НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Платформа для навчання навичкам екстреної медичної допомоги

Виконав : студент __4__ курсу, групи ІМ-11
(шифр групи)

_____Ватажко Михайло Андрійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Павлов В. Г._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Нікольський С. С._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____професор, д.т.н. Корнієнко Б. Я._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Ватажка Михайла Андрійовича

1. Тема проєкту *Платформа для навчання навичкам екстреної медичної допомоги*
керівник проєкту Павлов Валерій Георгійович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року №1705-с

2. Термін здачі студентом закінченого проєкту *2 червня 2025 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): актуальність теми розробки в умовах військового стану, існуючі програмні рішення для навчання екстреній допомозі, вибір інструментів та планування підходів розробки, опис процесу розробки платформи, загальне тестування розробленої платформи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, ER-діаграма бази даних (функціональна схема), алгоритм дій програмного забезпечення (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Нікольський С. С.		

7. Дата видачі завдання «10 вересня» 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	26.08.2024-25.01.2025	
2.	*Вивчення та аналіз завдання*	02.03.2025-24.04.2025	
3.	*Розробка архітектури та загальної структури системи*	27.04.2025-29.04.2025	
4.	*Розробка структур окремих підсистем*	30.04.2025-06.05.2025	
5.	*Програмна реалізація системи*	07.05.2025-19.05.2025	
6.	*Оформлення пояснювальної записки*	20.05.2025-02.06.2025	
7.	*Захист програмного продукту*	05.06.2025	
8.	*Передзахист*	06.06.2025	
9.	*Захист*	19.06.2025	

Студент-дипломник _____ Михайло ВАТАЖКО
(підпис)

Керівник проєкту _____ Валерій ПАВЛОВ
(підпис)

АНОТАЦІЯ

У дипломному проєкті розроблено інтерактивну веб-платформу для навчання навичкам екстреної медичної допомоги. Платформа реалізована з використанням сучасних технологій – Next.js, React, TailwindCSS, Prisma, PostgreSQL, Docker. Користувачі можуть проходити навчальні курси, виконувати симуляції екстрених ситуацій, а також користуватися швидким пошуком, у тому числі голосовим. Оригінальною частиною проєкту є впровадження інтерактивних симуляцій із покроковими діями реальних екстрених медичних сценаріїв, що дозволяє максимально наблизити навчання до реальних умов. У роботі наведено аналіз актуальності теми, описано архітектуру, структуру бази даних, основні компоненти клієнтської та серверної частин, а також результати тестування працездатності платформи. Основна увага приділена зручності користувача, безпеці та можливості подальшого розвитку системи.

Ключові слова: екстрена медична допомога, веб-платформа, симуляції, відеоінструкції, навчання.

ANNOTATION

This Bachelor's project presents the development of the interactive web platform for teaching emergency medical skills. The platform is built using modern technologies such as Next.js, React, TailwindCSS, Prisma, PostgreSQL and Docker. Users can take educational courses, perform emergency situation simulations, and use a fast search including voice input. The original feature of the project is the implementation of interactive simulations with step-by-step video guidance for real emergency medical scenarios, which brings the learning process as close as possible to real-life conditions. The work includes an analysis of the topic’s relevance, a description of the architecture, database structure, main client and server components, as well as the results of platform performance testing. Special attention is paid to user convenience, security, and the potential for further system development.

Keywords: emergency medical care, web platform, simulations, video instructions, training.

Справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Платформа для навчання	4		
			навичкам екстреної			
			медичної допомоги			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Платформа для навчання	125		
			навичкам екстреної			
			медичної допомоги			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Платформа для навчання	1		
			навичкам екстреної			
			медичної допомоги			
			Структурна схема системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Платформа для навчання	1		
			навичкам екстреної			
			медичної допомоги			
			ER-діаграма бази даних			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Платформа для навчання	1		
			навичкам екстреної			
			медичної допомоги			
			Алгоритм дій			
			програмного забезпечення			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Платформа для навчання	151		
			навичкам екстреної			
			медичної допомоги			
			Лістинг програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	Лист	№ докум.	Підп	Дата	
Розробив		Ватажко М. А.			*Платформа для навчання навичкам екстреної медичної допомоги* Опис альбому
Перевірив		Павлов В. Г.			
Реценз.		Корнієнко Б. Я.			
Н. Контр.		Нікольський С. С.			
Затв.		Новотарський М. А.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ "КПІ" ФІОТ ІМ-11

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для навчання навичкам екстреної медичної допомоги*»

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку інтерактивної платформи для навчання з надання навичок екстреної медичної допомоги, яка імітує широкий спектр ситуацій – від тяжких травм до невідкладних станів, таких як: зупинка серця, кровотечі, тяжкі опіки, анафілактичний шок, переломи, втручання при ДТП, теплових ударах та інших екстрених випадків.

Областю застосування платформи є підготовка медиків, навчання рятувальників, тренування волонтерів, освіта студентів-медиків, підвищення обізнаності громадян, вдосконалення навичок педагогів та тренерів з безпеки життєдіяльності.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної платформи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної роботи є створення інтерактивної платформи зі зручним інтерфейсом для навчання навичкам екстреної медичної допомоги, яка дозволить ефективно опановувати невідкладні процедури та поширювати медичну грамотність серед населення.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту з цієї теми є: офіційні документації, медичні стандарти, науково-технічна література, веб-сайти публікацій та статей у мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена платформа має виконувати такі вимоги:

- Простий та зручний інтерфейс.
- Локалізація для українських та іноземних користувачів.
- Реєстрація та авторизація користувачів за безпечними стандартами.
- Розподіл ролей користувачів з різним рівнем доступу до функціоналу.
- Швидкий пошук та фільтрація навчального контенту.
- Управління навчальними матеріалами.
- Підтримка різних форматів контенту (відео, тексти, тести, документи).
- Реалістичні сценарії екстрених ситуацій з можливістю повторного відпрацювання алгоритмів дій.
- Адаптація під різні типи пристроїв для зручності використання у будь-яких умовах.

5.2. Вимоги до програмного забезпечення

- ОС Windows, macOS чи Linux.
- Сучасний веб-браузер.

5.3. Вимоги до апаратної частини

- ЦП з кількістю ядер не менше ніж 4.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.
- Екран з роздільною здатністю не менше ніж 1280×720 пікселів.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	26.08.2024-25.01.2025
Вивчення та аналіз завдання	02.03.2025-24.04.2025
Розробка архітектури та загальної структури системи	27.04.2025-29.04.2025
Розробка структур окремих частин системи	30.04.2025-06.05.2025
Програмна реалізація системи	07.05.2025-19.05.2025
Виправлення помилок	20.05.2025-02.06.2025
Оформлення пояснювальної записки	08.06.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для навчання навичкам екстреної медичної допомоги*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ЕМД	Екстрена медична допомога
ДТП	Дорожньо-транспортна пригода
СЛР	Серцево-легенева реанімація
НС	Надзвичайна ситуація
МОЗ	Міністерство охорони здоров'я України
EMS	(Emergency Medical Services) Швидка медична допомога
NHS	(National Health Service) Національна служба охорони здоров'я Великобританії
АЕМД	Агенція Екстреної Медичної Допомоги
CAPCE	(Commission on Accreditation for Pre-Hospital Continuing Education) Комісія з акредитації безперервної догоспітальної освіти
LMS	(Learning management system) Система управління навчанням
FOAMed	(Free Open Access Medical Education) Безкоштовна медична освіта з відкритим доступом
PDF	(Portable Document Format) Портативний формат документа
URL	(Uniform Resource Locator) Уніфікований локатор ресурсів
API	(Application Programming Interface) Прикладний програмний інтерфейс
IDE	(Integrated Development Environment) Інтегроване середовище розробки
HTTP	(HyperText Transfer Protocol) Протокол передачі гіпертексту
JSON	(JavaScript Object Notation) Нотація об’єктів JavaScript
БД	База даних
ACID	(Atomicity, Consistency, Isolation, Durability) Атомарність, Узгодженість, Ізоляція, Довговічність
СУБД	Система керування базами даних

ORM	(Object-Relational Mapping) Об'єктно-реляційне відображення
SQL	(Structured Query Language) Структурована мова запитів
YAML	(YAML Ain’t Markup Language) YAML не є мовою розмітки
UI	(User Interface) Користувацький інтерфейс
SDK	(Software Development Kit) Комплект для розробки програмного забезпечення
JWT	(JSON Web Token) Веб-токен JSON
SMTP	(Simple Mail Transfer Protocol) Простий протокол передачі пошти
DOM	(Document Object Model) Об'єктна модель документа
CSS	(Cascading Style Sheets) Каскадні таблиці стилів
UUID	(Universally Unique Identifier) Універсальний унікальний ідентифікатор
CORS	(Cross-Origin Resource Sharing) Спільне використання ресурсів з різних джерел
CRUD	(Create Read Update Delete) Створити читати оновити видалити

ВСТУП

У сучасному світі навички екстреної медичної допомоги набувають критичного значення, особливо в умовах військового стану в Україні, де цивільне населення часто опиняється в ситуаціях, що вимагають невідкладних медичних дій. У багатьох дослідженнях показано, що проведення СЛР свідками або батьками підвищує шанси виживання на 50-500% (іншими словами, виживання збільшується в 0.5-5 разів порівняно з відсутністю допомоги) [1]. Це підтверджує, що своєчасна спеціалізована допомога суттєво підвищує шанси на виживання навіть при критичних станах, передумовою яких, наприклад, може бути обстріл населених пунктів, мінні-вибухи або інші обставини.

Рівень обізнаності населення щодо протоколів екстреної допомоги залишається надзвичайно низьким, а існуючі освітні ресурси часто недоступні широкому колу людей. Метою даної дипломної роботи є розробка інтерактивної платформи для навчання навичкам екстреної медичної допомоги, що забезпечить доступ до структурованих курсів і симуляцій як для фахівців, так і для немедичних осіб.

Таким чином, платформа буде спрямована на подолання розриву між теоретичними знаннями та практичними навичками через систематизацію матеріалів та впровадження інтерактивних елементів для закріплення знань.

Дипломний проєкт відповідає на нагальний запит українського суспільства щодо підвищення рівня готовності до дій у надзвичайних ситуаціях воєнного характеру та може бути використаний медичними закладами, підрозділами територіальної оборони, волонтерськими організаціями та широкими верствами населення.

РОЗДІЛ 1. АКТУАЛЬНІСТЬ ТЕМИ ТА ОГЛЯД СУЧАСНИХ ПЛАТФОРМ ДЛЯ НАВЧАННЯ НАВИЧКАМ ЕМД

1.1 Огляд системи ЕМД

1.1.1 Визначення поняття та його роль

Екстрена медична допомога (ЕМД) – це медична допомога, що потребує термінових дій медичних працівників, спрямованих на порятунок життя та здоров'я людини, яка перебуває в невідкладному стані, спричиненому хворобою, травмою або іншими чинниками [2].

Принципово ЕМД відрізняється від звичайних медичних послуг своєю невідкладністю, високим пріоритетом та безоплатністю для пацієнта, включно з діями під час НС [3].

Основними принципами функціонування системи є: постійна готовність, цілодобове реагування, доступність та безоплатність допомоги, її своєчасність, якість, безперервність надання, а також «екстериторіальність» – можливість виїзду бригад за межі області при потребі [3].

Отже, мета ЕМД – збереження життя людини, що реалізується через екстрену стабілізацію стану пацієнта та забезпечення його транспортування до лікарні з необхідним медичним супроводом.

1.1.2 Історія становлення та трансформації ЕМД-системи

У світі зачатки організованої допомоги при невідкладних станах сягають ще античності та релігійних благодійних ініціатив (відображено, зокрема, у біблійній притчі про доброго самарянина).

Перший цивільний підрозділ «рятувальної станції» було відкрито 1865 р. у Цинциннаті (США) [4]. З розвитком моторного транспорту та стерильних мед. технологій до середини XX ст. сформувався сучасний EMS (Emergency Medical Services): у США – завдяки впровадженню 911 і стандартам від Національного управління безпеки дорожнього руху (NHTSA) [5].

У Великобританії система розвивалася у рамках NHS (National Health Service), з національним номером екстреного виклику 999/112 і суворими нормативами швидкості реагування [6]. Військові конфлікти, особливо перші дві світові війни, значно прискорили появу швидких евакуаційних підрозділів з медиками на полі бою.

В Україні зародження «швидкої допомоги» датується початком XX ст. Перший «Рятувальний пункт» у Києві було створено 1881 р., а першу регулярну Рятувальну станцію швидкої допомоги відкрито 30 червня 1902 р. у Києві завдяки зусиллям громадських діячів [7].

Після Другої світової війни система швидкої допомоги розгорнулась у межах радянської медицини: до 1983 р. в УРСР діяло 88 станцій та 663 відділи швидкої допомоги, де працювало близько 4000 лікарів і 5000 медсестер; автопарк налічував приблизно 4000 санітарних автомобілів. У 1970-1980-х роках сформувалися спеціалізовані бригади (реанімаційні, кардіологічні, токсикологічні тощо) [7].

Після здобуття незалежності система ЕМД зберегла державну модель підпорядкування МОЗ (Міністерство охорони здоров'я), але потребувала суттєвої модернізації. У 1997 р. було засновано Український науковий центр екстреної медичної допомоги та медицини катастроф, який функціонував на базі Київської швидкої допомоги [8].

Ґрунтовна реформа ЕМД в Україні розпочалась із прийняттям у 2012 р. Закону «Про екстрену медичну допомогу». Закон передбачав створення Центрів ЕМД і медицини катастроф в областях та містах, впровадження єдиних диспетчерських служб та єдиної системи координування викликів [9].

Завдяки реформі було запроваджено єдині стандарти та нормативи забезпечення бригад засобами і транспортом, проте їхнє впровадження відбувалося нерівномірно: в регіонах, де реформу виконано повністю, система працює успішно; в інших – залишаються суттєві проблеми [9].

Зокрема, локальні дослідження констатують критичний стан автопарку: за даними медиків Чернігівщини, на початку 2020-х років близько 60% машин швидкої допомоги потребували капітального ремонту або списання, що значно уповільнювало реагування та прибуття на виклики [9].

З початку повномасштабної війни матеріально-технічну базу ЕМД було посилено завдяки міжнародній допомозі. Наприклад, у 2024 р. Україна отримала понад 50 сучасних автомобілів екстреної допомоги (дві партії по 27 машин) від уряду Об'єднаних Арабських Еміратів [10].

Працівники ЕМД проходять інтенсивні тренінги з надання допомоги в екстремальних умовах (тактична медицина, лікування травм від вибухів тощо) як в Україні, так і за кордоном. Багато команд підвищували кваліфікацію на спеціалізованих навчаннях у Польщі. Нині переважна більшість бригад «швидкої» працює за єдиними протоколами, а їхні автомобілі обладнані як мобільні реанімаційні відділення [11].

У зонах активних бойових дій система ЕМД зазнає екстремальних навантажень і ризиків. Наприклад, на правобережжі Херсонщини бригади «швидкої» у середньому обробляють 150-180 викликів на добу під постійними обстрілами; медики виїжджають у бронежилетах і касках [12].

Щоденна робота в умовах війни стає серйозним випробуванням для медичних працівників. З початку повномасштабного вторгнення майже 500 автомобілів ЕМД були пошкоджені, знищені або захоплені окупантами [10]. Це створює дефіцит транспортних засобів, незважаючи на міжнародні поставки нових автомобілів швидкої допомоги. Броньовані медичні автомобілі для евакуації поранених доступні лише на рівні близько 55% від

необхідної кількості [13], що суттєво ускладнює вивезення тяжкопоранених з лінії фронту під ворожим вогнем.

Робота в умовах постійних обстрілів, тривалі зміни та значна кількість критичних травм призводять до суттєвого емоційного та фізичного навантаження на медичний персонал. Для збереження їхньої працездатності та психологічного здоров'я необхідна регулярна психологічна підтримка, планова ротація кадрів та посилена підготовка для забезпечення безпеки й ефективності бригад [12].

Комплексний аналіз історичної трансформації системи ЕМД та її адаптації до сучасних викликів підкреслює гостру необхідність розробки інноваційних освітніх інструментів. Ситуація демонструє потребу у створенні сучасної інтерактивної медичної платформи, яка дозволить ефективно навчати навичкам надання екстреної допомоги не лише медиків, але й будь-якого сатусу громадян, що особливо актуально в умовах підвищених ризиків воєнного часу.

1.2 Актуальність теми розробки платформи ЕМД

У 2025 році в Україні актуальність теми розробки платформи ЕМД залишається надзвичайно високою, що зумовлено численними викликами, зокрема, наявними технічними проблемами та нестачею кваліфікованих кадрів.

Однією з головних причин актуальності розробки інтерактивної платформи для навчання навичкам ЕМД є потреба у швидкій адаптації медиків до сучасних реаліях роботи, особливо в умовах війни. Багато бригад «швидкої допомоги» зіштовхуються не лише з постійною загрозою обстрілу, а й з величезним обсягом викликів щоденно, що вимагає максимально ефективних дій та готовності при НС.

Додатковим фактором є включення парамедиків та екстрених медичних техніків до складу бригад екстреної медичної допомоги з 2025 року [14]. Таке рішення передбачає оновлення та стандартизацію навчальних програм для медиків по всій країні, включно з віддаленими та небезпечними регіонами.

Також варто зазначити той факт, що ситуація в Україні, особливо при обстрілах чи бойових зіткненнях, демонструє термінову потребу у навчанні не лише медиків, а й цивільного населення. Кожен може зіткнутися з необхідністю надання допомоги в екстренній ситуації.

Враховуючи ці обставини, дистанційна платформа навчання стане ефективним інструментом у навчанні людей навичкам ЕМД під час НС, що сприятиме підвищенню загального рівня готовності населення.

1.3 Огляд сучасних платформ для навчання навичкам ЕМД

Останнім часом система ЕМД удосконалюється швидкими темпами. Наприклад, у межах однієї з останніх реформ було запроваджено дистанційні онлайн-курси на платформі Академії Національної служби здоров'я України (НСЗУ) для спеціалістів ЕМД [15].

Враховуючи це, огляд сучасних платформ для навчання навичок ЕМД дозволить визначити сильні та слабкі сторони існуючих рішень, що дасть змогу окреслити напрямки вдосконалення нової платформи. У цьому пункті буде розглянуто чотири актуальні платформи (дві українського та дві іноземного походження), орієнтовані на підвищення навичок в екстреній медицині.

1.3.1 Агенція Екстреної Медичної Допомоги

Агенція екстреної медичної допомоги (АЕМД) – це українська медична навчальна платформа, яка публікує медичні новини, навчальні довідники,

лекції, рекомендації, проводить дистанційне навчання, тренінги, майстер класи, вебінари, тестування для перевірки знань, ситуаційні практичні задачі та багато іншого [16].

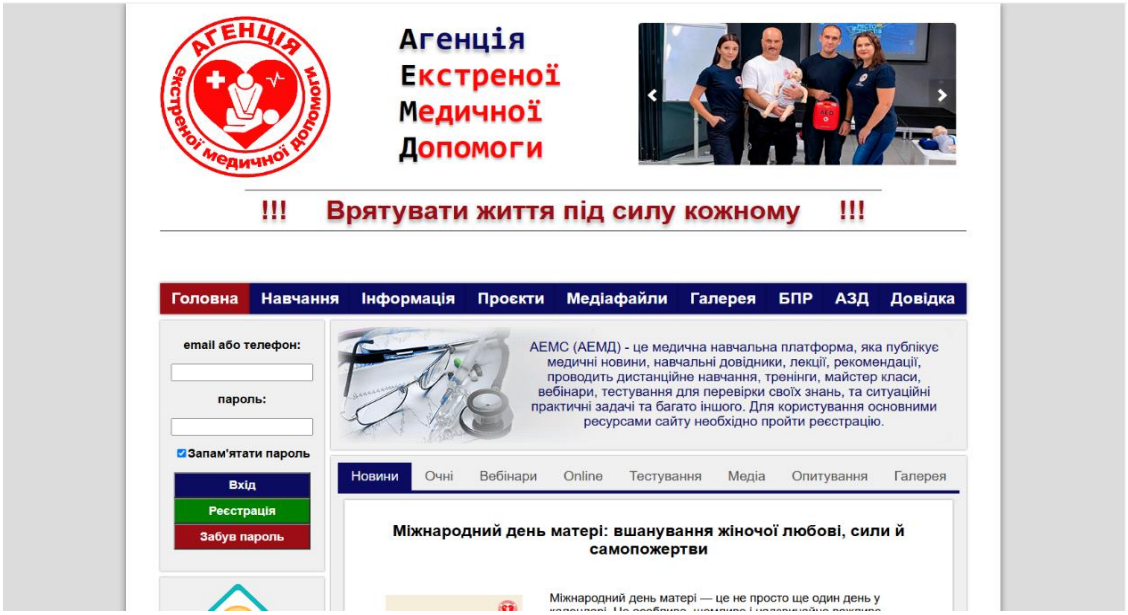


Рисунок 1.1 – Головна сторінка платформи АЕМД

Переваги:

- Широкий спектр матеріалів з тематики ЕМД (новини, статті, лекції, тестові завдання, курси тощо).
- Офіційний вміст від експертів; актуальні протоколи та рекомендації для медиків.
- Організація очних симуляційних тренінгів та майстер-класів.
- Безкоштовний доступ до більшості інформаційних ресурсів.
- Орієнтовність на практиків ЕМД – лікарів, фельдшерів, медсестер та студентів медичних закладів.

Недоліки:

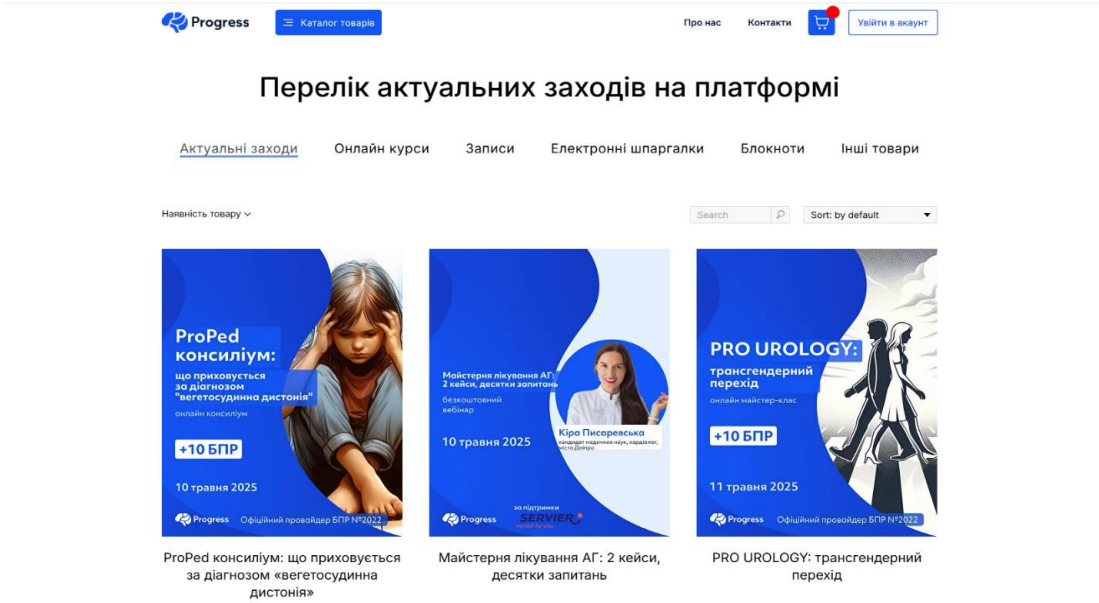
- Усі матеріали лише українською мовою, що обмежує аудиторію іноземних фахівців.
- Необхідність реєстрації для користування основними ресурсами.
- Платне проходження навчальних курсів.

- Відсутність інтерактивних онлайн-симуляцій з покроковим навчанням: платформа більш інформаційна, ніж тренувальна.
- Більшість навчальних матеріалів ЕМД орієнтовані на людей із медичною освітою або досвідом.
- Інтерфейс сайту доволі застарілий та перевантажений функціоналом, що може ускладнити навігацію для користувачів.

Отже, платформа АЕМД є корисним інструментом для практиків ЕМД в Україні, оскільки пропонує спеціалізований контент та велику кількість різних інформаційних ресурсів. Водночас для осіб без медичної освіти платформа залишається менш доступною через відсутність безкоштовних навчальних курсів та інтерактивних онлайн-симуляцій.

1.3.2 Progress

Progress – це українська освітня платформа для фахівців в сфері охорони здоров'я, які постійно прагнуть розвивати свої знання та навички. Це зручний ресурс, де можна знайти чинні у світі та Україні медичні протоколи, гайдлайни та рекомендації, останні новини зі світу медицини українською мовою [17].



ProPed консиліум: що приховується за діагнозом «вегетосудинна дистонія»
Майстерня лікування АГ: 2 кейси, десятки запитань
PRO UROLOGY: трансгендерний перехід

Рисунок 1.2 – Головна сторінка платформи Progress

Progress містить каталог онлайн-курсів, семінарів і вебінарів різної тематики; зокрема, тут доступний курс PRO EMERGENCY із невідкладних станів [18].

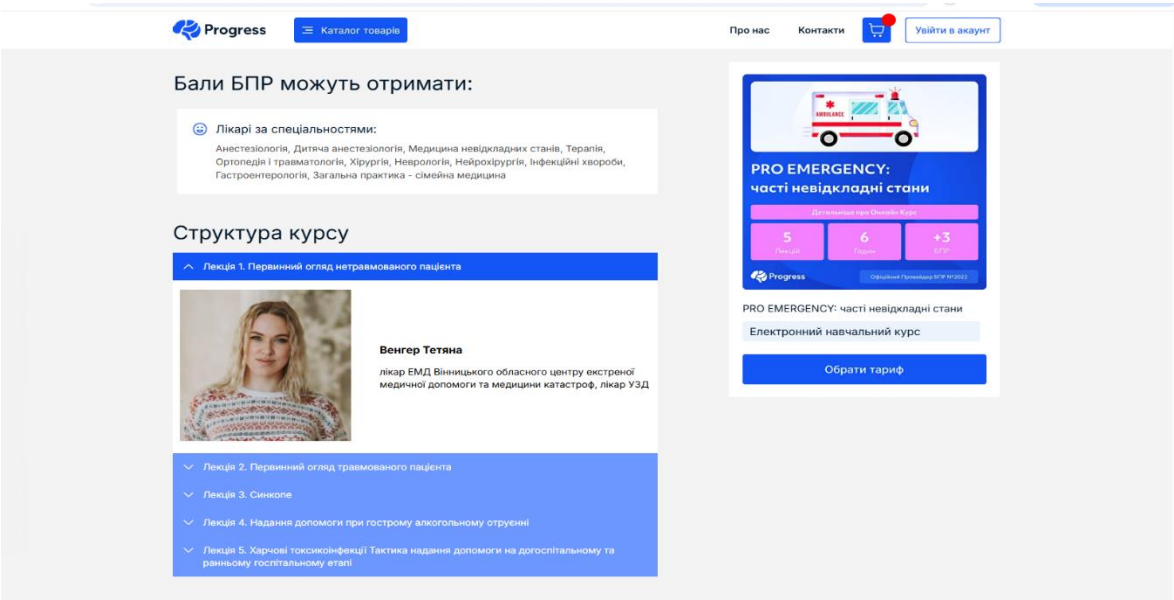


Рисунок 1.3 – Сторінка курсу PRO EMERGENCY на платформі Progress

Переваги:

- Сучасний інтерфейс.
- Можливість проходження курсів у зручному онлайн-форматі.
- Наявність спеціалізованих курсів з екстреної медицини (наприклад, PRO EMERGENCY) та матеріалів для медиків і студентів.
- Зручна подача контенту: структуровані лекції, алгоритми, таблиці та інтерактивні тести.
- Забезпечення сертифікації БПР (балів безперервного професійного розвитку) за успішне завершення курсів.

Недоліки:

- Усі матеріали лише українською мовою.
- Переважно комерційна модель: багато курсів є платними, а безкоштовний контент обмежений.
- Недостатньо інтерактивних форматів навчання (мало тестів, немає онлайн-симуляцій).

- Обмежена кількість курсів для немедичної аудиторії: основна увага приділяється медичним фахівцям, що зменшує доступність платформи для широкої аудиторії.
- Зосередженість не лише на ЕМД: представлені курси з інших медичних тем, що може відволікати від фокусу на екстреній допомозі.

Отже, платформа Progress орієнтована на лікарів та студентів, які прагнуть оновити свої знання, зокрема в галузі невідкладної медицини. Вона пропонує якісний контент та зручну систему навчання в Інтернеті. Однак платний характер та різноманітність тем курсів обмежують її доступність і зосередженість виключно на екстреній медицині.

1.3.3 Lexipol

Lexipol – це міжнародна освітня платформа для служб екстреної допомоги (EMS) та інших служб безпеки. Вона надає сотні онлайн-курсів і відеоматеріалів, призначених для виконання нормативних вимог і періодичного підвищення кваліфікації персоналу; зокрема, Lexipol пропонує близько 700 курсів акредитованих CAPCE та більше 440 годин CE (академічних кредитів) з EMS [19].

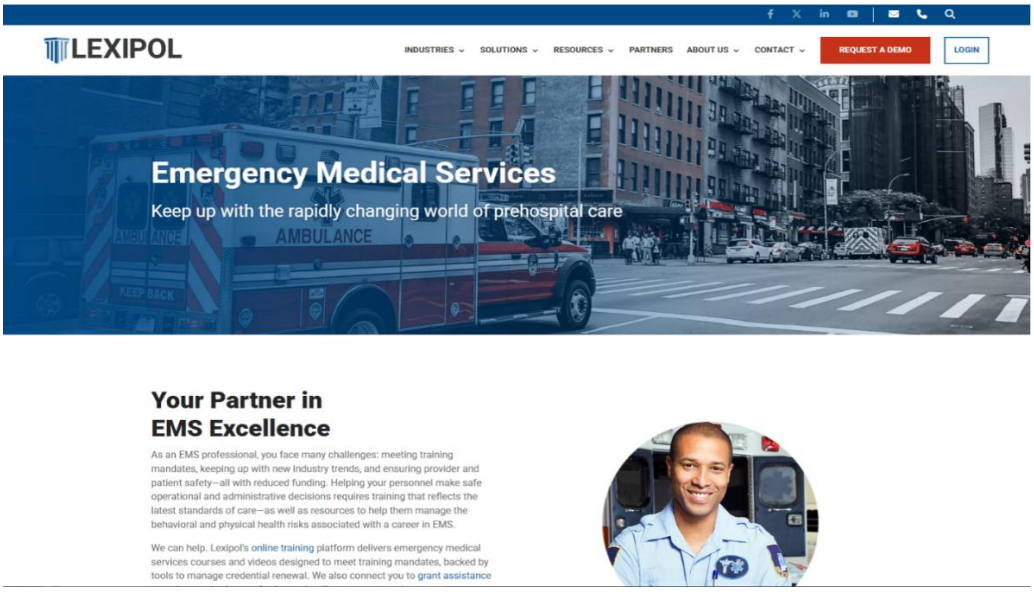


Рисунок 1.4 – Головна сторінка платформи Lexipol з EMS

Переваги:

- Дуже велика база навчальних матеріалів (відео, тестові питання, документи) з перевіреною якістю.
- Акредитація CAPCE гарантує визнання курсів службою сертифікації EMS, що забезпечує можливість поновлення ліцензії.
- Наявність мобільного застосунку та підтримка цілодобового навчання (можливість проходження курсів із мобільних пристроїв і планшетів).
- Інструменти для управління навчанням: контроль виконання курсів, автоматизація звітності та продовження сертифікації персоналу.

Недоліки:

- Платформа орієнтована на великі організації – вимагає підписки або корпоративного доступу (індивідуальному користувачеві важко отримати доступ).
- Матеріали переважно англійською мовою та заточені на стандарти США, що ускладнює використання платформи в інших країнах.
- Не передбачено можливості гостьового перегляду чи роботи без облікового запису – весь контент закритий для незареєстрованих користувачів.
- Інтерфейс і навігація розраховані на корпоративних адміністраторів, тому можуть бути складними для користувачів без досвіду роботи з LMS.

Отже, Lexipol – ефективний інструмент для фахівців у сфері EMC, який забезпечує повний цикл навчання з контролем якості. Платформа пропонує професійні курси з підтримкою сертифікації, але в основному призначена для організацій. Таким чином, вона може бути менш зручною для швидкого розповсюдження знань масовій аудиторії виключно через високі витрати й англомовність. Крім того, Lexipol не покриває потреб користувачів без реєстрації.

1.3.4 Médecine d’Urgence

Médecine d’Urgence (Невідкладна медицина) – це французька платформа, створена французьким експертом з невідкладної медицини Dr. Didier Thiercelin під егідою "Société Française de Médecine d’Urgence" (Французьке товариство невідкладної медицини) [20].

Рисунок 1.5 – Головна сторінка платформи Médecine d’Urgence

Платформа слугує «документарним простором», де зібрані референтні гайдлайни, рекомендації та протоколи, що стосуються саме ЕМД в долікарняному та післялікарняному середовищах [21].

Окрім текстових стандартів є добірки класифікацій, оцінкових шкал, щомісячний огляд публікацій, а також посилання на корисні медичні мобільні додатки. Вміст подано переважно французькою мовою і поступово оновлюється. Платформа повністю безкоштовна та реалізована за принципами FOAMed (Free Open Access Medical Education) – доступ до інформації відкритий для всіх [21].

Переваги:

- Великий обсяг авторитетних ресурсів з екстреної медицини (гайдлайни, протоколи, огляди) у центрі уваги.

- Постійні оновлення контенту.
- Повністю безкоштовний відкритий доступ за філософією FOAMed.
- Інтуїтивна рубрикова навігація.
- Матеріали покривають як стаціонарну, так і позалікарняну невідкладну допомогу.

Недоліки

- Контент суто французькою, що створює мовний бар'єр для міжнародних користувачів.
- Відсутність навчальних курсів чи інтерактивів (лише статичні текстові матеріали).
- Інформація суто професійна, мало пристосована для звичайних громадян.
- Для навчальних метеріалів немає швидкого пошуку, а фільтрація сильно обмежена.
- Платформа в одному єдиному екземплярі: її підтримка залежить від одного автора, що може призвести до нестабільності роботи.

Отже, Médecine d'Urgence є корисним довідником з постійним оновленням матеріалів. Сильною стороною є відкритість та доступність інформації. Проте платформа переважно спрямована на французько-мовних спеціалістів, що сильно обмежує її застосування на світовому ринку без локалізації. Відсутність інтерактивності та освітніх курсів зменшує її попит для користувачів, яким потрібно більш практичне навчання.

1.4 Аналіз порівняння платформ ЕМД

1.4.1 Таблиця порівняння

Нижче наведено порівняльну таблицю 1.1 чотирьох розглянутих платформ за ключовими критеріями їх функціональності та зручності.

Таблиця 1.1 – Порівняння розглянутих платформ

Назва / Критерій	АЕМД	Progress	Lexipol	Médecine d'Urgence
Основний вміст	Навчання ЕМД	Онлайн-курси з медицини	Регламенти навчання для EMS	Статті, протоколи з невідкладної допомоги
Орієнтованість на ЕМД	Повна (українська ЕМД)	Часткова (загальна медицина)	Повна (для служб EMS США)	Повна (французька ЕМД)
Цільова аудиторія	Громадяни, практики ЕМД	Студенти, медики	EMS-працівники	Французькі лікарі, студенти
Формат навчання	Курси, методички, вебінари	Онлайн-курси	Онлайн-тренінги, курси	Текстові матеріали
Навчальні матеріали для звичайних (немедичних) користувачів	Присутні (ситуаційні задачі, методичні матеріали)	Частково присутні (деякі відкриті вебінари)	Відсутні (не передбачені)	Відсутні (не надаються)
Доступ до навчальних матеріалів без реєстрації	Частковий (методичні матеріали)	Частковий (демо-матеріали)	Закритий	Відкритий
Наявність онлайн-курсів	Так	Так	Так	Ні
Наявність онлайн-симуляцій	Ні	Ні	Ні	Ні
Наявність навчальних відеоматеріалів	Так (у курсах)	Так (у курсах)	Так (у курсах, регламенти)	Ні
Наявність навчальних документацій	Так (методички, документи)	Так (лекції, PDF)	Так (регламенти)	Так (графіки, тексти)

Кінець таблиці 1.1

Назва / Критерій	АЕМД	Progress	Lexipol	Médecine d'Urgence
Наявність тестування знань	Так (у курсах)	Так (у курсах)	Так (у курсах)	Ні
Пошук/фільтрація навчальних матеріалів	Середня (повноцінний пошук є, мала кількість фільрів)	Висока (фільтрація за темами, категоріям, тегами)	Висока (фільтрація за тегами, категоріями)	Низька (відсутній повноцінний пошук)
Доступ до навчальних матеріалів	Переважно безкоштовний (онлайн-курси, вебінари – платні)	Платний (онлайн-курси, однак інколи бувають безкоштовні заходи)	Платний (доступ за підпискою)	Безкоштов-ний (відкриті статті та документи)
Завантаження навчальних матеріалів	Доступне (методички, PDF, відео)	Доступне (PDF, відео)	Доступне (PDF, відео)	Доступне (PDF, документи)
Управління контентом онлайн на платформі	Частково (заванта-ження методичок, документів)	Так (взаємодія з курсами)	Так (взаємодія з курсами)	Ні (не передбачено)
Сучасність та простота інтерфейсу	Низька (перевanта-жена, застаріла)	Висока	Середня (може бути складною без знань LMS)	Висока
Підтримка локалізації	Немає (українська)	Немає (українська)	Немає (англійська)	Немає (французька)
Мобільна підтримка	Так (веб-доступ)	Так (веб-доступ)	Так (додаток)	Так (веб-доступ)

1.4.2 Аналіз порівняння

Проаналізувавши таблицю 1.1, зроблено висновок, що жодна з розглянутих платформ (AEMД, Progress, Lexipol, Médecine d'Urgence) повноцінно не здатна задовольнити всі важливі потреби для ефективного навчання навичкам ЕМД.

Lexipol та Progress мають суто комерційний напрямок і переважно орієнтовані на професійну аудиторію. Такий підхід робить ці платформи менш привабливими для великої кількості користувачів, які прагнуть безкоштовно ознайомитися з базовими навичками ЕМД.

Médecine d'Urgence та AEMД у свою чергу мають обмежені або повністю відсутні функціональні можливості – наприклад, слабка пошукова спроможність або повна відсутність інтерактивних симуляцій. Ці обмеження роблять процес навчання менш наочним і суттєво знижують ефективність у вивченні матеріалу.

Усі користувачі потребують простих, інтуїтивно зрозумілих та доступних рішень для комфортного засвоєння практичних навичок ЕМД. У той же час доступ до більш складних матеріалів для медиків доцільно організовувати окремо, щоб не перевантажувати інформацією початківців.

Жодна з розглянутих платформ не має підтримки локалізації, що створює мовний бар'єр до навчального матеріалу користувачам з інших країн.

Важливо зазначити також те, що певні платформи, такі як Lexipol та Progress, вимагають обов'язкової реєстрації користувачів для доступу до навчальних матеріалів. Це рішення є розумним заходом для контролю якості освіти та персоналізації навчання. Проте навчальні ресурси не повинні бути цілковито заблоковані перед реєстраційною процедурою. Хорошим прикладом є платформа AEMД, де частина матеріалів доступна без необхідності увійти до системи. Варто забезпечити можливість користувачам швидко здобути базові навички ЕМД. Наприклад, при зупинці серця кожна

секунда має велике значення, тому доступ до інформації надання СЛР повинен бути якнайшвидшим.

Інтерфейс також є одним з найбільш значимих елементів ефективного навчання навичкам ЕМД. Наприклад, платформа Médecine d'Urgence має простий і зручний інтерфейс при використанні, однак ефективність навіть найзручнішого інтерфейсу значною мірою знижується за відсутності таких важливих функцій, як пошук та фільтрація. Саме ці інструменти допомагають швидко знаходити вкрай потрібну інформацію, особливо під час НС. Такий функціонал має стати невід'ємною частиною інтерфейсу.

Майбутня платформа значно перевершить розглянуті, якщо буде:

- Поєднувати найкращі рішення (онлайн-курси, підтримка мобільного формату).
- Переосмислювати стандартні підходи (інтерфейс зі зручним пошуком).
- Впроваджувати інноваційні рішення (інтерактивні симуляції, локалізація).
- Забезпечувати доступ до якісного матеріалу без зайвих обмежень як для новачків, так і для спеціалістів.

Завдяки описаним перевагам нова платформа навчання навичкам ЕМД стане важливим інструментом для підвищення готовності громадян до надання швидкої та ефективної ЕМД.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було освітлено важливість створення сучасної освітньою платформи на підставі аналізу існуючих навчальних рішень.

Було визначено значущість поняття ЕМД та його ключову роль у сучасному суспільстві разом з основними принципами функціонування цієї системи. Проаналізовано історичний шлях її розвитку з урахуванням особливостей України – від перших громадських ініціатив у XX століття до сучасних реформ і адаптації до військових умов.

На основі вивченої історії, виявлено актуальність у створенні інтерактивної платформи для швидкого навчання медичних працівників та громадян у ситуаціях підвищенної готовності як основний напрям розвитку цифрового навчання.

Було проаналізовано чотири сучасні платформи – АЕМД, Progress, Lexipol та Médecine d'Urgence, виявлено їх переваги та недоліки з подальшими висновками. Для упорядкування даних було створено порівняльну таблицю за ключовими критеріями: доступність матеріалу для навчання, формат навчального процесу, цільова аудиторія та функціонал платформи.

Аналіз даних показав, що жодна з існуючих платформ не відповідає усім потребам ефективного навчання навичкам ЕМД: переважно вони орієнтовані на професійну аудиторію, мають обмежений безкоштовний доступ або взагалі не мають інтерактивних симуляцій.

Отримані висновки показують, що потрібно створити нову платформу, яка поширить успішні практики існуючих аналогів, перегляне традиційні підходи та запропонує свою власну інноваційну стратегію. Це комплексне поліпшення допоможе усунути проблеми в галузі навчання навичкам ЕМД та забезпечить ефективну підготовку різних груп користувачів на випадок надзвичайних ситуацій.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПЛАТФОРМИ

2.1 Аналіз типів платформ

На етапі проєктування платформи ключовим моментом є вибір її типу. Існують три основні варіанти реалізації: веб-платформа, мобільна платформа та десктопна платформа.

У цьому підрозділі буде проаналізовано плюси та мінуси кожного з них, на основі аналізу складено порівняльну таблицю та обрано оптимальне рішення для реалізації платформи для навчання навичкам ЕМД.

2.1.1 Веб-платформа

Веб-платформа – це додаток, що працює через веб-браузер. Користувачі просто відкривають URL-адресу і користуються програмою без встановлення додаткового ПЗ [22].

Переваги:

- Кросплатформеність: веб-додаток запускається на будь-якій ОС (Windows, macOS, Linux) за наявності сучасного браузера.
- Доступність: користувачі мають доступ до платформи звідусіль за наявності лише інтернет-з’єднання.
- Відсутність інсталяції: немає потреби завантажувати та встановлювати окремий клієнт – оновлення відбуваються на сервері й відразу стають доступні всім користувачам.

Недоліки:

- Залежність від інтернету: веб-застосунки потребують постійного підключення, без доступу до мережі функціональність недоступна.

- Навантаження на мережу: при складних операціях вимагають передачі даних по мережі, через що може страждати швидкодія й відчуття відгуку інтерфейсу.

Отже, веб-платформа забезпечує максимальну універсальність і зручність доступу: немає прив’язки до конкретного пристрою чи ОС, просте оновлення та підтримка. Доцільно використовувати веб-платформу при широкому охопленні користувачів і гнучкому розгортанню.

2.1.2 Мобільна платформа

Мобільна платформа – це додаток, призначений для роботи на портативних пристроях, наприклад смартфонах або смарт-годинниках [23]. Він пишеться під конкретні ОС (iOS/Android) і поширюється через магазини додатків (App Store, Google Play).

Переваги:

- Доступність і мобільність: мобільний додаток дозволяє користувачу навчатися будь-де, незалежно від місцезнаходження.
- Інтуїтивний інтерфейс: додатки для мобільних оптимізовані під тач-управління, пропонують зручний та персоналізований інтерфейс.
- Робота офлайн: мобільні додатки можуть зберігати дані локально і працювати без інтернет-зв’язку.

Недоліки:

- Складність розробки: підтримка мобільної платформи потребує розробки окремих версій під iOS та Android.
- Підтримка та оновлення: застосунок потрібно регулярно оновлювати в магазинах додатків; розповсюдження нових версій потребує проходження перевірок і може затримуватися.
- Безпека: мобільні програми зберігають облікові дані та персональну інформацію локально, тому стають вразливими до атак.

- Розповсюдження: мобільні додатки незручно знаходити без реклами, оскільки потрібна промоція в App Store/Google Play, тоді як веб-додатки відкриваються за посиланням.

Отже, мобільна платформа добре підходить для задач, де важлива портативність та залучення через смартфон. Однак для масштабного навчального сервісу це потребує значних ресурсів на розповсюдження та підтримку двох операційних систем (iOS та Android).

2.1.3 Десктопна платформа

Десктопна платформа – це класичний локальний додаток, встановлений на персональному комп’ютері або ноутбуці. Програма виконується прямо на ПК, має доступ до усіх ресурсів системи і не вимагає браузера [24].

Переваги:

- Висока продуктивність: локальні програми оптимізовані під потужність комп’ютера; взаємодіють з апаратними компонентами, тому працюють швидко та плавно.
- Робота офлайн: десктопний застосунок не залежить від мережі – його можна запускати всюди.
- Багатовіконність і щільність інформації: на великому екрані ПК зручно працювати з кількома вікнами одночасно і розміщувати багато інформації на екрані.
- Контроль над даними: всі дані зберігаються на локальному комп’ютері, що дає користувачам більший контроль.

Недоліки:

- Аппаратна залежність: оскільки застосунок створюється для певної ОС та конфігурації, потрібно робити окремі версії для Windows, macOS тощо.

- Обмежена доступність: десктопні програми прив’язані до конкретного комп’ютера; користувачі можуть працювати лише на тих ПК, де встановлено ПЗ.
- Оновлення: доводиться поширювати окремо (новий інсталятор або патч), що часто потребує втручання адміністратора системи.

Отже, десктопна платформа забезпечує максимальну продуктивність та можливості, але найменш зручна для широкого розповсюдження. Її виправдано використовувати там, де потрібні дуже потужні локальні обчислення чи робота в офлайні, проте для навчальної системи, орієнтованої на масову аудиторію, це призведе до складнощів із доступністю.

2.1.4 Таблиця порівняння

Нижче наведено порівняльну таблицю 2.1, яка наочно демонструє переваги та недоліки трьох розглянутих типів платформ за основними критеріями.

Таблиця 2.1 – Переваги та недоліки типів платформ

Назва / Критерій	Веб-платформа	Мобільна платформа	Десктопна платформа
Кросплатформеність	+	±	−
Доступність	+	+	−
Легкість підтримки	+	−	−
Продуктивність	−	+	+
Зручність масштабування	+	−	−
Безпека даних	±	−	±
Офлайн-режим	−	+	+
Легкість тестування	+	−	±
Оновлення	+	−	−

2.1.5 Вибір найкращого рішення

На основі проведеного порівняльного аналізу таблиці 2.1 було визначено, що поганим рішенням є десктопна платформа. Хоч вона й пропонує добру швидкодію, але суттєво поступається за доступністю, вимагає складної підтримки та не мобільна.

Кращим рішенням вже є мобільна платформа. Вона зручна для користувачів «на ходу», але все так само не охоплює всі можливі сценарії навчання, обмежуючись лише пристроями з сенсорними екранами.

Найкращим рішенням реалізаціїї платформи для навчання навичкам ЕМД стане веб-орієнтованість, оскільки такий підхід поєднує найважливіші переваги: кросплатформеність, централізоване оновлення, масштабованість, простота у розробці та підтримці.

2.2 Аналіз архітектурних підходів

Наступним важливим етапом є вибір архітектурного підходу. Архітектурний підхід – це стратегія, яка визначає загальну структуру системи, її основні компоненти, способи їх взаємодії та принципи організації. Серед найпоширеніших архітектурних підходів у сучасній розробці ПЗ виділяють монолітну та мікросервісну архітектури [25].

У цьому підрозділі буде проведено аналіз цих архітектур з метою визначення їх сильних і слабких сторін, що у подальшому дозволить обґрунтовано обрати оптимальний варіант для реалізації платформи проєкту.

2.2.1 Монолітна архітектура

Монолітна архітектура – це традиційна модель побудови ПЗ, коли весь функціонал програми об’єднаний в одному кодовому сховищі. У такій

архітектурі всі компоненти (інтерфейс, логіка, доступ до даних) інтегровані в єдину систему [26].

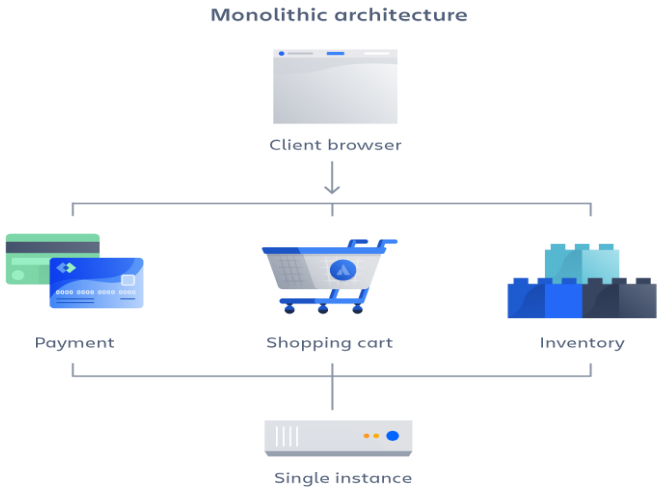


Рисунок 2.1 – Структура монолітної архітектури

Переваги:

- Простота розробки та розгортання: усі компоненти (інтерфейс, логіка, доступ до даних) інтегровані в єдину систему, що спрощує процес розробки та розгортання.
- Єдина кодова база: наявність однієї кодової бази полегшує відлагодження та тестування.
- Швидкий початок розробки: завдяки меншій кількості компонентів, початкова розробка відбувається швидше.
- Просте оновлення: оновлення програми відбувається одразу для всіх її частин, що зменшує складність управління версіями.

Недоліки:

- Складність масштабування: зі збільшенням обсягу функціоналу система стає «важкою» для масштабування; її важко розбити і розгорнути по частинах.
- Повне перезбирання при змінах: будь-яка зміна часто потребує повного перезбирання та повторного розгортання всього додатку.

- Ускладнення підтримки: ускладнення моноліту призводить до заплутаності коду і зростання трудомісткості підтримки.

Отже, моноліт є доцільним рішенням для простих проєктів зі стабільним обсягом функцій, де необхідність частого масштабування низька.

2.2.2 Мікросервісна архітектура

Мікросервісна архітектура – це модель, за якою додаток розбивається на набір невеликих незалежних сервісів, кожен із яких відповідає за певний набір функцій. Кожен мікросервіс має окремий життєвий цикл, кодову базу та (можливо) власну базу даних. Така незалежність дозволяє поєднувати різні технології в одному проєкті та оновлювати сервіси окремо [27].

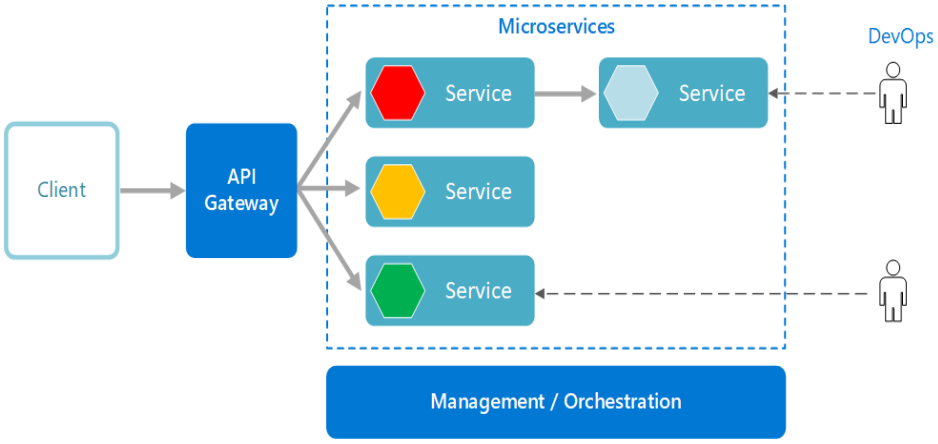


Рисунок 2.2 – Структура мікросервісної архітектури

Переваги:

- Модульність та масштабованість: кожен сервіс є ізольованим, тому збій одного не призводить до падіння всієї системи.
- Незалежне масштабування: можна масштабувати окремі сервіси, які потребують додаткових ресурсів (наприклад, критичні компоненти).
- Паралельна розробка: різні команди можуть працювати над окремими сервісами незалежно одна від одної, що прискорює розробку.
- Гнучкість: можна оновлювати чи змінювати один сервіс без втручання в решту системи.

Недоліки:

- Високі інфраструктурні вимоги: потрібна складна інфраструктура для розгортання (хостинг, контейнеризація, логування, моніторинг).
- Ускладнені міжсервісні взаємодії: потрібні компоненти, такі як API-шлюз, балансувальник навантаження, системи керування версіями.
- Складність тестування: налагодження системи ускладнене, адже потрібно відтворювати взаємодію між великою кількістю сервісів.
- Збільшення затримок: між окремими сервісами можуть виникати додаткові затримки у передачі даних, що впливає на загальну швидкодію системи.

Отже, мікросервісна архітектура забезпечує високу гнучкість, стійкість і масштабованість, але вимагає ретельного управління сервісами.

2.2.3 Таблиця порівняння

Нижче наведено порівняльну таблицю 2.2, яка демонструє переваги та недоліки двох розглянутих архітектурних підходів для веб-платформ.

Таблиця 2.2 – Переваги та недоліки архітектурних підходів

Назва / Критерій	Моноліт	Мікросервіси
Масштабованість	–	+
Швидкість розробки	+	–
Простота реалізації	+	–
Підтримка	–	+
Підхід для одного розробника	+	–
Продуктивність	+	–
Легкість розгортання	+	–
Тестування	+	–
Гнучкість	–	+

2.2.4 Вибір архітектури

На основі проведеного порівняльного аналізу таблиці 2.2 було визначено, що для розробки платформи навчання навичкам ЕМД найбільш раціональним рішенням є використання монолітної архітектури.

Швидка розробка, проста реалізація та легке розгортання – гарантовані переваги моноліту, які є надзвичайно важливими в умовах обмеженого часу. Крім того, він краще підходить для одного розробника, забезпечуючи вищу продуктивність і спрощуючи процес тестування.

Звісно, мікросервіси мають свої переваги у гнучкості чи масштабованості, проте їх складна реалізація, міжсервісні залежності та потреба в командній роботі роблять цей підхід менш доцільним для індивідуального проєкту.

2.3 Аналіз технологій реалізації

У цьому підрозділі було розглянуто основні технології, які будуть використовуватися під час розробки платформи, а саме: мову програмування, фронтенд та бекенд.

2.3.1 Мова програмування

При виборі мови програмування для реалізації веб-платформи навчання навичкам ЕМД важливо орієнтуватися на її функціональні можливості.

Основні критерії для оцінки виступають:

- Кросплатформеність.
- Продуктивність (простота синтаксису, підтримка сучасними інструментами розробки).
- Наявність фреймворків.
- Зручність тестування та відлагодження.

На основі цих критеріїв було розглянуто чотири мови програмування: TypeScript, JavaScript, Python та Java.

TypeScript (TS) – це мова програмування, розроблена Microsoft, яка є надбудовою над JavaScript. Вона транслюється у стандартний JavaScript та додає статичну типізацію, що спрощує роботу над великими проєктами. TS здатний працювати як на клієнті (в браузері), так і на сервері (Node.js) [28].

JavaScript (JS) – це мова програмування, яка є основою для веб-технологій. Вона виконується в браузері і відповідає за інтерактивність веб-сторінок, дозволяючи взаємодіяти з користувачами та обмінюватися даними з серверами [29].

Python – це мова програмування, відома своєю простотою та великим числом бібліотек для різних задач. Популярні фреймворки для веб-розробки на Python включають Django та Flask [30].

Java – це об’єктно-орієнтована мова програмування, яка з'явилася в 1995 році. Вона компілюється у байт-код та виконується на Java Virtual Machine. Java часто використовується на великих та корпоративних системах [31].

На основі всього вищеописаного для проєкту було обрано мову програмування TypeScript. Нижче наведено причини такого вибору:

- TypeScript проти JavaScript: На відміну від JavaScript, TS має статичну типізацію, що допомагає виявити помилки раніше – під час компіляції. Це полегшує технічне обслуговування великих проєктів завдяки автодоповненню та перевірки типів у сучасних IDE.
- TypeScript проти Python: Хоча Python зручний для сервера і має багато бібліотек, він переважно використовуються в бекенді. TypeScript же працює в веб-середовищах для фронтенду та бекенду.
- TypeScript проти Java: Обидві мови мають об'єктно-орієнтований підхід, але TypeScript зазвичай вимагає менше коду для тих самих задач. Крім того, TypeScript без проблем працює як на фронтенді, так і на сервері, що робить його гнучкішим вибором для розробки.

2.3.2 Фронтенд

Оскільки основною мовою проєкту було обрано TypeScript, фронтенд має базуватись на технології, яка підтримує її. У зв'язку з цим розглядаються три найпопулярніші інструменти для створення інтерфейсів користувача: React, Angular та Vue 3.

React – це JavaScript-бібліотека для побудови користувацьких інтерфейсів. Вона реалізує декларативний підхід і побудована на компонентній архітектурі, де інтерфейс розбивається на модульні компоненти, які ефективно оновлюються при зміні даних [32].

Angular – це веб-фреймворк від Google, призначений для створення швидких та масштабованих додатків. Він пропонує компонентну модель програмування, систему впровадження залежностей і набір інструментів для розробки великих організованих проектів [33].

Vue 3 – це прогресивний JavaScript-фреймворк для побудови інтерфейсів з акцентом на високу продуктивність і простоту освоєння. Vue відомий своєю зручністю і гнучкістю, забезпечуючи реактивну систему зв'язування даних, що дозволяє легко нарощувати функціонал інтерфейсу [34].

Нижче наведено порівняльну таблицю 2.3, яка наочно демонструє переваги та недоліки цих інструментів візуалізації за основними критеріями.

Таблиця 2.3 – Переваги та недоліки інструментів візуалізації

Назва / Критерій	React	Angular	Vue 3
Зручність для монолітної архітектури	+	+	±
Гнучкість інтеграції	+	−	+
Простота навчання	+	−	+
Продуктивність	+	−	+
Підтримка спільноти	+	+	±

На основі побудованої порівняльної таблиці 2.3 було проаналізовано, що невелику перевагу над усіма іншими має React. Отже, вибір впав саме на цей інструмент візуалізації.

Оскільки React є бібліотекою, то його можна використовуватися як самостійно (з будь-яким бекендом), так й у поєднанні з фреймворком Next.js.

Next.js – це React-фреймворк для побудови full-stack-додатків. Він надає вбудовані можливості маршрутизації, серверного рендерингу та створення API-ендпоїнтів. Зокрема, Next.js підтримує серверний рендеринг і статичну генерацію сторінок, а також автоматичну файлову маршрутизацію і вбудовані API-маршрути як серверні функції [35].

Підсумок: «Чистий» React (без Next.js) передбачає, що рендеринг відбувається на клієнті, а маршрутизація й обмін даними реалізуються вручну або за допомогою сторонніх бібліотек. У свою чергу комбінація React + Next.js забезпечує всі ці можливості «з коробки» без додаткової конфігурації.

Отже, для даного проєкту обрано саме React + Next.js – як сучасне, популярне та зручне рішення для побудови монолітної фронтенд-частини платформи для навчання навичкам ЕМД.

2.3.3 Бекенд

У попередньому пункті було обрано стек технологій React + Next.js. При цьому Next.js є не лише фронтенд-фреймворком, а й повноцінним full-stack рішенням, як зазначалось раніше.

Це означає, що серверна логіка реалізовується без потреби в окремому бекенд-сервері. Next.js сам обробляє HTTP-запити на сервері, паралельно з рендерингом сторінок.

Next.js Route Handler – це механізм створення API-ендпоінтів безпосередньо всередині Next.js-додатку. За офіційною документацією, файл

у каталозі app/api автоматично мапиться на маршрут виду /api/... та виконується на стороні сервера [36].

Рисунок 2.3 – Структура маршрутизації API у Next.js

Іншими словами, така файлова структура дозволяє писати серверні функції (обробники HTTP-запитів) у рамках одного проєкту. При отриманні запиту за відповідним URL виконується експортована функція-обробник, яка генерує відповідь (наприклад, повертає дані у форматі JSON). Таким чином Next.js Route Handler фактично дозволяють «будувати API не виходячи з додатку Next.js».

Next.js Route Handler ідеально вписується в монолітну архітектуру, оскільки він дозволяє тримати бекенд-логіку поряд із фронтендом.

Отже, враховуючи вищезазначені особливості, як бекенд-рішення для реалізації серверної логіки навчальної платформи ЕМД обрано саме Next.js Route Handler. Такий підхід дозволяє створювати необхідні серверні ендпоінти всередині існуючого Next.js-додатку, підтримуючи єдиний монолітний проєкт без запуску окремого зовнішнього бекенду.

2.4 Аналіз СУБД з інструментами взаємодії

База даних (БД) – це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації великих обсягів [37].

БД поділяють на два основних типи: реляційні (SQL) та нереляційні (NoSQL). Реляційні БД зберігають інформацію в таблицях із чітко визначеною структурою, що гарантує цілісність даних і підтримку транзакційних операцій.

Нереляційні БД зберігають інформацію у гнучких JSON-документах без жорстко визначеної форми структури для полегшення масштабованості та обробки великих обсягів неструктурованих даних [38].

Системи управління базами даних (СУБД) – це програмні засоби, які виступають посередником між БД та її користувачами [37].

Для вибору найкращої СУБД для проєкту було розглянуто можливість використання MySQL та PostgreSQL як найбільш популярних відкритих реляційних баз, а також MongoDB як провідної документо-орієнтованої NoSQL бази.

2.4.1 MySQL

MySQL – це реляційна СУБД з відкритим вихідним кодом, яка підтримує мову SQL. Ця технологія призначена для застосування у веб-програмах і забезпечування швидкості операцій читання та запису на високому рівні продуктивності [39].

Переваги:

- Простота використання: MySQL досить легко встановлювати та налаштовувати, навіть для початківців. Інтерфейсні інструменти, такі як MySQL Workbench, роблять роботу з ним зручною і зрозумілою.
- Висока швидкість читання даних: MySQL ефективно виконує операції читання, роблячи її оптимальним варіантом для проєктів з великим обсягом даних, наприклад, каталогів або блогів.

Недоліки:

- Проблеми зі складними запитами: При обробці складних запитів SQL, таких як вкладені підзапити або аналітичні функції, MySQL може демонструвати менш ефективну продуктивність.
- Робота з транзакціями менш гнучка: У порівнянні з іншими СУБД, особливо PostgreSQL, підтримка транзакцій у MySQL є менш

потужною. Це може призводити до складнощів у проєктах з великою кількістю модифікацій БД.

- Обмежені можливості розширення: Якщо навантаження зростають, MySQL може вимагати додаткових зусиль для забезпечення горизонтального розширення – наприклад, налаштування реплікацій або розподілу даних між кількома серверами.

Отже, MySQL є надійним варіантом для простих додатків з інтенсивним читанням даних, що забезпечує швидку обробку запитів та цілісність інформації. Однак СУБД має обмежену сферу функціональності: складні транзакції у високонавантажених системах та горизонтальне масштабування.

2.4.2 PostgreSQL

PostgreSQL – це відкрита реляційна СУБД, яка відома своєю надійністю, можливістю масштабування і строгим дотриманням стандартів SQL. Вона підтримує складні функціональні можливості (підзапити, віконні функції, тригери, подання тощо) та забезпечує високу стабільність роботи, навіть при великих обсягах даних [39].

Переваги:

- Гарантія цілісності даних: СУБД повністю дотримується принципів ACID для забезпечення стабільності та надійності транзакцій імовірних систем із важливою точністю обробки даних.
- Підтримка різноманітних структур: PostgreSQL дозволяє працювати не тільки з традиційними таблицями, а також з масивами даних, JSON-структурами, геоданими тощо. Такий підхід робить систему дуже зручною для зберігання інформації.
- Гнучкість та можливість розширення: Користувач може створювати власні функції, типи даних і оператори, налаштовуючи систему під конкретні завдання.

- Стійкість при великому навантаженні: PostgreSQL добре себе зарекомендувала як сучасна СУБД для складних та об'ємних проєктів. Вона надійно функціонуватиме навіть за високого навантаження та при значному обсязі одночасних операцій.

Недоліки:

- Більша складність освоювання: PostgreSQL може здатися важкою для новачків через широкий спектр можливостей цієї системи.
- Менш ефективна для простих запитів: У випадках використання лише простих операцій читання та запису PostgreSQL може не бути таким продуктивним рішенням як менш складні СУБД, наприклад MySQL.
- Масштабування у горизонтальному напрямку: Хоча це можливо, проте його впровадження потребуватиме детального налаштування, що може бути викликом для невеликих команд або проектів із обмеженими ресурсами.

Отже, PostgreSQL – надійна та гнучка СУБД, яка ідеально підходить для проєктів з високими вимогами до цілісності та складності даних. Вона підтримує сучасні типи даних та складні операції з транзакціями, хоча потребує трохи більше навичок для налаштування. Завдяки своєму стабільному функціонуванню та активному супроводженню – це чудовий варіант для об'ємних проєктів.

2.4.3 MongoDB

MongoDB – це нереляційна документо-орієнтована СУБД типу NoSQL, яка призначена для роботи з неструктурованими або частково-структурованими даними. Дані зберігаються у вигляді гнучких документів формату BSON (подібного до JSON), що надає можливість легко змінювати схему БД без необхідності в складних міграціях. MongoDB показує особливу ефективність при розробках веб-додатків, які швидко змінюються [39].

Переваги:

- Гнучка структура даних: MongoDB дозволяє зберігати документи без наперед визначеної схеми. Це зручно для проєктів, де структура даних може змінюватись з часом.
- Масштабованість: MongoDB автоматично підтримує розділення на частини даних (шардинг), що значно полегшує горизонтальне масштабування при збільшенні обсягу даних.
- Висока продуктивність: MongoDB підходить для роботи з великими обсягами даних у реальному часі завдяки швидкій обробці неструктурованої інформації та ефективному виконанню запитів на читання та запис.

Недоліки:

- Транзакційні обмеження: Гарантія цілісності діє лише в межах одного документа; виконання складних транзакцій залишається проблемою.
- Складні запити: Обмежений функціонал для складних запитів ускладнює взаємодію з даними через відсутність можливостей SQL, таких як багаторядкове об'єднання або аналітичні запити.
- Індексування: Часті зміни у БД створюють додаткове навантаження на систему, що впливає на її продуктивність.

Отже, MongoDB ідеально підходить для проєктів з динамічною структурою даних та великим обсягом навантаження, наприклад, у галузях аналітики або управління контентом. Однак через обмежену підтримку транзакцій та складних зв'язків, дана СУБД не є найкращим вибором для проєктів, де важлива цілісність даних, таких як навчальна платформа.

2.4.4 Таблиця порівняння

Нижче наведено порівняльну таблицю 2.4, яка наочно демонструє переваги та недоліки трьох розглянутих СУБД за основними критеріями.

Таблиця 2.4 – Переваги та недоліки розглянутих СУБД

Назва / Критерій	MySQL	PostgreSQL	MongoDB
Продуктивність	+	+	±
Масштабованість	±	±	+
Підтримка транзакцій	±	+	–
Простота у використанні	+	±	±
Підтримка JSON	±	+	+
Гнучкість схеми	–	±	+
Підтримка складних зв’язків	+	+	–

На основі аналізу порівняльної таблиці 2.4 зроблено висновок, що усі три СУБД мають свої сильні сторони під різні типи проєктів. Однак для розробки саме навчальної платформи MongoDB не підходит, оскільки вона не підтримує складні зв’язки між даними та має обмежену роботу з транзакціями.

При виборі між MySQL і PostgreSQL погляд пав на PostgreSQL через її кращу підтримку складних транзакцій, гнучкіше управління типами даних та можливість адаптування до складних структур.

2.4.5 Інструменти взаємодії

Оскільки у попередньому пункті було обрано в якості СУБД – PostreSQL, то для роботи з нею потрібно обрати інструмент взаємодії (ORM).

ORM (Object-Relational Mapping, Об'єктно-реляційне відображення) – це технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування [40].

У середовищі Node.js найпопулярнішими ORM-бібліотеками є Prisma, Sequelize та TypeORM. Усі троє підтримують PostgreSQL.

TypeORM – це ORM для TypeScript, яка дозволяє з’єднувати будь-який TypeScript-додаток із БД [41].

Sequelize – це сучасна бібліотека ORM для TypeScript та Node.js. Переваги використання Sequelize над TypeORM є повний набір міграцій та промісів [41].

Prisma – це ORM-бібліотека для Node.js з підтримкою TypeScript, яка суттєво відрізняється від TypeORM та Sequelize. Наприклад, вона не використовує класи для визначення моделей, а спирається на спеціальну схему (schema). Тобто, Prisma виступає як альтернатива звичайному SQL, складаючись при цьому з кількох компонентів: Prisma Client, Prisma Migrate та Prisma Studio [41].

В якості інструмета взаємодії було обрано саме Prisma через спрощення процесу розробки. Prisma автоматично створює готовий клієнт для взаємодії з БД на основі зручно-описаної структури даних, а її інструменти Prisma Migrate та Prisma Studio роблять роботу зі схемами та даними швидкою й зрозумілою.

Отже, остаточний набір технологій для взаємодії платформи з БД у цьому проєкті вийшов наступним: СУБД – PostgreSQL, ORM – Prisma.

2.5 Хмарне сховище

Навчальні матеріали (відео, презентації, документи) зазвичай займають від кількох одиниць (наприклад, PDF) до кількох сотень мегабайт (наприклад, відеофайли середньої якості). Зберігання таких великих файлів безпосередньо в БД є поганою практикою, оскільки це:

- збільшує навантаження на сервер,
- знижує швидкість обробки запитів,
- ускладнює масштабування та адміністрування системи.

Тому найкращим і сучасним підходом для усунення вищезазначених проблем є зберігання файлів у хмарному сховищі, а в БД лише посилань на них.

Хмарне сховище (Cloud storage) – це сервіс, який надає можливість зберігати дані на віддалених серверах хостингової компанії. Дані розміщуються у дата-центрах постачальника послуги, який відповідає за їх зберігання та доступність. Сховища масштабовані та доступні через Інтернет, тому вони зручні для зберігання великих обсягів інформації [42].

Популярні приклади хмарних сховищ: Amazon S3, Google Cloud Storage, Supabase Storage тощо. Більшість цих сервісів комерційні, тобто вони надають платні плани з якимось обмеженим безкоштовним тарифом, і то не завжди.

На основі аналізу хмарних сховищ для проєкту обрано сервіс Backblaze B2 Cloud Storage як оптимальний безкоштовний варіант збереження даних.

Backblaze B2 Cloud Storage сумісний з API S3, який підтримується великим набором SDKs (Software Development Kit) і бібліотек [43].

Характеристики Backblaze B2: 10 ГБ (гігабайт) сховища безкоштовні, після чого $0.006 за кожен додатковий ГБ на місяць; перші 2500 API-запитів на день безкоштовні, після чого $0.004 за кожні 1000 додаткових запитів [44]. Цей тариф дозволяє зберігати великі обсяги даних з мінімальними витратами.

2.6 Docker

При підтримці проєкту важливо мати зручний спосіб розгортання платформи. Для цього завдання обрано Docker як найкраще рішення.

Docker – це відкрита платформа для розробки, доставки та запуску застосунків. Він дозволяє відокремити програму від інфраструктури, щоб програма працювала однаково на різних машинах [45].

Ключові поняття Docker:

- Образ (Image) – шаблон із повним набором файлів та залежностей для запуску програми.
- Контейнер (Container) – запущений екземпляр образу; ізольоване середовище, в якому виконується програма.

- Dockerfile – текстовий файл зі сценарієм (інструкціями), за яким створюється образ.
- Docker Compose – інструмент для одночасного запуску й налаштування контейнерів як єдиного сервісу за допомогою YAML.

Процес розгортання з Docker:

1. Створення Dockerfile з описом середовища та додатку.
2. Побудова образу за допомогою docker build.
3. Запуск контейнера з побудованого образу через docker run.
4. Для багатокомпонентних додатків використовується docker-compose, що дозволяє запускати всі сервіси одночасно.

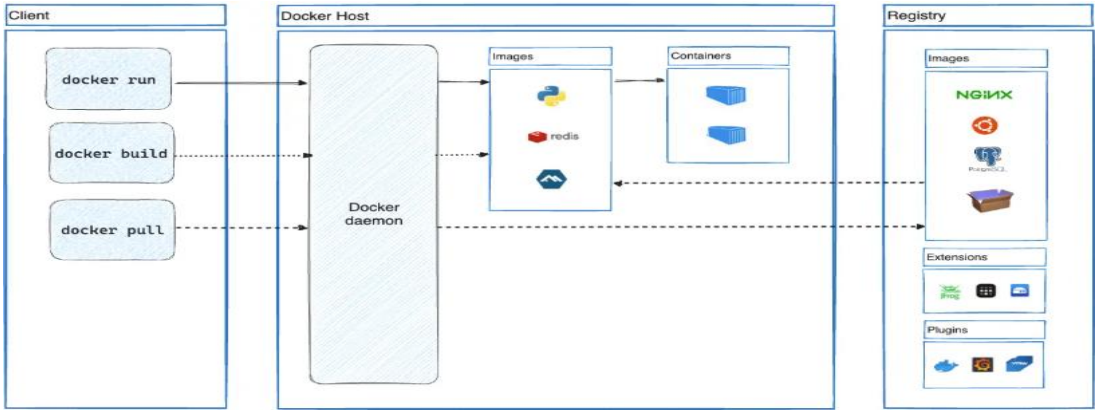


Рисунок 2.4 – Архітектура Docker

Отже, завдяки Docker розгортання платформи стає простішим, швидшим та менш залежним від особливостей конкретного середовища.

Таким чином структурна схема виглядає поки так:

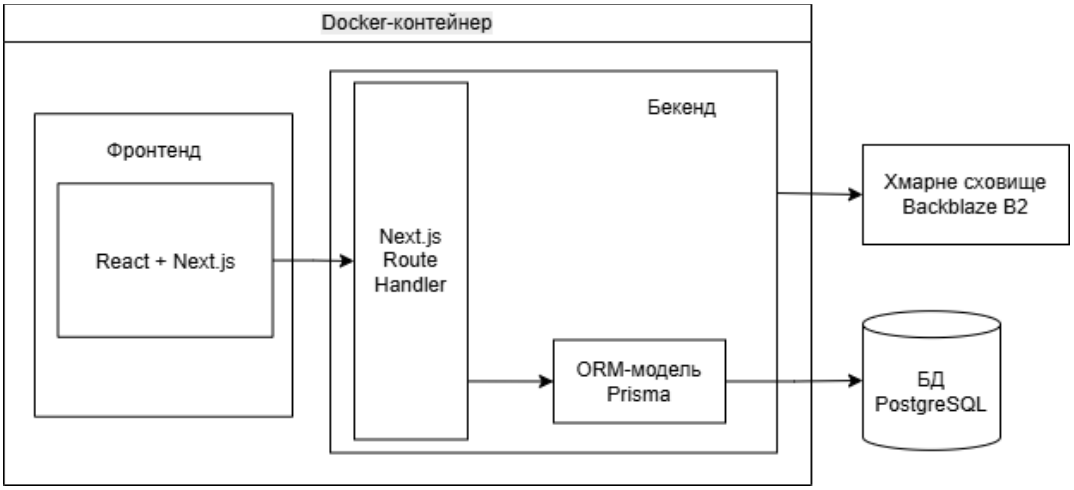


Рисунок 2.5 – Поточний вигляд структурної схеми

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було проведено комплексний аналіз основних компонентів, необхідних для розробки навчальної платформи. У результаті аналізу були зроблено наступні висновки.

Найкращим варіантом для реалізації платформи визначено веб-орієнтоване рішення, оскільки воно забезпечує кросплатформеність, легкість оновлення та широку доступність для користувачів. Монолітна архітектура була обрана через простоту розробки та підтримки, що особливо важливо на початковому етапі проєкту.

Для технічної реалізації обрано сучасний стек технологій: Next.js разом з TypeScript, що дозволяє ефективно поєднувати фронтенд і бекенд у єдиному рішенні. В якості СУБД вибрано PostgreSQL, яка відрізняється надійністю та підтримкою складних транзакцій. Для спрощення роботи з СУБД було вирішено використовувати інструмент взаємодії Prisma.

Для зберігання навчальних матеріалів, які зазвичай мають значний об'єм, обрано хмарне сховище Backblaze B2. Це рішення дозволяє ефективно керувати файлами без необхідності їх зберігання безпосередньо в БД. Для забезпечення розгортання платформи було прийнято рішення використовувати Docker.

Проведений аналіз та обрані технологічні рішення дозволять створити ефективну, зручну у використанні та підтримці навчальну платформу, яка буде відповідати сучасним вимогам та забезпечувати необхідний функціонал для навчання навичкам ЕМД.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ ПЛАТФОРМИ

3.1 Огранізація проєкту

У попередньому розділі було визначено технологічний стек для розробки платформи, який включає фронтенд, бакенд, СУБД з інструментом взаємодії, хмарне сховище. Надалі терміни «фронтенд» та «бекенд» будуть замінені на більш вузьконаправлені назви при розробці – клієнтська та серверна чистини відповідно. Отже, для клієнтської частини вибір пав на React + Next.js, а для серверної – Next.js Route Handler. СУБД було обрано PostgreSQL з Prisma, хмарне сховище – Backblaze B2.

У цьому пункті буде розглянуто організацію проєкту, тобто структуру, налаштування, залежності.

3.1.1 Структура директорій проєкту

Директорії проєкту структуровані за загальними принципами для Next.js з використанням React та Route Handler, а саме:

- src – головна робоча директорія, де зберігається весь вихідний код платформи для клієнтської та серверної частин.
- app – директорія, яка містить код сторінок клієнта та серверні обробники (Route Handler).
- api – директорія, в якій реалізовуються серверні маршрути обробки запитів клієнта.
- [page_name] – директорія з кодом окремої/групи сторінки(-ок) клієнта.
- components – директорія, де зберігаються багаторазові UI-комппоненти, які використовуються на різних сторінках платформи.
- hooks – директорія кастомних (створених розробником) React-хуків.
- utils – директорія з різними допоміжними функціями.

Загальний вигляд відображений на рис. 3.1.

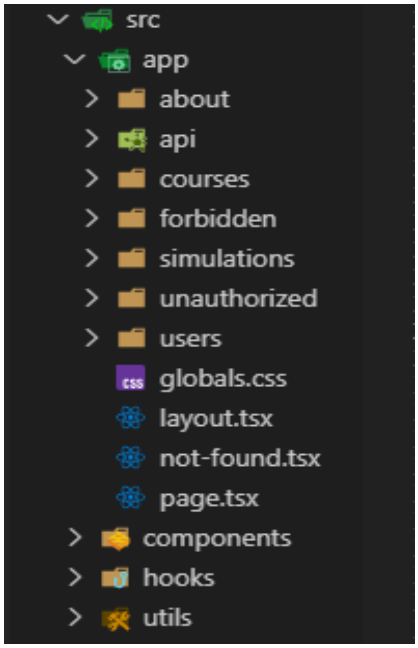


Рисунок 3.1 – Структура директорій проєкту

3.1.2 Огляд основних залежностей

При реалізації функціональної частини платформи використано багато бібліотек, які визначаються у файлі package.json. Нижче наведено їх список та де в основному застосовуються:

- @aws-sdk/client-s3 – SDK для роботи з S3-сумісними хмарними сховищами; застосовується при роботі з хмарним сховищем Backblaze B2, оскільки це просто сховище, яке не має свого API.
- @prisma/client – клієнтська бібліотека для роботи з ORM Prisma; застосовується при роботі з СУБД PostgreSQL для легшої взаємодії.
- bcryptjs – бібліотека для хешування паролів; застосовується при збереженні пароля в БД.
- jsonwebtoken – бібліотека для створення та перевірки JWT-токенів; застосовується при авторизації користувача.
- jwt-decode – бібліотека для декодування JWT-токенів; застосовується для отримання даних користувача з токену.

- md5 – бібліотека для створення MD5-хешів; застосовується для генерації ідентифікатора при отриманні фото користувача.
- next – основний фреймворк для розробки платформи.
- nodemailer – бібліотека для відправки email через SMTP; застосовується при реєстрації та відновлені паролю (таким чином через пошту перевіряється, чи є користувач дійсно власником облікового запису).
- react – основна бібліотека для побудови UI-компонентів та хуків.
- react-beautiful-dnd – бібліотека для drag-and-drop інтерфейсів у React; застосовується при перетаскувані файлів з файлової системи до компоненту з drop властивістю.
- react-dom – основна бібліотека для рендерингу React-компонентів у DOM (Document Object Model).
- tailwindcss – CSS-фреймворк з брекпоінтами, який дозволяє писати стилі для різних розмірів екранів, альтернатива кастомному CSS.
- uuid – бібліотека для генерації унікальних ідентифікаторів (UUID); застосовується при завантажені файлів у хмарне сховище або ідентифікації тестових питань/відповідей.

У наступних пунктах про деякі з них буде детальніше написано, оскільки залежності закладають фундаментальні рішення при розробці платформи.

```
"dependencies": {
  "@aws-sdk/client-s3": "^3.357.0",
  "@prisma/client": "^6.8.2",
  "bcryptjs": "^3.0.2",
  "jsonwebtoken": "^9.0.2",
  "jwt-decode": "^4.0.0",
  "md5": "^2.3.0",
  "next": "^14.0.3",
  "nodemailer": "^6.10.1",
  "react": "^18.2.0",
  "react-beautiful-dnd": "^13.1.1",
  "react-dom": "^18.2.0",
  "tailwindcss": "^3.4.17",
  "uuid": "^11.1.0"
},
```

Рисунок 3.2 – Залежності з версіями в package.json

3.1.3 Налаштування хмарного сховища

Як зазначалось раніше, для зберігання об’ємних файлових даних варто використовувати хмарне сховище. Однак його потрібно правильно налаштувати, щоб завантажувати файли в сховище не вручну, а безпосередньо через платформу.

Перший крок було зроблено, а саме обрано S3-сумісне API для взаємодії зі сховищем – S3 SDK. Наступні кроки такі:

1. Реєстрація та авторизація на сторінці Backblaze B2.
2. Створення однієї корзини (Bucket): сховище може містити багато корзин, в яких і зберігаються дані.
3. Налаштування корзини (Bucket Settings): обов’язково встановлюється публічний доступ до корзини (Public); налаштування життєвого циклу (Lifecycle Policy) – миттєве видалення файлів (проблема в тому, що при видалені файлу через API корзина логує його версію, тому розмір 2 версії буде дорівнювати 0 КБ, однак він буде відображатися); CORS правила також налаштовуються – ділитись з кожним джерелом.
4. Отримання ключів доступу (Access Key ID та Secret Access Key): ці ключі потрібні для з’єднання з корзиною, вони виконують функцію автентифікації (аналог логіна та пароля) для платформи до сховища.

Також важливо переглянути ендпоінти, оскільки вони формують URL-адресу до створеної корзини.

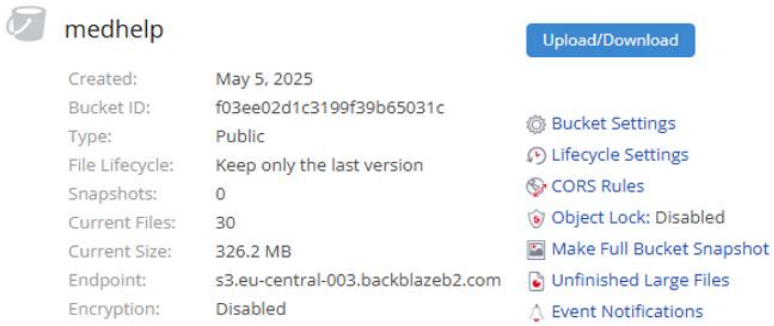


Рисунок 3.3 – Налаштування корзини medhelp

3.1.4 Задання змінних середовища

Для безпечного зберігання конфіденційних даних, таких як: дані підключення до БД, JWT секрет (по якому шифрується JWT-токен), дані пошти (відправника) по яким надсилаються поштові листи, пароль ROOT-користувача, дані сховища та корзини – використано змінні середовища, які визначаються у файлі .env. Доступ до змінних здійснюється через процес process.env у Next.js.

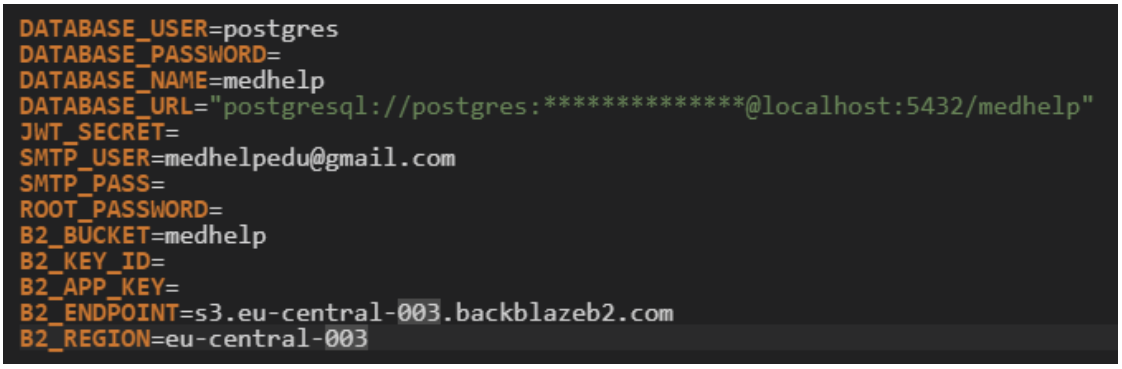


Рисунок 3.4 – Задані змінні середовища у файлі .env

На рис. 3.4 видно, що більшість змінних порожні, однак їх просто було ненадовго прибрано або замасковано, щоб не показувати конфіденційні дані. Просто потрібно зрозуміти, що в проєкті вони є та активно використовуються у серверній частині.

3.2 Структура БД

Як зазначалось раніше, для взаємодії з БД було обрано технологічні інструменти СУБД PostgreSQL з Prisma. При такому підході опис структури БД здійснюється у вигляді окремого файлу schema.prisma, де всі таблиці, зв’язки та типи даних представляються у вигляді моделей для Prisma та задаються у зручному декларативному форматі.

При розробці проєкту активно застосовувалась міграція (SQL-файл для оновлення структури БД без втрати даних) схеми при тестуванні (див. рис.

3.5), через що у БД створилась спеціальна таблиця історії змін міграцій – _prisma_migrations.

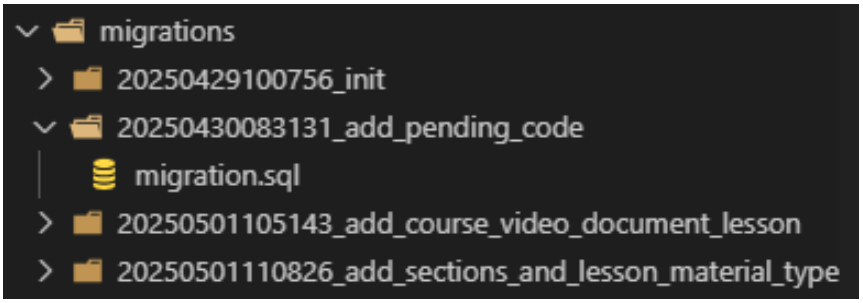


Рисунок 3.5 – Частина міграцій схеми

Далі буде розглянуто усі таблиці БД, які використовуються у проєкті, окрім технічної таблиці _prisma_migrations. Вона не несе смислового навантаження для самої платформи, таким чином не буде показана у функціональній схемі – Додаток Б.

3.2.1 Таблиця User

Таблиця User зберігає інформацію про всіх користувачів платформи. Призначена в основному для автентифікації та визначення прав доступу користувача. Описується в моделі User (див. рис. 3.6).

```
enum Role {
  USER
  MEDIC
  ROOT
}

model User {
  id              Int                 @id @default(autoincrement())
  email           String              @unique
  password        String
  name            String
  role            Role                @default(USER)
  createdAt       DateTime            @default(now())
  courses         Course[]            @relation("AuthoredCourses")
  courseProgress    UserCourseProgress[]
  completedLessons  CompletedLesson[]
  simulations     Simulation[]        @relation("AuthoredSimulations")
}
```

Рисунок 3.6 – Модель User у файлі schema.prisma

Для поля role використано enum (перелік лише заданих варіатів) Role, в якому визначається користувач за ролью: звичайний користувач (USER), медик (MEDIC) або адміністратор (ROOT).

Поля courses, courseProgress, completedLessons, simulations – не є фізичними полями таблиці User, оскільки поля-колекції (масиви інших моделей) – лише логічний опис зв'язків для ORM.

Нижче наведено таблицю 3.1, яка демонструє основні поля таблиці User у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.1 – Основні поля таблиці User

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор користувача (автоматично збільшується)
email	String	Унікальний email користувача
password	String	Хешований пароль користувача
name	String	Ім'я користувача
role	Role (enum)	Роль користувача: USER, MEDIC або ROOT (при створенні USER)
createdAt	DateTime	Дата створення користувача (запис при створенні)

3.2.2 Таблиця Course

Таблиця Course містить інформацію про навчальні курси, які доступні на платформі. Призначена в основному для організації навчального контенту в одну сутність. Описується в моделі Course (див. рис. 3.7).

```
model Course {
  id              Int                  @id @default(autoincrement())
  title           String
  slug            String               @unique
  description     String?
  imageUrl        String?
  author          User?                @relation("AuthoredCourses", fields: [authorId], references: [id], onDelete: SetNull)
  authorId        Int?
  keywords        String[]
  createdAt       DateTime             @default(now())
  sections        Section[]
  totalLessons    Int                  @default(0)
  userProgress    UserCourseProgress[]
  completedLessons CompletedLesson[]
  version         Int                  @default(1)
  editedAt        DateTime?
}
```

Рисунок 3.7 – Модель Course у файлі schema.prisma

Поля sections, userProgress, completedLessons – логічний опис зв’язків для ORM, які відображають відношення між курсом та відповідними сутностями (секціями, прогресом користувачів, завершеними уроками).

Поле author – логічний опис зв’язку для ORM, де поле authorId – зовнішній ключ (FK, foreign key), який посилається на поле id у таблиці User. Таким чином утворюється зв’язок таблиці Course з таблицею User за принципом «багато до одного»: один автор (користувач) може створити багато курсів. Якщо автора видалити – курс не видалиться, оскільки для події onDelete встановлено SetNull. Це означає, що в полі authorId замість ідентифікатора користувача буде стояти null.

Така поведінка дозволена, оскільки при проєктуванні моделі тип поля authorId визначений як необов’язковий (Int?), тобто поле може містити як значення, так і бути порожнім (null). Таким чином курси залишаються в БД навіть після видалення автора.

Нижче наведено таблицю 3.2, яка демонструє основні поля таблиці Course у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.2 – Основні поля таблиці Course

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор курсу (автоматично збільшується)
title	String	Назва курсу
slug	String	Унікальна URL-назва курсу, потрібна для перевірки на унікальність назви нового курсу
description	String	Опис курсу (може бути відсутній)
imageUrl	String	Посилання на зображення курсу (може бути відсутнє)
authorId	Int	Ідентифікатор автора курсу (може бути відсутній при видаленні автора)

Кінець таблиці 3.2

Назва поля	Тип даних	Семантика
keywords	String[]	Ключові слова для пошуку (можуть бути відсутні)
createAt	DateTime	Дата створення курсу
totalLessons	Int	Загальна кількість уроків у курсі
version	Int	Версія курсу (за замовчуванням 1)
editedAt	DateTime	Дата останнього редагування

3.2.3 Таблиця Section

Таблиця Section використовується для структурування курсу на окремі розділи (секції). Описується в моделі Section (див. рис. 3.8).

```
model Section {
  id        Int      @id @default(autoincrement())
  title     String
  order     Int
  course    Course   @relation(fields: [courseId], references: [id])
  courseId  Int
  lessons   Lesson[]
}
```

Рисунок 3.8 – Модель Section у файлі schema.prisma

Поле lessons, course – логічний опис зв’язків для ORM, тому у фізичній таблиці їх не буде. Поле courseId – зовнішній ключ до поля id у таблиці Course. Зв’язок між таблицями Course та Section – «один до багатьох».

Нижче наведено таблицю 3.3, яка демонструє основні поля таблиці Section у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.3 – Основні поля таблиці Section

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор секції (автоматично збільшується)
title	String	Назва секції
order	Int	Порядковий номер секції у курсі
courseId	Int	Ідентифікатор курсу

3.2.4 Таблиця Lesson

Таблиця Lesson містить інформацію про окремі уроки в межах розділу курсу. Признаяення: вміст конкретного навчального матеріалу. Описується в моделі Lesson (див. рис. 3.9).

```
enum LessonType {
  TEXT
  VIDEO
  DOCUMENT
  TEST
}

model Lesson {
  id           Int              @id @default(autoincrement())
  title        String
  type         LessonType
  content      String?
  videoUrl     String?
  documentUrl  String?
  test         Json?
  order        Int
  section      Section          @relation(fields: [sectionId], references: [id])
  sectionId    Int
  completedBy  CompletedLesson[]
}
```

Рисунок 3.9 – Модель Lesson у файлі schema.prisma

Для поля type використовується enum LessonType, який визначає тип уроку: текстовий (TEXT), відео (VIDEO), документ (DOCUMENT) або тест (TEST). Завдяки цьому підходу сутність уроку стає більш структурованою та передбачуваною: для кожного типу уроку чітко визначено, які поля мають бути заповнені, а які ні (null). Детальніше можна подивитись на рис. 3.10.

```
type
 │
 ├── TEXT      → content
 │
 ├── VIDEO     → videoUrl (+ optional content)
 │
 ├── DOCUMENT  → documentUrl (+ optional content)
 │
 └── TEST      → test (JSON)
```

Рисунок 3.10 – Логіка заповнення полів уроку

Поля section, completedBy – логічний опис зв’язків для ORM, поле sectionId – зовнішній ключ до поля id у таблиці Section. Зв’язок між таблицями Section та Lesson – «один до багатьох».

Нижче наведено таблицю 3.4, яка демонструє основні поля таблиці Lesson у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.4 – Основні поля таблиці Lesson

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор уроку (автоматично збільшується)
title	String	Назва уроку
Type	LessonType (enum)	Тип уроку: TEXT, VIDEO, DOCUMENT, TEST
content	String	Основний текстовий контент (якщо урок текст – обов’язково, відео або документа – опціонально, як опис)
videoUrl	String	Посилання на відео (якщо урок відео)
documentUrl	String	Посилання на документ (якщо урок документ)
test	JSON	Тестові завдання (якщо урок тест)
order	Int	Порядковий номер уроку в секції
sectionId	Int	Ідентифікатор секції

3.2.5 Таблиці UserCourseProgress та CompletedLesson

Таблиці UserCourseProgress та CompletedLesson виконують допоміжну роль у структурі БД платформи, бо вони не містять основного навчального контенту, а лише забезпечують зберігання та обробку інформації про навчальний прогрес користувачів.

Таблиця UserCourseProgress призначена для фіксації загального прогресу користувача у певному курсі. Описується в моделі UserCourseProgress (див. рис. 3.11). Таблиця CompletedLesson призначена для зберігання інформації про завершення конкретних уроків користувачем у межах курсу. Описується в моделі CompletedLesson (див. рис. 3.12).

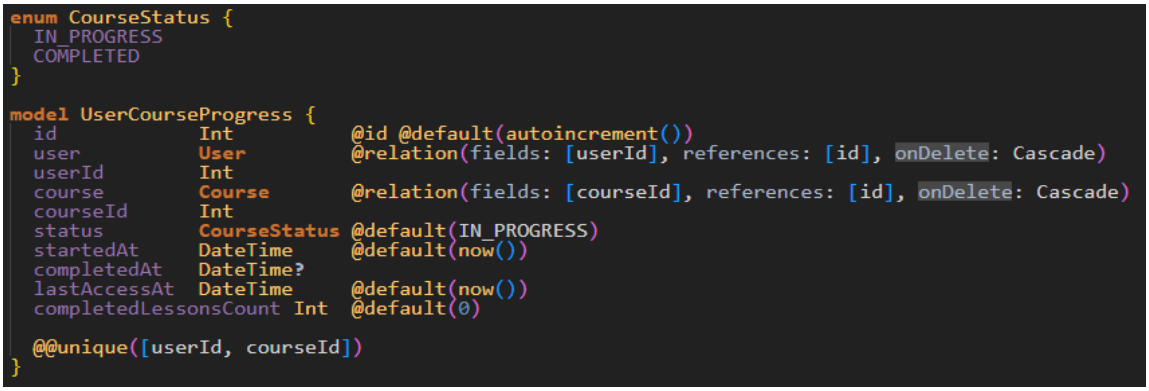


Рисунок 3.11 – Модель UserCourseProgress у файлі schema.prisma

Поле status використовує enum CourseStatus для статусу проходження курсу: IN_PROGRESS (у процесі) або COMPLETED (завершено).

Поля user, course – логічний опис зв'язків для ORM. Поля userId, courseId – зовнішні ключі до поля id у таблицях User та Course відповідно. Зв'язки між таблицями: User та UserCourseProgress – «один до багатьох», Course та UserCourseProgress – «один до багатьох».

Нижче наведено таблицю 3.5, яка демонструє основні поля таблиці UserCourseProgress у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.5 – Основні поля таблиці UserCourseProgress

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор запису (автоматично збільшується)
userId	Int	Ідентифікатор користувача
courseId	Int	Ідентифікатор курсу
status	CourseStatus (enum)	Статус проходження: IN_PROGRESS або COMPLETED (за замовчуванням IN_PROGRESS)
startedAt	DateTime	Дата початку проходження
completedAt	DateTime	Дата завершення (може бути відсутня)
lastAccessAt	DateTime	Дата останнього доступу
completedLessonsCount	Int	Кількість завершених уроків

```
model CompletedLesson {
  id          Int       @id @default(autoincrement())
  user        User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId      Int
  lesson      Lesson    @relation(fields: [lessonId], references: [id], onDelete: Cascade)
  lessonId    Int
  course      Course    @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId    Int
  completedAt DateTime  @default(now())
  timeSpent   Int       @default(0)

  @@unique([userId, lessonId])
  @@index([courseId, userId])
}
```

Рисунок 3.12 – Модель CompletedLesson у файлі schema.prisma

Поля user, lesson, course – логічний опис зв'язків для ORM. Поля userId, lessonId, courseId – зовнішні ключі до поля id у таблицях User, Lesson та Course відповідно. Зв'язки між таблицями: User та CompletedLesson – «один до багатьох», Lesson та CompletedLesson – «один до багатьох», Course та CompletedLesson – «один до багатьох».

Нижче наведено таблицю 3.6, яка демонструє основні поля таблиці CompletedLesson у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.6 – Основні поля таблиці CompletedLesson

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор запису (автоматично збільшується)
userId	Int	Ідентифікатор користувача
lessonId	Int	Ідентифікатор уроку
courseId	Int	Ідентифікатор курсу
completedAt	DateTime	Дата завершення уроку
timeSpent	Int	Час, витрачений на проходження уроку (секунди)

Обидві таблиці використовують каскадну форму видалення зв'язків з іншими таблицями onDelete – Cascade. Це означає, що при видаленні будь-якого запису зі зв'язаних таблиць будуть видалятись записи й для розглянутих.

3.2.6 Таблиці Simulation та Step

Таблиця Simulation містить дані про симуляції – це окремі інтерактивні сценарії, які користувач може проходити для відпрацювання навичок. Описується в моделі Simulation (див. рис. 3.13).

```
model Simulation {
  id          Int        @id @default(autoincrement())
  title       String
  slug        String     @unique
  description String?
  imageUrl    String?
  createdAt   DateTime   @default(now())
  editedAt    DateTime?
  version     Int        @default(1)
  author      User?       @relation("AuthoredSimulations", fields: [authorId], references: [id], onDelete: SetNull)
  authorId    Int?
  steps       Step[]
  keywords    String[]
}
```

Рисунок 3.13 – Модель Simulation у файлі schema.prisma

У представлені таблиця Simulation майже нічим не відрізняється в розглянутої таблиці Course – містить ті самі базові поля та зв’язки з таблицею User, відрізняючись лише одним полем – step замість lesson, тобто каркас один і той же, а ось вміст інший.

Поле step – логічний опис зв’язків для ORM, тому структура таблиці Simulation майже ідентична структурі таблиці Course.

Таблиця Step зберігає кроки для кожної симуляції. Кожен крок – це окремий етап у сценарії симуляції. Описується в моделі Step (див. рис. 3.14).

```
model Step {
  id              Int        @id @default(autoincrement())
  title           String
  content         String
  videoUrl        String
  videoPreviewUrl String?
  order           Int
  simulation      Simulation @relation(fields: [simulationId], references: [id], onDelete: Cascade)
  simulationId    Int
}
```

Рисунок 3.14 – Модель Step у файлі schema.prisma

Поле simulation – логічний опис зв’язків для ORM. Поле simulationId – зовнішній ключ до поля id у таблиці Simulation. Зв’язок між таблицями Simulation та Step – «один до багатьох». Звісно каскадна модель видалення.

Нижче наведено таблицю 3.7, яка демонструє основні поля таблиці Step у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.7 – Основні поля таблиці Step

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор кроку (автоматично збільшується)
title	String	Назва кроку
content	String	Опис кроку
videoUrl	String	Посилання на відео симуляції
videoPreviewUrl	String	Прев'ю відео (може бути відсутнє)
order	Int	Порядковий номер кроку у симуляції
simulationId	Int	Ідентифікатор симуляції

3.2.7 Таблиця PendingCode

Таблиця PendingCode використовується для зберігання тимчасових кодів підтвердження, які надсилаються користувачам під час реєстрації. Це дозволяє реалізувати підтвердження email перед створенням повноцінного акаунта. Описується в моделі PendingCode (див. рис. 3.15).

```
model PendingCode {
  id        String   @id @default(uuid())
  email     String   @unique
  name      String
  password  String
  code      String
  expiresAt DateTime
  createdAt DateTime @default(now())

  @@index([email])
}
```

Рисунок 3.15 – Модель PendingCode у файлі schema.prisma

Нижче наведено таблицю 3.8, яка демонструє основні поля таблиці PendingCode у БД із зазначенням їх назв, типів даних та семантики.

Таблиця 3.8 – Основні поля таблиці PendingCode

Назва поля	Тип даних	Семантика
id	Int	Унікальний ідентифікатор (UUID)
email	String	Email користувача, для якого створено код

Кінець таблиці 3.8

Назва поля	Тип даних	Семантика
name	String	Ім’я користувача
password	String	Хешований пароль, який буде збережено після підтвердження
code	String	Код підтвердження, який надсилається на email
expiresAt	DateTime	Час, до якого код дійсний
createdAt	DateTime	Дата створення запису

3.3 Розробка серверної частини

Серверну частину платформи побудовано на основі Next.js Route Handler у директорії app/api з якої йде побудова ендпоінтів сервера. Базова логіка така, що усі запити з клієнта надходять на сервер, де обробляються через CRUD-операції (створення, читання, оновлення, видалення) з СУБД PostgreSQL через Prisma, а також відбувається взаємодія з зовнішніми сервісами (наприклад, S3 SDK для зберігання файлів, Google Translate API для перекладу).

Для підключення до СУБД через Prisma та до Backblaze B2 через S3 використано патерн синглтон. Це означає, що створюється лише один екземпляр клієнта для кожного сервісу. Такий підхід зменшує навантаження на СУБД та сторонні сервіси та підвищує стабільність роботи. Під час розробки це було сильно відчутно, оскільки при створенні великої кількості з’єднань (екземплярів) доходило до їх примусового розривання.

```
import { PrismaClient } from '@prisma/client';

const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    log: ['warn', 'error'],
  });

if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = prisma;
```

Рисунок 3.16 – Синглтон prisma (PrismaClient)

```ts
import { S3Client } from '@aws-sdk/client-s3';

const B2_KEY_ID = process.env.B2_KEY_ID!;
const B2_APP_KEY = process.env.B2_APP_KEY!;
const B2_ENDPOINT = process.env.B2_ENDPOINT!;
const B2_REGION = process.env.B2_REGION!;

const globalForS3 = global as unknown as { s3: S3Client };

export const s3 =
  globalForS3.s3 ||
  new S3Client({
    region: B2_REGION,
    endpoint: `https://${B2_ENDPOINT}`,
    credentials: {
      accessKeyId: B2_KEY_ID,
      secretAccessKey: B2_APP_KEY,
    },
    forcePathStyle: true,
  });

if (process.env.NODE_ENV !== 'production') globalForS3.s3 = s3;
```

Рисунок 3.17 – Синглтон s3 з даними підключення (S3Client)

Майже для всіх API-роутів клієнт надсилає запит з параметром lang – обрана клієнтом мова (якщо не вказано, то за замовчуванням сервер обирає мову відповіді українську – ua). В utils є словник перекладу відповідей для сервера за ключем – message.ts, який підтримує 2 мови – англійську (en) та українську (ua). Також словник підтримує адаптивний тип відповіді користувачу з відправкою даних, наприклад, при відправці коду на пошту з вказанням імені користувача (див. рис. 3.20).

```ts
ua: {
  required: "Усі поля обов'язкові",
  notFound: "Користувача не знайдено",
```

Рисунок 3.18 – Фрагмент словника перекладу message.ts (ua)

```ts
},
en: {
  required: "All fields are required",
  notFound: "User not found",
```

Рисунок 3.19 – Фрагмент словника перекладу message.ts (en)

```ts
registrationSubject: (name: string) => `Код підтвердження | ${name} (MedHelp)`,
registrationText: (code: string) => `Ваш код підтвердження: ${code}`,
```

Рисунок 3.20 – Фрагмент адаптивної відповіді словника message.ts

3.3.1 Авторизація та реєстрація

Нижче наведено таблицю 3.9, яка наочно демонструє маршрутизацію сервера для авторизації чи реєстрації користувача.

Таблиця 3.9 – Таблиця маршрутів при авторизації та реєстрації

Шлях до файлу	Ендпоінт	CRUD	Призначення
/api/login/route.ts	/login	POST	авторизація користувача
/api/register/request/route.ts	/register/request	POST	запит на реєстрацію (відправка коду на email)
/api/register/confirm/route.ts	/register/confirm	POST	підтвердження реєстрації
/api/password/forgot/route.ts	/password/forgot	POST	запит на відновлення паролю
/api/password/reset/route.ts	/password/reset	POST	скидання паролю

Далі детальніше розписано про алгоритм роботи кожного ендпоінту при взаємодії клієнт надіслав – сервер отримав.

/login (POST):

Клієнт надсилає запит з email та паролем. Сервер отримує дані, перевіряє формат пошти та наявність користувача у БД (таблиця User), порівнює хеш паролю, у разі успіху повертає клієнту JWT-токен авторизації (зашифровані дані користувача з БД, окрім паролю) підписаний секретним ключем (JWT_SECRET). У разі помилки повертає клієнту відповідний статус та повідомлення.

```
const valid = await bcrypt.compare(password, user.password);
if (!valid) {
  return NextResponse.json({ error: t.wrongPassword }, { status: 401 });
}
const token = jwt.sign({ id: user.id, email: user.email, name: user.name, role: user.role }, JWT_SECRET as string,
return NextResponse.json({ token, user: { id: user.id, email: user.email, name: user.name, role: user.role } });
```

Рисунок 3.21 – Фрагмент коду порівняння хеш паролю та відправкою JWT

/register/request (POST):

Клієнт надсилає запит з email, ім’ям, паролем. Сервер отримує дані, перевіряє коректність email, унікальність користувача, генерує код

підтвердження з життям у 15 хвилин, хешує пароль, зберігає дані у таблицю PendingCode, надсилає сформовані параметри листа з кодом SMTP-серверу. У разі успіху або помилки повертає клієнту відповідний статус та повідомлення.

```
await prisma.pendingCode.upsert({
  where: { email },
  update: { code, expiresAt, name, password: hashedPassword },
  create: { email, code, expiresAt, name, password: hashedPassword },
});
```

Рисунок 3.22 – Фрагмент коду запису в таблицю PendingCode

/register/confirm (POST):

Клієнт надсилає запит з email та кодом. Сервер отримує дані, перевіряє код у таблиці PendingCode, створює нового користувача у БД (таблиця User), видаляє тимчасовий запис коду, у разі успіху повертає клієнту статус ok. У разі помилки співпадінння з кодом повертає клієнту статус та повідомлення.

```
const pending = await prisma.pendingCode.findUnique({ where: { email } });
if (!pending || pending.code !== code || pending.expiresAt < new Date()) {
  return NextResponse.json({ error: t.invalidOrExpiredCode }, { status: 400 });
}
```

Рисунок 3.23 – Фрагмент коду перевірки коду в PendingCode

/password/forgot (POST):

Клієнт надсилає запит з email. Сервер отримує дані, перевіряє наявність користувача, генерує код підтвердження, зберігає у таблиці PendingCode, надсилає сформовані параметри листа з кодом SMTP-серверу. У разі успіху або помилки повертає клієнту відповідний статус та повідомлення.

```
await prisma.pendingCode.upsert({
  where: { email },
  update: { code, expiresAt },
  create: { email, code, expiresAt, name: '', password: '' },
});

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});
```

Рисунок 3.24 – Фрагмент коду формування транспортеру

/password/reset (POST):

Клієнт надсилає запит з email, кодом та новим паролем. Сервер отримує дані, перевіряє код, хешує новий пароль, оновлює запис пароля у БД (таблиця

User), видаляє тимчасовий запис коду. У разі успіху або помилки повертає клієнту відповідний статус та повідомлення.

3.3.2 Взаємодія з курсами та уроками

Нижче наведено таблицю 3.10, яка наочно демонструє маршрутизацію сервера для взаємодії з курсами та уроками.

Таблиця 3.10 – Таблиця маршрутів при взаємодії з курсами та уроками

Шлях до файлу	Ендпоінт	CRUD	Призначення
/api/courses/available/route.ts	/courses/available	GET	отримання доступних курсів
/api/courses/progress/route.ts	/courses/progress	GET	отримання курсів у процесі
/api/courses/completed/route.ts	/courses/completed	GET	отримання завершених курсів
/api/courses/control/route.ts	/courses/control	GET, POST	керування курсами
/api/courses/[id]/route.ts	/courses/[id]	GET, PUT, DELETE	отримання, оновлення, видалення курсу
/api/courses/start/route.ts	/courses/start	POST	старт курсу
/api/lessons/complete/route.ts	lessons/complete	POST	завершення уроку

Для взаємодії з цими серверними ендпоітами потрібна вже автентифікація користувача на платформі. Для її перевірки в utils/auth.ts реалізовано функцію

getUserFromRequest. Працює все наступним чином: сервер у цю функцію параметром передає HTTP-запит клієнта; у змінну auth кладеться значення з заголовоку 'authorization'; якщо auth не визначена (undefined) – заголовка 'authorization' немає, тобто користувача не автентифіковано; інакше з заголовка авторизації дістається токен (прибравши його тип ‘Bearer’), токен декодується через JWT секрет, функція повертає серверу об’єкт користувача, які були зашифровані в токені (id, email, name, role). Власне таким чином сервер й отримує інформацію про користувача (через JWT-токен).

```
export function getUserFromRequest(req: NextRequest) {
  const auth = req.headers.get('authorization');
  if (!auth) return null;
  const token = auth.replace('Bearer ', '');

  try {
    if (!JWT_SECRET) throw new Error('JWT_SECRET is not defined');
    const decoded: any = jwt.verify(token, JWT_SECRET);
    return {
      id: decoded.id,
      email: decoded.email,
      name: decoded.name,
      role: decoded.role,
    };
  } catch {
    return null;
  }
}
```

Рисунок 3.25 – Функція getUserFromRequest

У цьому ж файлі utils/auth.ts реалізовано функцію, яка перевіряє потрібні ролі для користувача – requireRole.

```
export function requireRole(user: any, roles: string[]) {
  if (!user || !roles.includes(user.role)) {
    throw new Error('Insufficient rights');
  }
}
```

Рисунок 3.26 – Функція requireRole

Далі детальніше розписано про алгоритм роботи кожного ендпоінту при взаємодії клієнт надіслав – сервер отримав.

/courses/available (GET):

Клієнт надсилає запит для отримання доступних курсів. Сервер автентифіковує користувача; визначає, які курси доступні користувачу (якщо записів у поля userProgress для id користувача немає – курс доступний (див.рис. 3.27)), отримує їх з таблиці Course, повертає клієнту. У разі помилки повертає клієнту статус та повідомлення.

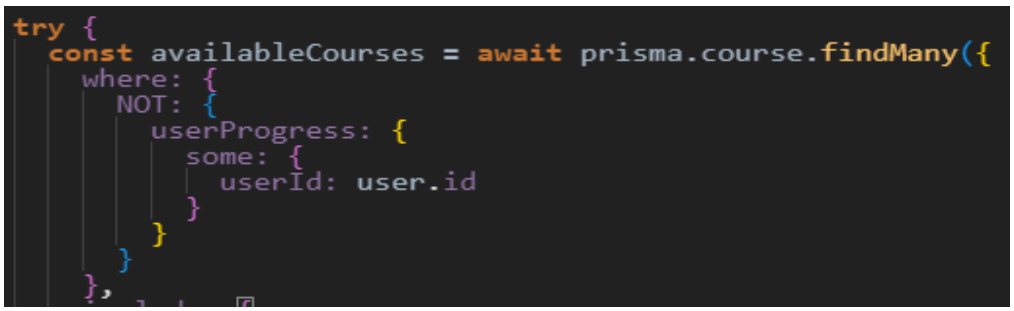


Рисунок 3.27 – Фрагмент коду визначення доступних користувачу курсів

Рисунок 3.28 – Фрагмент коду автентифікації користувача

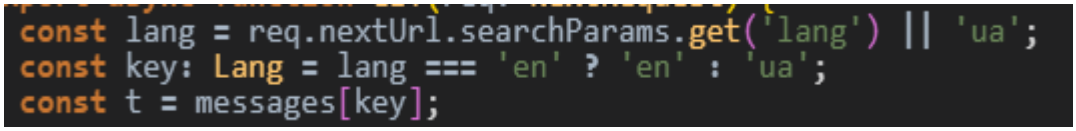


Рисунок 3.29 – Фрагмент коду локалізації відповідей від сервера

/courses/progress (GET):

Клієнт надсилає запит для отримання курсів у процесі. Сервер автентифікує користувача, звертається до таблиці UserCourse, отримує курси у процесі для користувача за полями userID та status – IN_PROGRESS (див. рис. 3.30), повертає їх клієнту. У разі помилки повертає клієнту статус та повідомлення.

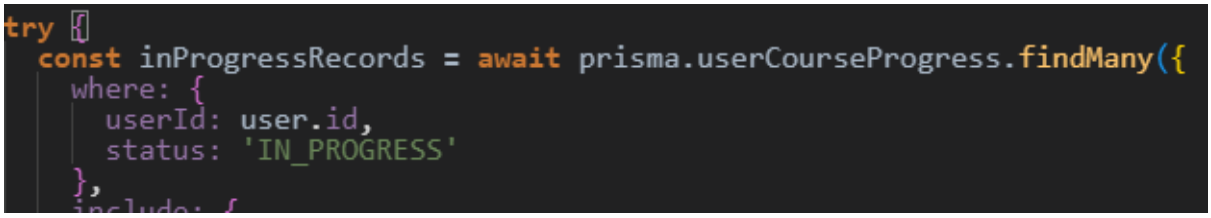


Рисунок 3.30 – Фрагмент коду визначення курсів у прогресі користувача

/courses/completed (GET):

Аналогічно курсам у процесі, тільки тут пошук за status – COMPLETED.

/courses/control (GET):

Клієнт надсилає запит для отримання списку курсів. Сервер автентифікує користувача, перевіряє роль (права доступу) користувача:

- Якщо роль користувача ROOT – звертається до таблиці Course, отримує увесь список курсів, повертає їх клієнту.

- Якщо роль користувача MEDIC – звертається до таблиці Course, отримує список курсів у яких authorId збігається з user.id (власні курси), повертає їх клієнту.

У разі помилки повертає клієнту статус та повідомлення.

Рисунок 3.31 – Фрагмент коду повернення курсів залежно від ролі

/courses/control (POST):

Клієнт надсилає запит з даними нового курсу. Сервер отримує дані, автентифікує користувача, перевіряє необхідну роль, перевіряє URL-назву курсу за співпадінням з іншими (функція sligify (див. рис. 3.32)), створює новий запис у таблиці Course (див. рис. 3.33), у разі успіху повертає повідомлення та дані нового курсу. У разі помилки повертає клієнту статус та повідомлення.

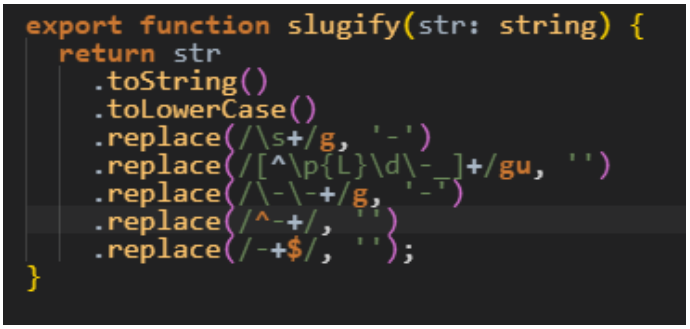


Рисунок 3.32 – Функція slugify

Рисунок 3.33 – Фрагмент коду формування нового запису курсу

/courses/[id] (GET):

Клієнт надсилає запит із ідентифікатором курсу. Сервер отримує id з параметрів запиту, автентифікує користувача, перевіряє чи являється id дійсно цифровим значенням, звертається до БД (таблиця Course), знаходить відповідний курс та повертає його дані клієнту. У разі помилки повертає клієнту статус та повідомлення.

/courses/[id] (DELETE):

Клієнт надсилає запит із ідентифікатором курсу. Сервер отримує id з параметрів запиту, автентифікує користувача, перевіряє необхідну роль (ROOT може видалити будь-який курс, а MEDIC тільки власний (див. рис. 3.34)), перевіряє чи являється id дійсно цифровим значенням, збирає усі посилання з БД таблиць Course та Lesson, відправляє запит на видалення кожного файлу курсу з хмарного сховища за посиланням з S3 (див. рис. 3.35), повертає клієнту підтвердження видалення або помилку.

Реалізовувати видалення інших зв’язних частин курсу не потрібно, оскільки це було вказано у структурі БД.

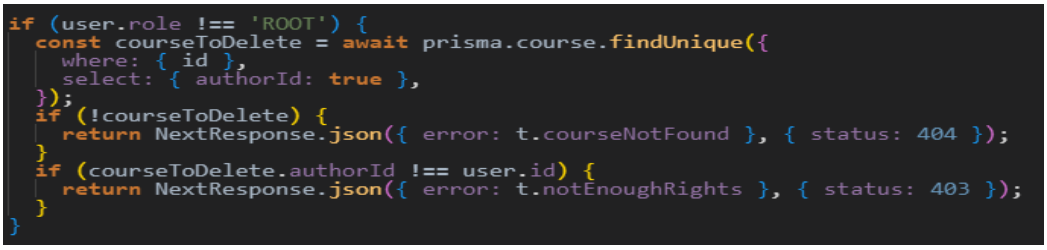


Рисунок 3.34 – Фрагмент коду ролі для видалення курсу

```
const fileUrls: string[] = [];
if (course.imageUrl) fileUrls.push(course.imageUrl);
for (const section of course.sections) {
  for (const lesson of section.lessons) {
    if (lesson.videoUrl) fileUrls.push(lesson.videoUrl);
    if (lesson.documentUrl) fileUrls.push(lesson.documentUrl);
  }
}

for (const url of fileUrls) {
  try {
    const urlObj = new URL(url);
    let key = urlObj.pathname;
    if (key.startsWith('/')) key = key.slice(1);
    if (key.startsWith(`${B2_BUCKET}/`)) key = key.slice(B2_BUCKET.length + 1);
    if (key) {
      await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: key }));
    }
  } catch (error: any) {
    console.error('Error deleting file from B2:', url, error);
  }
}
```

Рисунок 3.35 – Фрагмент коду збирання та видалення матеріалів

/courses/[id] (PUT):

Клієнт надсилає запит із ідентифікатором курсу та новими даними курсу. Сервер отримує id та нові дані, автентифікує користувача, перевіряє роль, перевіряє slug назву, при новому посиланні на зображення курсу – відправляє запит на видалення зі сховища поточного зображення (див. рис. 3.36), оновлює запис у таблиці Course (див. рис. 3.37), збирає нові посилання та поточні з БД таблиць Course та Lesson, відправляє запит на видалення старих файлів курсу з хмарного сховища за посиланням з S3, оновлює розділи та уроки (див. рис. 3.38), очищає прогрес користувачів, повертає оновлену інформацію про курс та повідомлення. У разі помилки повертає відповідний статус та повідомлення.

```
if (current.imageUrl && current.imageUrl !== data.imageUrl) {
  try {
    const urlObj = new URL(current.imageUrl);
    let imageKey = urlObj.pathname;
    if (imageKey.startsWith('/')) imageKey = imageKey.slice(1);
    if (imageKey.startsWith(`${B2_BUCKET}/`)) imageKey = imageKey.slice(B2_BUCKET.length + 1);
    if (imageKey) {
      await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: imageKey }));
      console.log('Old course image deleted from cloud:', imageKey);
    }
  } catch (error: any) {
    console.error('Error deleting old course image from cloud:', error);
  }
}
```

Рисунок 3.36 – Фрагмент коду запиту на видалення поточного зображення

```
try {
  course = await prisma.course.update({
    where: { id },
    data: {
      title: data.title,
      description: data.description,
      imageUrl: data.imageUrl,
      keywords: data.keywords,
      ...(slug ? { slug } : {}),
      editedAt: new Date(),
      version: (current?.version || 1) + 1,
      totalLessons
    }
  });
```

Рисунок 3.37 – Фрагмент коду оновлення запису в таблиці Course

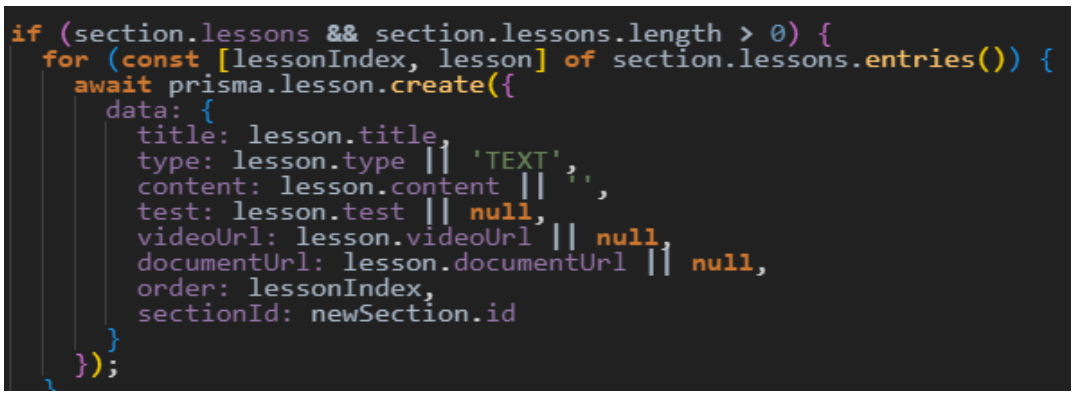


Рисунок 3.38 – Фрагмент коду оновлення запису в таблиці Lesson

/courses/start (POST):

Клієнт надсилає запит для старту курсу з його id. Сервер отримує дані, автентифікує користувача, створює запис про початок проходження курсу у таблиці userCourseProgress, повертає повідомлення підтвердження клієнту. У разі помилки повертає відповідний статус та повідомлення.

/lessons/complete (POST):

Клієнт надсилає запит із ідентифікатором уроку (lessonId). Сервер отримує дані, автентифіковує користувача, перевіряє наявність lessonId, шукає урок у таблиці Lesson; перевіряє, чи вже завершував користувач цей урок (таблиця CompletedLesson). Якщо урок ще не завершено, у транзакції створюється запис про завершення уроку (CompletedLesson), оновлюється або створюється прогрес користувача по курсу (UserCourseProgress), підраховується кількість завершених уроків (Course). Якщо всі уроки завершено, статус курсу змінюється на COMPLETED. У відповідь клієнту повертається статус, повідомлення. У разі помилки повертається відповідний статус та повідомлення.

3.3.3 Взаємодія з симуляціями та кроками

Нижче наведено таблицю 3.11, яка наочно демонструє маршрутизацію сервера для взаємодії з симуляціями та кроками.

Таблиця 3.11 – Таблиця маршрутів при взаємодії з симуляціями та кроками

Шлях до файлу	Ендпоінт	CRUD	Призначення
/api/simulations/available/route.ts	/simulations/available	GET	отримання симуляцій
/api/simulations/control/route.ts	/simulations/control	GET, POST	керування симуляціями
/api/simulations/[id]/route.ts	/simulations/[id]/	GET, PUT, DELETE	отримання, оновлення, видалення симуляції

Як видно, CRUD-операції та ендпоінти таблиці 3.11 схожі з таблицею 3.10, оскільки сутність курс та симуляція майже ідентичні при своїй побудові, відрізняючись лише наповненням матеріалу всередині них. Алгоритм роботи кожного ендпоінту залишається незмінним порівняно з курсами, тому детальний опис кожної операції можна опустити, щоб уникнути дублювання інформації.

Єдина відмінність виникає лише при методі GET: сервер не автентифікує користувача як для однієї, так і для багатьох симуляцій, оскільки симуляція – інструмент швидкого реагування на подію.

Такий підхід забезпечує зручність використання симуляцій як інструменту для оперативного навчання та відпрацювання навичок у будь-який момент.

3.3.4 Взаємодія з користувачами

Нижче наведено таблицю 3.12, яка наочно демонструє маршрутизацію сервера для взаємодії з користувачами.

Таблиця 3.12 – Таблиця маршрутів при взаємодії з користувачами

Шлях до файлу	Ендпоінт	CRUD	Призначення
/api/users/all/route.ts	/users/all	GET	отримання всіх користувачів
/api/users/simple/route.ts	users/simple	GET	отримання простих користувачів
/api/users/medics/route.ts	users/medics	GET	отримання медиків
/api/users/[id]/route.ts	users/[id]	PATCH, DELETE	оновлення, видалення користувача

Далі детальніше розписано про алгоритм роботи кожного ендпоінту при взаємодії клієнт надіслав – сервер отримав.

/users/all (GET):

Клієнт надсилає запит для отримання списку всіх користувачів. Сервер автентифікує користувача, перевіряє, чи має він роль ROOT. Далі звертається до БД (таблиця User), вибирає всіх користувачів, у яких роль не ROOT, повертає їх. У разі помилки повертає статус та повідомлення.

/users/simple (GET) та /users/medics (GET) – алгоритм той самий, тільки повертає користувачів з ролями USER та MEDIC відповідно.

/users/[id] (DELETE):

Клієнт надсилає запит для видалення свого акаунта. Сервер отримує id з параметрів запиту, автентифікує користувача; перевіряє, чи id є числовим значенням і чи id співпадає з id авторизованого користувача. Якщо так – видаляє користувача з БД (таблиця User), повертає підтвердження видалення. У разі помилки повертає статус та повідомлення.

/users/[id] (PATCH):

Клієнт надсилає запит із новою роллю для користувача. Сервер отримує id з параметрів запиту, автентифікує користувача; перевіряє, чи має він роль ROOT; перевіряє, чи id є числовим значенням і чи не намагається змінити свою власну роль. Далі сервер перевіряє, чи його роль не ROOT. Якщо роль у тілі запиту коректна (USER або MEDIC) – оновлює роль користувача у БД (таблиця User) (див. рис. 3.39). Повертає клієнту повідомлення про успіх або помилку.

```
  const body = await req.json();
  if (!['USER', 'MEDIC'].includes(body.role)) {
    return NextResponse.json({ error: t.invalidRole }, { status: 400 });
  }

  await prisma.user.update({ where: { id }, data: { role: body.role } });
  return NextResponse.json({ message: t.roleChanged });
} catch (error) {
  return NextResponse.json({ error: t.errorChangingRole }, { status: 404 });
```

Рисунок 3.39 – Фрагмент коду оновлення ролі користувача

3.3.5 Завантаження файлів та локалізація

Завантаження файлів з сервера реалізовано у файлі /api/upload/route.ts, ендпоінт: /upload методом POST.

Загальний алгоритм наступний: клієнт надсилає запит із файлом (через form-data) та типом файлу (fileType). Сервер отримує form-data, перевіряє наявність файлу. Якщо файл не передано – повертає клієнту помилку. Далі сервер генерує унікальний ідентифікатор (uuid), формує ім’я файлу, визначає шлях збереження у хмарному сховищі (Backblaze B2) залежно від типу файлу (videos, images, documents) (див. рис. 3.40). Сервер зчитує файл у буфер, надсилає запит на збереження файлу у хмару з публічним доступом (див. рис. 3.41). У разі успіху формує публічне посилання на файл і повертає його клієнту. Якщо під час завантаження виникає помилка – повертає клієнту статус та повідомлення про помилку.

```
const fileType = String(formData.get('fileType') || 'other');
const uniqueId = uuidv4().replace(/-/g, '').substring(0, 8);
const fileName = file.name.replace(/[^\p{L}\d._-]+/gu, '_');

let key = `${uniqueId}-${fileName}`;
if (fileType === 'videos' || fileType === 'images' || fileType === 'documents') {
  key = `${fileType}/${uniqueId}-${fileName}`;
}
```

Рисунок 3.40 – Фрагмент коду формування шляху файла

```
const arrayBuffer = await file.arrayBuffer();
const buffer = Buffer.from(arrayBuffer);

try {
  const putParams = {
    Bucket: B2_BUCKET,
    Key: key,
    Body: buffer,
    ContentType: file.type,
    ACL: 'public-read' as const,
  };
  await s3.send(new PutObjectCommand(putParams));
} catch (e: any) {
```

Рисунок 3.41 – Фрагмент коду запиту на збереження файлу

Локалізацію з сервера реалізовано у файлі /api/translate/route.ts, ендпоінт: /translate методом POST.

Загальний алгоритм наступний: клієнт надсилає запит із текстом (text) для перекладу та цільовою мовою (target). Сервер отримує дані, перевіряє їхню коректність. Далі сервер визначає, чи потрібно розбивати текст на частини (якщо текст довгий). Для кожної частини або для всього тексту сервер надсилає запит до Google Translate API (див. рис. 3.42), отримує переклад, повертає клієнту перекладений текст. У разі помилки повертає оригінальний текст та статус.

```
async function translateChunk(text: string, target: string) {
  const url = `https://translate.google.com/translate_a/single?client=gtx&sl=uk&tl=${target}&dt=t&q=${encodeURIComponent(text)}`;
  try {
    const response = await fetch(url);
    const data = await response.json();
    return data[0][0][0] || text;
  } catch {
    return text;
  }
}
```

Рисунок 3.42 – Фрагмент коду запиту до Google Translate API

3.4 Розробка клієнтської частини

Розробка клієнтської частини включає в себе, по-перше, створення та збірку UI-компонентів, які потім рендеряться на сторінках додатку (платформи).

Як правило, спочатку розробляються базові компоненти (наприклад, картка), потім ці компоненти можуть використовуватись у більш складних, наприклад, у списках або модальних вікнах. Таким чином, відбувається поступова збірка: прості компоненти об'єднуються у складніші, складніші у ще складніші і т.д. до тих пір, поки вже з них не буде сформовані основні компоненти сторінок. На фінальному етапі основні компоненти збираються разом для створення повноцінних сторінок. Також компоненті можна використовувати й у інших місцях проєкту не один раз.

По-друге, HTTP-запити до сервера. Важливою функцією у цій частині проєкту є fetchWithAuth. Після авторизації користувача сервер повертає клієнту JWT-токен, кастомний React-хук useAuth зберігає його у LocalStorage (див. рис. 3.43). Такий підхід дуже зручний, оскільки навіть при закритті сторінки – користувач залишиться автентифікованим.

```
const login = useCallback((token: string) => {
  if (isTokenExpired(token)) {
    localStorage.removeItem('token');
    setUser(null);
    return;
  }
  localStorage.setItem('token', token);
  const decoded: any = jwtDecode(token);
  setUser({
    id: decoded.id,
    email: decoded.email,
    name: decoded.name,
    role: decoded.role,
  });
}, []);
```

Рисунок 3.43 – Фрагмент коду додавання JWT-токена в LocalStorage

Так ось, під час подальших запитів з клієнта до сервера використовується функція fetchWithAuth, яка автоматично додає токен авторизації до заголовків кожного запиту (звісно якщо токен є). Це дозволяє забезпечити захищену взаємодію з сервером, оскільки CRUD-опервції в основному потребують автентифікації користувача.

Автентифіковані користувачі отримують усі можливості для навчанння, однак незареєстровані користувачі неповністю обмежені.

Принцип роботи функції fetchWithAuth (розташування /utils/auth.ts):

Перевіряє наявність токена у LocalStorage, додає його до Authorization Header, а також може обробляти помилки, пов’язані з авторизацією,

наприклад, автоматично перенаправляти користувача на сторінку входу у разі втрати токена або отримання помилки 401.

```
export async function fetchWithAuth(url: string, options: RequestInit = {}) {
  const token = typeof window !== 'undefined' ? localStorage.getItem('token') : null;

  if (token && isTokenExpired(token)) {
    localStorage.removeItem('token');
    window.location.reload();
    return Promise.reject(new Error('Token expired'));
  }

  let headers: any = { ...(options.headers || {}) };
  if (!(options.body instanceof FormData)) {
    headers['Content-Type'] = 'application/json';
  }
  if (token) {
    headers['Authorization'] = `Bearer ${token}`;
  }

  const response = await fetch(url, { ...options, headers });

  if (response.status === 401) {
    return Promise.reject(new Error('Unauthorized'));
  }
  if (response.status === 403) {
    return Promise.reject(new Error('Forbidden'));
  }

  return response;
}
```

Рисунок 3.44 – Функція fetchWithAuth

3.4.1 Розроблені UI- компоненти

У проєкті реалізовано як базові, так і основні UI-компоненти.

Деякі базові компоненти:

- Modal.tsx – універсальне модальне вікно для відображення повідомлень чи форм.
- CardGrid.tsx – компонент для організації карток у вигляді сітки.
- Spinner.tsx – простий індикатор завантаження, який використовується під час очікування відповіді від сервера.
- Notification.tsx – компонент для відображення коротких сповіщень користувачу.
- LanguageSwitcher.tsx – перемикач мови інтерфейсу, дозволяє змінювати мову платформи.

Основні компоненти (використовуються на сторінках):

- Navbar.tsx – головний навігаційний компонент (Next.js Hooks), відповідає за переміщення користувача між основними розділами платформи, містить логіку авторизації/реєстрації.

- Footer.tsx – нижній колонтитул сторінки, містить додаткову інформацію.
- UserList.tsx – компонент для відображення списку користувачів у вигляді сітки, використовує UserCard (картка з інформацією про окремого користувача).
- CardList.tsx – відображає список навчальних карток, підтримує додавання, редагування, видалення, застосовується разом з CardGrid – сітка розміщення карток.
- Carousel.tsx – відображає список навчальних карток у вигляді каруселі, тобто карти в деякий час змінюють розташування.
- PageErrorBlock.tsx – компонент для відображення повідомлень про помилки (наприклад, 404 Not Found).
- LessonModal.tsx – модальне вікно для перегляду або проходжння уроку, підтримує різні типи уроків (текст, відео, документ, тест) – кожен урок використовується LessonDocumentUI / LessonVideoUI / LessonTestUI.
- SimulationModal.tsx – модальне вікно для запуску симуляційного навчання з покроковим переглядом відео та контенту.
- TranslatedText.tsx – компонент для відображення перекладеного тексту, використовується на сторінках з мультимовною підтримкою.
- Form.tsx – універсальний компонент для відображення та обробки форм (додавання/редагування курсів чи симуляцій). Для курсів використовуються компоненти – Section, Lesson; для симуляцій – Step.
- Filter.tsx – компонент для фільтрації даних на сторінках, містить логіку та UI для сортування, пошуку, фільтрації.
- SearchBar.tsx – компонент для відображення панелі пошуку або навігації в межах сторінки, підтримує голосовий пошук.

Усі компоненти у проєкті стилізовані за допомогою TailwindCSS, як писалось раніше – розробка компонентів під адаптацію екранів. Більшість

активно використовують React-хуки (useState, useEffect, useRef тощо), через які дуже зручно керувати станом інтерфейсу.

3.4.2 Розроблені сторінки платформи

Було розроблено наступні сторінки платформи:

- Головна сторінка (/).
- Сторінка курсів (/courses).
- Сторінка курсу (/courses/[id]).
- Сторінка симуляцій (/simulations).
- Про проєкт (/about).
- Сторінка користувачів (/users).
- Сторінки обробники помилок:
  - Сторінка не знайдено 404.
  - Сторінка неавторизовано 401 (/unauthorized).
  - Сторінка заборонено 403 (/forbidden).

Нижче наведено таблицю 3.13, яка наочно демонструє використані компоненти та функціонал сторінок.

Таблиця 3.13 – UI-компоненти та функціонал сторінок

Назва	UI-Компоненти	Функціонал
Головна сторінка	Navbar, Footer, TranslatedText, SearchBar, Carousel, SimulationModal	Точка входу на платформу, пошук симуляцій (текстовий і голосовий), перегляд результатів, запуск симуляцій у модальному вікні, ознайомлення з можливостями платформи. Увесь текст локалізовано.
Сторінка курсів	Navbar, Footer, CardList, CardGrid, Filter, Notification, Form, TranslatedText	Відображення курсів у вигляді карток з табами (доступні, у процесі, завершені, мої). Фільтрація, пошук, додавання/редагування/видалення курсів (для медиків/адмінів). Увесь текст локалізовано.

Кінець таблиці 3.13

Назва	UI-Компоненти	Функціонал
Сторінка симуляцій	Navbar, Footer, CardList, CardGrid, Filter, Notification, Form, SimulationModal, TranslatedText	Схожа на сторінку курсів: таби (доступні, мої, всі), фільтрація, пошук, додавання/редагування/видалення симуляцій (для медиків/адмінів), запуск симуляції у модальному вікні. Увесь текст локалізовано.
Сторінка курсу	Navbar, Footer, CardGrid, Notification, TranslatedText, LessonModal, LessonDocumentUI, LessonVideoUI, LessonTestUI	Відображає детальну інформацію про вибраний курс: назву, опис, список уроків (у вигляді карток). Користувач може переглядати уроки різних типів (текст, відео, документ, тест) через модальні вікна. Для курсу відображається прогрес, а після проходження тесту – результат. Увесь текст локалізовано.
Сторінка користувачів	Navbar, Footer, UserList, UserCard, Filter, Notification, Modal, TranslatedText	Доступна лише адміністраторам. Таби (усі, користувачі, медики), відображає користувачів у вигляді сітки карток, фільтрація за полем пошуку, зміна ролі користувача, перегляд інформації на карточці. Увесь текст локалізовано.
Про проєкт	Navbar, Footer, TranslatedText	Інформація про платформу, автора, цілі, використані технології, контакти. Увесь текст локалізовано.
Сьорінки 404, 403, 401	PageErrorBlock	Відображення відповідних повідомлень про помилки.

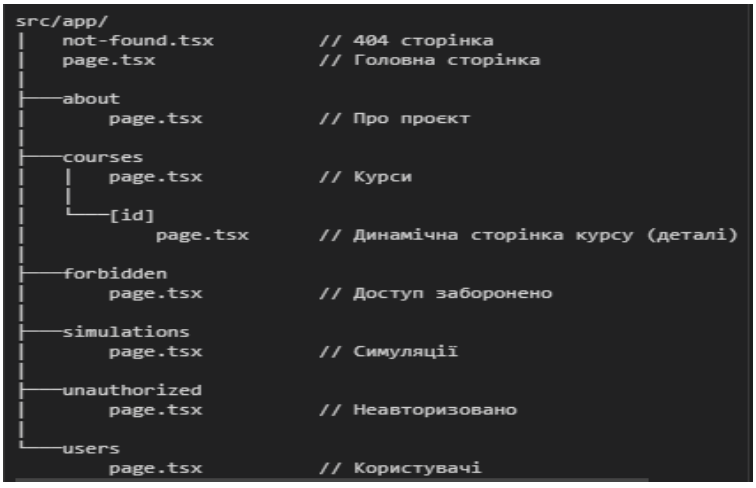


Рисунок 3.45 – Структура сторінок клієнтської частини

3.5 Розгортання проєкту

Для розгортання проєкту в першу чергу було створено Dokerfile (див. рис. 3.46), у якому описані всі кроки для збирання та запуску платформи: встановлення залежностей, копіювання коду, генерація Prisma, збірка Next.js та запуск у продакшн-режимі.

```
Dockerfile
FROM node:18-alpine AS base

WORKDIR /app

COPY package*.json ./

RUN npm ci --legacy-peer-deps

COPY . .

RUN npx prisma generate
RUN npm run build

FROM node:18-alpine AS prod
WORKDIR /app
COPY --from=base /app/package*.json ./
COPY --from=base /app/.next ./.next
COPY --from=base /app/node_modules ./node_modules
COPY --from=base /app/tsconfig.json ./
COPY --from=base /app/src ./src

ENV NODE_ENV=production

EXPOSE 3000

CMD ["npm", "start"]
```

Рисунок 3.46 – Dockerfile

Далі було створено файл docker-compose.yml (див. рис. 3.47), який дозволяє легко запускати сервіс на основі зібраного образу, підключати змінні середовища (env) та відкривати потрібний порт.

```
docker-compose.yml
1   version: '3.9'
2   services:
3     app:
4       build: .
5       env_file:
6         - .env
7       environment:
8         DATABASE_URL: ${DATABASE_URL}
9       ports:
0         - '3000:3000'
1       command: npm run start
2       restart: always
```

Рисунок 3.47 – Файл docker-compose.yml

Для збірки та запуску платформи (контейнера) потрібно в термінал ввести команду docker-compose up --build –d.

На рис. 3.48 можна побачити, що контейнер успішно створився та запустився.

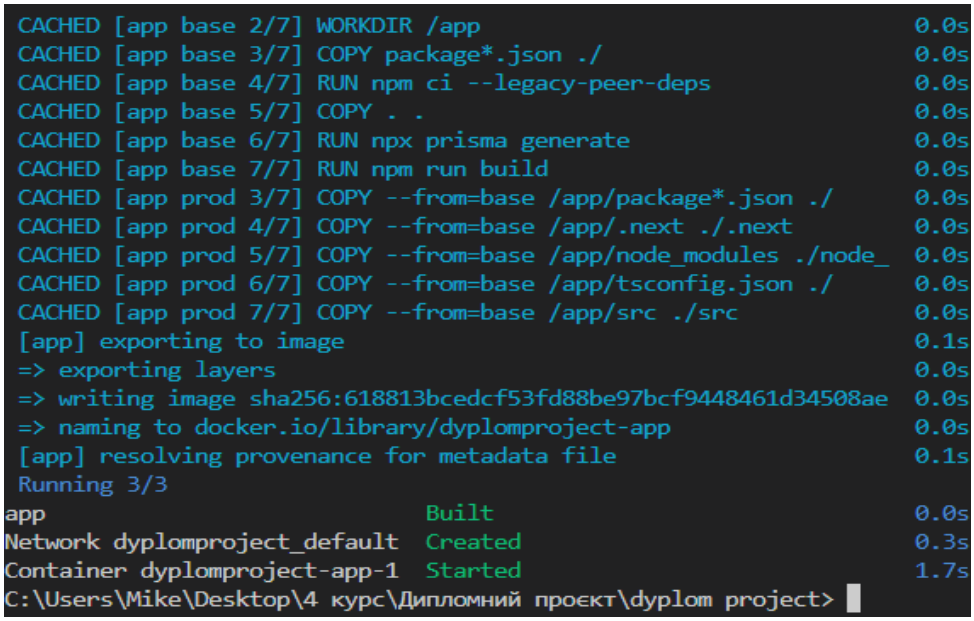


Рисунок 3.48 – Логи створення та запуску Docker-контейнера

При переході на сторінку платформа працює, значить розгортання проєкту пройшло успішно.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було виконано детальну розробку платформи для навчання навичкам ЕМД. Основна увага приділялася реалізації ключових функціональних модулів для усіх частин.

Розробку почато з проектування архітектури з чітким поділом на клієнтську та серверну частини. Для клієнта створено систему компонентів з продуманою ієрархією, де базові елементи використовуються у складних блоках. Це дозволило легко збирати сторінки та підтримувати єдиний стиль інтерфейсу.

На серверній стороні реалізовано модульну систему обробки запитів. Маршрути сгруповано за функціональним призначенням, для кожного ендпоінта визначено послідовність перевірок та обробки даних. Особливу увагу приділено обробці помилок для забезпечення зрозумілої відповіді системи.

Роботу з БД організовано через централізований шар доступу. Описано моделі сутностей з урахуванням всіх зв'язків, для складних операцій використано транзакції. Систему автентифікації реалізовано з кількома рівнями захисту, включаючи хешування паролів та перевірку прав доступу.

Функціонал роботи з файлами інтегровано в основну архітектуру платформи, що забезпечило централізоване управління медіаресурсами в рамках єдиної системи. Для підтримки мультимовності створено шаблони повідомлень. Тестування проводилось на всіх рівнях - від окремих компонентів до повноцінних сценаріїв використання.

У результаті отримано модульну систему з чіткою структурою, яка дозволяє легко розширювати функціонал та підтримує масштабування. Архітектура платформи забезпечує стабільну роботу під навантаженням та зручну підтримку в майбутньому.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ ПЛАТФОРМИ

У цьому розділі буде проведено дослідження та аналіз роботи розробленої платформи, зокрема її функціональних можливостей, інтерактивності та відповідності поставленим вимогам.

4.1 Головна сторінка

На рис. 4.1 показано вигляд головної сторінки при вході користувача на платформу. Вона має досить інтуїтивно-зрозумілий інтерфейс, де розташовано її назву «Платформа для навчання навичкам екстреної медичної допомоги», навігаційний блок з доступними сторінками, перемикачем мови та кнопками Увійти/Реєстрація. Під назвою відображається інструкція та пошукове поле симуляцій з підтримкою голосового вводу. Також є перелік доступних симуляцій для базового ознайомлення користувачем кроків дій під час екстрених ситуацій.

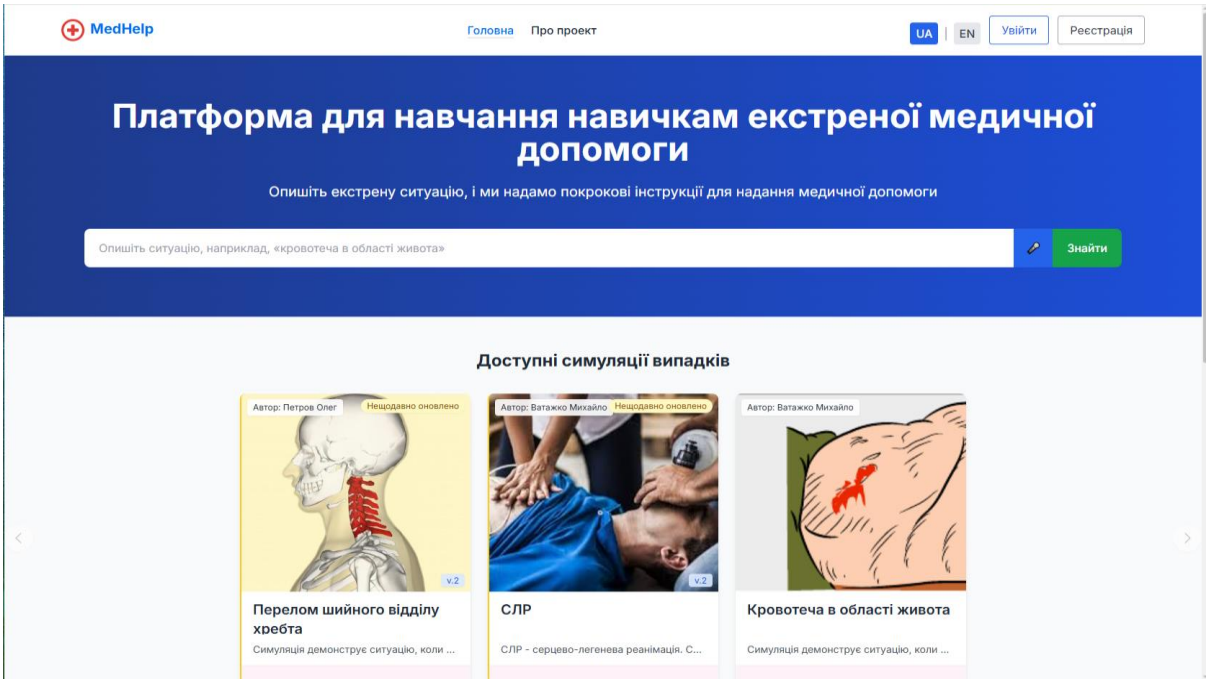


Рисунок 4.1 – Головна сторінка

При натисканні на пошукове поле користувач здійснює звичний текстовий ввід через клавіатуру або екран. При натисканні на кнопку мікрофона біля пошукового поля користувач здійснює голосовий ввід для швидкого знаходження потрібної симуляції (див. рис. 4.2).

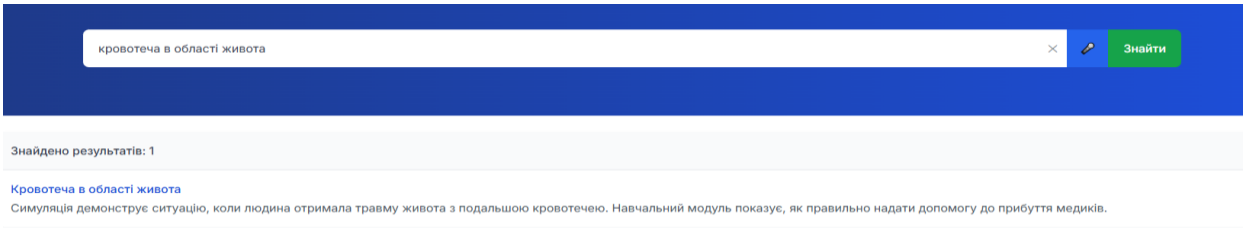


Рисунок 4.2 – Знайдена симуляція

Пошук підтримується за ключовими словами. Наприклад, для тестування обрано слово «реанімація», яке не міститься в назві симуляцій (див. рис. 4.3).

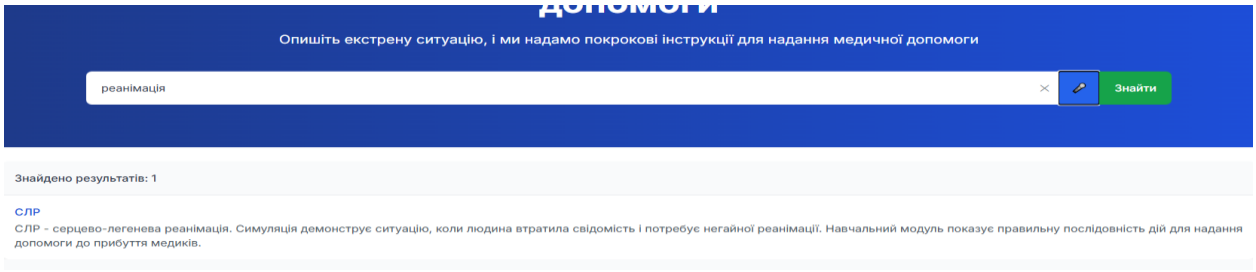


Рисунок 4.3 – Пошук за ключовим словом

При перемиканні мови на EN (англійська) можна побачити локалізовану сторінку для іноземних користувачів (рис. 4.4). Взагалі локалізація присутня всюди, де є текстова інформація.

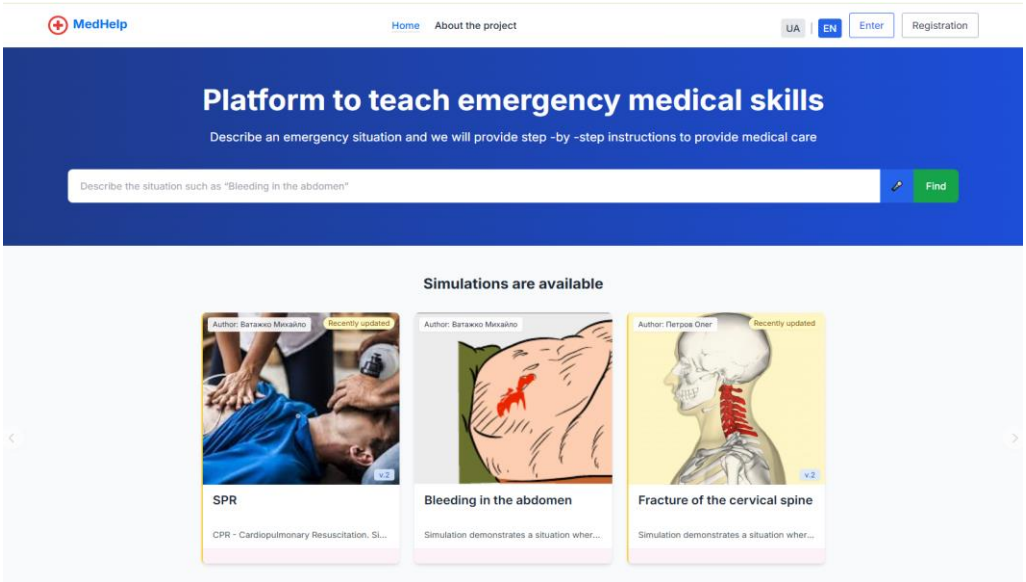


Рисунок 4.4 – Перекладена сторінка

4.2 Сторінка про проєкт

На рис. 4.5 показано вигляд сторінки про проєкт. Вона містить ознайомчу інформацію для користувача про розроблений проєкт (мета, вміст, технології розробки тощо), тобто без функціональної частини.

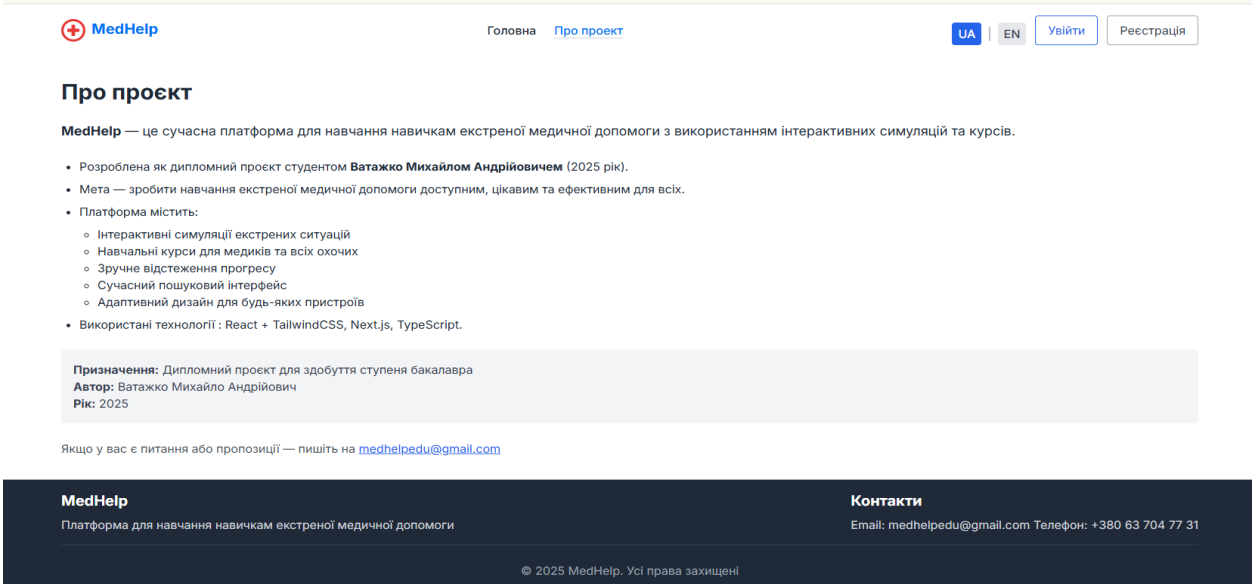


Рисунок 4.5 – Сторінка про проєкт

4.3 Реєстрація та авторизація

Неавторизований користувач може проходити тільки симуляції на головній сторінці. При спробі неавторизованого користувача перейти на сторінку не з вищезазначених – відкривається сторінка помилки статуса 401 Неавторизовано.

Щоб користувач зміг авторизуватись на платформі йому потрібно натиснути в навігаційному блоці на кнопку Увійти. Для загального вигляду процесу автентифікації буде продемонстровано повний процес реєстрації та авторизації користувача покроково.

1. Користувач повинен у навігаційному блоці натиснути на кнопку Реєстрація, після чого відкриється модальне вікно з формою реєстрації (рис. 4.6).

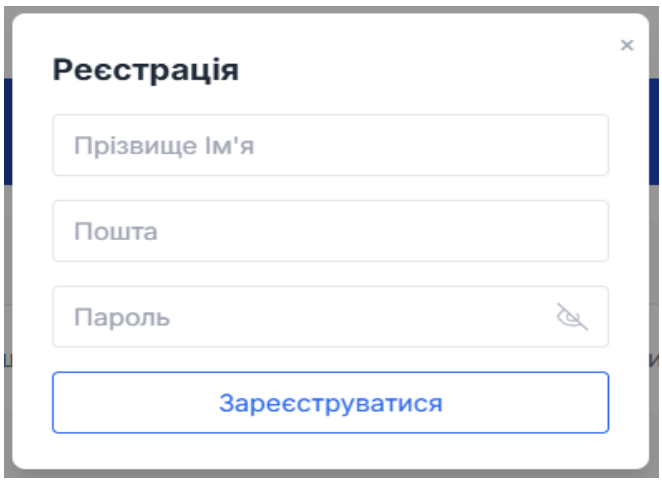


Рисунок 4.6 – Форма реєстрації

2. Користувач повинен ввести усі поля форми: прізвище ім’я, пошта, пароль. Пошта обов’язково повинна бути gmail та не використана на цій платформі. Якщо не виконати цієї інструкції – будуть помилки (приклад див. рис. 4.7).

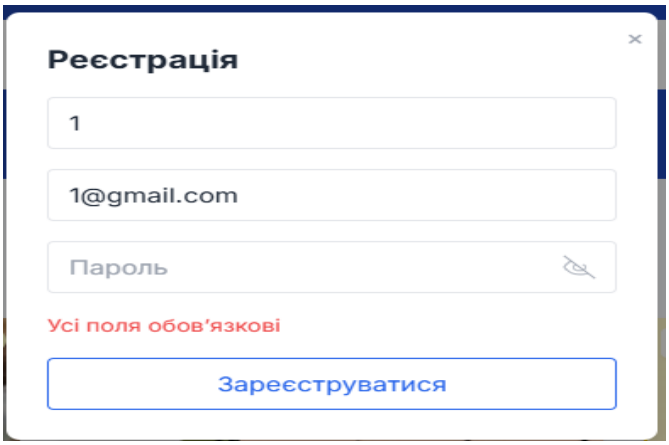


Рисунок 4.7 – Приклад помилки полів

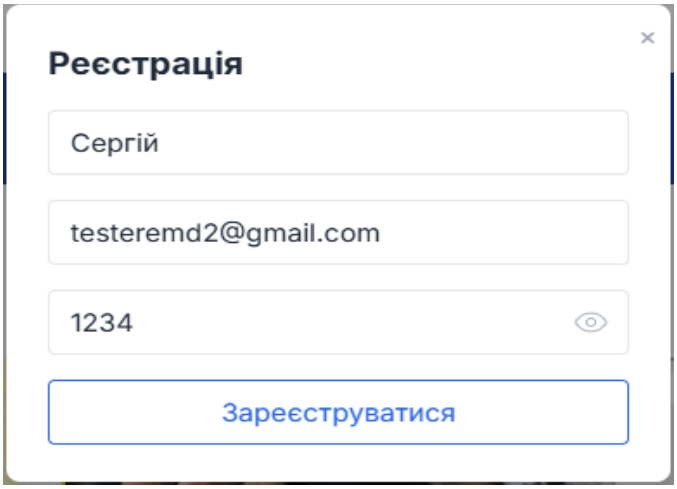


Рисунок 4.8 – Приклад правильної форми реєстрації

3. Користувача перенаправлено на модальне вікно форми підтвердження пошти (рис. 4.9), де він повинен ввести код підтвердження направлений йому на вказану пошту (рис. 4.10). Код діє 15 хвилин. Якщо код не надійшов – можна повторити відправку. Так само помилки при введені коду враховуються (див. рис. 4.11).

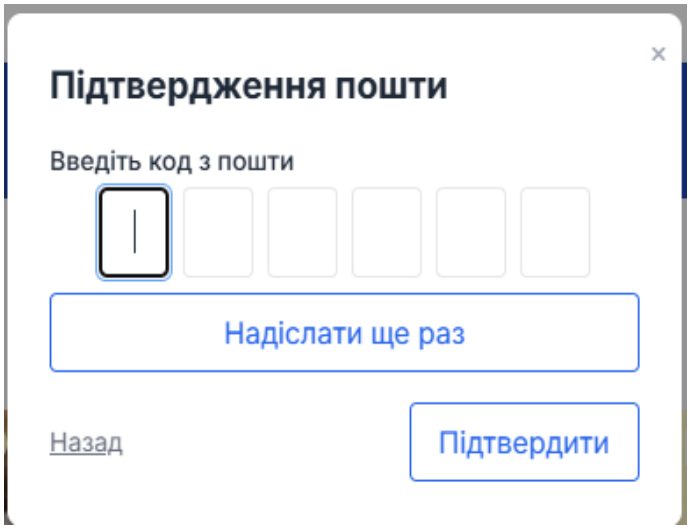


Рисунок 4.9 – Форма підтвердження коду

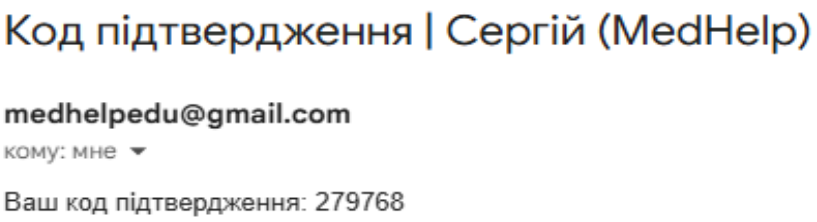


Рисунок 4.10 – Приклад коду з пошти

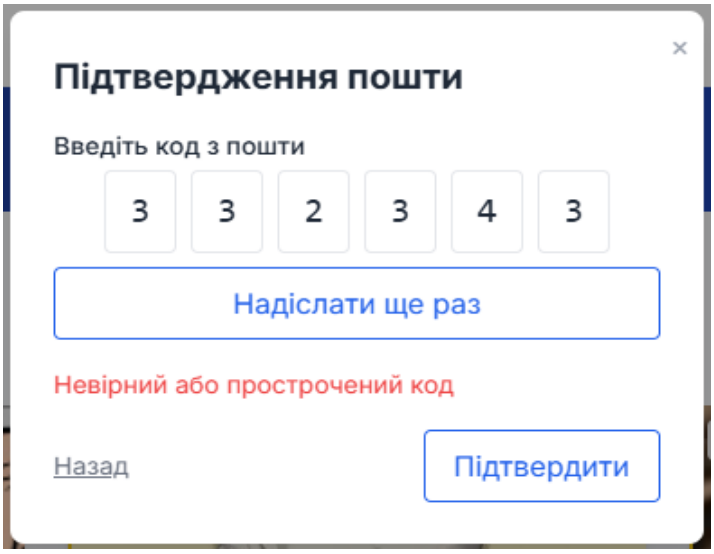


Рисунок 4.11 – Приклад помилки коду

4. Якщо код правильний – користувач зареєстрований у системі. Користувача перенаправлено на модальне вікно успішної реєстрації, потім користувач переходить до модального вікна форми авторизації (рис. 4.12). Користувач вводить пошту, пароль та відправляє форму авторизації. Якщо помилок не виникло (див. рис. 4.13) – користувача авторизовано на платформі.

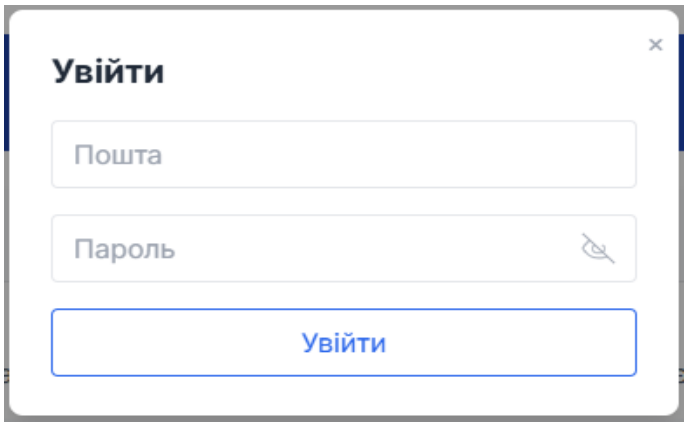


Рисунок 4.12 – Форма авторизації

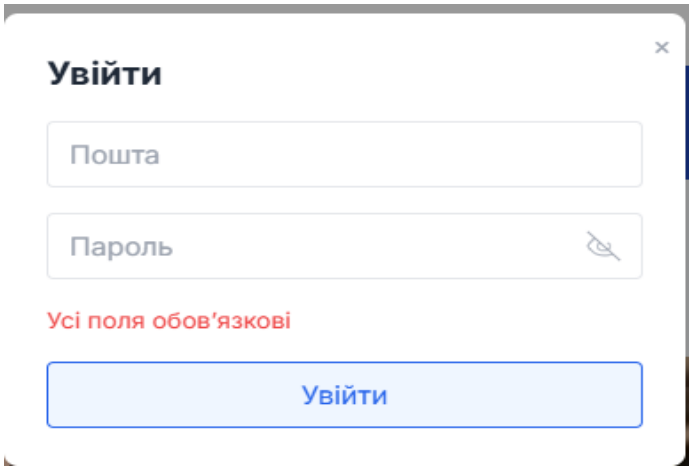


Рисунок 4.13 – Приклад помилки полів

5. Якщо при авторизації користувач вводить неправильні паролі, то він може натиснути на рядок «Забули пароль?» (рис. 4.14). Користувача перенапралено на модальне вікно відновлення пароля, де він підтверджує відправку кода на пошту. Потім користувача перенаправлено на модальне вікно форми зміни пароля, де йому потрібно ввести новий пароль та підтвердити код з пошти (рис. 4.15). Якщо все зроблено правильно – пароль користувача успішно змінено.

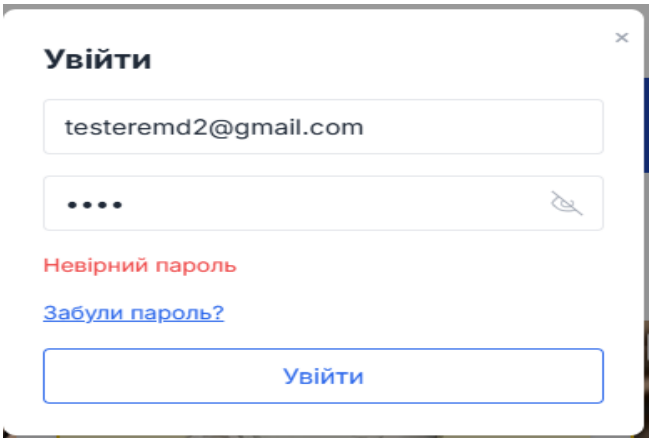


Рисунок 4.14 – Поле «Забули пароль?» при невірному паролі

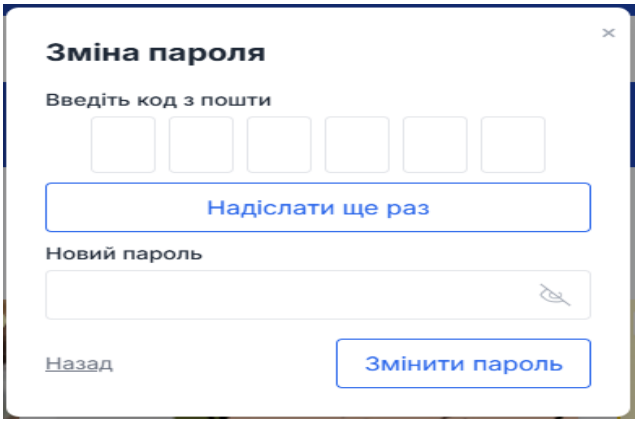


Рисунок 4.15 – Форма зміни пароля

На рис. 4.16 показано головну сторінку після успішної авторизації користувача. У навігаційному блоці з’явились нові вкладки навчальних матеріалів, а замість кнопок Увійти/Реєстрація – іконка користувача з поштою. Натиснувши на іконку, користувач може вийти з сесії або видалити акаунт.

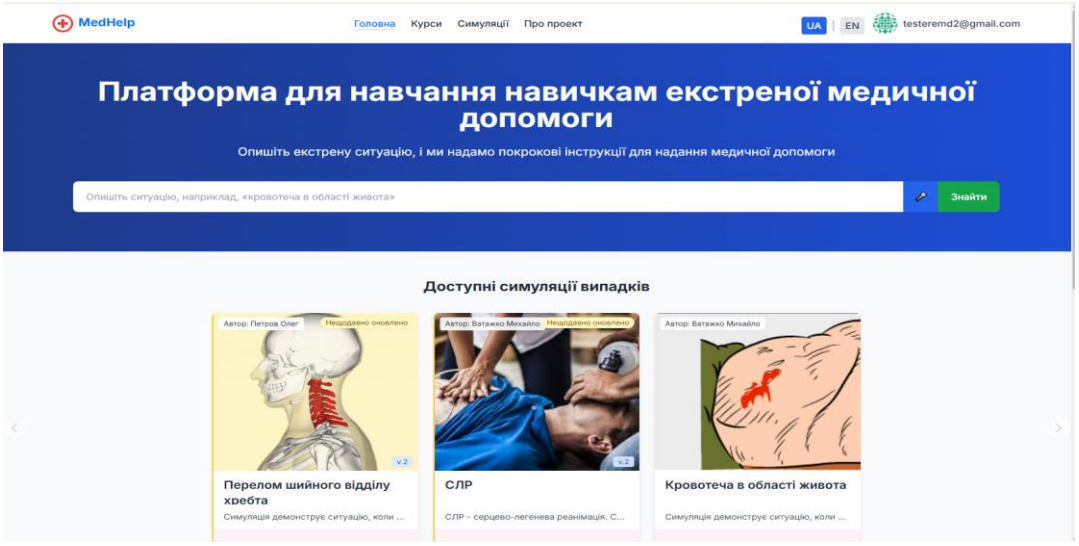


Рисунок 4.16 – Головна сторінка після авторизації користувача

4.4 Сторінка курсів

На рис. 4.17 показано інтерфейс сторінки курсів, де розміщено назва сторінки, вкладки можливих станів курсів (доступні – ще не розпочаті; у процесі – розпочаті, але не завершені; завершені – повністю пройдені), пошукове поле з функціоналом фільтра (фільтр можна сказати), картки курсів.

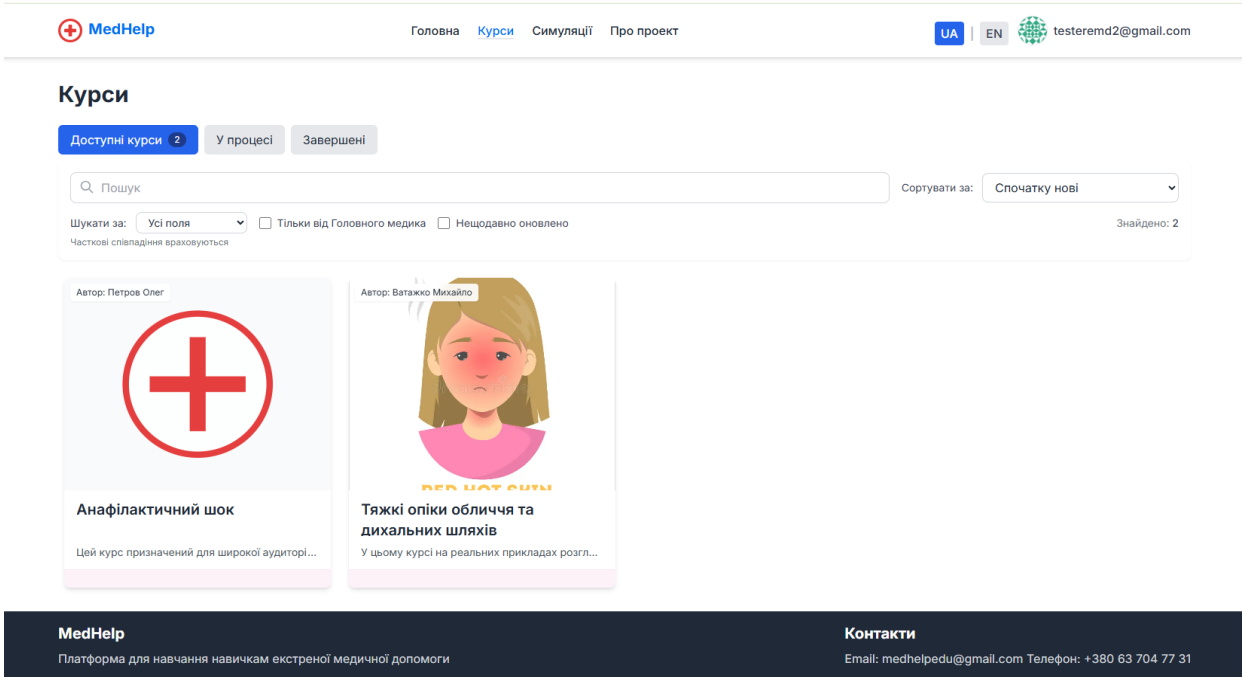


Рисунок 4.17 – Сторінка курсів

У попередньому пункті при демонстрації процесу реєстрації/авторизації було створено нового користувача, тому як видно з рис. 4.17 у новоствореного користувача ще немає курсів у процесі, оскільки він не розпочав процес проходження курсу.

Варто дослідити функціональність фільтра. Як видно з рис. 4.17 фільтр має пошукове поле, сортування за певним критерієм, пошук за певним критерієм, можливість відмітити курси: тільки від Головного медика (ROOT) та нещодавно оновлені. За цим набором зрозуміло, що фільтр повинен комбінувати ці складові для ефективно-націленого пошуку.

На рис. 4.18 показано, що фільтр правильно працює з ключовими словами, оскільки показав тільки цей курс.

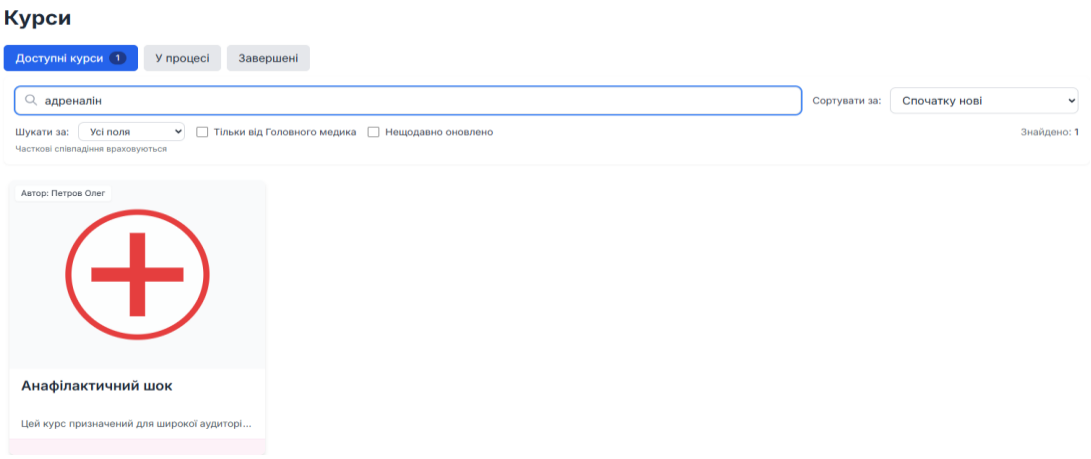


Рисунок 4.18 – Приклад роботи фільтра для ключових слів

На рис. 4.19 пошук за відміткою тільки від Головного медика правильний, оскільки автор Ватажко Михайло є ROOT.

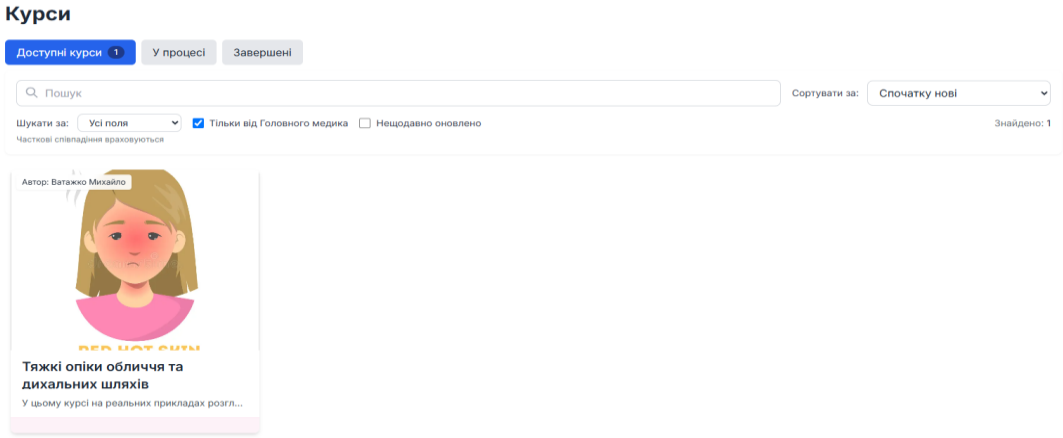


Рисунок 4.19 – Приклад пошуку за відміткою

На рис. 4.20 показано за якими конкретними полями можна виконати пошук.

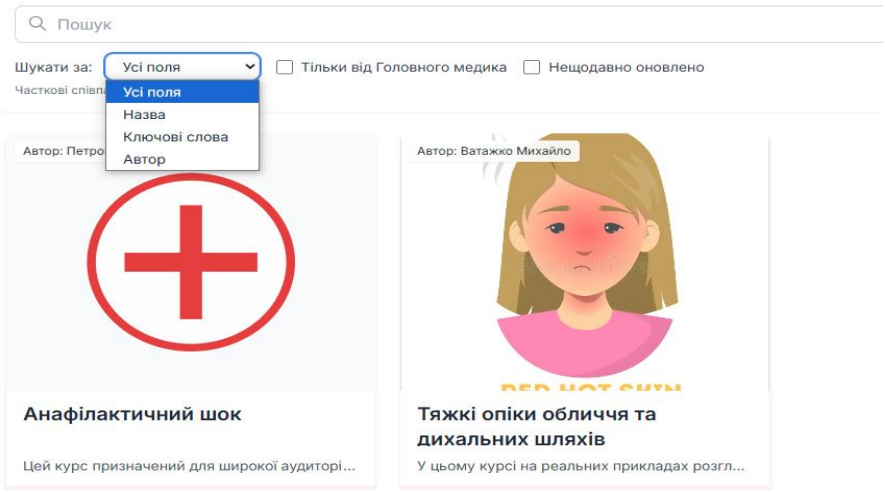


Рисунок 4.20 – Вкладка фільтру «Шукати за:»

На рис. 4.21 показано вкладку «Сортувати за:», в якій можна побачити досить велику кількість можливості сортування.

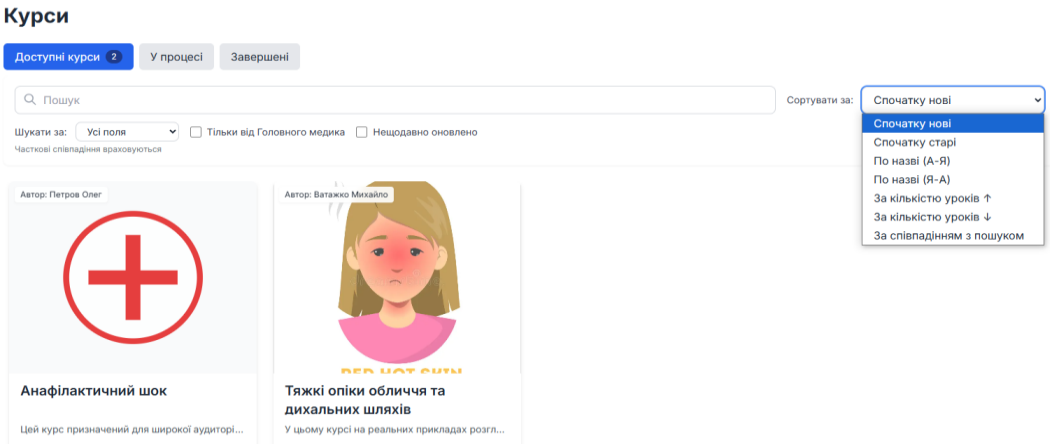


Рисунок 4.21 – Вкладка фільтру «Сортувати за:»

4.5 Сторінка курсу

При натисканні на карточку курсу на сторінці курсів, користувач переходить на сторінку курсу. На рис. 4.22 показано інтерфейс сторінки курсу, де розташовано назва, опис, назва розділів та власне самі навчальні матеріали у вигляді текстві, документів, відеоматеріалів, тестів. По іконках уроку відразу можна визначити, до якого типу вони відносяться. Уроки для проходження курсу зачинені, оскільки проходження курсу ще не розпочато.

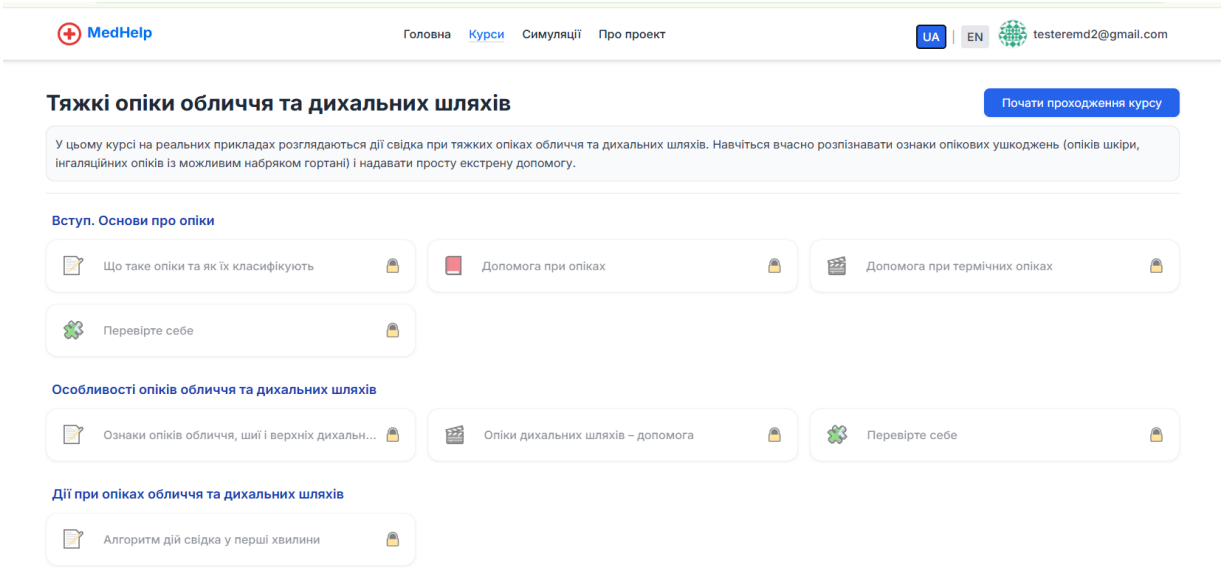


Рисунок 4.22 – Сторінка курсу

Після натискання на кнопку початку проходження курсу відкривається доступ до усіх уроків, з’являється шкала прогресу (рис. 4.23)

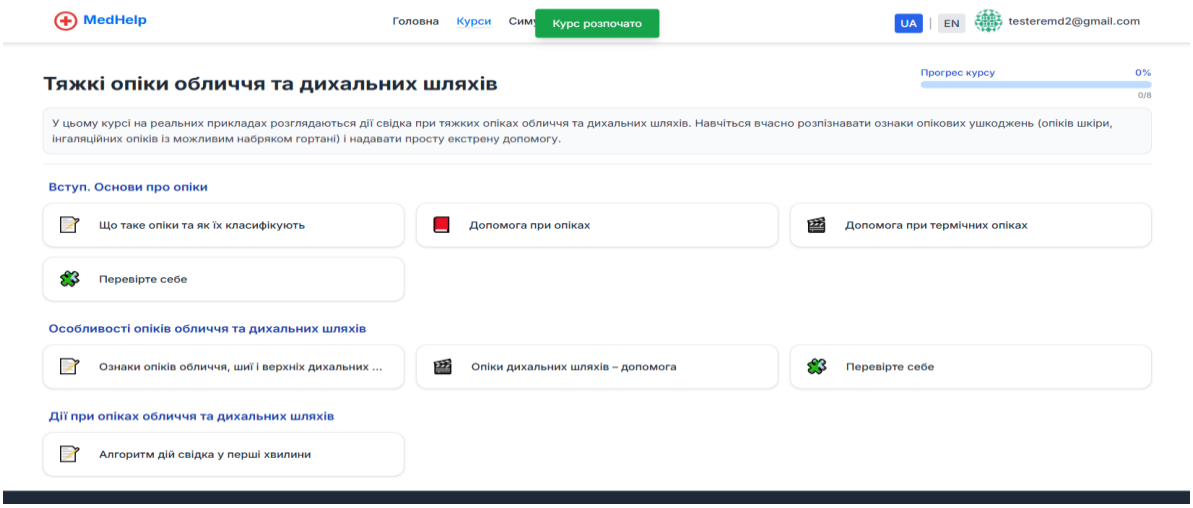


Рисунок 4.23 – Початок проходження курсу

Усі уроки курсу представлені у вигляді модального вікна (як на формах реєстрації/авторизації). Таке відображення навчального контенку досить зручне та сучасне, бо охоплює різні типи екранів. Нижче будуть наведені короткі описи стосовно матеріалів з подальшою нумерацією рисунків.

На рис. 4.24 показано модальне вікно уроку типу текст (TEXT). Такий тип зручний для швидкого опрацювання матеріалу, оскільки містить у собі лише текст.

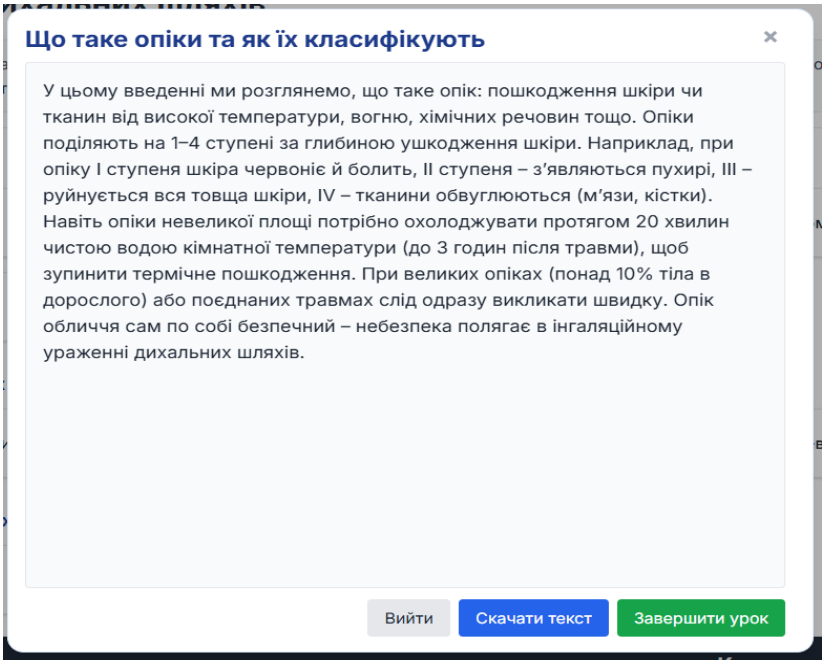


Рисунок 4.24 – Урок типу текст

Будь-який тип уроку можна завантажити та відкрити для перегляду.

На рис. 4.25 видно, що урок до курсу завершився (стоїть зелена галочка) після натискання кнопки «Завершити урок», а прогрес курсу збільшився.

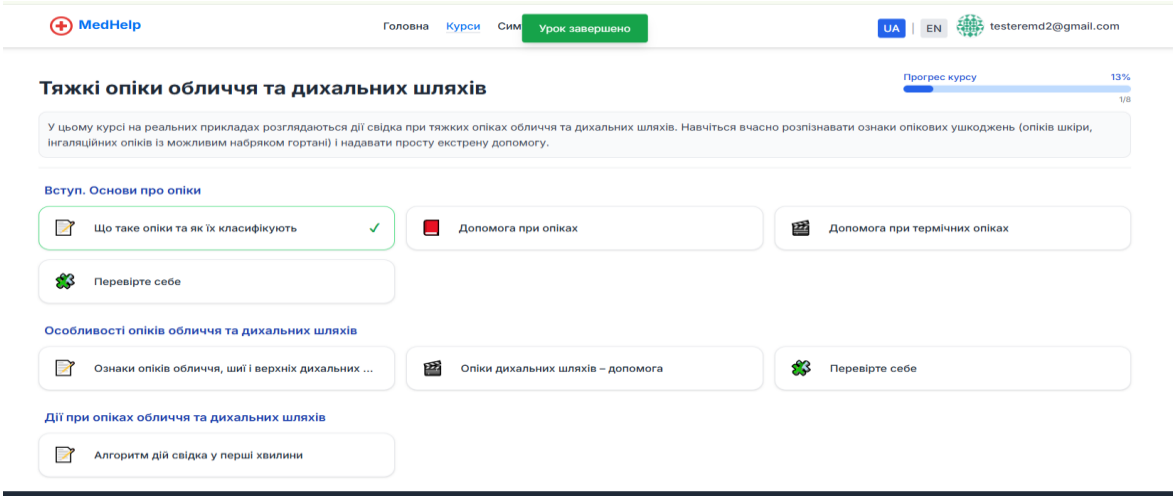


Рисунок 4.25 – Завершення уроку

На рис. 4.26 показано модальне вікно уроку типу документ (DOCUMENT). Такий тип зручний для детального аналізу з поглибленим вивченням матеріалу, оскільки містить у собі не короткий текст, а вже цілі розділи матеріалів, можливо з інфографіками (якщо мова йде про PDF).

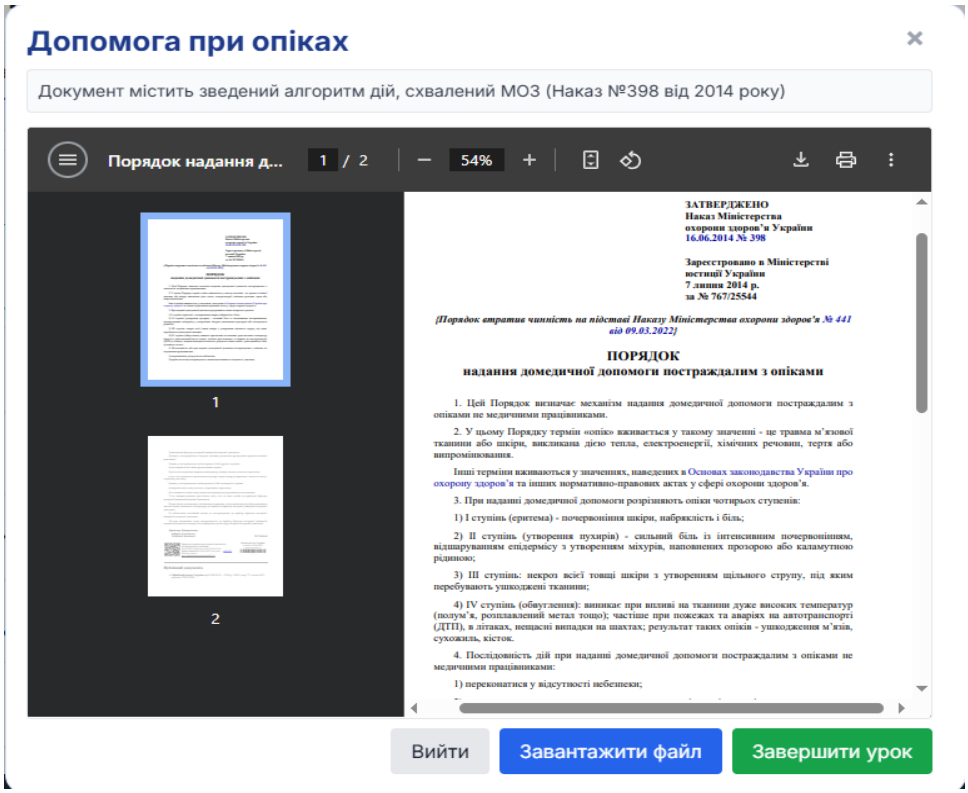


Рисунок 4.26 – Урок типу документ

На рис. 4.27 показано модальне вікно уроку типу відео (VIDEO). Такий тип зручний для більшої інтерактивності та наочності матеріалу, оскільки складається з візуальної картини, яка краще сприймається.

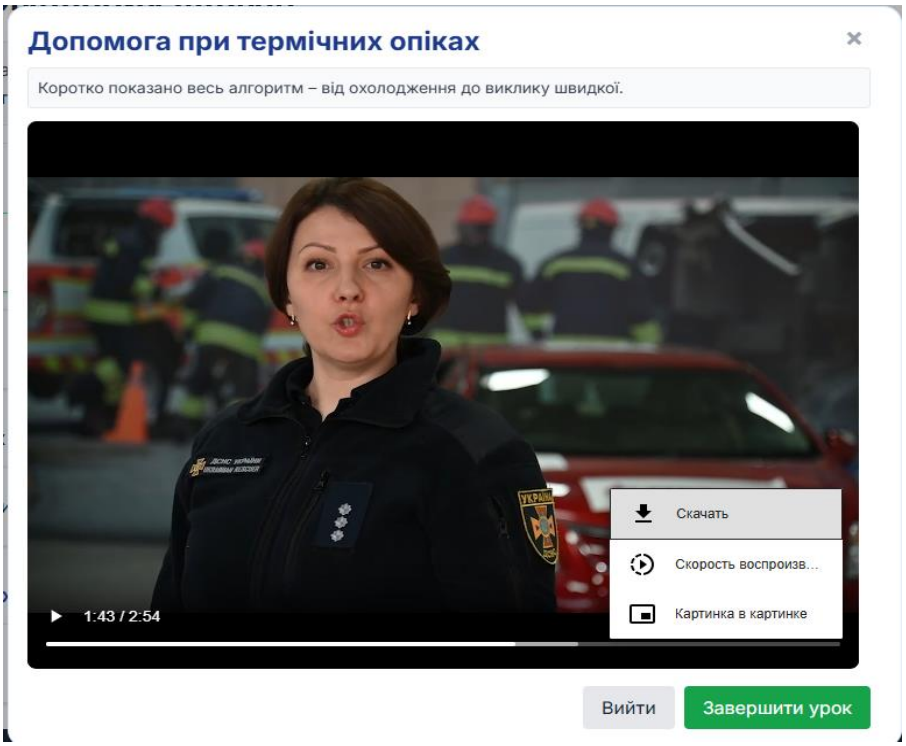


Рисунок 4.27 – Урок типу відео

На рис. 4.28 показано модальне вікно уроку типу тест (TEST). Такий тип потрібен для контролю рівня знань на поточний момент.

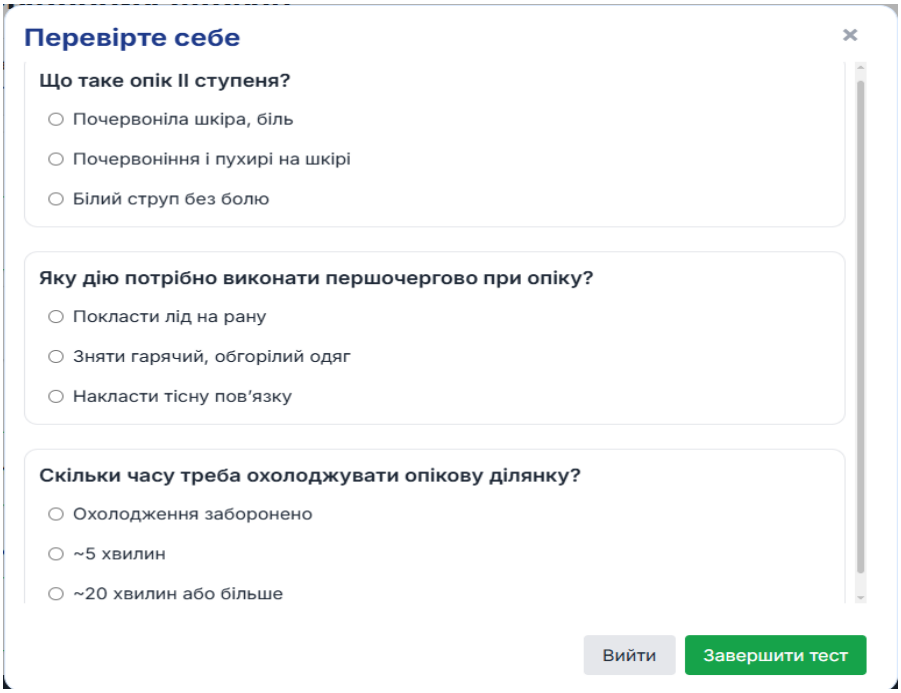


Рисунок 4.28 – Урок типу тест

Тест містить у собі набір запитань та варіантів відповідей. Запитання може мати як одну, так і декілька правильних відповідей, звісно якщо це передбачено власником теста. Після проходження тесту користувач повинен отримати результат його проходження, а також правильні відповіді за потреби. Усі запитання повинні мати відповідь при закінченні тесту.

На рис. 4.29 та рис. 4.30 продемонстровано пройдений користувачем тест. Знизу назви тесту виведено результат з правильними та частково правильними відповідями. Частково правильна відповідь розрахована для питань з декількома правильними відповідями. Зеленим кольором світиться фон питань, на які користувач надав правильну відповідь. Червоним кольором – неправильну, та знизу блока питання зазначається правильний варіант.

Рисунок 4.29 – Приклад пройденого тесту

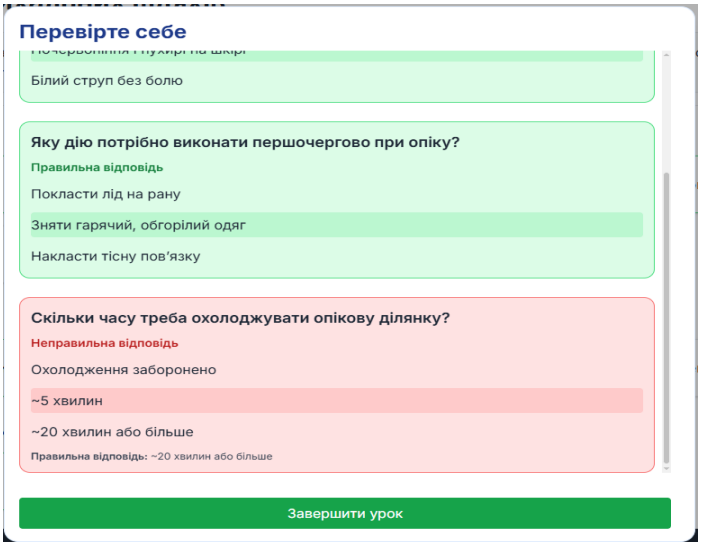


Рисунок 4.30 – Приклад помилки в тесті

Після проходження останнього уроку курсу, платформа вітає користувача з успішним завершенням курсу та перенаправляє на сторінку курсів, де пройдений курс знаходиться у вкладці завершені, а на картці відображається статус Завершено (рис. 4.31). На цьому взаємодія звичайного користувача з курсами завершена.

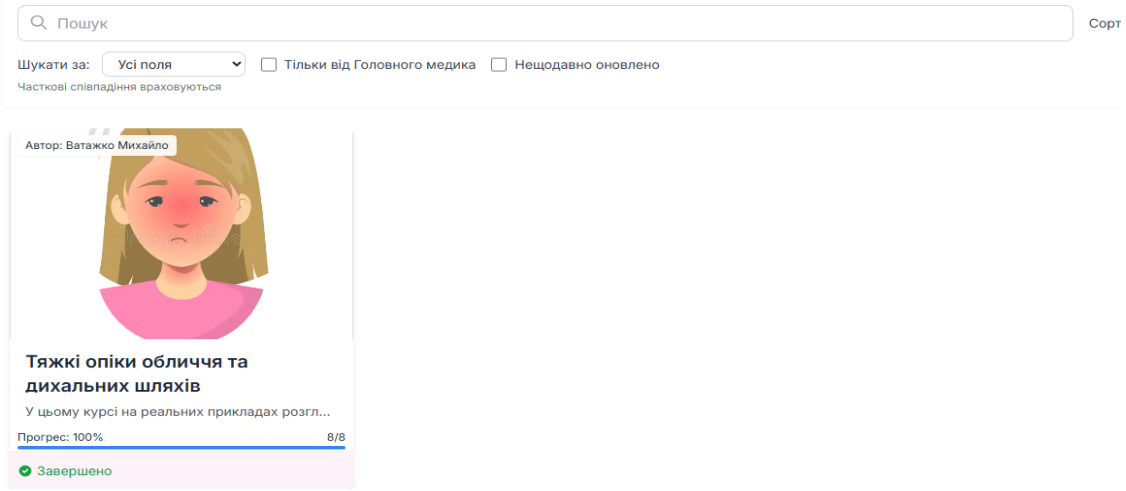


Рисунок 4.31 – Завершений курс

4.6 Сторінка симуляцій

На рис. 4.32 показано інтерфейс сторінки симуляцій, який майже один в один повторює сторінку курсів. Різниця тіль в тому, що вкладок на сторінці симуляцій немає. Логіка компонентів на різних сторінках не змінюється.

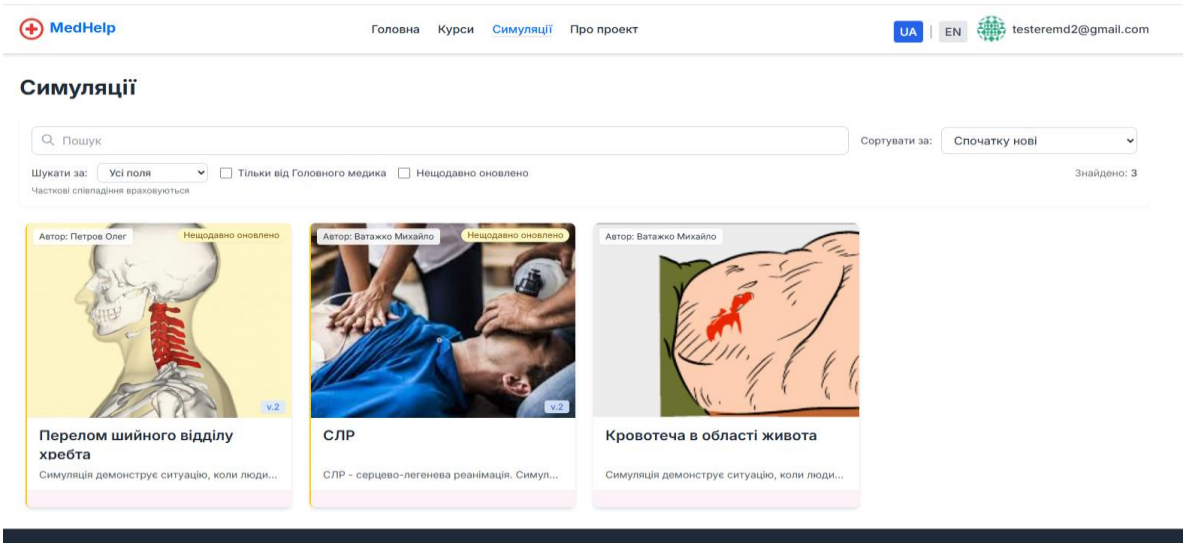


Рисунок 4.32 – Сторінка симуляцій

При натисканні на картку відкривається модальне вікно, в якому розпочинається запуск симуляції (рис. 4.33). Взагалі симуляція – покрокова інструкція дій під час НС. У кроці зациклюється відео з можливістю його прискорення або уповільнення та описуються кроки, які можна робити безліч разів, до поки не завершити симуляцію.

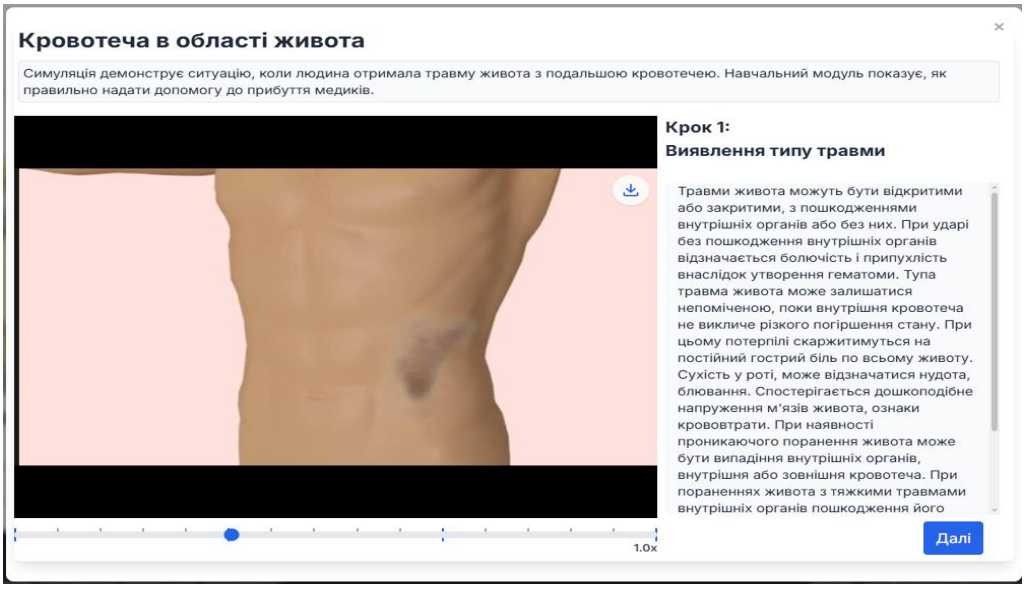


Рисунок 4.33 – Запуск симуляції

Кнопки Назад/Далі слугують певною навігацією для користувача. При переході до останього кроку симуляції замість кнопки Далі з’являється кнопка Завершити, яка й означає кінець симуляції (рис. 4.34).

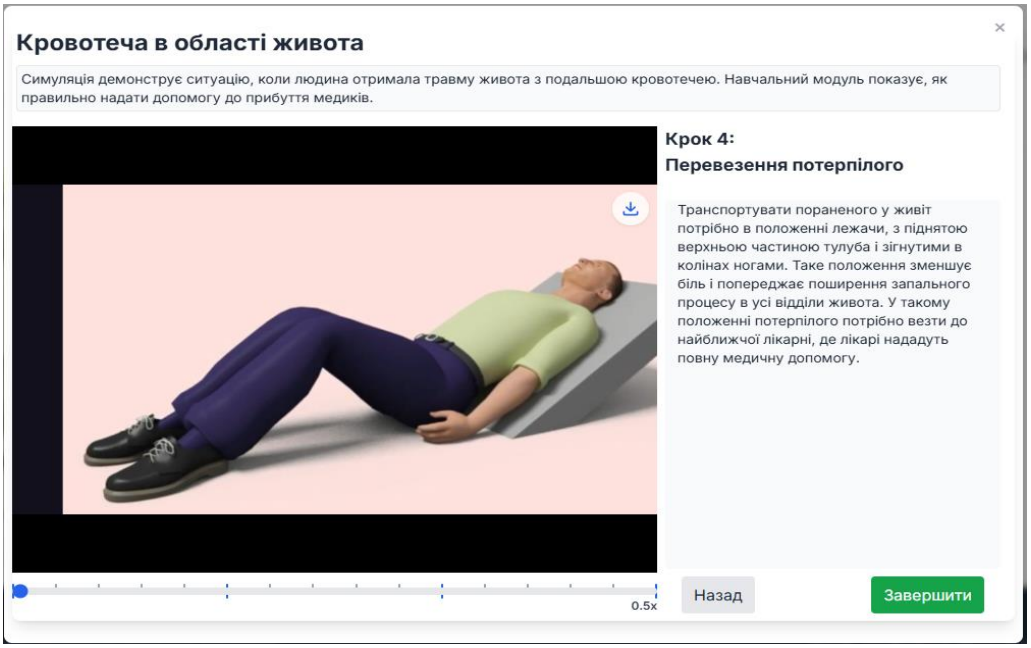


Рисунок 4.34 – Останній крок симуляції

На рис. 4.35 показано запуск симуляції на головній сторінці платформи.

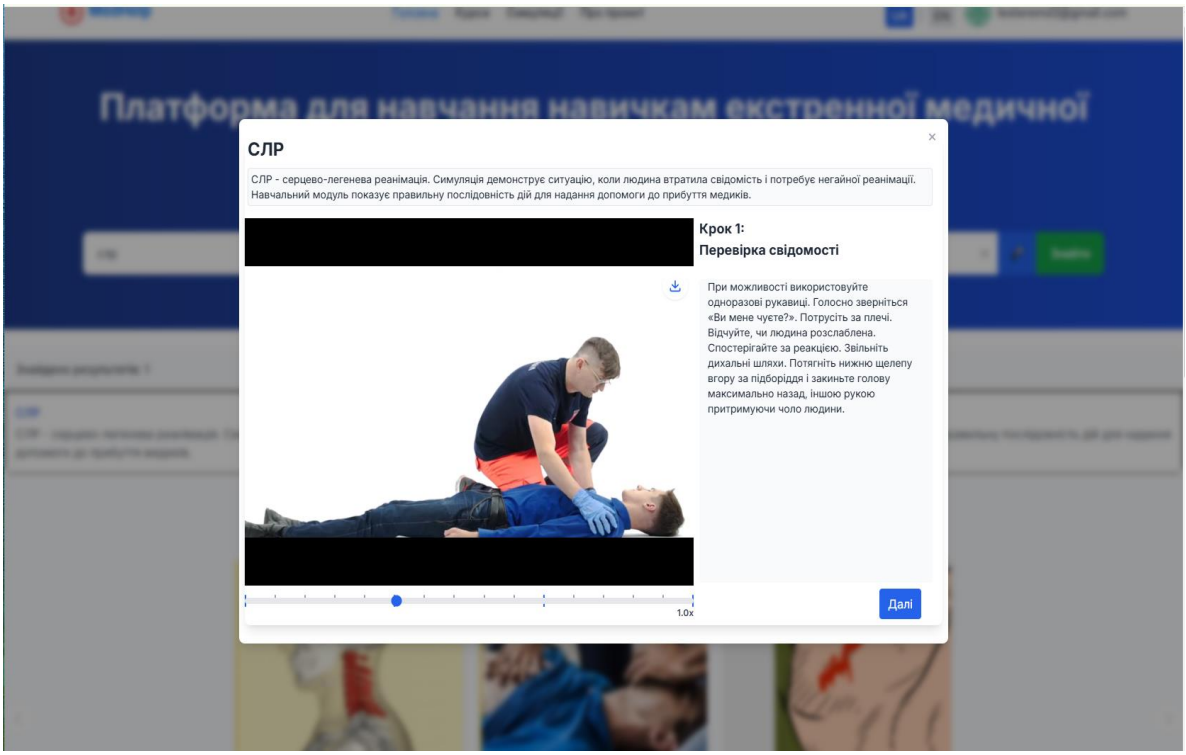


Рисунок 4.35 – Симуляція на головній сторінці

4.7 Сторінка користувачів

Сторінку користувачів призначено тільки для користувача з ролью в системі ROOT. Якщо звичайний користувач (USER) або медик (MEDIC) спробують зайти за ендпоінтом сторінки користувачів, то їм буде продемонстровано сторінку помилки статус 403 – Немає прав доступу до сторінки. Таким чином взаємодіяти з сторінкою користувачів може тільки ROOT.

У навігаційному блоці ROOT натискає на іконку профілю, там знаходиться вкладка користувачі тільки для ROOT. Натиснувши на неї, ROOT переходить на сторінку користувачів, на якій знаходяться наступні елементи: вкладки (Усі користувачі, окрім ROOT; користувачі – USERS; медики – MEDICS), пошукове поле та картки користувачів (рис. 4.36).

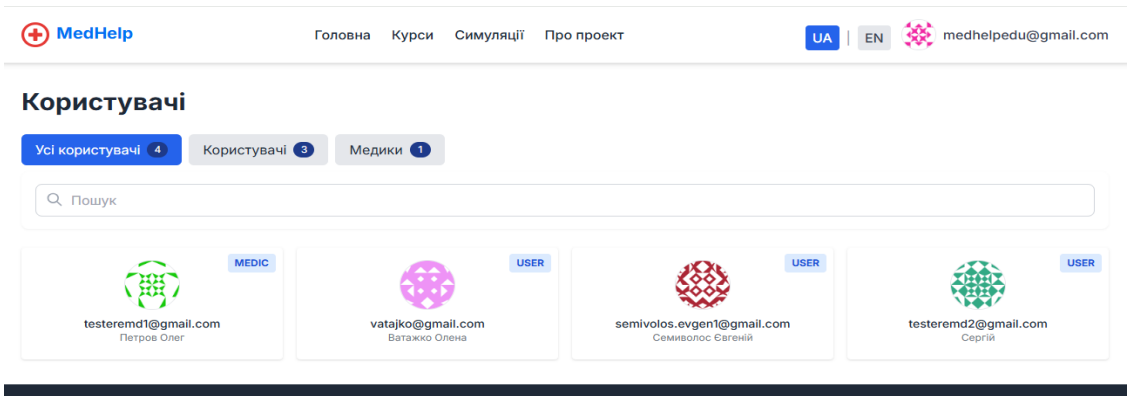


Рисунок 4.36 – Сторінка користувачів

Сторінку користувачів було розроблено тільки для управління ролями користувачів: USER → MEDIC та USER ← MEDIC. Пошук здійснюється за частковим співпадінням назв пошти або імені користувача.

Оскільки весь шлях дослідження платформи проходив через новоствореного користувача testeremd2@gmail.com – було обрано надати йому роль медика (MEDIC) через модальне вікно при натисканні на картку користувача (див. рис. 4.37) для подальшого дослідження платформи користувачем з іншою ролью.

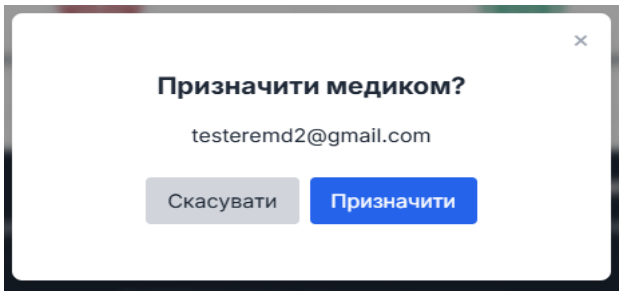


Рисунок 4.37 – Модальне вікно призначення MEDIC

На рис. 4.38 видно, що користувач з поштою testeremd2@gmail.com знаходиться на вкладці медиків, а отже, роль користувача успішно змінена.

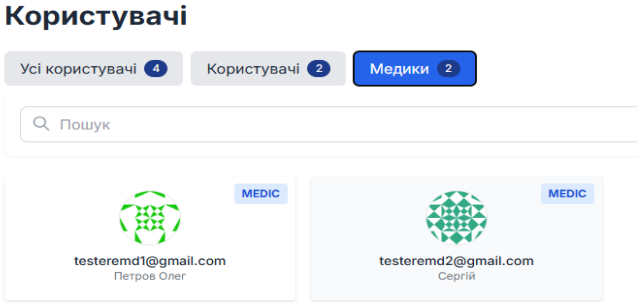


Рисунок 4.38 – Вкладка медиків

4.8 Управління курсами та симуляціями

На платформі авторизовано користувача testeremd2@gmail.com з новою ролью MEDIC. На рис. 4.39 показано сторінку курсів з новою вкладкою Мої курси для користувачів з ролью MEDIC.

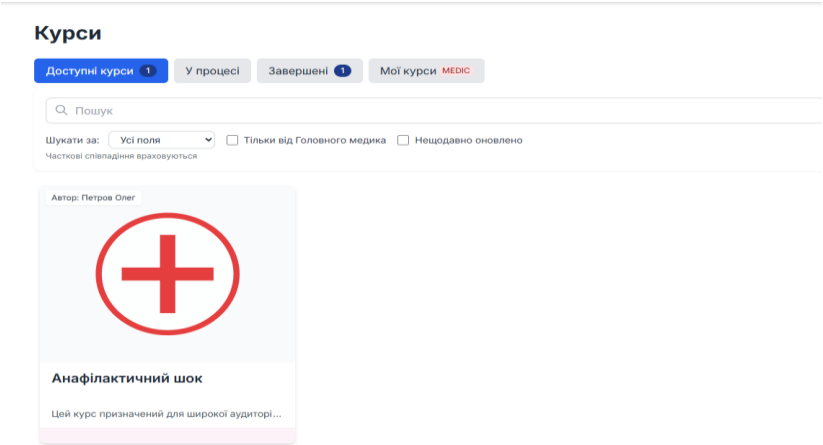


Рисунок 4.39 – Сторінка курсів з новою вкладкою

Вкладка Мої курси (MEDIC) служить для управління власними курсами, на якій є картка з назвою «Створити». При натиснені на неї користувачу відкривається модальне вікно форми для створення курсу (рис. 4.40).

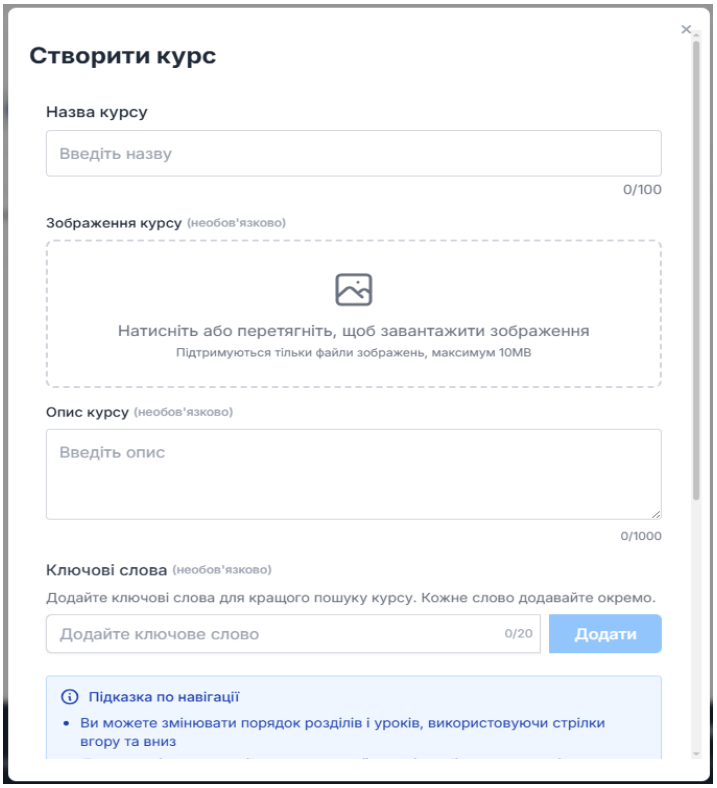


Рисунок 4.40 – Форма для створення курсу

На рис. 4.41 показано продовження форми для створення курсу, а саме частина з розділами та уроками.

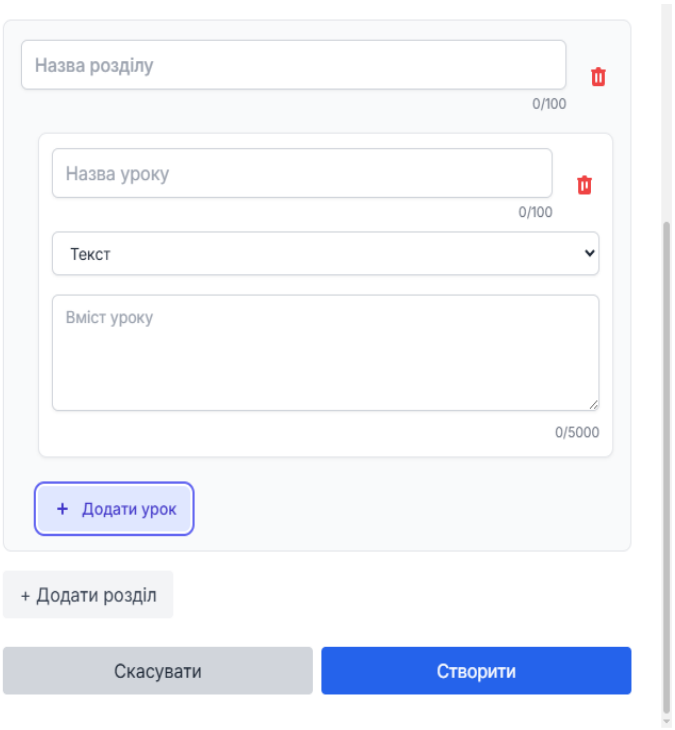


Рисунок 4.41 – Розділи та уроки форми

Далі буде продемонстровано та досліджено увесь процес заповнення форми для створення майбутнього курсу.

Форма має наступні поля для заповнення; назва, зображення, опис, ключові слові, розділи та уроки. Форма не може бути відправлена при помилці заповнення. Спочатку файли завантажуються до форми. Після відправки форми без помилок починається процес завантаження їх у сховище.

На рис. 4.42 та рис. 4.43 показано помилки при відправленні пустої форми. Назви курсу, розділу та уроку не можуть бути порожні та вміст уроку не може бути порожнім (для текстового уроку).

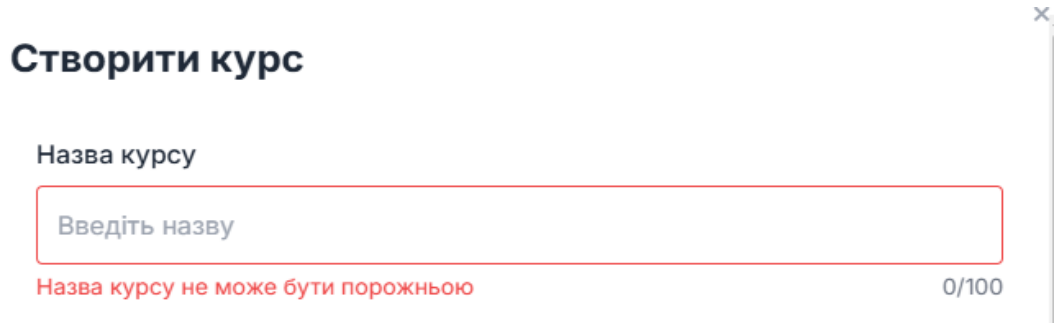


Рисунок 4.42 – Помилка назви курсу

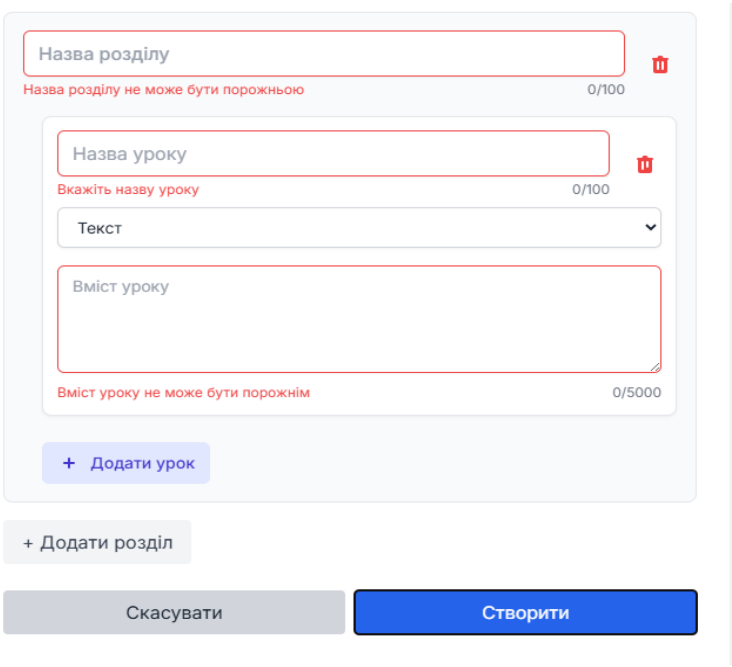


Рисунок 4.43 – Помилки назв та вмісту розділів-уроків

Було заповнено назву курсу. При завантаженні зображення курсу можуть виникнути наступні помилки: файл більше 10 мегабайт та файл не формату зображень (рис. 4.44). В іншому випадку помилок не буде (див. рис. 4.45).

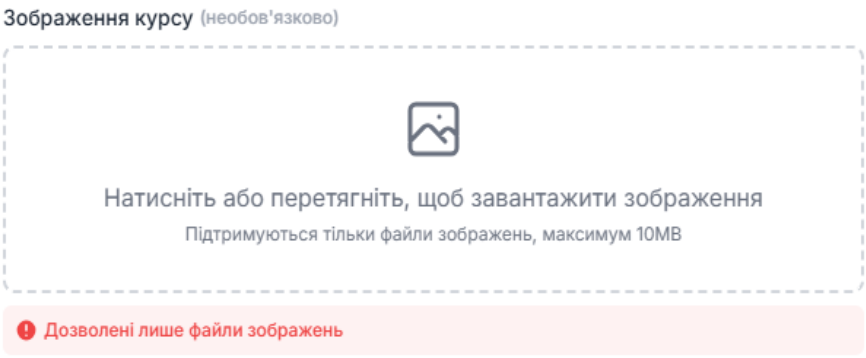


Рисунок 4.44 – Помилка формату файлу

Рисунок 4.45 – Успішне завантаження зображення

Було заповнено опис курсу. Було введено ключове слово (рис. 4.46). При вводі ключових слів можуть виникнути наступні помилки: відразу додається декілька ключових слів та слово вже існує (рис. 4.47).

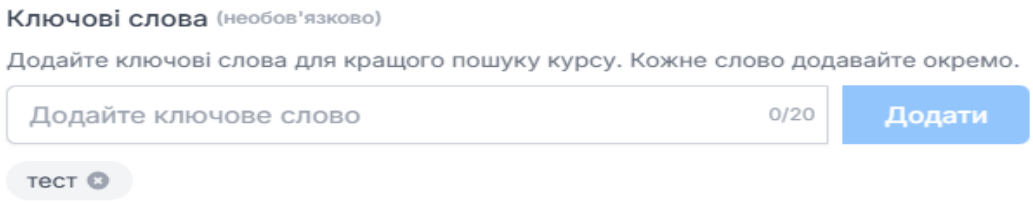


Рисунок 4.46 – Успішне додавання ключового слова

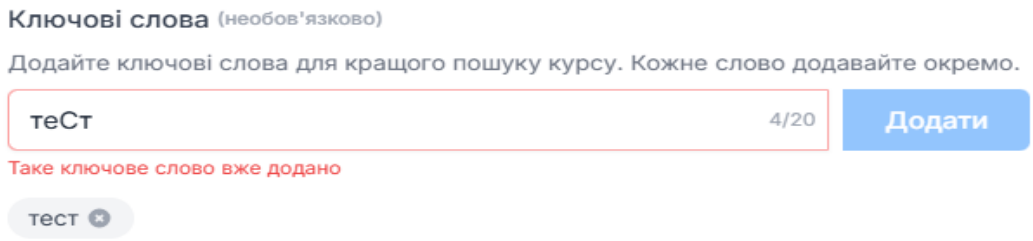


Рисунок 4.47 – Помилка додавання існуючого слова

На рис. 4.48 показано панель з типами уроків, які можна обрати.

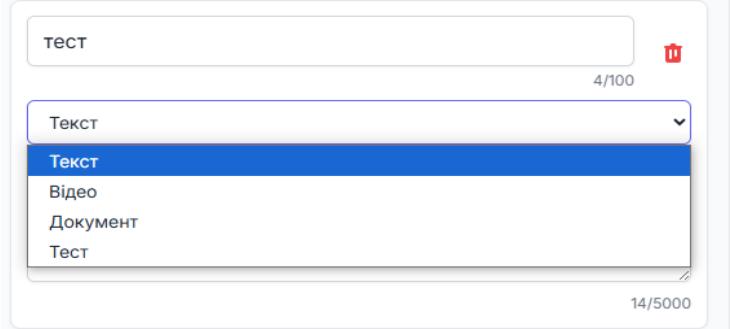


Рисунок 4.48 – Панель з типами уроків

На рис. 4.49 показано урок типу Текст, який містить обов'язкове поле вводу тексту до 5000 символів. Поле вводу можна розтягувати.

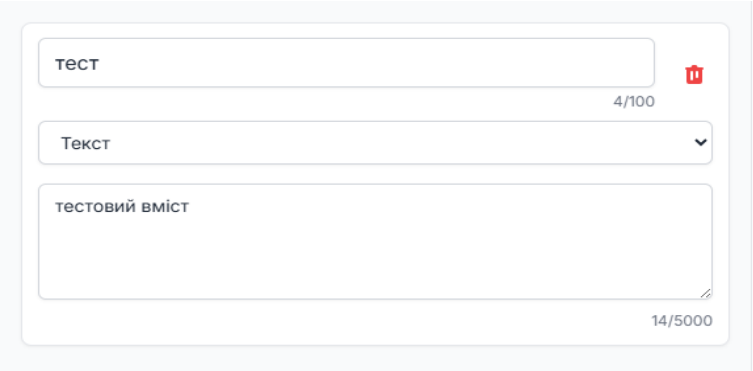


Рисунок 4.49 – Блок уроку типу Текст

На рис. 4.50 показано урок типу Відео, який містить обов'язкове поле завантаження відеофайлу та необов'язкове поле опису до 500 символів.

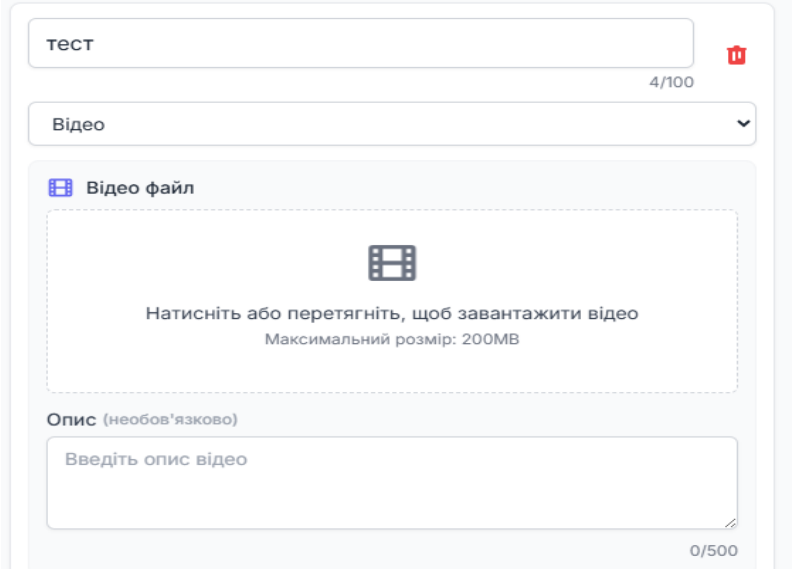


Рисунок 4.50 – Блок уроку типу Відео

При завантаженні відеофайлу до уроку можуть виникнути наступні помилки: файл не обрано, файл не формату відео та файл більше 200 мегабайт (рис. 4.51). В іншому випадку помилок не буде (див. рис. 4.52).

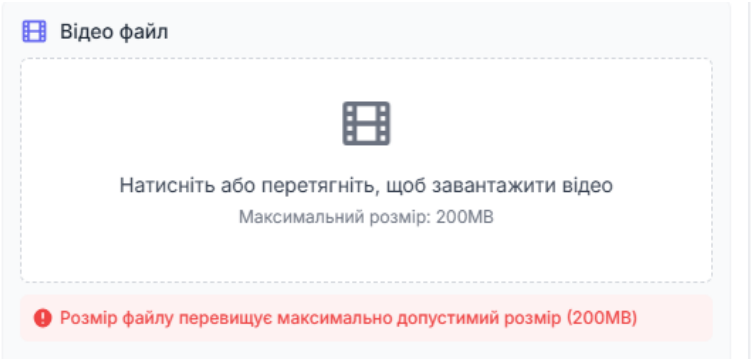


Рисунок 4.51 – Помилка максимального розміру файлу

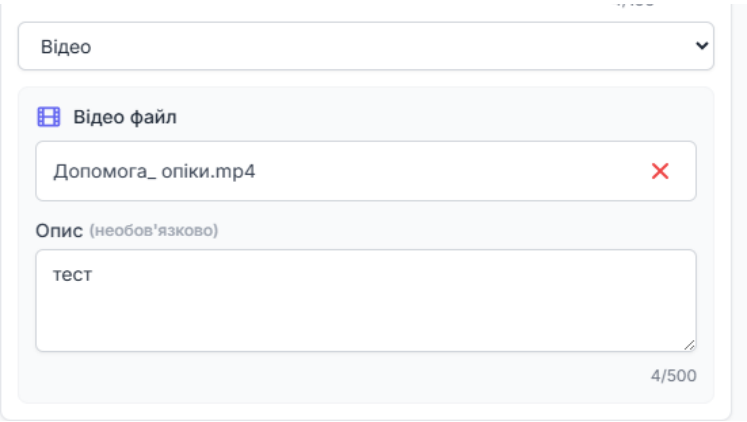


Рисунок 4.52 – Приклад завантаженого відеофайлу

Якщо натиснути на крестик біля завантаженого файлу, то відкриється модальне вікно з можливістю його прибирання.

На рис. 4.53 показано урок типу Документ, який містить обов'язкове поле завантаження файлу підтримуваних форматів та необов'язкове поле опису до 500 символів. Помилки ідентичні до помилок відео: неправильний формат файлу, перевищення максимального розміру, пустий урок.

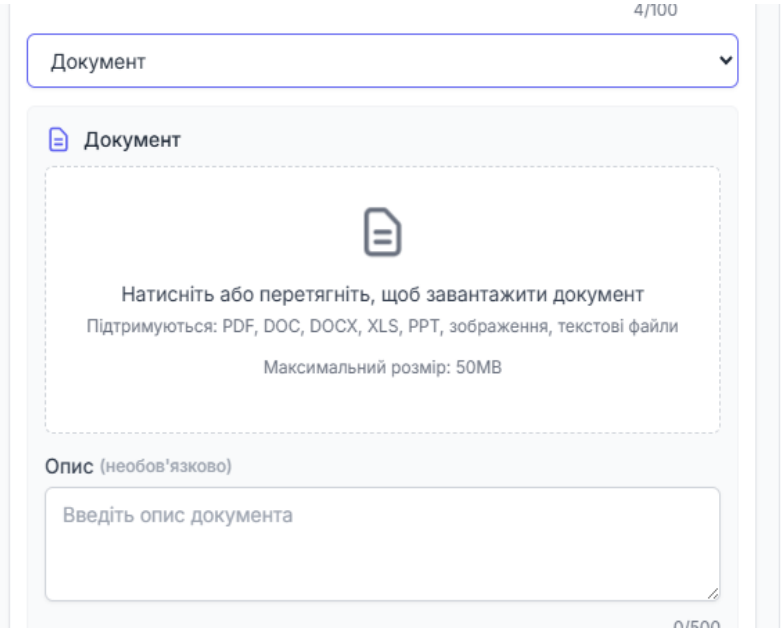


Рисунок 4.53 – Блок уроку типу Документ

На рис. 4.54 показано урок типу Документ з завантаженим файлом.

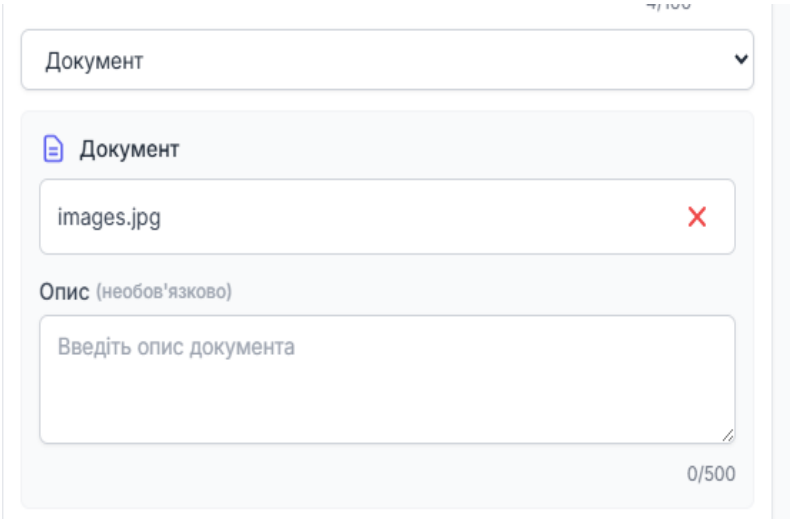


Рисунок 4.54 – Приклад завантаженого файлу для Документ

На рис. 4.55 показано форму для заповнення уроку типу Тест, яка розділена на два блоки: в 1 блоці (зліва) розташовані питання до тесту, кнопки

Додати питання та Зберегти стан тесту; в 2 блоці (справа) заповнюється питання, обирається тип питання, будуються різні варіанти відповідей.

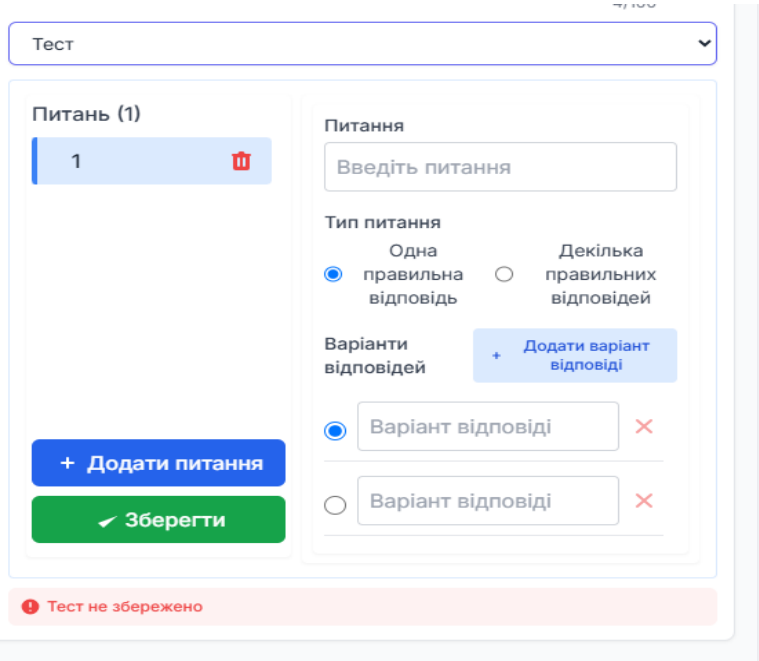


Рисунок 4.55 – Блок уроку типу Тест

На рис. 4.56 показано, що збереження тесту без варіантів відповідей або питання видає помилку.

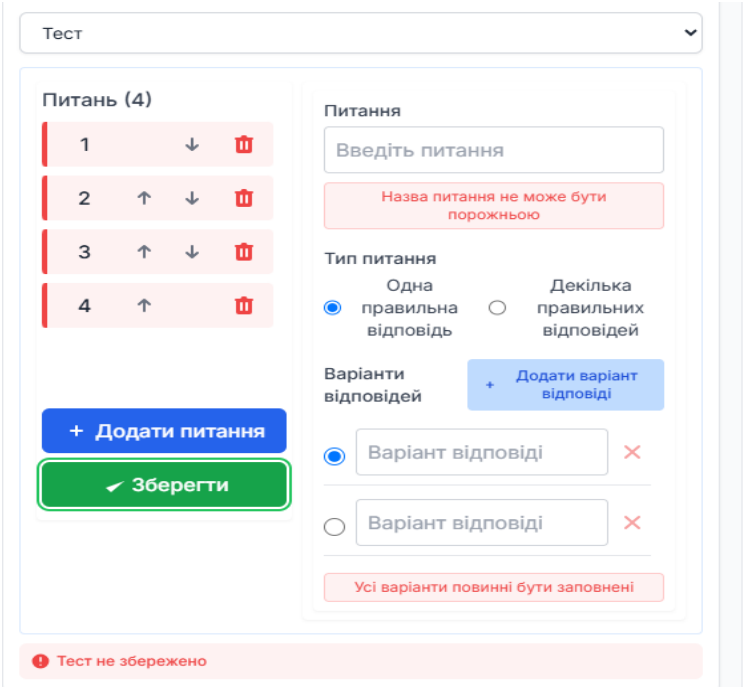


Рисунок 4.56 – Помилка збереження тесту

На рис. 4.57 показано, що помилка для кожного питання тесту зникає тільки тоді, коли питання та варіанти відповідей повністю заповнені.

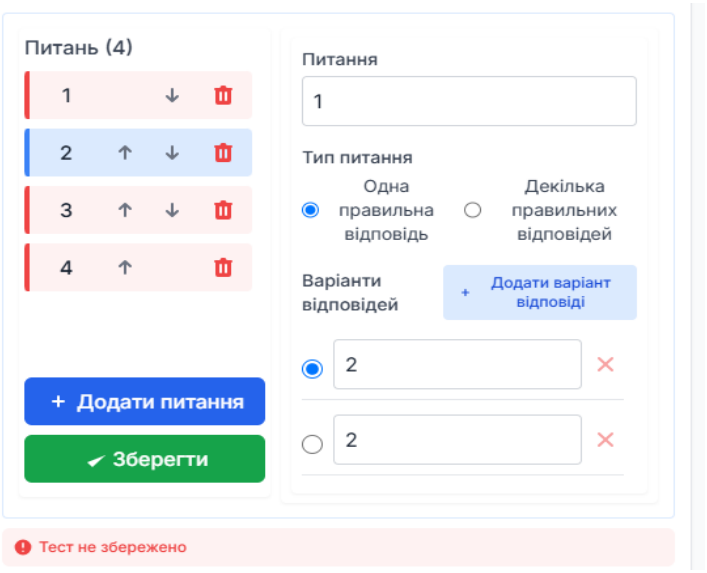


Рисунок 4.57 – Виправлення помилки тесту

Після виправлення усіх помилок тест було успішно збережено (рис. 4.58). Тепер тест можна редагувати або видалити.

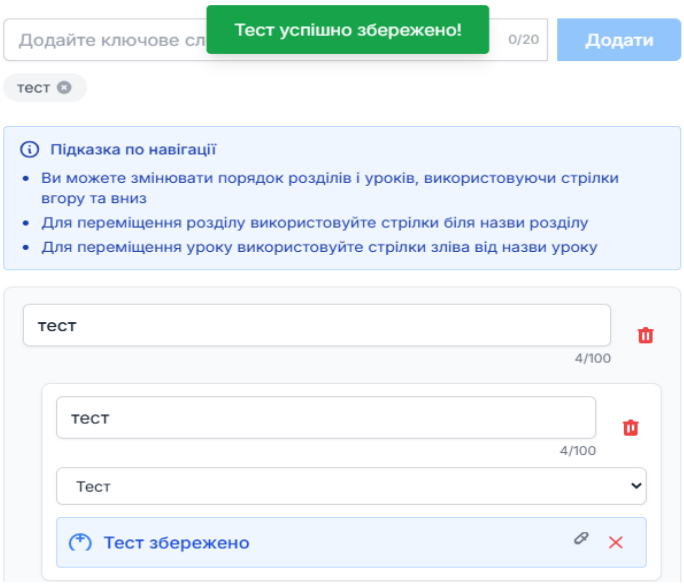


Рисунок 4.58 – Успішне збереження тесту

Ознайомлення з заповненням форми для створення курсів завершено. Усі передбачувальні ситуації реалізовано.

На основі всіх введених полів форми при дослідженні, було створено базову структуру курсу, який складається з 2 розділів по 2 уроки кожного з типів. Після відправки форми було отримано повідомлення про успішне створення курсу, а на вкладках з курсами з’явилась картка новоствореного курсу (див. рис. 4.59).

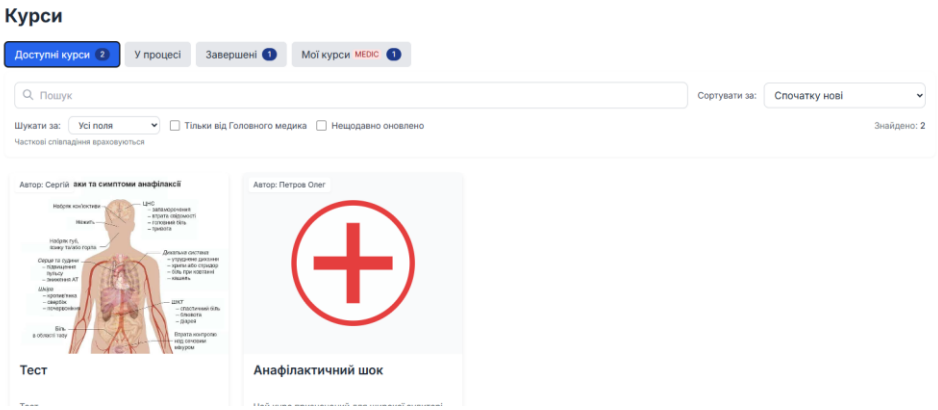


Рисунок 4.59 – Новостворений курс на вкладці доступних курсів

На рис. 4.60 показано вміст уроків новоствореного курсу.

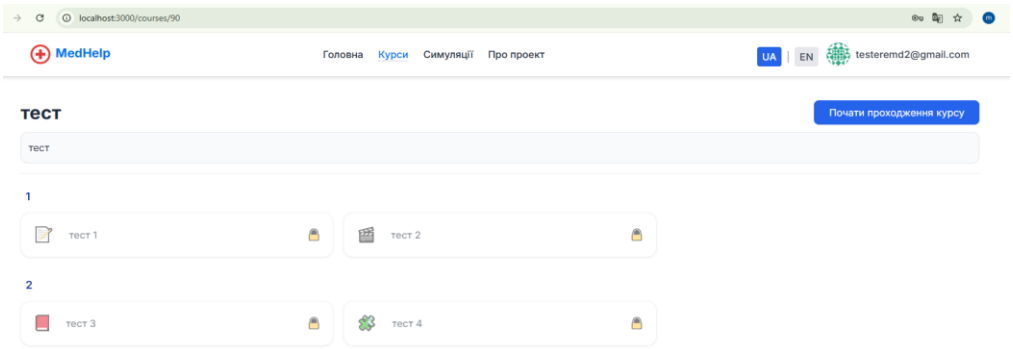


Рисунок 4.60 – Сторінка новоствореного курсу

Якщо відкрити вкладку Мої курси, то можна побачити на кожній картці кнопки управління курсом: редагування та видалення (рис. 4.61).

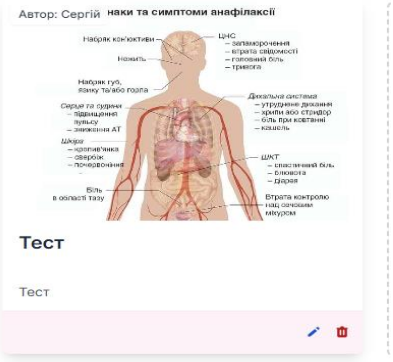


Рисунок 4.61 – Картка з інструментами управління

При натисканні користувачем кнопки редагування курсу, відкривається модальне вікно з формою для редагування курсу (рис. 4.62). Ця форма для редагування ідентична формі для створення, єдина відмінність, що дані підтягуються з БД для редагування курсу.

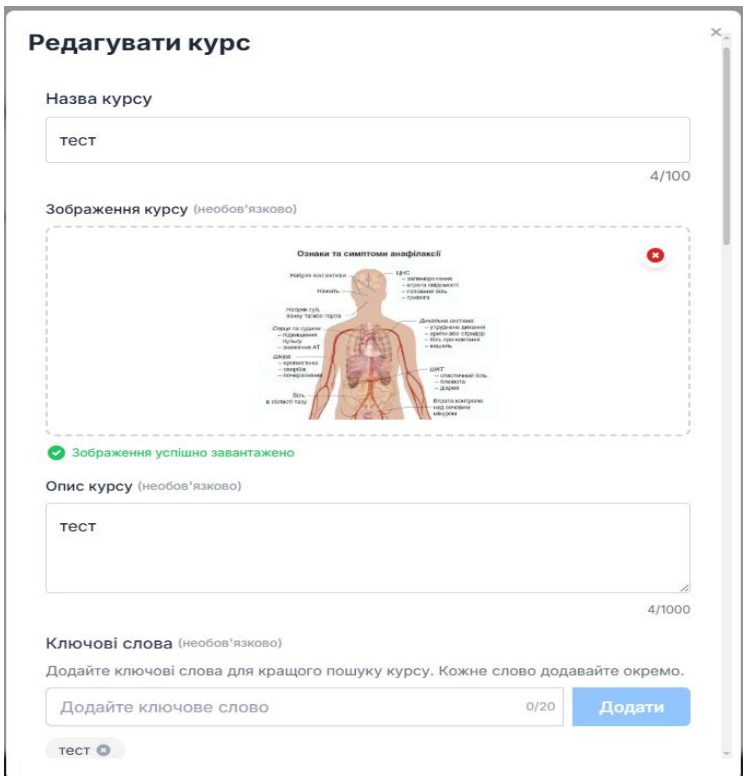


Рисунок 4.62 – Форма для редагування курсу

Форма відображає усі прикріплені файли як завантажені, оскільки в БД є посилання на них. Тест також збирається перед відображенням форми.

Якщо, наприклад, вибрати інший файл чи тип для уроку – зелена галочка зникне і при підтвердженні редагування почнеться процес завантадення файлу у сховище. На рис. 4.63 такий приклад продемонстровано для уроку тест 3.

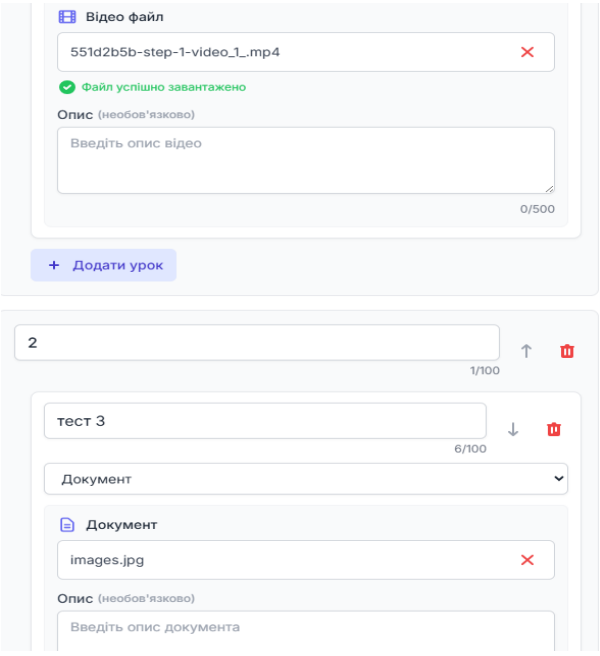


Рисунок 4.63 – Прикріплення нового файлу уроку

Після внесених змін можна підтверджувати форму редагування курсу. На рис. 4.64 показано оновлення стилю картки після успішного редагування.

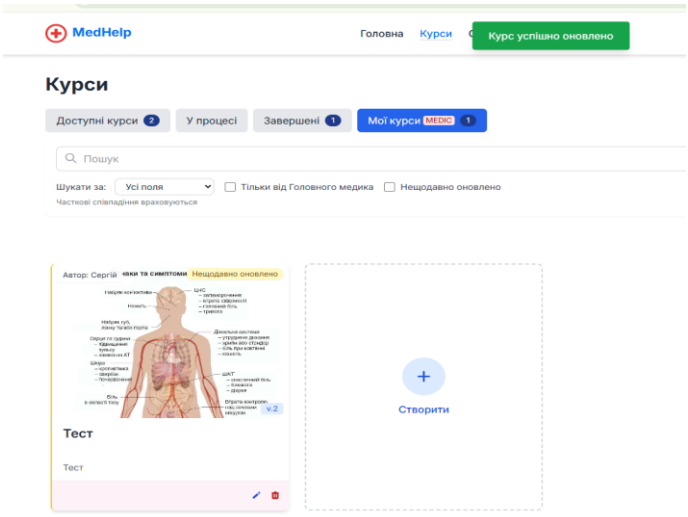


Рисунок 4.64 – Оновлений стиль картки

На рис. 4.65 продемонстровано сторінку відредагованого курсу, де можна побачить, що зміни пройшли успішно, оскільки 3 урок курсу змінив відображення іконки с червоної книжки (PDF) на картинку (зображення).

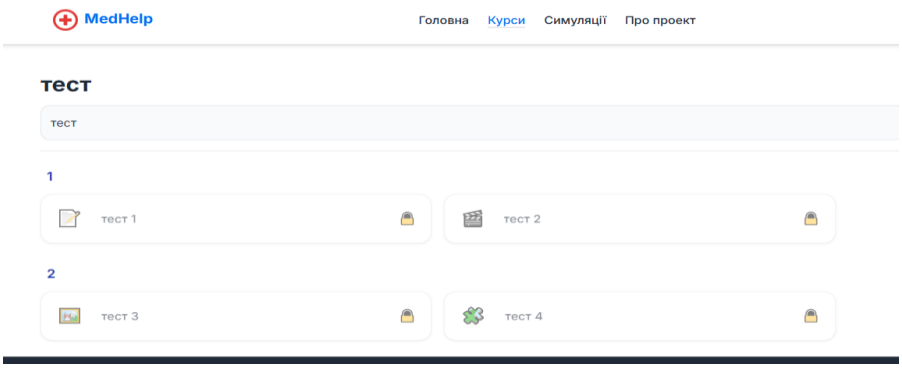


Рисунок 4.65 – Сторінка оновленого курсу

Після досліджень курсу можна його видалити. Для цього потрібно натиснути на відповідну кнопку видалення, після чого відкривається модальне вікно підтвердження дій на конкретному курсі на видалення (рис. 4.66)

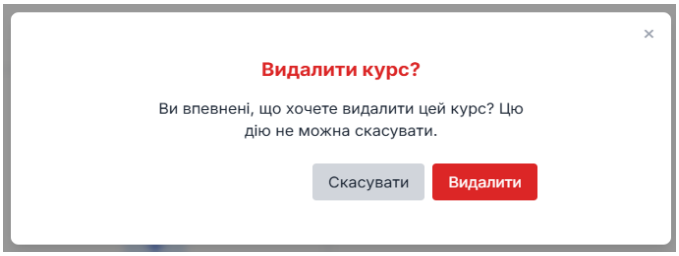


Рисунок 4.66 – Модальне вікно видалення курсу

Перевірка симуляцій. На рис. 4.67 видно, що на сторінці симуляцій також з’явилася вкладка Мої симуляції для MEDIC користувача.

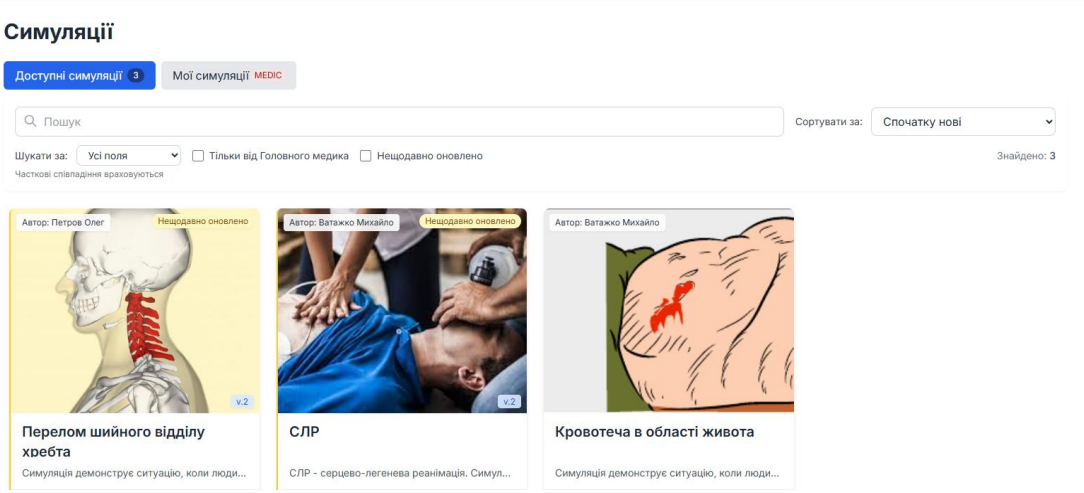


Рисунок 4.67 – Нова вкладка Мої симуляції

Алгоритм дій такий самий, як для курсів: перехід на вкладку Мої симуляції, вибір створення симуляції (картка «Створити»), відкривається модальне вікно з формою для створення симуляції (рис. 4.68).

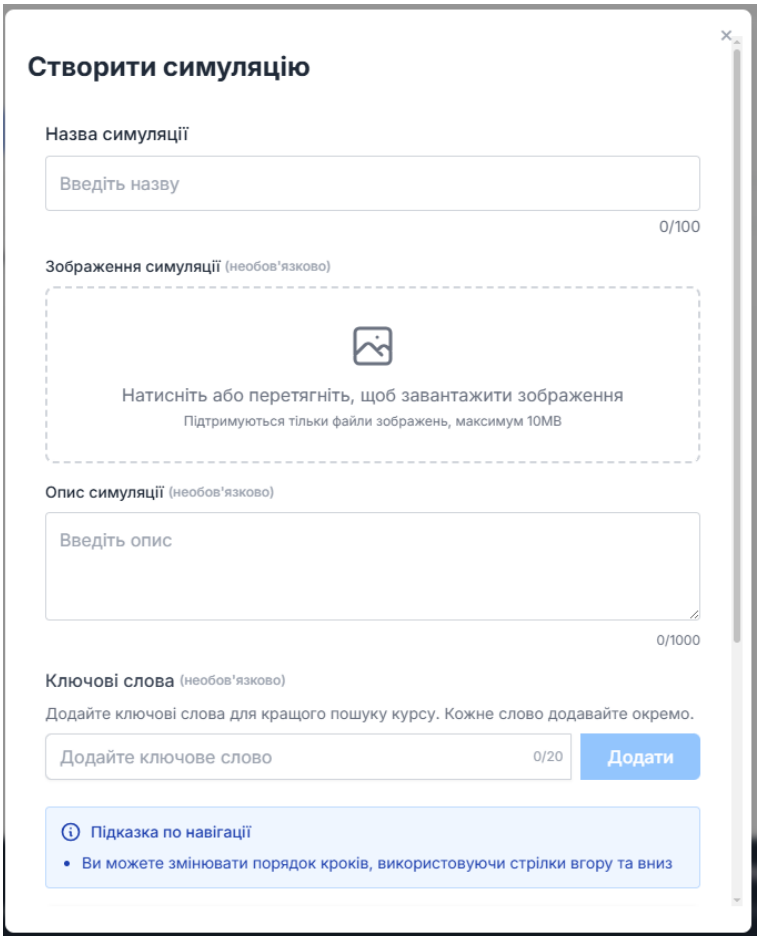


Рисунок 4.68 – Форма для створення симуляції

Вміст форми для створення симуляцій відрізняється від форми для створення курсів лише наповненням навчального контенту. У симуляцій це кроки. На рис. 4.69 показано вигляд симуляції у формі.

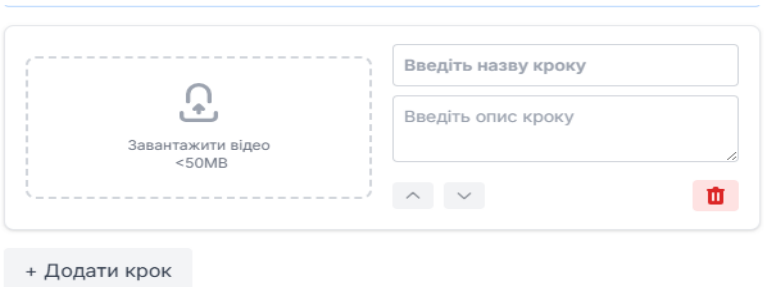


Рисунок 4.69 – Крок симуляції для форми

У формі для створення симуляції крок складається з наступного: лівий блок (1 частина) містить у собі відеоматеріал з підтримкою помилок (рис. 4.70); правий блок (2 частина) містить у собі назву та поле опису – не можуть бути порожніми.

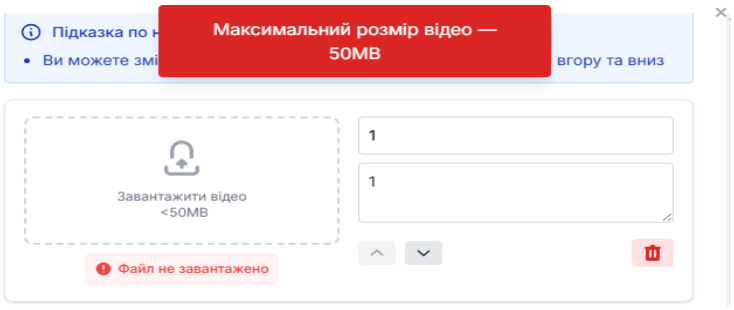


Рисунок 4.70 – Помилка максимального розміру файлу

При успішному завантаженні файлу до форми, показується превью відео та помилки знакають (рис. 4.71).

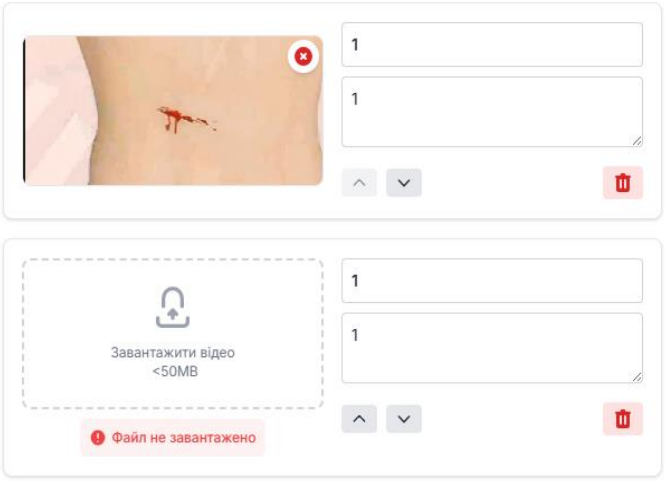


Рисунок 4.71 – Запис відеоматеріалу

У результаті було створену базову симуляцію з 1 кроком працювання. На рис. 4.72 показано запуск новоствореної симуляції. Форма для редагування симуляції дублює логіку з форми для редагування курсів. Видалення також.

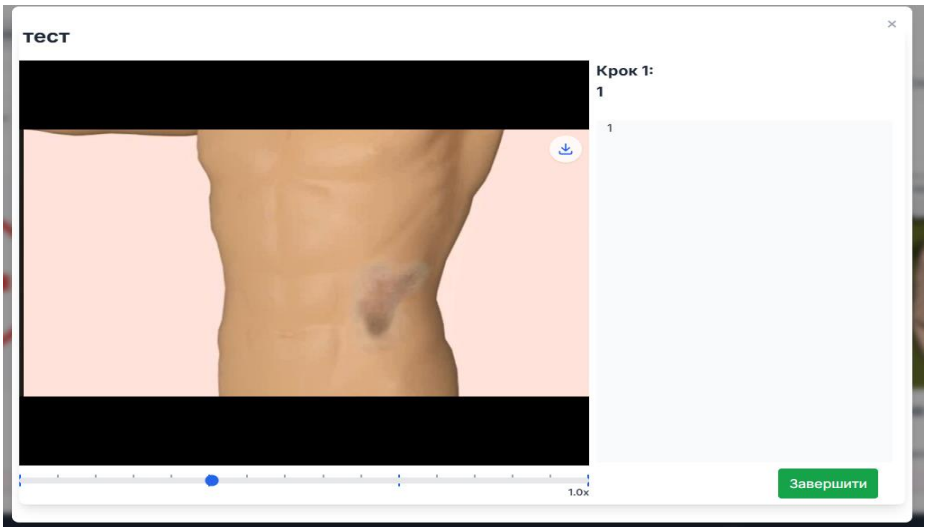


Рисунок 4.72 – Запуск новоствореної симуляції

4.9 Мобільний вигляд

Також варто було протестувати вид розробленої платформи для мобільного дисплею. Тому на рис. 4.73-80 будуть надані скріншоти з мобільного екрана веб-браузера, де була відкрита платформа.

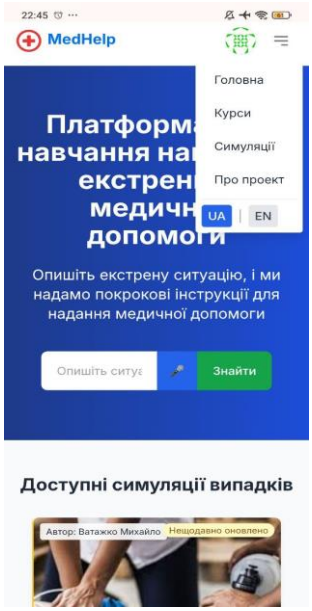


Рисунок 4.73 – Головна сторінка

Рисунок 4.74 – Доступні симуляції

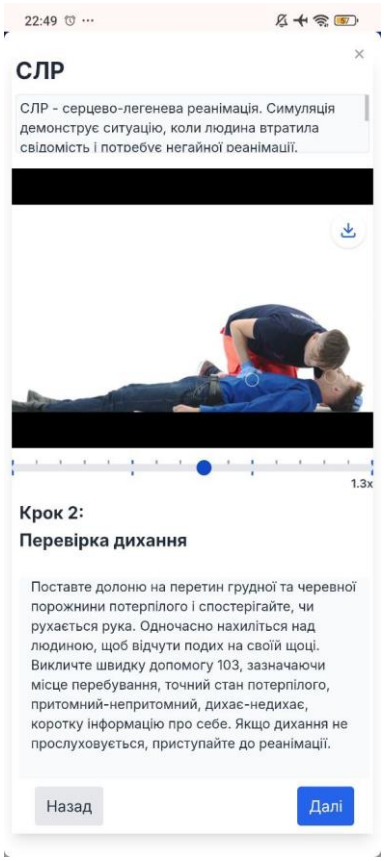


Рисунок 4.75 – Симуляція

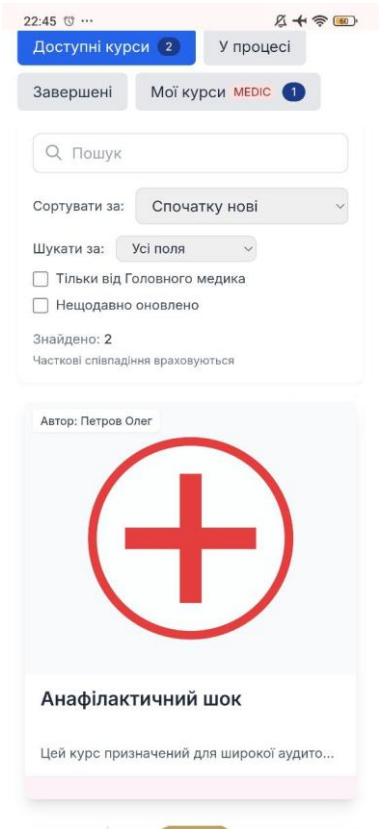


Рисунок 4.76 – Сторінка курсів

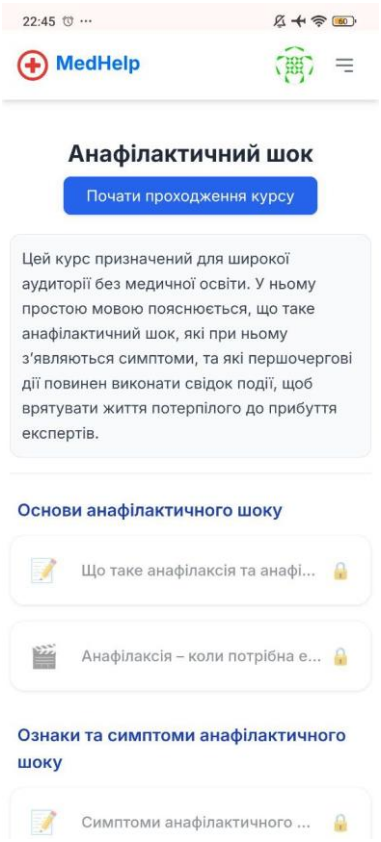


Рисунок 4.77 – Сторінка курсу

Рисунок 4.78 – Відеоурок

Рисунок 4.79 – Документ урок

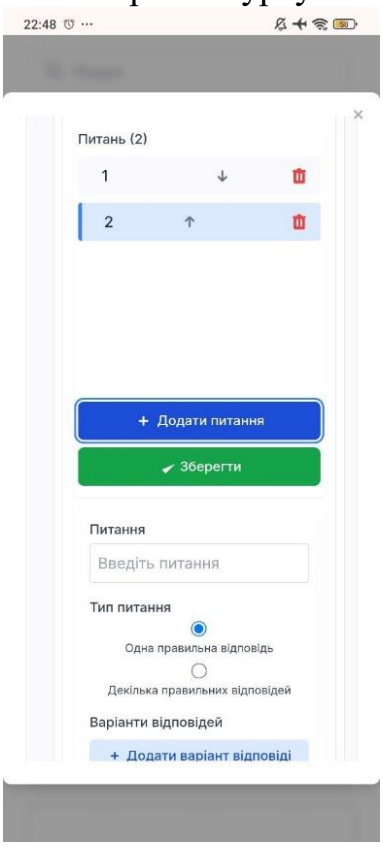


Рисунок 4.80 – Форма для створення тесту

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі дослідження та аналіз роботи платформи підтвердило її відповідність технічним вимогам.

Інтерфейс виявився інтуїтивно зрозумілим, з чіткою структурою навігації та ефективною системою пошуку, яка підтримує як текстові, так і голосові запити. Локалізація працює коректно, забезпечуючи зручне користування для українськомовних та іноземних користувачів.

Система автентифікації реалізована з дотриманням безпечних стандартів, з чітким розподілом ролей та обмеженням доступу. Функціонал фільтрації дозволяє швидко знаходити потрібні матеріали за різними критеріями.

Для користувачів з правами MEDIC та ROOT доступний повний цикл роботи з контентом: створення, редагування та видалення курсів і симуляцій. Платформа підтримує різні формати навчальних матеріалів, включаючи тексти, документи, відео та інтерактивні тести. Симуляції реалізовані у вигляді покрокових інструкцій з можливістю повторного відпрацювання.

Адаптивний дизайн забезпечує коректне відображення на різних пристроях. Загалом, платформа демонструє стабільну роботу та відповідає поставленим завданням.

ВИСНОВКИ

Розробка платформи для навчання навичкам ЕМД стала відповіддю на критичні проблеми, висвітлені у вступі, зокрема низьку доступність якісних навчальних ресурсів для цивільного населення та недостатній рівень практичної підготовки. Дослідження показало, що існуючі системи не задовольняють сучасні потреби у навчанні ЕМД, особливо в умовах військового стану.

У першому розділі було виявлено, що переважна більшість сучасних платформ орієнтовані виключно на професійних медиків, не враховуючи потреб цивільного населення. Аналіз чотирьох провідних систем (AEMД, Progress, Lexipol та Médecine d'Urgence) показав відсутність комплексного підходу, що поєднував би теоретичну підготовку з практичними тренуваннями. Це обґрунтувало необхідність створення нової платформи, здатної охопити різні групи користувачів.

У другому розділі було спрамовано технічну реалізацію на забезпечення стабільності та масштабованості системи. Використання сучасного технологічного стеку (Next.js, TypeScript, PostgreSQL) дозволило створити продуктивне рішення, здатне витримувати високе навантаження при масовому використанні. Інтеграція з хмарним сховищем Backblaze B2 та застосування Docker забезпечили надійність і простоту розгортання платформи.

У третьому розділі особливу увагу приділено практичній реалізації функціоналу, зокрема:

- Рольовій системі, що дозволяє використовувати платформу як навчальний модуль у медичних університетах.
- Інтерактивним симуляціям, які максимально наближають умови навчання до реальних ситуацій.
- Адаптивному інтерфейсу, зрозумілому для користувачів з різним рівнем підготовки.

У четвертому розділі було проведено дослідження платформи, яке підтвердило її ефективність та відповідність поставленим вимогам. Платформа демонструє стабільну роботу на різних пристроях, забезпечуючи зручний доступ до навчальних матеріалів та можливість відпрацювання практичних навичок.

Для подальшого вдосконалення платформи можна реалізувати :

- Впровадження VR-симуляцій для більш реалістичного тренування.
- Розширення бібліотеки практичних кейсів.
- Інтеграцію з телемедичними сервісами.

Отже, розроблена платформа повністю відповідає поставленим завданням, пропонуючи сучасне, доступне та ефективне рішення для навчання навичкам ЕМД.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Graham R., McCoy M. A., Schultz A. M. Strategies to Improve Cardiac Arrest Survival: A Time to Act. Washington, DC, USA: The National Academies Press, 2015. P. 106 [Електронний ресурс] – Режим доступу до ресурсу: https://ohiohealthems.com/wp-content/uploads/2015/06/Graham_Strategies-to-Improving-Cardiac-Arrest-Survival-2015-06.pdf.
2. Журба О. Екстрена медична допомога: в МОЗ пояснили, що це, коли надається та чи потрібно за неї платити. Юридична газета. 2024. 16 січ [Електронний ресурс] – Режим доступу до ресурсу: https://yur-gazeta.com/golovna/ekstrena-medichna-dopomoga-v-moz-poyasnili-shcho-ce-koli-nadaetsya-ta-chi-potribno-za-neyi-platiti.html.
3. Про екстрену медичну допомогу : Закон України від 05.07.2012, № 5081-VI [Електронний ресурс] – Режим доступу до ресурсу: https://ukc.gov.ua/knowdledge/systema-ekstrenoyi-medychnoyi-dopomogy/.
4. Emergency medical services. Wikipedia. Early civilian ambulances [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Emergency_medical_services.
5. Emergency medical services in the United States. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Emergency_medical_services_in_the_United_States.
6. Emergency and urgent care services // Internet Archive’s Wayback Machine [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20091106000344/http:/www.nhs.uk/nhsengland/aboutnhsservices/emergencyandurgentcareservices/pages/ambulanceservices.aspx

7. Швидка медична допомога. Wikipedia. Україна [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Швидка_медична_допомога.
8. Про створення державної служби медицини катастроф : Постанова Кабінет Міністрів України від 14.04.1997, № 827 [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20111114182253/http://uazakon.com/document/spart82/inx82993.htm.
9. Дубіль В. Чи виживе екстрена медична допомога після «реформи» МОЗу [Електронний ресурс] – Режим доступу до ресурсу: https://lb.ua/blog/valeriy_dubil/363723_chi_vizhive_ekstrena_medichna_dopomoga.html.
10. Державний Заклад Медицини Катастроф. Українські медики отримали 50 швидких від уряду ОАЕ. Державний Заклад Медицини Катастроф. Новини. 06.02.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://dzmk.org.ua/2024/02/06/ukrainski-medyky-otrymaly-50-shvydkykh-vid-uriadu-oae/.
11. Завгородня О. Герої у білих халатах: як у Дніпрі працює екстрена медична допомога під час війни. Наше місто. Новини Дніпра. 29.04.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://nashemisto.dp.ua/2024/04/29/heroi-u-bilykh-khalatakh-iak-u-dnipri-pratsiuie-ekstrena-medychna-dopomoha-pid-chas-viiny/.
12. Мокляк А., Самко Я. Цілодобово під обстрілами: як працює екстрена медична допомога на Херсонщині. Суспільне Херсон. Новини. 18.01.2025 [Електронний ресурс] – Режим доступу до ресурсу: https://suspilne.media/kherson/925161-cilodobovo-pid-obstrilami-ak-pracue-ekstrena-medicna-dopomoga-na-hersonsini/.
13. Укрзалізниця випустила ще кілька вагонів для медичної евакуації. Укрінформ. Суспільство. 26.12.2024 [Електронний ресурс] – Режим

доступу до ресурсу: https://www.ukrinform.ua/rubric-society/3941832-ukrzaliznica-vipustila-se-kilka-vagoniv-dla-medicnoi-evakuacii.html.

14. Департамент охорони здоров'я та реабілітації Вінницької ОДА. З 2025 року до складу бригад екстреної медичної допомоги мають входити парамедики та екстрені медичні техніки. Facebook [Електронний ресурс] – Режим доступу до ресурсу: https://www.facebook.com/story.php?id=100057090714503&story_fbid=1055904202989260.
15. Устінов О. В. Доступний курс «Надання екстреної медичної допомоги на догоспітальному етапі». Український Медичний Часопис. Новини. 18.12.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://umj.com.ua/uk/novyna-261279-dostupnij-kurs-nadannya-ekstrenoyi-medichnoyi-dopomogi-na-dogospitalnomu-etapi.
16. АЕМД [Електронний ресурс] – Режим доступу до ресурсу: https://aemc.org.ua/.
17. Progress [Електронний ресурс] – Режим доступу до ресурсу: https://progressplatform.org/.
18. Progress. Курс PRO EMERGENCY [Електронний ресурс] – Режим доступу до ресурсу: https://progressplatform.org/pro-kurs-emergency.
19. Lexipol [Електронний ресурс] – Режим доступу до ресурсу: https://www.lexipol.com/industries/emergency-medical-services/.
20. Médecine d’Urgence [Електронний ресурс] – Режим доступу до ресурсу: https://medecinedurgence.fr/.
21. Thiercelin D. Le site Internet. Médecine d’Urgence. A propos [Електронний ресурс] – Режим доступу до ресурсу: https://medecinedurgence.fr/a-propos.
22. Shah D. Advantages and Disadvantages of Developing Web Applications. Solvios Technology. 30.07.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://solvios.technology/advantages-and-disadvantages-of-web-applications/.

23. Бондаренко С. Мобільні застосунки: їх види та особливості. Highload. 26.01.2023 [Електронний ресурс] – Режим доступу до ресурсу: https://highload.tech/uk/mobilni-zastosunki-yih-vidi-ta-osoblivosti/.
24. Desktop Vs Web App – The Main Advantages And Disadvantages. Technology Talker. APPS. 18.12.2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.technologytalker.com/desktop-vs-web-app-the-main-advantages-and-disadvantages/.
25. Software architecture. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_architecture.
26. Про архітектуру додатків. FoxmindEd. 28.03.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/arkhitektura-zastosunku/.
27. @Text Writer. Мікросервісна архітектура: плюси та мінуси. ITEDU. Статті. 09.02.2022 [Електронний ресурс] – Режим доступу до ресурсу: https://itedu.center/ua/blog/articles/microservices-architecture-advantages-and-disadventages/.
28. TypeScript. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/TypeScript.
29. JavaScript. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript.
30. Python. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python.
31. Java. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java.
32. React. Meta Open Source [Електронний ресурс] – Режим доступу до ресурсу: https://opensource.fb.com/projects/react/.
33. What is Angular?. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.dev/overview.
34. VueJS [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/.

35. NextJS vs React — Which One is Better for Web Development?. UXPin. Blog [Електронний ресурс] – Режим доступу до ресурсу: https://www.uxpin.com/studio/blog/nextjs-vs-react/.
36. Route Handlers. NextJS. Routing [Електронний ресурс] – Режим доступу до ресурсу: https://nextjs.org/docs/app/building-your-application/routing/route-handlers.
37. Герасимик І. Що таке база даних?. АПЕПС. Новини [Електронний ресурс] – Режим доступу до ресурсу: https://apeps.kpi.ua/shco-take-basa-danykh.
38. Vitaly S. SQL чи NoSQL – ось в чому питання. Альтернативна Наука. Програмування. 12.03.2021 [Електронний ресурс] – Режим доступу до ресурсу: https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/.
39. Comparing MySQL, PostgreSQL, and MongoDB. GeeksforGeeks. 26.09.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/comparing-mysql-postgresql-and-mongodb/.
40. Об’єктно-реляційне відображення. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Об%27єктно-реляційне_відображення.
41. Shariq A. Popular ORMs and Their Difference: Prisma, TypeORM, and Sequelize. Medium. @shariq.ahmed525. 18.01.2024 [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@shariq.ahmed525/popular-orms-and-their-difference-prisma-typeorm-and-sequelize-564a83575eea.
42. What is Cloud Storage?. Google Cloud. Cloud storage [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/learn/what-is-cloud-storage.
43. About Backblaze B2 Cloud Storage. Backblaze [Електронний ресурс] – Режим доступу до ресурсу: https://www.backblaze.com/docs/cloud-storage-about-backblaze-b2-cloud-storage.

44. Pricing Organized by API Calls. Backblaze [Електронний ресурс] – Режим доступу до ресурсу: https://www.backblaze.com/cloud-storage/transaction-pricing.
45. What is Docker?. Docker [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/get-started/docker-overview/.

ДОДАТОК А

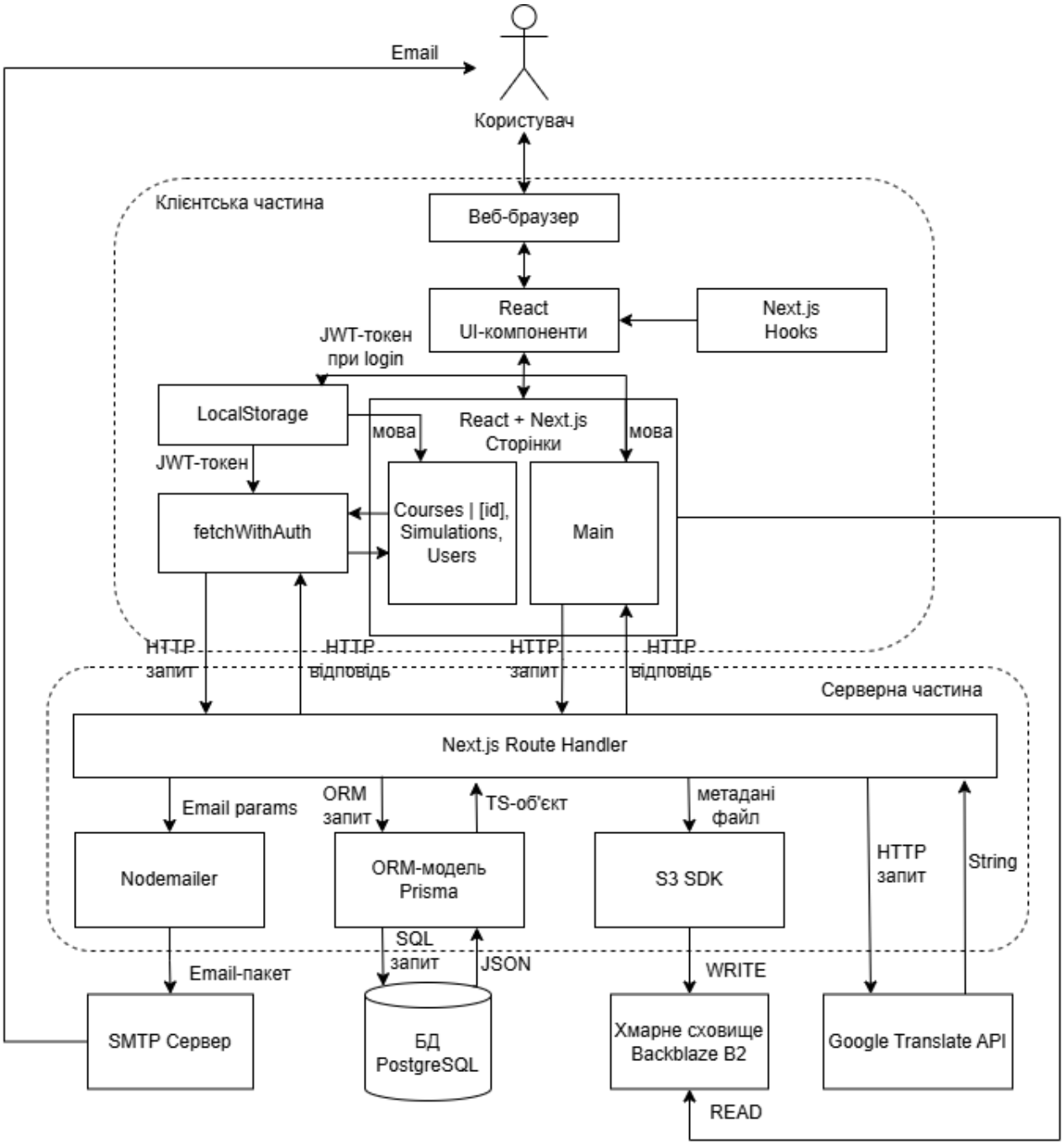
Платформа для навчання навичкам екстреної медичної допомоги

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Ватажко М. А.			**Платформа для навчання навичкам екстреної медичної допомоги** Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Павлов В. Г.					
Реценз.		Корнієнко Б. Я.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11	
Н. Контр.		Нікольський С. С.					
Затвердив		Новотарський М. А.					

ДОДАТОК Б

Платформа для навчання навичкам екстреної медичної допомоги

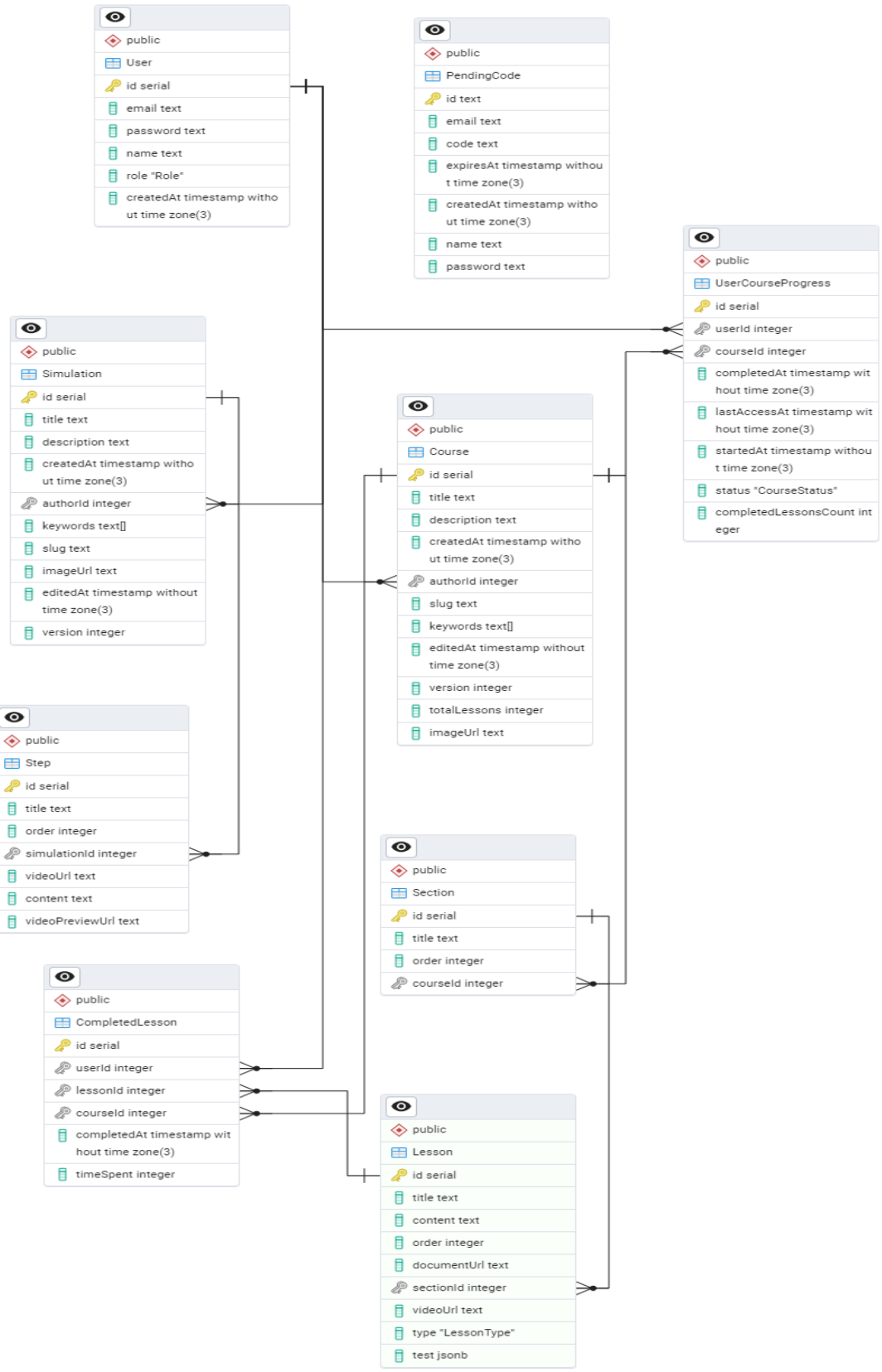
ER-діаграма бази даних

(функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 Д2
		№ докум.	Підпис	Дата	
Розробив		Ватажко М. А.			**Платформа для навчання навичкам екстреної медичної допомоги** ER-діаграма бази даних
Перевірив		Павлов В. Г.			Літ. / Аркуш 1 / Аркушів 1
Реценз.		Корнієнко Б. Я.			НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11
Н. Контр.		Нікольський С. С.			
Затвердив		Новотарський М. А.			

ДОДАТОК В

Платформа для навчання навичкам екстреної медичної допомоги

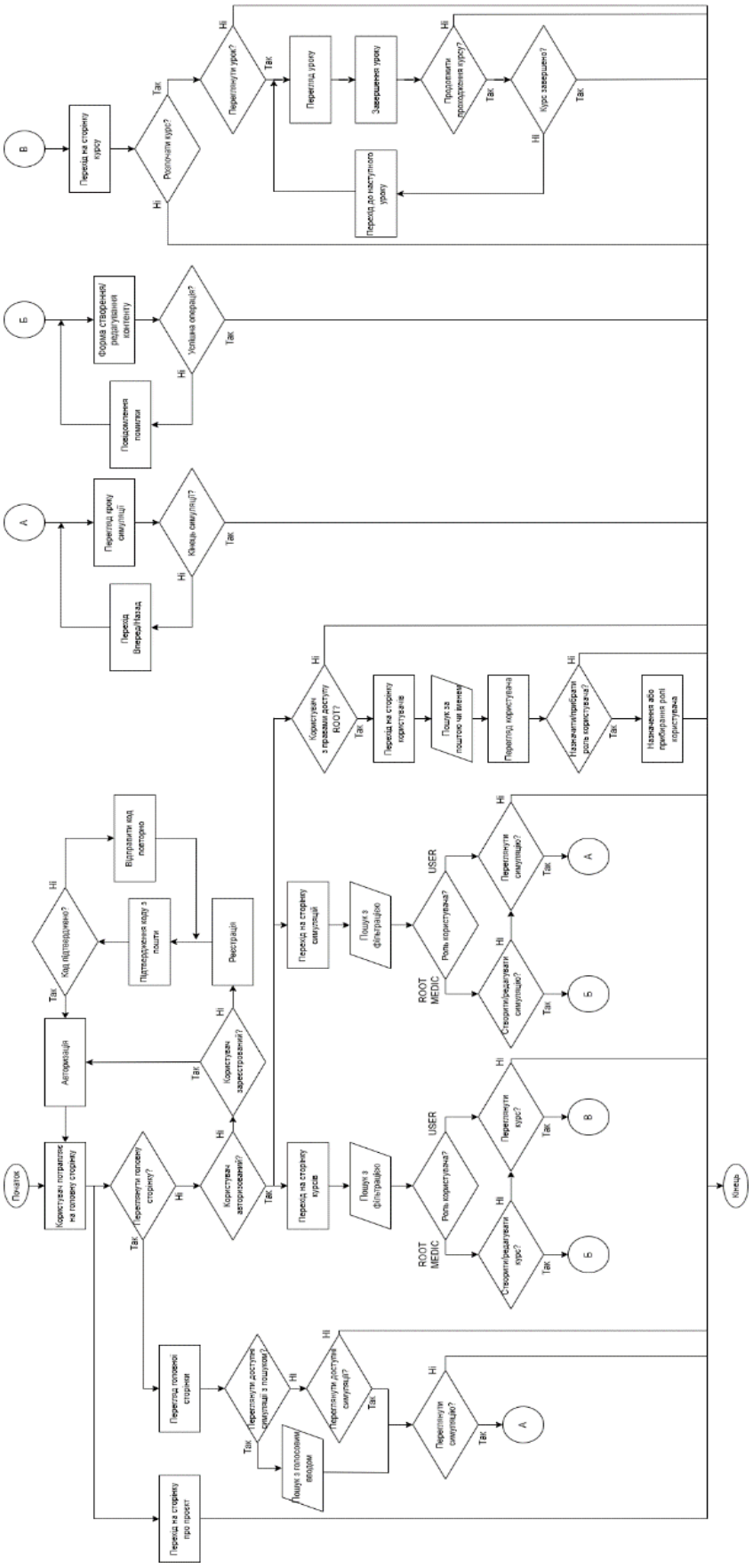
Алгоритм дій програмного забезпечення

(принципова схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Ватажко М. А.			**Платформа для навчання навичкам екстреної медичної допомоги** Алгоритм дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Павлов В. Г.				
Реценз.		Корнієнко Б. Я.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11
Н. Контр.		Нікольський С. С.				
Затвердив		Новотарський М. А.				

ДОДАТОК Г

Платформа для навчання навичкам екстреної медичної допомоги

Лістинг програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 151

Київ 2025 р

schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

enum Role {
  USER
  MEDIC
  ROOT
}

enum LessonType {
  TEXT
  VIDEO
  DOCUMENT
  TEST
}

enum CourseStatus {
  IN_PROGRESS
  COMPLETED
}

model User {
  id              Int                  @id @default(autoincrement())
  email           String               @unique
  password        String
  name            String
  role            Role                 @default(USER)
  createdAt       DateTime             @default(now())
  courses         Course[]             @relation("AuthoredCourses")
  courseProgress    UserCourseProgress[]
  completedLessons  CompletedLesson[]
  simulations     Simulation[]         @relation("AuthoredSimulations")
}

model Course {
  id            Int                  @id @default(autoincrement())
  title         String
  slug          String               @unique
  description   String?
  imageUrl      String?
  author        User?                @relation("AuthoredCourses", fields: [authorId], references:
      [id], onDelete: SetNull)
  authorId      Int?
  keywords      String[]
  createdAt     DateTime             @default(now())
  sections      Section[]
  totalLessons  Int                  @default(0)
  userProgress  UserCourseProgress[]
  completedLessons CompletedLesson[]
  version       Int                  @default(1)
  editedAt      DateTime?
}

model Section {
  id       Int      @id @default(autoincrement())
  title    String
  order    Int
  course   Course   @relation(fields: [courseId], references: [id])
  courseId Int
  lessons  Lesson[]
}

model Lesson {
  id           Int              @id @default(autoincrement())
  title        String
  type         LessonType
  content      String?
  videoUrl     String?
  documentUrl  String?
  test         Json?
  order        Int
  section      Section          @relation(fields: [sectionId], references: [id])
  sectionId    Int
  completedBy  CompletedLesson[]
}

model UserCourseProgress {
  id            Int         @id @default(autoincrement())
  user          User        @relation(fields: [userId], references: [id], onDelete: Cascade)
```

```
  userId        Int
  course        Course       @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId      Int
  status        CourseStatus @default(IN_PROGRESS)
  startedAt     DateTime     @default(now())
  completedAt   DateTime?
  lastAccessAt  DateTime     @default(now())
  completedLessonsCount Int  @default(0)

  @@unique([userId, courseId])
}

model CompletedLesson {
  id          Int       @id @default(autoincrement())
  user        User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId      Int
  lesson      Lesson    @relation(fields: [lessonId], references: [id], onDelete: Cascade)
  lessonId    Int
  course      Course    @relation(fields: [courseId], references: [id], onDelete: Cascade)
  courseId    Int
  completedAt DateTime  @default(now())
  timeSpent   Int       @default(0)

  @@unique([userId, lessonId])
  @@index([courseId, userId])
}

model Simulation {
  id          Int         @id @default(autoincrement())
  title       String
  slug        String      @unique
  description String?
  imageUrl    String?
  createdAt   DateTime    @default(now())
  editedAt    DateTime?
  version     Int         @default(1)
  author      User?        @relation("AuthoredSimulations", fields: [authorId], references: [id],
      onDelete: SetNull)
  authorId    Int?
  steps       Step[]
  keywords    String[]
}

model Step {
  id              Int        @id @default(autoincrement())
  title           String
  content         String
  videoUrl        String
  videoPreviewUrl String?
  order           Int
  simulation      Simulation @relation(fields: [simulationId], references: [id], onDelete: Cascade)
  simulationId    Int
}

model PendingCode {
  id        String   @id @default(uuid())
  email     String   @unique
  name      String
  password  String
  code      String
  expiresAt DateTime
  createdAt DateTime @default(now())

  @@index([email])
}
```

prisma\seed.ts

```
import { prisma } from '@/utils/prisma';
import bcrypt from 'bcryptjs';

async function main() {
  const root = await prisma.user.findFirst({ where: { role: 'ROOT' } });
  if (!root) {
    const password = process.env.ROOT_PASSWORD;
    if (!password) {
      throw new Error('ROOT_PASSWORD env variable is not set!');
    }
    const hash = await bcrypt.hash(password, 10);
    await prisma.user.create({
      data: {
        email: 'medhelpedu@gmail.com',
        password: hash,
        name: 'Ватажко Михайло',
        role: 'ROOT'
      }
    });
    console.log('Root user created!');
  } else {
    console.log('Root user already exists.');
  }
}

main()
  .catch(e => { console.error(e); process.exit(1); })
  .finally(() => prisma.$disconnect());
```

api\courses\[id]\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { DeleteObjectCommand } from '@aws-sdk/client-s3';
import { messages, Lang } from '@/utils/messages';
import { s3 } from '@/utils/s3';
import { slugify } from '@/utils/slugify';

const B2_BUCKET = process.env.B2_BUCKET!;

export async function DELETE(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const id = Number(params.id);
    if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });

    if (user.role !== 'ROOT') {
      const courseToDelete = await prisma.course.findUnique({
        where: { id },
        select: { authorId: true },
      });
      if (!courseToDelete) {
        return NextResponse.json({ error: t.courseNotFound }, { status: 404 });
      }
      if (courseToDelete.authorId !== user.id) {
        return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
      }
    }

    const course = await prisma.course.findUnique({
      where: { id },
      include: { sections: { include: { lessons: true } } },
    });
    if (!course) return NextResponse.json({ error: t.courseNotFound }, { status: 404 });

    const fileUrls: string[] = [];
    if (course.imageUrl) fileUrls.push(course.imageUrl);
    for (const section of course.sections) {
      for (const lesson of section.lessons) {
        if (lesson.videoUrl) fileUrls.push(lesson.videoUrl);
        if (lesson.documentUrl) fileUrls.push(lesson.documentUrl);
      }
    }

    for (const url of fileUrls) {
      try {
        const urlObj = new URL(url);
        let key = urlObj.pathname;
        if (key.startsWith('/')) key = key.slice(1);
        if (key.startsWith(`${B2_BUCKET}/`)) key = key.slice(B2_BUCKET.length + 1);
        if (key) {
          await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: key }));
        }
      } catch (error: any) {
        console.error('Error deleting file from B2:', url, error);
      }
    }

    const sectionIds = course.sections.map((s: any) => s.id);
    await prisma.userCourseProgress.deleteMany({ where: { courseId: id } });
    await prisma.completedLesson.deleteMany({ where: { courseId: id } });

    if (sectionIds.length > 0) {
      await prisma.lesson.deleteMany({ where: { sectionId: { in: sectionIds } } });
    }

    await prisma.section.deleteMany({ where: { courseId: id } });
    await prisma.course.delete({ where: { id } });
    return NextResponse.json({ ok: true, message: t.courseDeleted });
  } catch (error: any) {
    console.error('Error deleting course:', error);
    return NextResponse.json({ error: t.courseDeletedError, details: error?.message }, { status: 500
      });
  }
}

export async function PUT(req: NextRequest, { params }: { params: { id: string } }) {
  const data = await req.json();
  const lang = data.lang || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
```

```
if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

try {
  requireRole(user, ['MEDIC', 'ROOT']);
} catch {
  return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
}

const id = Number(params.id);
if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });
const slug = data.title ? slugify(data.title) : undefined;

if (slug) {
  const existing = await prisma.course.findFirst({ where: { slug, NOT: { id } } });
  if (existing) return NextResponse.json({ error: t.courseAlreadyExists }, { status: 409 });
}

const isOnlySlugCheck = req.headers.get('X-Only-Check-Slug') === 'true';
if (isOnlySlugCheck) {
  return NextResponse.json({ ok: true, message: 'Slug is available' });
}

const current = await prisma.course.findUnique({
  where: { id },
  select: { version: true, imageUrl: true }
});

if (!current) {
  return NextResponse.json({ error: t.courseNotFound }, { status: 404 });
}

if (user.role !== 'ROOT') {
  const courseToUpdate = await prisma.course.findUnique({
    where: { id },
    select: { authorId: true },
  });
  if (!courseToUpdate) {
    return NextResponse.json({ error: t.courseNotFound }, { status: 404 });
  }
  if (courseToUpdate.authorId !== user.id) {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }
}

if (current.imageUrl && current.imageUrl !== data.imageUrl) {
  try {
    const urlObj = new URL(current.imageUrl);
    let imageKey = urlObj.pathname;
    if (imageKey.startsWith('/')) imageKey = imageKey.slice(1);
    if (imageKey.startsWith(`${B2_BUCKET}/`)) imageKey = imageKey.slice(B2_BUCKET.length + 1);
    if (imageKey) {
      await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: imageKey }));
      console.log('Old course image deleted from cloud:', imageKey);
    }
  } catch (error: any) {
    console.error('Error deleting old course image from cloud:', error);
  }
}

const totalLessons = (data.sections || []).reduce(
  (sum: number, section: any) => sum + (section.lessons?.length || 0),
  0
);

let course;
try {
  course = await prisma.course.update({
    where: { id },
    data: {
      title: data.title,
      description: data.description,
      imageUrl: data.imageUrl,
      keywords: data.keywords,
      ...(slug ? { slug } : {}),
      editedAt: new Date(),
      version: (current?.version || 1) + 1,
      totalLessons
    }
  });
} catch (error: any) {
  console.error('Error updating course:', error);
  return NextResponse.json({ error: t.courseUpdatedError, details: error?.message }, { status: 500
    });
}

try {
  const sections = await prisma.section.findMany({ where: { courseId: id }, select: { id: true }
    });
  const sectionIds = sections.map((s: any) => s.id);
  const existingLessons = await prisma.lesson.findMany({
    where: { sectionId: { in: sectionIds } },
    select: { id: true, videoUrl: true, documentUrl: true, type: true }
  });

  const oldFileUrls: string[] = [];
  existingLessons.forEach((lesson: any) => {
    if (lesson.videoUrl) oldFileUrls.push(lesson.videoUrl);
```

```
      if (lesson.documentUrl) oldFileUrls.push(lesson.documentUrl);
    });

    await prisma.userCourseProgress.deleteMany({ where: { courseId: id } });
    await prisma.completedLesson.deleteMany({ where: { courseId: id } });

    if (data.sections && data.sections.length > 0) {
      const newFileUrls: string[] = [];
      for (const section of data.sections) {
        if (section.lessons) {
          for (const lesson of section.lessons) {
            if (lesson.videoUrl) newFileUrls.push(lesson.videoUrl);
            if (lesson.documentUrl) newFileUrls.push(lesson.documentUrl);
          }
        }
      }

      for (const url of oldFileUrls) {
        if (!newFileUrls.includes(url)) {
          try {
            const urlObj = new URL(url);
            let key = urlObj.pathname;
            if (key.startsWith('/')) key = key.slice(1);
            if (key.startsWith(`${B2_BUCKET}/`)) key = key.slice(B2_BUCKET.length + 1);
            if (key) {
              await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: key }));
              console.log('Deleted unused lesson file:', key);
            }
          } catch (error: any) {
            console.error('Error deleting unused file:', url, error);
          }
        }
      }

      if (sectionIds.length > 0) {
        await prisma.lesson.deleteMany({ where: { sectionId: { in: sectionIds } } });
      }
      await prisma.section.deleteMany({ where: { courseId: id } });

      for (const [sectionIndex, section] of data.sections.entries()) {
        const newSection = await prisma.section.create({
          data: {
            title: section.title,
            order: sectionIndex,
            courseId: id
          }
        });

        if (section.lessons && section.lessons.length > 0) {
          for (const [lessonIndex, lesson] of section.lessons.entries()) {
            await prisma.lesson.create({
              data: {
                title: lesson.title,
                type: lesson.type || 'TEXT',
                content: lesson.content || '',
                test: lesson.test || null,
                videoUrl: lesson.videoUrl || null,
                documentUrl: lesson.documentUrl || null,
                order: lessonIndex,
                sectionId: newSection.id
              }
            });
          }
        }
      }
    }
  } catch (error: any) {
    console.error('Error updating course sections/lessons:', error);
    return NextResponse.json({ error: t.courseUpdatedError, details: error?.message }, { status: 500
      });
  }
  return NextResponse.json({ course, message: t.courseUpdated });
}

export async function GET(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const id = Number(params.id);
  if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 400 });

  const user = getUserFromRequest(req);

  try {
    const course = await prisma.course.findUnique({
      where: { id },
      include: {
        author: {
          select: {
            id: true,
            name: true
          }
        },
        sections: {
          orderBy: { order: 'asc' },
          include: {
```

```
          lessons: {
            orderBy: { order: 'asc' }
          }
        }
      }
    }
  });

  if (!course) {
    return NextResponse.json({ error: t.courseNotFound }, { status: 404 });
  }

  if (user) {
    const userProgress = await prisma.userCourseProgress.findUnique({
      where: {
        userId_courseId: { userId: user.id, courseId: id }
      }
    });

    const completedLessons = await prisma.completedLesson.findMany({
      where: {
        userId: user.id,
        courseId: id
      },
      select: {
        lessonId: true
      }
    });

    const completedLessonIds = completedLessons.map((cl: any) => cl.lessonId);

    const sectionsWithCompletedInfo = course.sections.map((section: any) => ({
      ...section,
      lessons: section.lessons.map((lesson: any) => ({
        ...lesson,
        completed: completedLessonIds.includes(lesson.id)
      }))
    }));

    return NextResponse.json({
      ...course,
      sections: sectionsWithCompletedInfo,
      userProgress: userProgress || null
    });
  }

  return NextResponse.json(course);
 } catch (error: any) {
  console.error('Error loading course:', error);
  return NextResponse.json({ error: t.coursesLoadError }, { status: 500 });
 }
}
```

api\courses\available\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    const availableCourses = await prisma.course.findMany({
      where: {
        NOT: {
          userProgress: {
            some: {
              userId: user.id
            }
          }
        }
      },
      include: {
        sections: {
          include: {
            lessons: true
          }
        },
        author: true
      },
      orderBy: { createdAt: 'desc' }
    });

    return NextResponse.json(availableCourses);
  } catch (error) {
    return NextResponse.json({ error: t.availableCoursesLoadError }, { status: 500 });
  }
}
```

api\courses\completed\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    const completedRecords = await prisma.userCourseProgress.findMany({
      where: {
        userId: user.id,
        status: 'COMPLETED'
      },
      include: {
        course: {
          include: {
            sections: {
              include: {
                lessons: true
              }
            },
            author: true
          }
        }
      },
      orderBy: {
        completedAt: 'desc'
      }
    });

    if (completedRecords.length === 0) {
      return NextResponse.json([]);
    }

    const completedCourses = completedRecords.map((record: any) => ({
      ...record.course,
      progress: 100,
      completedAt: record.completedAt,
      completedLessonsCount: record.completedLessonsCount,
      totalLessons: record.course.totalLessons || 0
    }));

    return NextResponse.json(completedCourses);
  } catch (error) {
    console.error('Error loading completed courses:', error);
    return NextResponse.json({ error: t.completedCoursesLoadError }, { status: 500 });
  }
}
```

api\courses\progress\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = await getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    const inProgressRecords = await prisma.userCourseProgress.findMany({
      where: {
        userId: user.id,
        status: 'IN_PROGRESS'
      },
      include: {
        course: {
          include: {
            sections: {
              include: {
                lessons: true
              }
            },
            author: true
          }
        }
      },
      orderBy: {
        lastAccessAt: 'desc'
      }
    });

    if (inProgressRecords.length === 0) {
      return NextResponse.json([]);
```

```
    }

    const inProgressCourses = inProgressRecords.map((record: any) => {
      const totalLessons = record.course.totalLessons || 0;
      const progress = totalLessons > 0
        ? Math.round((record.completedLessonsCount / totalLessons) * 100)
        : 0;

      return {
        ...record.course,
        progress,
        lastAccessAt: record.lastAccessAt,
        completedLessonsCount: record.completedLessonsCount,
        totalLessons
      };
    });

    return NextResponse.json(inProgressCourses);
  } catch (error) {
    return NextResponse.json({ error: t.inProgressCoursesLoadError }, { status: 500 });
  }
}
```

api\courses\control\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';
import { slugify } from '@/utils/slugify';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    if (user.role === 'ROOT') {
      const courses = await prisma.course.findMany({
        include: {
          sections: { include: { lessons: true } },
          author: { select: { id: true, name: true, email: true } }
        },
        orderBy: { createdAt: 'desc' },
      });
      return NextResponse.json(courses);
    }

    const courses = await prisma.course.findMany({
      where: { authorId: user.id },
      include: {
        sections: { include: { lessons: true } },
        author: { select: { id: true, name: true, email: true } }
      },
      orderBy: { createdAt: 'desc' },
    });

    return NextResponse.json(courses);
  } catch (error) {
    console.error('Error loading courses:', error);
    return NextResponse.json({ error: t.coursesLoadError }, { status: 500 });
  }
}

export async function POST(req: NextRequest) {
  const data = await req.json();
  const lang = data.lang || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const slug = slugify(data.title);
    const existing = await prisma.course.findFirst({ where: { slug } });
    if (existing) {
      return NextResponse.json({ error: t.courseAlreadyExists }, { status: 409 });
    }

    const isOnlySlugCheck = req.headers.get('X-Only-Check-Slug') === 'true';
    if (isOnlySlugCheck) {
```

```
      return NextResponse.json({ ok: true, message: 'Slug is available' });
    }

    const totalLessons = (data.sections || []).reduce(
      (sum: number, section: any) => sum + (section.lessons?.length || 0),
      0
    );

    const course = await prisma.course.create({
      data: {
        title: data.title,
        slug,
        description: data.description || '',
        imageUrl: data.imageUrl || null,
        authorId: user.id,
        keywords: data.keywords || [],
        version: 1,
        totalLessons,
        sections: {
          create: (data.sections || []).map((section: any, sIdx: number) => ({
            title: section.title,
            order: sIdx,
            lessons: {
              create: (section.lessons || []).map((lesson: any, lIdx: number) => ({
                title: lesson.title,
                type: lesson.type ? lesson.type.toUpperCase() : 'TEXT',
                content: lesson.content || '',
                test: lesson.test || null,
                videoUrl: lesson.videoUrl || null,
                documentUrl: lesson.documentUrl || null,
                order: lIdx,
              }))
            }
          }))
        },
      },
      include: {
        sections: {
          include: { lessons: true },
          orderBy: { order: 'asc' }
        }
      },
    });

    return NextResponse.json({ course, message: t.courseCreated });
  } catch (error) {
    console.error('Error creating course:', error);
    return NextResponse.json({ error: t.courseCreatedError }, { status: 500 });
  }
}
```

api\courses\start\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function POST(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  const { courseId } = await req.json();
  if (!courseId) return NextResponse.json({ error: t.courseIdRequired }, { status: 400 });

  try {
    const course = await prisma.course.findUnique({
      where: { id: courseId },
      include: {
        sections: {
          include: {
            lessons: true
          }
        }
      }
    });

    if (!course) {
      return NextResponse.json({ error: t.courseNotFound }, { status: 404 });
    }

    const existingProgress = await prisma.userCourseProgress.findUnique({
      where: {
        userId_courseId: { userId: user.id, courseId }
      }
    });

    if (existingProgress) {
      await prisma.userCourseProgress.update({
        where: { id: existingProgress.id },
        data: {
          lastAccessAt: new Date(),
          status: existingProgress.status === 'COMPLETED' ? 'IN_PROGRESS' : existingProgress.status
```

```
      }
    });

    return NextResponse.json({
      ok: true,
      message: t.courseProgressUpdated,
      status: existingProgress.status === 'COMPLETED' ? 'IN_PROGRESS' : existingProgress.status
    });
  }

  let totalLessons = 0;
  course.sections.forEach((section: any) => {
    totalLessons += section.lessons.length;
  });

  await prisma.userCourseProgress.create({
    data: {
      userId: user.id,
      courseId,
      status: 'IN_PROGRESS',
      startedAt: new Date(),
      lastAccessAt: new Date(),
      completedLessonsCount: 0
    }
  });

  if (course.totalLessons !== totalLessons) {
    await prisma.course.update({
      where: { id: courseId },
      data: { totalLessons }
    });
  }

  return NextResponse.json({
    ok: true,
    message: t.courseStarted,
    progress: 0,
    status: 'IN_PROGRESS'
  });

 } catch (error) {
  console.error('Error starting course:', error);
  return NextResponse.json({ error: t.courseProgressError }, { status: 500 });
 }
}
```

api\lessons\complete\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { Prisma } from '@prisma/client';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function POST(req: NextRequest) {
 const lang = req.nextUrl.searchParams.get('lang') || 'ua';
 const key: Lang = lang === 'en' ? 'en' : 'ua';
 const t = messages[key];

 const user = getUserFromRequest(req);
 if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

 const { lessonId, timeSpent } = await req.json();
 if (!lessonId) return NextResponse.json({ error: t.lessonIdRequired }, { status: 400 });

 try {
  const lesson = await prisma.lesson.findUnique({
    where: { id: lessonId },
    include: {
      section: {
        select: {
          courseId: true
        }
      }
    }
  });

  if (!lesson) {
    return NextResponse.json({ error: t.lessonNotFound }, { status: 404 });
  }

  const courseId = lesson.section.courseId;
  const existingCompletion = await prisma.completedLesson.findUnique({
    where: { userId_lessonId: { userId: user.id, lessonId } }
  });

  if (existingCompletion) {
    return NextResponse.json({
      ok: true,
      message: t.lessonCompleted,
      alreadyCompleted: true
    });
  }

  const result = await prisma.$transaction(async (tx: Prisma.TransactionClient) => {
    await tx.completedLesson.create({
      data: {
```

```
          userId: user.id,
          lessonId,
          courseId,
          timeSpent: timeSpent || 0,
          completedAt: new Date()
        }
      });

      let courseProgress = await tx.userCourseProgress.findUnique({
        where: {
          userId_courseId: { userId: user.id, courseId }
        }
      });

      if (!courseProgress) {
        courseProgress = await tx.userCourseProgress.create({
          data: {
            userId: user.id,
            courseId,
            status: 'IN_PROGRESS',
            startedAt: new Date(),
            lastAccessAt: new Date(),
            completedLessonsCount: 1
          }
        });
      } else {
        courseProgress = await tx.userCourseProgress.update({
          where: { id: courseProgress.id },
          data: {
            completedLessonsCount: {
              increment: 1
            },
            lastAccessAt: new Date()
          }
        });
      }

      const course = await tx.course.findUnique({
        where: { id: courseId },
        select: { totalLessons: true }
      });

      let totalLessons = course?.totalLessons || 0;
      if (totalLessons === 0) {
        const lessonsCount = await tx.lesson.count({
          where: {
            section: {
              courseId
            }
          }
        });

        totalLessons = lessonsCount;

        await tx.course.update({
          where: { id: courseId },
          data: { totalLessons: lessonsCount }
        });
      }

      const isCompleted = courseProgress.completedLessonsCount === totalLessons && totalLessons > 0;
      if (isCompleted) {
        await tx.userCourseProgress.update({
          where: { id: courseProgress.id },
          data: {
            status: 'COMPLETED',
            completedAt: new Date()
          }
        });
      }

      const progress = totalLessons > 0
        ? Math.round((courseProgress.completedLessonsCount / totalLessons) * 100)
        : 0;

      return {
        completedLessonsCount: courseProgress.completedLessonsCount,
        totalLessons,
        progress,
        isCompleted
      };
    });

    return NextResponse.json({
      ok: true,
      message: result.isCompleted ? t.courseCompleted : t.lessonCompleted,
      progress: result.progress,
      completedLessonsCount: result.completedLessonsCount,
      totalLessons: result.totalLessons,
      isCompleted: result.isCompleted,
    });
  } catch (error) {
    console.error('Error completing lesson:', error);
    return NextResponse.json({ error: t.courseProgressError }, { status: 500 });
  }
}
```

api\login\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { messages, Lang } from '@/utils/messages';

const JWT_SECRET = process.env.JWT_SECRET;

export async function POST(req: NextRequest) {
  const { email, password, lang = 'ua' } = await req.json();
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  if (!email || !password) {
    return NextResponse.json({ error: t.required }, { status: 400 });
  }
  if (!/^[\w.+-]+@gmail\.com$/.test(email)) {
    return NextResponse.json({ error: t.onlyGmail }, { status: 400 });
  }

  const user = await prisma.user.findUnique({ where: { email } });
  if (!user) {
    return NextResponse.json({ error: t.notFound }, { status: 404 });
  }
  const valid = await bcrypt.compare(password, user.password);
  if (!valid) {
    return NextResponse.json({ error: t.wrongPassword }, { status: 401 });
  }
  const token = jwt.sign({ id: user.id, email: user.email, name: user.name, role: user.role },
      JWT_SECRET as string, { expiresIn: '7d' });
  return NextResponse.json({ token, user: { id: user.id, email: user.email, name: user.name, role:
      user.role } });
}
```

api\password\forgot\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import nodemailer from 'nodemailer';
import { prisma } from '@/utils/prisma';
import { messages, Lang } from '@/utils/messages';

function generateCode() {
  return Math.floor(100000 + Math.random() * 900000).toString();
}

export async function POST(req: NextRequest) {
  const { email, lang = 'ua' } = await req.json();
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const code = generateCode();
  const expiresAt = new Date(Date.now() + 15 * 60 * 1000);

  await prisma.pendingCode.deleteMany({ where: { expiresAt: { lt: new Date() } } });

  await prisma.pendingCode.upsert({
    where: { email },
    update: { code, expiresAt },
    create: { email, code, expiresAt, name: '', password: '' },
  });

  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.SMTP_USER,
      pass: process.env.SMTP_PASS,
    },
  });

  const subject = t.passwordResetSubject;
  const text = t.passwordResetText(code);

  try {
    await transporter.sendMail({
      from: process.env.SMTP_USER,
      to: email,
      subject,
      text,
    });
    return NextResponse.json({ ok: true, message: t.instructionsSent });
  } catch (e: any) {
    return NextResponse.json({ error: t.emailSend }, { status: 500 });
  }
}
```

api\password\reset\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import bcrypt from 'bcryptjs';
import { messages, Lang } from '@/utils/messages';

export async function POST(req: NextRequest) {
  const { email, code, newPassword, lang = 'ua' } = await req.json();
```

```
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  await prisma.pendingCode.deleteMany({ where: { expiresAt: { lt: new Date() } } });

  const pending = await prisma.pendingCode.findUnique({ where: { email } });
  if (!pending || pending.code !== code || pending.expiresAt < new Date()) {
    return NextResponse.json({ error: t.invalidOrExpiredCode }, { status: 400 });
  }

  const hashed = await bcrypt.hash(newPassword, 10);
  await prisma.user.update({
    where: { email },
    data: { password: hashed },
  });
  await prisma.pendingCode.delete({ where: { email } });
  return NextResponse.json({ message: t.passwordChanged });
}
```

api\register\confirm\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { messages, Lang } from '@/utils/messages';

export async function POST(req: NextRequest) {
  const { email, code, lang = 'ua' } = await req.json();
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  await prisma.pendingCode.deleteMany({ where: { expiresAt: { lt: new Date() } } });

  const pending = await prisma.pendingCode.findUnique({ where: { email } });
  if (!pending || pending.code !== code || pending.expiresAt < new Date()) {
    return NextResponse.json({ error: t.invalidOrExpiredCode }, { status: 400 });
  }

  await prisma.user.create({ data: { email, password: pending.password, name: pending.name } });
  await prisma.pendingCode.delete({ where: { email } });
  return NextResponse.json({ ok: true });
}
```

api\register\request\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import nodemailer from 'nodemailer';
import { prisma } from '@/utils/prisma';
import { messages, Lang } from '@/utils/messages';
import bcrypt from 'bcryptjs';

function generateCode() {
  return Math.floor(100000 + Math.random() * 900000).toString();
}

export async function POST(req: NextRequest) {
  const { email, name, password, lang = 'ua' } = await req.json();
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  if (!email || !name || !password) {
    return NextResponse.json({ error: t.required }, { status: 400 });
  }
  if (!/^[\w.+-]+@gmail\.com$/.test(email)) {
    return NextResponse.json({ error: t.onlyGmail }, { status: 400 });
  }

  const existingUser = await prisma.user.findUnique({ where: { email } });
  if (existingUser) {
    return NextResponse.json({ error: t.exists }, { status: 400 });
  }

  const code = generateCode();
  const hashedPassword = await bcrypt.hash(password, 10);
  const expiresAt = new Date(Date.now() + 15 * 60 * 1000);

  await prisma.pendingCode.upsert({
    where: { email },
    update: { code, expiresAt, name, password: hashedPassword },
    create: { email, code, expiresAt, name, password: hashedPassword },
  });

  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.SMTP_USER,
      pass: process.env.SMTP_PASS,
    },
  });

  const subject = t.registrationSubject(name);
  const text = t.registrationText(code);

  try {
    await transporter.sendMail({
      from: process.env.SMTP_USER,
      to: email,
      subject,
```

```
      text,
    });
    return NextResponse.json({ ok: true });
  } catch (e: any) {
    if (e && e.message && (e.message.includes('Invalid recipient') || e.message.includes('No
      recipients defined') || e.message.includes('Invalid email'))) {
      return NextResponse.json({ error: t.invalidEmail }, { status: 400 });
    }
    return NextResponse.json({ error: t.emailSend }, { status: 500 });
  }
}
```

api\simulations\[id]\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { DeleteObjectCommand } from '@aws-sdk/client-s3';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';
import { s3 } from '@/utils/s3';
import { slugify } from '@/utils/slugify';

const B2_BUCKET = process.env.B2_BUCKET!;

export async function DELETE(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const id = Number(params.id);
    if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });

    if (user.role !== 'ROOT') {
      const simulationToDelete = await prisma.simulation.findUnique({
        where: { id },
        select: { authorId: true },
      });
      if (!simulationToDelete) {
        return NextResponse.json({ error: t.simulationNotFound }, { status: 404 });
      }
      if (simulationToDelete.authorId !== user.id) {
        return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
      }
    }

    const simulation = await prisma.simulation.findUnique({
      where: { id },
      include: { steps: true }
    });
    if (!simulation) return NextResponse.json({ error: t.simulationNotFound }, { status: 404 });

    const fileUrls: string[] = [];
    if (simulation.imageUrl) fileUrls.push(simulation.imageUrl);
    for (const step of simulation.steps) {
      if (step.videoUrl) fileUrls.push(step.videoUrl);
    }

    for (const url of fileUrls) {
      try {
        const urlObj = new URL(url);
        let key = urlObj.pathname;
        if (key.startsWith('/')) key = key.slice(1);
        if (key.startsWith(`${B2_BUCKET}/`)) key = key.slice(B2_BUCKET.length + 1);
        if (key) {
          await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: key }));
        }
      } catch (error: any) {
        console.error('Error deleting file from B2:', url, error);
      }
    }

    await prisma.step.deleteMany({ where: { simulationId: id }});
    await prisma.simulation.delete({ where: { id } });
    return NextResponse.json({ ok: true, message: t.simulationDeleted });
  } catch (error: any) {
    console.error('Error deleting simulation:', error);
    return NextResponse.json({ error: t.simulationDeletedError, details: error?.message }, { status:
      500 });
  }
}

export async function PUT(req: NextRequest, { params }: { params: { id: string } }) {
  const data = await req.json();
  const lang = data.lang || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];
```

```
const user = getUserFromRequest(req);
if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

try {
  requireRole(user, ['MEDIC', 'ROOT']);
} catch {
  return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
}

const id = Number(params.id);
if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });
const slug = data.title ? slugify(data.title) : undefined;

if (slug) {
  const existing = await prisma.simulation.findFirst({ where: { slug, NOT: { id } }});
  if (existing) return NextResponse.json({ error: t.simulationAlreadyExists }, { status: 409 });
}

const isOnlySlugCheck = req.headers.get('X-Only-Check-Slug') === 'true';
if (isOnlySlugCheck) {
  return NextResponse.json({ ok: true, message: 'Slug is available' });
}

const current = await prisma.simulation.findUnique({
  where: { id },
  select: { version: true, imageUrl: true }
});

if (!current) {
  return NextResponse.json({ error: t.simulationNotFound }, { status: 404 });
}

if (user.role !== 'ROOT') {
  const simulationToUpdate = await prisma.simulation.findUnique({
    where: { id },
    select: { authorId: true },
  });
  if (!simulationToUpdate) {
    return NextResponse.json({ error: t.simulationNotFound }, { status: 404 });
  }
  if (simulationToUpdate.authorId !== user.id) {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }
}

if (current.imageUrl && current.imageUrl !== data.imageUrl) {
  try {
    const urlObj = new URL(current.imageUrl);
    let imageKey = urlObj.pathname;
    if (imageKey.startsWith('/')) imageKey = imageKey.slice(1);
    if (imageKey.startsWith(`${B2_BUCKET}/`)) imageKey = imageKey.slice(B2_BUCKET.length + 1);
    if (imageKey) {
      await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: imageKey }));
      console.log('Old simulation image deleted from cloud:', imageKey);
    }
  } catch (error: any) {
    console.error('Error deleting old simulation image from cloud:', error);
  }
}

let simulation;
try {
  simulation = await prisma.simulation.update({
    where: { id },
    data: {
      title: data.title,
      description: data.description,
      imageUrl: data.imageUrl,
      keywords: data.keywords,
      ...(slug ? { slug } : {}),
      editedAt: new Date(),
      version: (current?.version || 1) + 1,
    }
  });
} catch (error: any) {
  console.error('Error updating simulation:', error);
  return NextResponse.json({ error: t.simulationUpdatedError, details: error?.message }, { status:
    500 });
}

try {
  const existingSteps = await prisma.step.findMany({ where: { simulationId: id }, select: { id:
    true, videoUrl: true } });

  const oldFileUrls: string[] = [];
  existingSteps.forEach((step: any) => {
    if (step.videoUrl) oldFileUrls.push(step.videoUrl);
  });

  if (data.steps && data.steps.length > 0) {
    const newFileUrls: string[] = [];
    for (const step of data.steps) {
      if (step.videoUrl) newFileUrls.push(step.videoUrl);
    }

    for (const url of oldFileUrls) {
      if (!newFileUrls.includes(url)) {
```

```
          try {
            const urlObj = new URL(url);
            let key = urlObj.pathname;
            if (key.startsWith('/')) key = key.slice(1);
            if (key.startsWith(`${B2_BUCKET}/`)) key = key.slice(B2_BUCKET.length + 1);
            if (key) {
              await s3.send(new DeleteObjectCommand({ Bucket: B2_BUCKET, Key: key }));
              console.log('Deleted unused step file:', key);
            }
          } catch (error: any) {
            console.error('Error deleting unused file:', url, error);
          }
        }
      }

      await prisma.step.deleteMany({ where: { simulationId: id } });

      for (const [stepIndex, step] of data.steps.entries()) {
        await prisma.step.create({
          data: {
            title: step.title,
            content: step.content,
            videoUrl: step.videoUrl,
            videoPreviewUrl: step.videoPreviewUrl,
            order: stepIndex,
            simulationId: id
          }
        });
      }
    }
  } catch (error: any) {
    console.error('Error updating simulation:', error);
    return NextResponse.json({ error: t.simulationUpdatedError, details: error?.message }, { status:
      500 });
  }
  return NextResponse.json({ simulation, message: t.simulationUpdated });
}

export async function GET(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const id = Number(params.id);
  if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });

  try {
    const simulation = await prisma.simulation.findUnique({
      where: { id},
      include: {
        author: {
          select: {
            id: true,
            name: true
          }
        },
        steps: {
          orderBy: { order: 'asc' }
        }
      }
    });

    if (!simulation) {
      return NextResponse.json({ error: t.simulationNotFound }, { status: 404 });
    }
    return NextResponse.json(simulation);
  } catch (error) {
    console.error('Error loading simulation:', error);
    return NextResponse.json({ error: t.simulationsLoadError }, { status: 500 });
  }
}
```

api\simulations\available\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  try {
    const allSimulations = await prisma.simulation.findMany({
      include: {
        author: true,
        steps: true
      },
      orderBy: { createdAt: 'desc' }
    });
    return NextResponse.json(allSimulations);
  } catch (error) {
    return NextResponse.json({ error: t.simulationsLoadError }, { status: 500 });
  }
}
```

api\simulations\control\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';
import { slugify } from '@/utils/slugify';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  const allSimulations = req.nextUrl.searchParams.get('all') === '1';

  try {
    if (user.role === 'ROOT' && allSimulations) {
      const simulations = await prisma.simulation.findMany({
        include: {
          steps: true,
          author: { select: { id: true, name: true, email: true } }
        },
        orderBy: { createdAt: 'desc' },
      });
      return NextResponse.json(simulations);
    }

    const simulations = await prisma.simulation.findMany({
      where: { authorId: user.id },
      include: {
        author: { select: { id: true, name: true, email: true } },
        steps: true
      },
      orderBy: { createdAt: 'desc' }
    });

    return NextResponse.json(simulations);
  } catch (error) {
    console.error('Error loading simulations:', error);
    return NextResponse.json({ error: t.simulationsLoadError }, { status: 500 });
  }
}

export async function POST(request: NextRequest) {
  const data = await request.json();
  const lang = data.lang || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(request);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['MEDIC', 'ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const slug = slugify(data.title);
    const existing = await prisma.simulation.findFirst({ where: { slug } });
    if (existing) {
      return NextResponse.json({ error: t.simulationAlreadyExists }, { status: 409 });
    }

    const isOnlySlugCheck = request.headers.get('X-Only-Check-Slug') === 'true';
    if (isOnlySlugCheck) {
      return NextResponse.json({ ok: true, message: 'Slug is available' });
    }

    const simulation = await prisma.simulation.create({
      data: {
        title: data.title,
        slug,
        description: data.description || '',
        imageUrl: data.imageUrl || null,
        authorId: user.id,
        keywords: data.keywords || [],
        version: 1,
        steps: {
          create: (data.steps || []).map((step: any, sIdx: number) => ({
            title: step.title,
            order: sIdx,
            content: step.content || '',
            videoUrl: step.videoUrl || null,
            videoPreviewUrl: step.videoPreviewUrl || null
          }))
        },
```

```
      },
      include: {
        steps: { orderBy: { order: 'asc' } }
      }
    });

    return NextResponse.json({ simulation, message: t.simulationCreated });
  } catch (error) {
    console.error('Error creating simulation:', error);
    return NextResponse.json({ error: t.simulationCreatedError }, { status: 500 });
  }
}
```

api\translate\route.ts

```
import { NextRequest } from 'next/server';

const CHUNK_SIZE = 500;

async function translateChunk(text: string, target: string) {
  const url =
      `https://translate.google.com/translate_a/single?client=gtx&sl=uk&tl=${target}&dt=t&q=${encode
     URIComponent(text)}` ;
  try {
    const response = await fetch(url);
    const data = await response.json();
    return data[0][0][0] || text;
  } catch {
    return text;
  }
}

export async function POST(req: NextRequest) {
  const { text, target } = await req.json();

  if (typeof text !== 'string' || !target) {
    return Response.json({ translatedText: text }, { status: 400 });
  }

  const safeTextTemp = text.replace(/.\n/g, ' @.');
  const safeText = safeTextTemp.replace(/\./g, ' &');

  let translated = '';
  if (safeText.length <= CHUNK_SIZE) {
    translated = await translateChunk(safeText, target);
  } else {
    const lines = safeText.split('@');
    for (const line of lines) {
      translated += await translateChunk(line, target);
      translated += '@';
    }
  }

  const resultTemp = translated.replace(/ @./g, '.\n');
  const result = resultTemp.replace(/ &/g, '.');
  return Response.json({ translatedText: result });
}
```

api\upload\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { PutObjectCommand } from '@aws-sdk/client-s3';
import { v4 as uuidv4 } from 'uuid';
import { messages } from '@/utils/messages';
import { s3 } from '@/utils/s3';

const B2_BUCKET = process.env.B2_BUCKET!;
const B2_ENDPOINT = process.env.B2_ENDPOINT!;

export async function OPTIONS() {
  return new NextResponse(null, {
    status: 204,
    headers: {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Methods': 'POST, OPTIONS',
      'Access-Control-Allow-Headers': 'Content-Type',
    },
  });
}

export async function POST(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') === 'en' ? 'en' : 'ua';
  const t = messages[lang];

  const formData = await req.formData();
  const file = formData.get('file') as File;
  if (!file) {
    return NextResponse.json({ error: t.fileRequired }, { status: 400, headers: { 'Access-Control-
      Allow-Origin': '*' } });
  }

  const fileType = String(formData.get('fileType') || 'other');
  const uniqueId = uuidv4().replace(/-/g, '').substring(0, 8);
  const fileName = file.name.replace(/[^\p{L}\d._-]+/gu, '_');

  let key = `${uniqueId}-${fileName}`;
  if (fileType === 'videos' || fileType === 'images' || fileType === 'documents') {
```

```
    key = `${fileType}/${uniqueId}-${fileName}`;
  }

  const arrayBuffer = await file.arrayBuffer();
  const buffer = Buffer.from(arrayBuffer);

  try {
    const putParams = {
      Bucket: B2_BUCKET,
      Key: key,
      Body: buffer,
      ContentType: file.type,
      ACL: 'public-read' as const,
    };
    await s3.send(new PutObjectCommand(putParams));
  } catch (e: any) {
    return NextResponse.json({ error: t.uploadError, details: e }, { status: 500, headers: {
      'Access-Control-Allow-Origin': '*' } });
  }
  const url = `https://${B2_ENDPOINT}/${B2_BUCKET}/${key}`;

  return NextResponse.json({ url }, { headers: { 'Access-Control-Allow-Origin': '*' } });
}
```

api\users\simple\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const simpleUsers = await prisma.user.findMany({
      where: { role: 'USER' },
      select: { id: true, email: true, name: true, role: true }
    });
    return NextResponse.json(simpleUsers);
  } catch (error) {
    return NextResponse.json({ error: t.simpleUsersLoadError }, { status: 500 });
  }
}
```

api\users\medics\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const medics = await prisma.user.findMany({
      where: { role: 'MEDIC' },
      select: { id: true, email: true, name: true, role: true }
    });
    return NextResponse.json(medics);
  } catch (error) {
    return NextResponse.json({ error: t.medicsLoadError }, { status: 500 });
  }
}
```

api\users\all\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function GET(req: NextRequest) {
```

```
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try {
    requireRole(user, ['ROOT']);
  } catch {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    const users = await prisma.user.findMany({
      where: { role: { not: 'ROOT' } },
      select: { id: true, email: true, name: true, role: true }
    });
    return NextResponse.json(users);
  } catch (error) {
    return NextResponse.json({ error: t.usersLoadError }, { status: 500 });
  }
}
```

api\users\[id]\route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/utils/prisma';
import { getUserFromRequest, requireRole } from '@/utils/auth';
import { messages, Lang } from '@/utils/messages';

export async function PATCH(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  try{
    requireRole(user, ['ROOT']);
  } catch (error) {
    return NextResponse.json({ error: t.notAuthorized }, { status: 403 });
  }

  const id = Number(params.id);
  if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });

  if (user.id === id) return NextResponse.json({ error: t.cannotChangeYourOwnRole }, { status: 400
     });

  try {
    const target = await prisma.user.findUnique({ where: { id } });
    if (!target || target.role === 'ROOT') {
      return NextResponse.json({ error: t.cannotChangeRoleForThisUser }, { status: 400 });
    }

    const body = await req.json();
    if (!['USER', 'MEDIC'].includes(body.role)) {
      return NextResponse.json({ error: t.invalidRole }, { status: 400 });
    }

    await prisma.user.update({ where: { id }, data: { role: body.role } });
    return NextResponse.json({ message: t.roleChanged });
  } catch (error) {
    return NextResponse.json({ error: t.errorChangingRole }, { status: 404 });
  }
}

export async function DELETE(req: NextRequest, { params }: { params: { id: string } }) {
  const lang = req.nextUrl.searchParams.get('lang') || 'ua';
  const key: Lang = lang === 'en' ? 'en' : 'ua';
  const t = messages[key];

  const user = getUserFromRequest(req);
  if (!user) return NextResponse.json({ error: t.notAuthorized }, { status: 401 });

  const id = Number(params.id);
  if (isNaN(id)) return NextResponse.json({ error: t.notFoundId }, { status: 404 });

  if (user.id !== id) {
    return NextResponse.json({ error: t.notEnoughRights }, { status: 403 });
  }

  try {
    await prisma.user.delete({ where: { id } });
    return NextResponse.json({ ok: true, message: t.accountDeleted });
  } catch (error) {
    return NextResponse.json({ error: t.errorDeletingAccount }, { status: 500 });
  }
}
```

about\page.tsx

```
import Navbar from '@/components/Navbar';
import Footer from '@/components/Footer';
import { TranslatedText } from '@/components/TranslatedText';
```

```
export default function AboutPage() {
  return (
    <>
      <Navbar />
      <main className="min-h-[60vh] bg-white">
        <div className="container-wrapper py-8">
          <h1 className="text-3xl font-bold section-spacing"><TranslatedText text="Про проєкт"
     /></h1>
          <p className="mb-6 text-lg">
            <b>MedHelp</b> — <TranslatedText text="це сучасна платформа для навчання навичкам
     екстреної медичної допомоги з використанням інтерактивних симуляцій та курсів." />
          </p>
          <ul className="list-disc pl-6 space-y-2 mb-6 text-base">
            <li><TranslatedText text="Розроблена як дипломний проєкт студентом" />
     <b><TranslatedText text="Ватажко Михайлом Андрійовичем" /></b> <TranslatedText text="(2025
     рік)" />.</li>
            <li><TranslatedText text="Мета — зробити навчання екстреної медичної допомоги доступним,
     цікавим та ефективним для всіх." /></li>
            <li><TranslatedText text="Платформа містить:" /></li>
              <ul className="list-[circle] pl-6 mt-1">
                <li><TranslatedText text="Інтерактивні симуляції екстрених ситуацій" /></li>
                <li><TranslatedText text="Навчальні курси для медиків та всіх охочих" /></li>
                <li><TranslatedText text="Зручне відстеження прогресу" /></li>
                <li><TranslatedText text="Сучасний пошуковий інтерфейс" /></li>
                <li><TranslatedText text="Адаптивний дизайн для будь-яких пристроїв" /></li>
              </ul>
            <li><TranslatedText text="Використані технології" /> : React + TailwindCSS, Next.js,
     TypeScript.</li>
          </ul>
          <div className="bg-gray-100 rounded p-4 text-gray-700 mb-6">
            <b><TranslatedText text="Призначення" />:</b> <TranslatedText text="Дипломний проєкт для
     здобуття ступеня бакалавра" /><br/>
            <b><TranslatedText text="Автор" />:</b> <TranslatedText text="Ватажко Михайло
     Андрійович" /><br/>
            <b><TranslatedText text="Рік" />:</b> 2025
          </div>
          <p className="text-base text-gray-600">
            <TranslatedText text="Якщо у вас є питання або пропозиції — пишіть на" /> <a
     href="mailto:medhelpedu@gmail.com" className="text-blue-600
     underline">medhelpedu@gmail.com</a>
          </p>
        </div>
      </main>
      <Footer />
    </>
  )
}
```

courses\page.tsx

```
'use client'

import React, { useState, useEffect, useCallback, useMemo } from 'react';
import { useAuth } from '@/hooks/useAuth';
import { TranslatedText } from '@/components/TranslatedText';
import Navbar from '@/components/Navbar';
import Footer from '@/components/Footer';
import Modal from '@/components/Modal';
import { fetchWithAuth } from '@/utils/auth';
import CourseList from '@/components/cards/list/CardList';
import CourseForm from '@/components/form/Form';
import { useRouter } from 'next/navigation';
import Filter from '@/components/search/filter/Filter';
import Notification from '@/components/Notification';

const TABS = [
  { key: 'available', label: <TranslatedText text="Доступні курси" /> },
  { key: 'inProgress', label: <TranslatedText text="У процесі" /> },
  { key: 'completed', label: <TranslatedText text="Завершені" /> },
];

const CoursesPage = () => {
  const { user } = useAuth();
  const router = useRouter();

  const getInitialTab = () => {
    if (typeof window !== 'undefined') {
      return localStorage.getItem('coursesActiveTab') || 'available';
    }
    return 'available';
  };

  const [activeTab, setActiveTab] = useState(getInitialTab);

  const [availableCourses, setAvailableCourses] = useState<any[]>([]);
  const [inProgressCourses, setInProgressCourses] = useState<any[]>([]);
  const [completedCourses, setCompletedCourses] = useState<any[]>([]);
  const [myCourses, setMyCourses] = useState<any[]>([]);

  const [allAvailableCourses, setAllAvailableCourses] = useState<any[]>([]);
  const [allInProgressCourses, setAllInProgressCourses] = useState<any[]>([]);
  const [allCompletedCourses, setAllCompletedCourses] = useState<any[]>([]);
  const [allMyCourses, setAllMyCourses] = useState<any[]>([]);

  const [loading, setLoading] = useState(false);
```

```
const [notification, setNotification] = useState<{ type: 'success' | 'error'; message: string } |
    null>(null);
const [showCreate, setShowCreate] = useState(false);
const [showEdit, setShowEdit] = useState(false);
const [editCourseId, setEditCourseId] = useState<string | number | null>(null);
const [deleteCourseId, setDeleteCourseId] = useState<string | number | null>(null);
const [showDeleteModal, setShowDeleteModal] = useState(false);
const [reloadFlag, setReloadFlag] = useState(0);

const [title, setTitle] = useState('');
const [description, setDescription] = useState('');
const [sections, setSections] = useState<any[]>([]);
const [editCourse, setEditCourse] = useState<any | null>(null);

const handleFiltered = useCallback(
  ({ available, inProgress, completed, my }: any) => {
    setAvailableCourses(available);
    setInProgressCourses(inProgress);
    setCompletedCourses(completed);
    setMyCourses(my);
  },
  []
);

const items = useMemo(() => ({
  available: allAvailableCourses,
  inProgress: allInProgressCourses,
  completed: allCompletedCourses,
  my: allMyCourses
}), [allAvailableCourses, allInProgressCourses, allCompletedCourses, allMyCourses]);

useEffect(() => {
  const fetchAll = async () => {
    setLoading(true);
    try {
      const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

      const resAvailable = await fetchWithAuth(`/api/courses/available?lang=${currentLang}`);
      if (resAvailable.ok) {
        const notStartedCourses = await resAvailable.json();
        setAllAvailableCourses(notStartedCourses || []);
      }

      const resInProgress = await fetchWithAuth(`/api/courses/progress?lang=${currentLang}`);
      if (resInProgress.ok) {
        const inProgressCourses = await resInProgress.json();
        setAllInProgressCourses(inProgressCourses || []);
      }

      const resCompleted = await fetchWithAuth(`/api/courses/completed?lang=${currentLang}`);
      if (resCompleted.ok) {
        const completedCourses = await resCompleted.json();
        setAllCompletedCourses(completedCourses || []);
      }

      if (user?.role === 'MEDIC' || user?.role === 'ROOT') {
        const endpoint = user.role === 'ROOT'
          ? `/api/courses/control?all=1&lang=${currentLang}`
          : `/api/courses/control?lang=${currentLang}`;

        const resMine = await fetchWithAuth(endpoint);
        const myCourseData = resMine.ok ? await resMine.json() : [];
        setAllMyCourses(myCourseData);
      }
    } catch (error: any) {
      if (error instanceof Error && error.message === 'Unauthorized') {
        router.push('/unauthorized');
      } else {
        setNotification({ type: 'error', message: 'Error loading courses' });
      }
    } finally {
      setLoading(false);
    }
  };
  fetchAll();
}, [reloadFlag]);

const handleTabChange = (tab: string) => {
  setActiveTab(tab);
  if (typeof window !== 'undefined') {
    localStorage.setItem('coursesActiveTab', tab);
  }
};

const handleEditCourse = (course: any) => {
  setEditCourseId(course.id);
  setTitle(course.title);
  setDescription(course.description);
  setSections(course.sections || [{ title: '', lessons: [] }]);
  setEditCourse(course);
  setShowEdit(true);
};

const handleDeleteCourseConfirmed = async () => {
  if (!deleteCourseId) return;
  setShowDeleteModal(false);
  try {
```

```
    const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

    const res = await fetchWithAuth(`/api/courses/${deleteCourseId}?lang=${currentLang}`, {
    method: 'DELETE' });

    if (!res.ok) {
      const errorData = await res.json();
      throw new Error(errorData.error);
    }

    const result = await res.json();
    setNotification({ type: 'success', message: result.message });
    setReloadFlag(f => f + 1);
  } catch (error: any) {
    if (error instanceof Error && error.message === 'Unauthorized') {
      router.push('/unauthorized');
    } else {
      setNotification({ type: 'error', message: error.message });
    }
  }
  setDeleteCourseId(null);
};

const handleCourseCardClick = (course: any) => {
  if (course.id === 'create') {
    setShowCreate(true);
  } else {
    router.push(`/courses/${course.id}`);
  }
};

const getActiveCoursesForTab = () => {
  switch (activeTab) {
    case 'available': return availableCourses;
    case 'inProgress': return inProgressCourses;
    case 'completed': return completedCourses;
    case 'my': return myCourses;
    default: return [];
  }
};

return (
  <>
    <Navbar />
    {notification && (
      <Notification
        type={notification.type}
        message={notification.message}
        onClose={() => setNotification(null)}
      />
    )}
    <div className="container-wrapper py-8">
      <h1 className="text-3xl font-bold section-spacing"><TranslatedText text="Курси" /></h1>
      <div className="section-spacing flex gap-2 flex-wrap mb-2">
        {TABS.map(tab => (
          <button
            key={tab.key}
            className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
    tab.key ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
            onClick={() => handleTabChange(tab.key)}
          >
            <span>{tab.label}</span>
            {tab.key === 'available' && availableCourses.length > 0 && (
              <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
    [1.5rem] text-center">{availableCourses.length}</span>
            )}
            {tab.key === 'inProgress' && inProgressCourses.length > 0 && (
              <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
    [1.5rem] text-center">{inProgressCourses.length}</span>
            )}
            {tab.key === 'completed' && completedCourses.length > 0 && (
              <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
    [1.5rem] text-center">{completedCourses.length}</span>
            )}
          </button>
        ))}
        {(user?.role === 'MEDIC' || user?.role === 'ROOT') && (
          <button
            className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
    'my' ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
            onClick={() => handleTabChange('my')}
          >
            <span>
              <TranslatedText text={user?.role === 'ROOT' ? 'Усі курси' : 'Мої курси'} />
            </span>
            {user?.role === 'ROOT' && (
              <span className="ml-1 text-xs bg-yellow-100 text-yellow-800 px-1
    rounded">ROOT</span>
            )}
            {user?.role === 'MEDIC' && (
              <span className="ml-1 text-xs bg-red-100 text-red-800 px-1 rounded">MEDIC</span>
            )}
            {myCourses.length > 0 && (
              <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
    [1.5rem] text-center">{myCourses.length}</span>
            )}
          </button>
```

```
          )}
        </div>
        <Filter
          items={items}
          onFiltered={handleFiltered}
          total={getActiveCoursesForTab().length}
        />
        {loading ? (
          <div className="min-h-[200px] flex items-center justify-center section-spacing">
            <div className="animate-spin rounded-full h-10 w-10 border-b-2 border-blue-500"></div>
          </div>
        ) : (
          <CourseList
            items={getActiveCoursesForTab()}
            user={user}
            showProgress={activeTab === 'inProgress' || activeTab === 'completed'}
            completed={activeTab === 'completed'}
            isMine={activeTab === 'my'}
            onEdit={handleEditCourse}
            onDelete={(course) => { setDeleteCourseId(course.id); setShowDeleteModal(true); }}
            onStart={handleCourseCardClick}
          />
        )}
      </div>
      <Footer />

      {showCreate && (
        <Modal onClose={() => { setShowCreate(false); }}>
          <div className="p-3 w-full max-w-2xl overflow-y-auto max-h-[80vh] overscroll-contain">
            <h2 className="text-2xl font-bold mb-4"><TranslatedText text="Створити курс" /></h2>
            <CourseForm
              type="course"
              initialTitle=""
              initialDescription=""
              initialSections={[{ title: '', lessons: [] }]}
              onCancel={() => setShowCreate(false)}
              user={user}
              onSuccess={(message: string) => {
                setNotification({ type: 'success', message });
                setShowCreate(false);
                setReloadFlag(f => f + 1);
              }}
            />
          </div>
        </Modal>
      )}
      {showEdit && (
        <Modal onClose={() => { setShowEdit(false); }}>
          <div className="p-3 w-full max-w-2xl overflow-y-auto max-h-[80vh] overscroll-contain">
            <h2 className="text-2xl font-bold mb-4"><TranslatedText text="Редагувати курс" /></h2>
            <CourseForm
              type="course"
              initialTitle={title}
              initialDescription={description}
              initialSections={sections}
              initialImageUrl={editCourse?.imageUrl || ''}
              initialKeywords={editCourse?.keywords || []}
              entityId={editCourseId ?? undefined}
              onCancel={() => setShowEdit(false)}
              submitText="Зберегти"
              user={user}
              onSuccess={(message: string) => {
                setNotification({ type: 'success', message });
                setShowEdit(false);
                setReloadFlag(f => f + 1);
              }}
            />
          </div>
        </Modal>
      )}
      {showDeleteModal && (
        <Modal onClose={() => setShowDeleteModal(false)}>
          <div className="p-6 w-full max-w-md mx-auto">
            <h2 className="text-xl font-bold mb-4 text-center text-red-600">
              <TranslatedText text="Видалити курс?" />
            </h2>
            <p className="mb-6 text-center">
              <TranslatedText text="Ви впевнені, що хочете видалити цей курс? Цю дію не можна
      скасувати." />
            </p>
            <div className="flex justify-end space-x-2">
              <button className="px-4 py-2 bg-gray-300 rounded" onClick={() =>
      setShowDeleteModal(false)}>
                <TranslatedText text="Скасувати" />
              </button>
              <button className="px-4 py-2 bg-red-600 text-white rounded"
      onClick={handleDeleteCourseConfirmed}>
                <TranslatedText text="Видалити" />
              </button>
            </div>
          </div>
        </Modal>
      )}
    </>
  );
}
```

```
export default CoursesPage;
```

courses\[id]\page.tsx

```
"use client";

import React, { useEffect, useState } from 'react';
import { useParams, useRouter } from 'next/navigation';
import { useAuth } from '@/hooks/useAuth';
import { fetchWithAuth } from '@/utils/auth';
import CourseStartHeader from '@/components/course/basic/CourseStartHeader';
import CourseSectionList, { Lesson } from '@/components/course/basic/CourseSectionList';
import LessonModal from '@/components/course/modal/LessonModal';
import Notification from '@/components/Notification';
import Navbar from '@/components/Navbar';
import Footer from '@/components/Footer';
import ErrorBlock from '@/components/page-error/PageErrorBlock';

const CoursePage = () => {
  const params = useParams();
  const router = useRouter();
  const { user } = useAuth();

  const courseId = params?.id;
  const [course, setCourse] = useState<any>(null);
  const [error, setError] = useState('');
  const [started, setStarted] = useState(false);
  const [progress, setProgress] = useState(0);
  const [selectedLesson, setSelectedLesson] = useState<any>(null);
  const [showLessonModal, setShowLessonModal] = useState(false);
  const [completedLessons, setCompletedLessons] = useState<number[]>([]);
  const [lessonLoading, setLessonLoading] = useState(false);
  const [notification, setNotification] = useState<{ type: 'success' | 'error'; message: string } |
      null>(null);

  const fetchCourse = async () => {
    setError('');
    try {
      const lang = localStorage.getItem('preferredLanguage') || 'ua';
      const res = await fetchWithAuth(`/api/courses/${courseId}?lang=${lang}`);
      const data = await res.json();

      if (!res.ok) throw new Error(data.error);
      setCourse(data);
      setStarted(!!data.userProgress);

      setCompletedLessons(data.sections
        ? data.sections.flatMap((s: any) => s.lessons.filter((l: any) => l.completed).map((l: any)
      => l.id))
        : []);

        if (data.userProgress && data.totalLessons) {
        const completedCount = data.sections
          ? data.sections.flatMap((s: any) => s.lessons.filter((l: any) => l.completed)).length
          : 0;
        setProgress(Math.round((completedCount / data.totalLessons) * 100));
      } else {
        setProgress(0);
      }
    } catch (error: any) {
      setError(error.message);
    }
  };

  useEffect(() => {
    if (courseId && !user) {
      router.push('/unauthorized');
    } else if (courseId && user) {
      fetchCourse();
    }
  }, [courseId, user]);

  const handleStart = async () => {
    try {
      const lang = localStorage.getItem('preferredLanguage') || 'ua';
      const res = await fetchWithAuth(`/api/courses/start?lang=${lang}`, {
        method: 'POST',
        body: JSON.stringify({ courseId: Number(courseId), lang }),
      });
      const data = await res.json();

      if (!res.ok) throw new Error(data.error);

      setStarted(true);
      setNotification({ type: 'success', message: data.message });
      setProgress(data.progress);

      await fetchCourse();
    } catch (error: any) {
      if (error instanceof Error && error.message === 'Unauthorized') {
        router.push('/unauthorized');
      } else {
        setNotification({ type: 'error', message: error.message });
      }
    }
  };
```

```
  const handleLessonClick = (lesson: Lesson) => {
    if (!started && !lesson.completed) return;
    setSelectedLesson(lesson);
    setShowLessonModal(true);
  };

  const handleCloseLesson = () => {
    setShowLessonModal(false);
    setSelectedLesson(null);
  };

  const handleCompleteLesson = async () => {
    if (!selectedLesson) return;
    setLessonLoading(true);
    try {
      const lang = localStorage.getItem('preferredLanguage') || 'ua';
      const res = await fetchWithAuth(`/api/lessons/complete?lang=${lang}`, {
        method: 'POST',
        body: JSON.stringify({ lessonId: selectedLesson.id, lang }),
      });
      const data = await res.json();

      if (!res.ok) throw new Error(data.error);

      setNotification({ type: 'success', message: data.message });
      setProgress(data.progress);

      if (data.isCompleted) {
        setNotification({ type: 'success', message: data.message });
        setTimeout(() => {
          router.push('/courses');
        }, 3000);
      } else {
        await fetchCourse();
      }
    } catch (error: any) {
      if (error instanceof Error && error.message === 'Unauthorized') {
        router.push('/unauthorized');
      } else {
        setNotification({ type: 'error', message: error.message });
      }
    } finally {
      setLessonLoading(false);
      setShowLessonModal(false);
      setSelectedLesson(null);
    }
  };

  if (error) return <ErrorBlock message={error} />;
  if (!course) return null;

  return (
    <>
      <Navbar />
      <div className="container mx-auto px-2 md:px-0 py-6">
        {notification && (
          <Notification type={notification.type} message={notification.message} onClose={() =>
      setNotification(null)} />
        )}
        <CourseStartHeader
          started={started}
          progress={progress}
          onStart={handleStart}
          title={course.title}
          description={course.description}
          completedCount={completedLessons.length}
          totalCount={course.totalLessons}
        />
        <CourseSectionList
          sections={course.sections}
          onLessonClick={handleLessonClick}
          locked={!started}
        />
        <LessonModal
          open={showLessonModal}
          onClose={handleCloseLesson}
          lesson={selectedLesson}
          onComplete={handleCompleteLesson}
          completed={!!selectedLesson?.completed}
          loading={lessonLoading}
        />
      </div>
      <Footer />
    </>
  );
};

export default CoursePage;
```

forbidden\page.tsx

```
import ForbiddenClient from '@/components/page-error/ForbiddenClient';

export default function Forbidden() {
  return <ForbiddenClient />;
}
```

simulations\page.tsx

```
'use client';

import React, { useState, useEffect, useCallback, useMemo } from 'react';
import { useAuth } from '@/hooks/useAuth';
import Navbar from '@/components/Navbar';
import Footer from '@/components/Footer';
import Modal from '@/components/Modal';
import SimulationForm from '@/components/form/Form';
import { fetchWithAuth } from '@/utils/auth';
import SimulationList from '@/components/cards/list/CardList';
import Filter from '@/components/search/filter/Filter';
import { TranslatedText } from '@/components/TranslatedText';
import Notification from '@/components/Notification';
import { useRouter } from 'next/navigation';
import SimulationModal from '@/components/simulation/SimulationModal';

const TABS = [
  { key: 'available', label: <TranslatedText text="Доступні симуляції" /> },
  { key: 'my', label: <TranslatedText text="Мої симуляції" /> },
  { key: 'all', label: <TranslatedText text="Усі симуляції" /> },
];

const SimulationsPage = () => {
  const { user } = useAuth();
  const router = useRouter();

  const getInitialTab = () => {
    if (typeof window !== 'undefined') {
      return localStorage.getItem('simulationsActiveTab') || 'available';
    }
    return 'available';
  };

  const [activeTab, setActiveTab] = useState(getInitialTab);

  const [availableSimulations, setAvailableSimulations] = useState<any[]>([]);
  const [mySimulations, setMySimulations] = useState<any[]>([]);

  const [allAvailableSimulations, setAllAvailableSimulations] = useState<any[]>([]);
  const [allMySimulations, setAllMySimulations] = useState<any[]>([]);

  const [loading, setLoading] = useState(false);
  const [notification, setNotification] = useState<{ type: 'success' | 'error'; message: string } |
    null>(null);
  const [showCreate, setShowCreate] = useState(false);
  const [showEdit, setShowEdit] = useState(false);
  const [editSimulationId, setEditSimulationId] = useState<string | number | null>(null);
  const [deleteSimulationId, setDeleteSimulationId] = useState<string | number | null>(null);
  const [showDeleteModal, setShowDeleteModal] = useState(false);
  const [reloadFlag, setReloadFlag] = useState(0);

  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [steps, setSteps] = useState<any[]>([]);
  const [editSimulation, setEditSimulation] = useState<any | null>(null);

  const [selectedSimulation, setSelectedSimulation] = useState<any | null>(null);
  const [showSimulationModal, setShowSimulationModal] = useState(false);

  const handleFiltered = useCallback(
    ({ available, my }: any) => {
      setAvailableSimulations(available);
      setMySimulations(my);
    },
    []
  );

  const items = useMemo(() => ({
    available: allAvailableSimulations,
    my: allMySimulations
  }), [allAvailableSimulations, allMySimulations]);

  useEffect(() => {
    const fetchAll = async () => {
      setLoading(true);
      try {
        const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

        const resAvailable = await fetchWithAuth(`/api/simulations/available?lang=${currentLang}`);
        if (resAvailable.ok) {
          const availableSimulations = await resAvailable.json();
          setAllAvailableSimulations(availableSimulations || []);
        }

        if (user?.role === 'MEDIC' || user?.role === 'ROOT') {
          const endpoint = user.role === 'ROOT'
            ? `/api/simulations/control?all=1&lang=${currentLang}`
            : `/api/simulations/control?lang=${currentLang}`;

          const resMine = await fetchWithAuth(endpoint);
          const mySimulationsData = resMine.ok ? await resMine.json() : [];
          setAllMySimulations(mySimulationsData);
        }
      } catch (error: any) {
        if (error instanceof Error && error.message === 'Unauthorized') {
```

```
        router.push('/unauthorized');
      } else {
        setNotification({ type: 'error', message: 'Error loading simulations' });
      }
    } finally {
      setLoading(false);
    }
  };
  fetchAll();
}, [reloadFlag]);

const handleTabChange = (tab: string) => {
  setActiveTab(tab);
  if (typeof window !== 'undefined') {
    localStorage.setItem('simulationsActiveTab', tab);
  }
};

const handleEditSimulation = (simulation: any) => {
  setEditSimulationId(simulation.id);
  setTitle(simulation.title);
  setDescription(simulation.description);
  setSteps(simulation.steps || []);
  setEditSimulation(simulation);
  setShowEdit(true);
};

const handleDeleteSimulationConfirmed = async () => {
  if (!deleteSimulationId) return;
  setShowDeleteModal(false);
  try {
    const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

    const res = await fetchWithAuth(`/api/simulations/${deleteSimulationId}?lang=${currentLang}`,
    { method: 'DELETE' });

    if (!res.ok) {
      const errorData = await res.json();
      throw new Error(errorData.error);
    }

    const result = await res.json();
    setNotification({ type: 'success', message: result.message });
    setReloadFlag(f => f + 1);
  } catch (error: any) {
    if (error instanceof Error && error.message === 'Unauthorized') {
      router.push('/unauthorized');
    } else {
      setNotification({ type: 'error', message: error.message });
    }
  }
  setDeleteSimulationId(null);
};

const handleSimulationCardClick = (simulation: any) => {
  if (simulation.id === 'create') {
    setShowCreate(true);
  } else {
    setSelectedSimulation(simulation);
    setShowSimulationModal(true);
  }
};

const getActiveSimulationsForTab = () => {
  switch (activeTab) {
    case 'available': return availableSimulations;
    case 'my': return mySimulations;
    default: return [];
  }
};

return (
  <>
    <Navbar />
    {notification && (
      <Notification
        type={notification.type}
        message={notification.message}
        onClose={() => setNotification(null)}
      />
    )}
    <div className="container-wrapper py-8">
      <h1 className="text-3xl font-bold section-spacing"><TranslatedText text="Симуляції" /></h1>
      <div className={`section-spacing flex gap-2 flex-wrap mb-2 ${user?.role === 'USER' ?
    'hidden' : ''}`}>
        <button
          className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
    TABS[0].key ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
          onClick={() => handleTabChange(TABS[0].key)}
        >
          <span>{TABS[0].label}</span>
          {availableSimulations.length > 0 && (
            <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
    [1.5rem] text-center">{availableSimulations.length}</span>
          )}
        </button>
        {user?.role === 'MEDIC' && (
```

```
      <button
        className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
TABS[1].key ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
        onClick={() => handleTabChange(TABS[1].key)}
      >
        <span>{TABS[1].label}</span>
        <span className="ml-1 text-xs bg-red-100 text-red-800 px-1 rounded">MEDIC</span>
        {mySimulations.length > 0 && (
          <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
[1.5rem] text-center">{mySimulations.length}</span>
        )}
      </button>
    )}
    {user?.role === 'ROOT' && (
      <button
        className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
TABS[1].key ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
        onClick={() => handleTabChange(TABS[1].key)}
      >
        <span>{TABS[2].label}</span>
        <span className="ml-1 text-xs bg-yellow-100 text-yellow-800 px-1 rounded">ROOT</span>
        {mySimulations.length > 0 && (
          <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
[1.5rem] text-center">{mySimulations.length}</span>
        )}
      </button>
    )}
  </div>
  <Filter
    items={items}
    onFiltered={handleFiltered}
    total={getActiveSimulationsForTab().length}
  />
  {loading ? (
    <div className="min-h-[200px] flex items-center justify-center section-spacing">
      <div className="animate-spin rounded-full h-10 w-10 border-b-2 border-blue-500"></div>
    </div>
  ) : (
    <SimulationList
      items={getActiveSimulationsForTab()}
      user={user}
      showProgress={false}
      completed={false}
      isMine={activeTab === 'my'}
      onEdit={handleEditSimulation}
      onDelete={(simulation) => { setDeleteSimulationId(simulation.id);
setShowDeleteModal(true); }}
      onStart={handleSimulationCardClick}
    />
  )}
</div>
<Footer />

{showCreate && (
  <Modal onClose={() => { setShowCreate(false); }}>
    <div className="p-3 w-full max-w-2xl overflow-y-auto max-h-[80vh] overscroll-contain">
      <h2 className="text-2xl font-bold mb-4"><TranslatedText text="Створити симуляцію"
/></h2>
      <SimulationForm
        type="simulation"
        initialTitle=""
        initialDescription=""
        initialSteps={[{ title: '', content: '', videoUrl: ''}]}
        onCancel={() => setShowCreate(false)}
        user={user}
        onSuccess={(message: string) => {
          setNotification({ type: 'success', message });
          setShowCreate(false);
          setReloadFlag(f => f + 1);
        }}
      />
    </div>
  </Modal>
)}
{showEdit && (
  <Modal onClose={() => { setShowEdit(false); }}>
    <div className="p-3 w-full max-w-2xl overflow-y-auto max-h-[80vh] overscroll-contain">
      <h2 className="text-2xl font-bold mb-4"><TranslatedText text="Редагувати симуляцію"
/></h2>
      <SimulationForm
        type="simulation"
        initialTitle={title}
        initialDescription={description}
        initialSteps={steps}
        initialImageUrl={editSimulation?.imageUrl || ''}
        initialKeywords={editSimulation?.keywords || []}
        entityId={editSimulationId ?? undefined}
        onCancel={() => setShowEdit(false)}
        submitText="Зберегти"
        user={user}
        onSuccess={(message: string) => {
          setNotification({ type: 'success', message });
          setShowEdit(false);
          setReloadFlag(f => f + 1);
        }}
      />
    </div>
```

```
        </Modal>
      )}
      {showSimulationModal && selectedSimulation && (
        <SimulationModal
          steps={selectedSimulation.steps}
          title={selectedSimulation.title}
          description={selectedSimulation.description}
          onClose={() => { setShowSimulationModal(false); setSelectedSimulation(null); }}
        />
      )}
      {showDeleteModal && (
        <Modal onClose={() => setShowDeleteModal(false)}>
          <div className="p-6 w-full max-w-md mx-auto">
            <h2 className="text-xl font-bold mb-4 text-center text-red-600">
              <TranslatedText text="Видалити симуляцію?" />
            </h2>
            <p className="mb-6 text-center">
              <TranslatedText text="Ви впевнені, що хочете видалити цю симуляцію? Цю дію не можна
скасувати." />
            </p>
            <div className="flex justify-end space-x-2">
              <button className="px-4 py-2 bg-gray-300 rounded" onClick={() =>
setShowDeleteModal(false)}>
                <TranslatedText text="Скасувати" />
              </button>
              <button className="px-4 py-2 bg-red-600 text-white rounded"
onClick={handleDeleteSimulationConfirmed}>
                <TranslatedText text="Видалити" />
              </button>
            </div>
          </div>
        </Modal>
      )}
    </>
  );
};

export default SimulationsPage;
```

unauthorized\page.tsx

```
import UnauthorizedClient from '@/components/page-error/UnauthorizedClient';

export default function Unauthorized() {
  return <UnauthorizedClient />;
}
```

users\page.tsx

```
'use client';

import { useState, useCallback, useMemo, useEffect } from 'react';
import Navbar from '@/components/Navbar';
import { useAuth } from '@/hooks/useAuth';
import Footer from '@/components/Footer';
import Filter from '@/components/search/filter/Filter';
import UserList from '@/components/user/UserList';
import { useRouter } from 'next/navigation';
import Notification from '@/components/Notification';
import { TranslatedText } from '@/components/TranslatedText';
import { fetchWithAuth } from '@/utils/auth';
import Modal from '@/components/Modal';

const TABS = [
  { key: 'all', label: <TranslatedText text="Усі користувачі" /> },
  { key: 'users', label: <TranslatedText text="Користувачі" /> },
  { key: 'medics', label: <TranslatedText text="Медики" /> },
];

const UsersPage = () => {
  const { user } = useAuth();
  const router = useRouter();

  const getInitialTab = () => {
    if (typeof window !== 'undefined') {
      return localStorage.getItem('usersActiveTab') || 'all';
    }
    return 'all';
  };

  const [activeTab, setActiveTab] = useState(getInitialTab);

  const [users, setUsers] = useState<any[]>([]);
  const [simpleUsers, setSimpleUsers] = useState<any[]>([]);
  const [medics, setMedics] = useState<any[]>([]);

  const [allUsers, setAllUsers] = useState<any[]>([]);
  const [allSimpleUsers, setAllSimpleUsers] = useState<any[]>([]);
  const [allMedics, setAllMedics] = useState<any[]>([]);

  const [loading, setLoading] = useState(false);
  const [notification, setNotification] = useState<{ type: 'success' | 'error'; message: string } |
    null>(null);
  const [reloadFlag, setReloadFlag] = useState(0);

  const [selectedUser, setSelectedUser] = useState<any | null>(null);
  const [showUserStatusCard, setShowUserStatusCard] = useState(false);
```

```
const handleFiltered = useCallback(
  ({ users, simpleUsers, medics }: any) => {
    setUsers(users);
    setSimpleUsers(simpleUsers);
    setMedics(medics);
  },
  []
);

const items = useMemo(() => ({
  users: allUsers,
  simpleUsers: allSimpleUsers,
  medics: allMedics
}), [allUsers, allSimpleUsers, allMedics]);

useEffect(() => {
  const fetchAll = async () => {
    setLoading(true);
    try {
      const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

      const resUsers = await fetchWithAuth(`/api/users/all?lang=${currentLang}`);
      if (resUsers.ok) {
        const usersData = await resUsers.json();
        setAllUsers(usersData);
      }

      const resSimpleUsers = await fetchWithAuth(`/api/users/simple?lang=${currentLang}`);
      if (resSimpleUsers.ok) {
        const simpleUsersData = await resSimpleUsers.json();
        setAllSimpleUsers(simpleUsersData);
      }

      const resMedics = await fetchWithAuth(`/api/users/medics?lang=${currentLang}`);
      if (resMedics.ok) {
        const medicsData = await resMedics.json();
        setAllMedics(medicsData);
      }
    } catch (error: any) {
      if (error instanceof Error && error.message === 'Unauthorized') {
        router.push('/unauthorized');
      }
      if (error instanceof Error && error.message === 'Forbidden') {
        router.push('/forbidden');
      }
      setNotification({ type: 'error', message: 'Помилка завантаження користувачів' });
    } finally {
      setLoading(false);
    }
  };
  fetchAll();
}, [reloadFlag]);

const handleTabChange = (tab: string) => {
  setActiveTab(tab);
  if (typeof window !== 'undefined') {
    localStorage.setItem('usersActiveTab', tab);
  }
};

const handleCardClick = (user: any) => {
  setSelectedUser(user);
  setShowUserStatusCard(true);
};

const handleEditUserRole = async () => {
  setLoading(true);

  try {
    const currentLang = localStorage.getItem('preferredLanguage') || 'ua';
    const res = await fetchWithAuth(`/api/users/${selectedUser.id}?lang=${currentLang}`, {
      method: 'PATCH',
      body: JSON.stringify({
        role: selectedUser.role === 'USER' ? 'MEDIC' : 'USER'
      })
    });
    if (!res.ok) {
      const errorData = await res.json();
      throw new Error(errorData.error);
    }
    const result = await res.json();
    setNotification({ type: 'success', message: result.message });
    setShowUserStatusCard(false);
    setReloadFlag(f => f + 1);
  } catch (error: any) {
    if (error instanceof Error && error.message === 'Unauthorized') {
      router.push('/unauthorized');
    }
    if (error instanceof Error && error.message === 'Forbidden') {
      router.push('/forbidden');
    }
    setNotification({ type: 'error', message: error.message });
  } finally {
    setLoading(false);
  }
}
```

```
  const getActiveUsersForTab = () => {
    switch (activeTab) {
      case 'all': return users;
      case 'users': return simpleUsers;
      case 'medics': return medics;
      default: return [];
    }
  };

  return (
    <>
      <Navbar />
      {notification && (
        <Notification
          type={notification.type}
          message={notification.message}
          onClose={() => setNotification(null)}
        />
      )}
      <div className="container-wrapper py-8">
        <h1 className="text-3xl font-bold section-spacing"><TranslatedText text="Користувачі"
      /></h1>
        <div className="section-spacing flex gap-2 flex-wrap mb-2">
          {TABS.map(tab => (
            <button
              key={tab.key}
              className={`px-4 py-2 rounded-md flex items-center transition-colors ${activeTab ===
      tab.key ? 'bg-blue-600 text-white' : 'bg-gray-200 hover:bg-gray-300 text-gray-800'}`}
              onClick={() => handleTabChange(tab.key)}
            >
              <span>{tab.label}</span>
              {tab.key === 'all' && users.length > 0 && (
                <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
      [1.5rem] text-center">{users.length}</span>
              )}
              {tab.key === 'users' && simpleUsers.length > 0 && (
                <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
      [1.5rem] text-center">{simpleUsers.length}</span>
              )}
              {tab.key === 'medics' && medics.length > 0 && (
                <span className="ml-2 px-2 py-0.5 bg-blue-900 text-white text-xs rounded-full min-w-
      [1.5rem] text-center">{medics.length}</span>
              )}
            </button>
          ))}
        </div>
        <Filter
          isUsers={true}
          items={items}
          onFiltered={handleFiltered}
          total={getActiveUsersForTab().length}
        />
        {loading ? (
          <div className="min-h-[200px] flex items-center justify-center section-spacing">
            <div className="animate-spin rounded-full h-10 w-10 border-b-2 border-blue-500"></div>
          </div>
        ) : (
          <UserList
            users={getActiveUsersForTab()}
            onClick={handleCardClick}
          />
        )}
      </div>
      <Footer />

      {showUserStatusCard && selectedUser && (
        <Modal className="w-full max-w-md" onClose={() => setShowUserStatusCard(false)}>
          <div className="p-6 w-full max-w-md mx-auto">
            <h2 className="text-xl font-bold mb-4 text-center">
              {selectedUser.role === 'USER'
                ? <TranslatedText text="Призначити медиком?" />
                : <TranslatedText text="Прибрати роль медика?" />}
            </h2>
            <p className="mb-6 text-center">
              {selectedUser.email}
            </p>
            <div className="flex justify-center space-x-2">
              <button className="px-4 py-2 bg-gray-300 rounded" onClick={() =>
      setShowUserStatusCard(false)}>
                <TranslatedText text="Скасувати" />
              </button>
              <button
                className="px-4 py-2 bg-blue-600 text-white rounded"
                onClick={handleEditUserRole}
              >
                {selectedUser.role === 'USER'
                  ? <TranslatedText text="Призначити" />
                  : <TranslatedText text="Прибрати" />}
              </button>
            </div>
          </div>
        </Modal>
      )}
    </>
  );
}
```

```
export default UsersPage;
```

not-found.tsx

```
import NotFoundClient from '@/components/page-error/NotFoundClient';

export default function NotFound() {
  return <NotFoundClient />;
}
```

app\page.tsx (main page)

```
'use client'

import { useState, useEffect, useRef } from 'react';
import Link from 'next/link';
import Navbar from '@/components/Navbar';
import Footer from '@/components/Footer';
import { TranslatedText } from '@/components/TranslatedText';
import SearchBar from '@/components/search/bar/Bar';
import Carousel from '@/components/cards/carousel/Carousel';
import Notification from '@/components/Notification';
import SimulationModal from '@/components/simulation/SimulationModal';
import Register from '@/components/register/register_modal/Register';
import ConfirmCode from '@/components/register/confirm_code_modal/ConfirmCode';
import RegisterSuccess from '@/components/register/register_success_modal/RegisterSuccess';

declare global {
  interface Window {
    SpeechRecognition: any;
    webkitSpeechRecognition: any;
  }
}

const Home = () => {
  const [loading, setLoading] = useState(false);
  const [simulations, setSimulations] = useState<any[]>([]);
  const [notification, setNotification] = useState<{ type: 'success' | 'error'; message: string } |
      null>(null);
  const [searchResults, setSearchResults] = useState<any[]>([]);
  const [showResults, setShowResults] = useState(false);
  const [selectedSimulation, setSelectedSimulation] = useState<any | null>(null);
  const [showSimulationModal, setShowSimulationModal] = useState(false);
  const [showRegisterModal, setShowRegisterModal] = useState(false);
  const [registerEmail, setRegisterEmail] = useState('');
  const [registerName, setRegisterName] = useState('');
  const [registerPassword, setRegisterPassword] = useState('');
  const [registerStep, setRegisterStep] = useState<'register' | 'registerCode' |
      'registerSuccess'>('register');
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  useEffect(() => {
    const fetchSimulations = async () => {
      setLoading(true);
      try {
        const currentLang = localStorage.getItem('preferredLanguage') || 'ua';

        const res = await fetch(`/api/simulations/available?lang=${currentLang}`);
        if (res.ok) {
          const all = await res.json();
          setSimulations(all || []);
        } else {
          setSimulations([]);
        }
      } catch {
        setNotification({ type: 'error', message: 'Error loading simulations' });
      } finally {
        setLoading(false);
      }
    }
    fetchSimulations();
  }, []);

  return (
    <>
      <Navbar />
      {notification && (
        <Notification
          type={notification.type}
          message={notification.message}
          onClose={() => setNotification(null)}
        />
      )}
      <div className="bg-gradient-to-r from-blue-900 to-blue-700 text-white py-16">
        <div className="container-wrapper">
          <div className="flex flex-col items-center text-center">
            <h1 className="text-4xl md:text-5xl font-bold mb-6">
              <TranslatedText text="Платформа для навчання навичкам екстреної медичної допомоги" />
            </h1>
            <p className="text-xl mb-10 w-full">
              <TranslatedText text="Опишіть екстрену ситуацію, і ми надамо покрокові інструкції для
      надання медичної допомоги" />
            </p>
            <SearchBar
              simulations={simulations}
              onFilteredResults={(results) => {
```

```
            setSearchResults(results);
            setShowResults(true);
          }}
          withVoice={true}
        />
      </div>
    </div>
  </div>

  {showResults && (
    <div className="w-full mt-6 bg-white rounded-lg shadow-lg overflow-hidden text-left">
      <div className="p-4 bg-gray-50 border-b border-gray-200">
        <h3 className="font-medium text-gray-700">
          {searchResults.length > 0
            ? <TranslatedText text={`Знайдено результатів: ${searchResults.length}`} />
            : <TranslatedText text="Результатів не знайдено" />}
        </h3>
      </div>

      {searchResults.length > 0 ? (
        <div className="divide-y divide-gray-200">
          {searchResults.map(result => (
            <button
              key={result.id}
              className="block w-full text-left p-4 hover:bg-blue-50"
              onClick={() => {
                setSelectedSimulation(result);
                setShowSimulationModal(true);
              }}
            >
              <h4 className="font-medium text-blue-700 overflow-hidden">
                <TranslatedText text={result.title} />
              </h4>
              <p className="text-gray-600 mt-1 overflow-hidden">
                <TranslatedText text={result.description} />
              </p>
            </button>
          ))}
        </div>
      ) : (
        <div className="p-6 text-center text-gray-500">
          <p><TranslatedText text="Спробуйте інший пошуковий запит або перегляньте можливі
випадки нижче" /></p>
        </div>
      )}
    </div>
  )}

  <div className="py-12 bg-gray-50">
    <div className="container-wrapper=">
      <h2 className="text-2xl font-bold mb-8 text-center">
        <TranslatedText text="Доступні симуляції випадків" />
      </h2>
      <div>
        <Carousel
          items={simulations}
          loading={loading}
          emptyText="Немає симуляцій для відображення"
          onCardClick={(simulation) => {
            setSelectedSimulation(simulation);
            setShowSimulationModal(true);
          }}
        />
      </div>
    </div>
  </div>

  <div className="py-16 bg-white">
    <div className="container-wrapper">
      <h2 className="text-3xl font-bold mb-12 text-center">
        <TranslatedText text="Особливості платформи" />
      </h2>

      <div className="grid md:grid-cols-3 gap-10">
        <div className="text-center">
          <div className="bg-blue-100 rounded-full w-20 h-20 flex items-center justify-center
mx-auto mb-4">
            <span className="text-3xl">🔍</span>
          </div>
          <h3 className="text-xl font-semibold mb-3">
            <TranslatedText text="Швидкий пошук" />
          </h3>
          <p className="text-gray-600">
            <TranslatedText text="Знайдіть інструкції з надання екстренної медичної допомоги за
ключовими словами або голосовим запитом" />
          </p>
        </div>

        <div className="text-center">
          <div className="bg-blue-100 rounded-full w-20 h-20 flex items-center justify-center
mx-auto mb-4">
            <span className="text-3xl">▶</span>
          </div>
          <h3 className="text-xl font-semibold mb-3">
            <TranslatedText text="Інтерактивні симуляції" />
          </h3>
          <p className="text-gray-600">
```

```
            <TranslatedText text="Покрокові інструкції симуляцій для відпрацювання навичок
надання екстренної медичної допомоги" />
          </p>
        </div>

        <div className="text-center">
          <div className="bg-blue-100 rounded-full w-20 h-20 flex items-center justify-center
mx-auto mb-4">
            <span className="text-3xl">🎓</span>
          </div>
          <h3 className="text-xl font-semibold mb-3">
            <TranslatedText text="Навчальні курси" />
          </h3>
          <p className="text-gray-600">
            <TranslatedText text="Поглиблені курси з медицини для тих, хто хоче розвинути свої
навички. Доступні після реєстрації" />
          </p>
        </div>
      </div>
    </div>
  </div>

  <div className="bg-blue-900 text-white py-12">
    <div className="container-wrapper text-center">
      <h2 className="text-3xl font-bold mb-6">
        <TranslatedText text="Готові вивчати медичні навички глибше?" />
      </h2>
      <p className="text-xl mb-8 w-full mx-auto">
        <TranslatedText text="Зареєструйтеся, щоб отримати доступ до повних курсів та
відстежувати свій прогрес навчання" />
      </p>
      <div className="flex flex-col sm:flex-row justify-center space-y-4 sm:space-y-0 sm:space-
x-4">
        <button
          className="btn-secondary px-8 py-3"
          onClick={() => {
            setShowRegisterModal(true);
            setRegisterStep('register');
          }}
        >
          <TranslatedText text="Зареєструватися" />
        </button>
        <Link href="/courses" className="btn-outline-white px-8 py-3">
          <TranslatedText text="Переглянути курси" />
        </Link>
      </div>
    </div>
  </div>

  <Register
    open={showRegisterModal && registerStep === 'register'}
    onClose={() => setShowRegisterModal(false)}
    onSuccess={(email, name, password) => {
      setRegisterEmail(email);
      setRegisterName(name);
      setRegisterPassword(password);
      setRegisterStep('registerCode');
    }}
  />
  <ConfirmCode
    open={showRegisterModal && registerStep === 'registerCode'}
    email={registerEmail}
    name={registerName}
    password={registerPassword}
    lang={lang}
    onClose={() => setShowRegisterModal(false)}
    onBack={() => setRegisterStep('register')}
    onSuccess={() => setRegisterStep('registerSuccess')}
  />
  <RegisterSuccess
    open={showRegisterModal && registerStep === 'registerSuccess'}
    onLogin={() => setShowRegisterModal(false)}
  />

  {showSimulationModal && selectedSimulation && (
    <SimulationModal
      steps={selectedSimulation.steps}
      title={selectedSimulation.title}
      description={selectedSimulation.description}
      onClose={() => { setShowSimulationModal(false); setSelectedSimulation(null); }}
    />
  )}
  <Footer />
</>
  );
}

export default Home;
```

utils\auth.ts

```
import { NextRequest } from 'next/server';
import jwt from 'jsonwebtoken';
import { jwtDecode } from 'jwt-decode';

const JWT_SECRET = process.env.JWT_SECRET;
```

```
export function isTokenExpired(token: string): boolean {
  try {
    const payload: any = jwtDecode(token);
    if (!payload.exp) return false;
    return payload.exp * 1000 < Date.now();
  } catch {
    return true;
  }
}

export function getUserFromRequest(req: NextRequest) {
  const auth = req.headers.get('authorization');
  if (!auth) return null;
  const token = auth.replace('Bearer ', '');

  try {
    if (!JWT_SECRET) throw new Error('JWT_SECRET is not defined');
    const decoded: any = jwt.verify(token, JWT_SECRET);
    return {
      id: decoded.id,
      email: decoded.email,
      name: decoded.name,
      role: decoded.role,
    };
  } catch {
    return null;
  }
}

export function requireRole(user: any, roles: string[]) {
  if (!user || !roles.includes(user.role)) {
    throw new Error('Insufficient rights');
  }
}

export async function fetchWithAuth(url: string, options: RequestInit = {}) {
  const token = typeof window !== 'undefined' ? localStorage.getItem('token') : null;

  if (token && isTokenExpired(token)) {
    localStorage.removeItem('token');
    window.location.reload();
    return Promise.reject(new Error('Token expired'));
  }

  let headers: any = { ...(options.headers || {}) };
  if (!(options.body instanceof FormData)) {
    headers['Content-Type'] = 'application/json';
  }
  if (token) {
    headers['Authorization'] = `Bearer ${token}`;
  }

  const response = await fetch(url, { ...options, headers });

  if (response.status === 401) {
    return Promise.reject(new Error('Unauthorized'));
  }
  if (response.status === 403) {
    return Promise.reject(new Error('Forbidden'));
  }

  return response;
}
```

utils\messages.ts

```
export type Lang = 'ua' | 'en';

export const messages = {
  ua: {
    required: "Усі поля обов’язкові",
    notFound: "Користувача не знайдено",
    wrongPassword: "Невірний пароль",
    onlyGmail: "Дозволена лише пошта @gmail.com",
    exists: "Користувач з такою поштою вже існує",
    emailSend: "Не вдалося надіслати лист. Перевірте адресу пошти або налаштування сервера.",
    passwordResetSubject: "Код для відновлення пароля (MedHelp)",
    passwordResetText: (code: string) => `Ваш код для відновлення пароля: ${code}`,
    invalidOrExpiredCode: "Невірний або прострочений код",
    passwordChanged: "Пароль змінено",
    registrationSubject: (name: string) => `Код підтвердження | ${name} (MedHelp)`,
    registrationText: (code: string) => `Ваш код підтвердження: ${code}`,
    invalidEmail: "Некоректна адреса пошти.",
    instructionsSent: "Інструкції надіслано на вашу пошту",
    courseCreated: "Курс успішно створено",
    courseCreatedError: "Помилка створення курсу",
    courseUpdated: "Курс успішно оновлено",
    courseUpdatedError: "Помилка оновлення курсу",
    courseDeleted: "Курс успішно видалено",
    courseDeletedError: "Помилка видалення курсу",
    courseNotFound: "Курс не знайдено",
    notAuthorized: "Неавторизовано",
    notEnoughRights: "Недостатньо прав",
    fileRequired: "Файл не завантажено",
    uploadError: "Помилка завантаження файлу",
    courseStarted: "Курс розпочато",
    courseProgressUpdated: "Прогрес курса оновлено",
```

```
    courseCompleted: "Курс завершено",
    lessonNotFound: "Урок не знайдено",
    lessonCompleted: "Урок завершено",
    lessonIdRequired: "Необхідно вказати ID уроку",
    courseIdRequired: "Необхідно вказати ID курсу",
    courseProgressError: "Помилка при оновленні прогресу курсу",
    coursesLoadError: "Помилка отримання курсу",
    availableCoursesLoadError: "Помилка отримання доступних курсів",
    inProgressCoursesLoadError: "Помилка отримання курсів у процесі навчання",
    completedCoursesLoadError: "Помилка отримання завершених курсів",
    courseAlreadyExists: "Курс з такою назвою вже існує",
    simulationsLoadError: "Не вдалося завантажити симуляції",
    simulationAlreadyExists: "Симуляція з такою назвою вже існує",
    simulationCreated: "Симуляцію створено",
    simulationCreatedError: "Помилка створення симуляції",
    notFoundId: "Неправильний ID",
    simulationNotFound: "Симуляція не знайдена",
    simulationDeleted: "Симуляцію успішно видалено",
    simulationDeletedError: "Помилка видалення симуляції",
    simulationUpdated: "Симуляцію успішно оновлено",
    simulationUpdatedError: "Помилка оновлення симуляції",
    usersLoadError: "Помилка отримання користувачів",
    simpleUsersLoadError: "Помилка отримання звичайних користувачів",
    medicsLoadError: "Помилка отримання медиків",
    cannotChangeYourOwnRole: "Не можете змінити власну роль",
    cannotChangeRoleForThisUser: "Не можете змінити роль для цього користувача",
    invalidRole: "Неправильна роль",
    roleChanged: "Роль успішно змінена",
    errorChangingRole: "Помилка зміни ролі",
    accountDeleted: "Акаунт успішно видалено",
    errorDeletingAccount: "Помилка видалення акаунта"
  },
  en: {
    required: "All fields are required",
    notFound: "User not found",
    wrongPassword: "Incorrect password",
    onlyGmail: "Only @gmail.com email is allowed",
    exists: "A user with this email already exists",
    emailSend: "Failed to send email. Check the email address or server settings.",
    passwordResetSubject: "Password reset code (MedHelp)",
    passwordResetText: (code: string) => `Your password reset code: ${code}`,
    invalidOrExpiredCode: "Invalid or expired code",
    passwordChanged: "Password changed",
    registrationSubject: (name: string) => `Confirmation code | ${name} (MedHelp)`,
    registrationText: (code: string) => `Your confirmation code: ${code}`,
    invalidEmail: "Invalid email address.",
    instructionsSent: "Instructions sent to your email",
    courseCreated: "Course created successfully",
    courseCreatedError: "Error creating course",
    courseUpdated: "Course updated successfully",
    courseUpdatedError: "Error updating course",
    courseDeleted: "Course deleted successfully",
    courseDeletedError: "Error deleting course",
    courseNotFound: "Course not found",
    notAuthorized: "Not authorized",
    notEnoughRights: "Not enough rights",
    fileRequired: "No file uploaded",
    uploadError: "File upload error",
    courseStarted: "Course started",
    courseProgressUpdated: "Course progress updated",
    courseCompleted: "Course completed",
    lessonNotFound: "Lesson not found",
    lessonCompleted: "Lesson completed",
    lessonIdRequired: "Lesson ID is required",
    courseIdRequired: "Course ID is required",
    courseProgressError: "Error updating course progress",
    coursesLoadError: "Error loading course",
    availableCoursesLoadError: "Error loading available courses",
    inProgressCoursesLoadError: "Error loading courses in progress",
    completedCoursesLoadError: "Error loading completed courses",
    courseAlreadyExists: "A course with this name already exists",
    simulationsLoadError: "Failed to load simulations",
    simulationAlreadyExists: "A simulation with this name already exists",
    simulationCreated: "Simulation created successfully",
    simulationCreatedError: "Error creating simulation",
    notFoundId: "Invalid ID",
    simulationNotFound: "Simulation not found",
    simulationDeleted: "Simulation deleted successfully",
    simulationDeletedError: "Error deleting simulation",
    simulationUpdated: "Simulation updated successfully",
    simulationUpdatedError: "Error updating simulation",
    usersLoadError: "Error loading users",
    simpleUsersLoadError: "Error loading simple users",
    medicsLoadError: "Error loading medics",
    cannotChangeYourOwnRole: "Cannot change your own role",
    cannotChangeRoleForThisUser: "Cannot change role for this user",
    invalidRole: "Invalid role",
    roleChanged: "Role changed successfully",
    errorChangingRole: "Error changing role",
    accountDeleted: "Account deleted successfully",
    errorDeletingAccount: "Error deleting account"
  }
};
```

utils\prisma.ts

```
import { PrismaClient } from '@prisma/client';
```

```
const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    log: ['warn', 'error'],
  });

if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = prisma;
```

utils\s3.ts

```
import { S3Client } from '@aws-sdk/client-s3';

const B2_KEY_ID = process.env.B2_KEY_ID!;
const B2_APP_KEY = process.env.B2_APP_KEY!;
const B2_ENDPOINT = process.env.B2_ENDPOINT!;
const B2_REGION = process.env.B2_REGION!;

const globalForS3 = global as unknown as { s3: S3Client };

export const s3 =
  globalForS3.s3 ||
  new S3Client({
    region: B2_REGION,
    endpoint: `https://${B2_ENDPOINT}`,
    credentials: {
      accessKeyId: B2_KEY_ID,
      secretAccessKey: B2_APP_KEY,
    },
    forcePathStyle: true,
  });

if (process.env.NODE_ENV !== 'production') globalForS3.s3 = s3;
```

utils\slugify.ts

```
export function slugify(str: string) {
  return str
    .toString()
    .toLowerCase()
    .replace(/\s+/g, '-')
    .replace(/[^\p{L}\d\-_]+/gu, '')
    .replace(/\-\-+/g, '-')
    .replace(/^-+/, '')
    .replace(/-+$/, '');
}
```

hooks\useAuth.ts

```
export { useAuth } from '../components/AuthContext';
export type { AuthUser } from '../components/AuthContext';
```

/components

AuthContext.tsx

```
'use client';

import React, { createContext, useContext, useState, useEffect, useCallback } from 'react';
import { jwtDecode } from 'jwt-decode';
import { isTokenExpired } from '@/utils/auth';

export interface AuthUser {
  id: number;
  email: string;
  name?: string;
  role?: string;
}

interface AuthContextType {
  user: AuthUser | null;
  login: (token: string) => void;
  logout: () => void;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [user, setUser] = useState<AuthUser | null>(null);

  useEffect(() => {
    const token = typeof window !== 'undefined' ? localStorage.getItem('token') : null;
    if (token) {
      if (isTokenExpired(token)) {
        localStorage.removeItem('token');
        setUser(null);
        return;
      }
      try {
        const decoded: any = jwtDecode(token);
        setUser({
          id: decoded.id,
          email: decoded.email,
          name: decoded.name,
```

```
        role: decoded.role,
      });
    } catch {
      setUser(null);
    }
  } else {
    setUser(null);
  }
}, []);

const login = useCallback((token: string) => {
  if (isTokenExpired(token)) {
    localStorage.removeItem('token');
    setUser(null);
    return;
  }
  localStorage.setItem('token', token);
  const decoded: any = jwtDecode(token);
  setUser({
    id: decoded.id,
    email: decoded.email,
    name: decoded.name,
    role: decoded.role,
  });
}, []);

const logout = useCallback(() => {
  localStorage.removeItem('token');
  localStorage.removeItem('coursesActiveTab');
  localStorage.removeItem('simulationsActiveTab');
  setUser(null);
}, []);

return (
  <AuthContext.Provider value={{ user, login, logout }}>
    {children}
  </AuthContext.Provider>
);
};

export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) throw new Error('useAuth must be used within AuthProvider');
  return ctx;
}
```

Navbar.tsx

```
'use client'

import Link from 'next/link';
import { TranslatedText } from './TranslatedText';
import { LanguageSwitcher } from './LanguageSwitcher'
import { usePathname, useRouter } from 'next/navigation';
import Login from './login/login_modal/Login';
import Register from './register/register_modal/Register';
import ForgotPassword from './login/forgot_password_modal/ForgotPassword';
import ResetPassword from './login/reset_password_modal/ResetPassword';
import ConfirmCode from './register/confirm_code_modal/ConfirmCode';
import RegisterSuccess from './register/register_success_modal/RegisterSuccess';
import { useState, useRef, useEffect } from 'react';
import { useAuth } from '@/hooks/useAuth'
import md5 from 'md5';
import PreloaderWrapper from './PreloaderWrapper';
import Modal from './Modal';
import { fetchWithAuth } from '@/utils/auth';

const RedCrossIcon = () => (
  <svg width="32" height="32" viewBox="0 0 80 80" fill="none" xmlns="http://www.w3.org/2000/svg">
    <circle cx="40" cy="40" r="36" stroke="#E53935" strokeWidth="8" fill="#fff" />
    <rect x="34" y="20" width="12" height="40" rx="3" fill="#E53935" />
    <rect x="20" y="34" width="40" height="12" rx="3" fill="#E53935" />
  </svg>
);

export default function Navbar() {
  const pathname = usePathname();
  const router = useRouter();
  const { user, logout } = useAuth();
  const [menuOpen, setMenuOpen] = useState(false);
  const menuRef = useRef<HTMLDivElement>(null);

  const [modal, setModal] = useState<'login' | 'forgot' | 'reset' | 'register' | 'registerCode' |
      'registerSuccess' | null>(null);
  const [loginEmail, setLoginEmail] = useState('');
  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [registerEmail, setRegisterEmail] = useState('');
  const [registerName, setRegisterName] = useState('');
  const [registerPassword, setRegisterPassword] = useState('');
  const [loadingLogout, setLoadingLogout] = useState(false);
  const navRef = useRef<HTMLDivElement>(null);
  const [isEmailOverflow, setIsEmailOverflow] = useState(false);
  const [showDeleteModal, setShowDeleteModal] = useState(false);

  useEffect(() => {
    function handleClick(e: MouseEvent) {
      if (menuRef.current && !menuRef.current.contains(e.target as Node)) {
```

```
      setMenuOpen(false);
    }
  }
  if (menuOpen) document.addEventListener('mousedown', handleClick);
  return () => document.removeEventListener('mousedown', handleClick);
}, [menuOpen]);

useEffect(() => {
  setMenuOpen(false);
}, [user]);

useEffect(() => {
  function handleResize() {
    if (!navRef.current) return;
    const emailSpan = navRef.current.querySelector('.user-email-span') as HTMLSpanElement;
    if (emailSpan) {
      setIsEmailOverflow(emailSpan.offsetWidth + emailSpan.offsetLeft > navRef.current.offsetWidth
    - 120); // 120px запас под меню
    }
  }
  window.addEventListener('resize', handleResize);
  handleResize();
  return () => window.removeEventListener('resize', handleResize);
}, [user]);

const handleLogout = async () => {
  setLoadingLogout(true);
  logout();
  setLoadingLogout(false);
  router.push('/');
};

const handleDeleteAccount = async () => {
  if (!user) return;
  setShowDeleteModal(false);
  try {
    setLoadingLogout(true);
    const currentLang = localStorage.getItem('preferredLanguage') || 'ua';
    const res = await fetchWithAuth(`/api/users/${user.id}?lang=${currentLang}`, {
      method: 'DELETE',
    });
    if (!res.ok) {
      const errorData = await res.json();
      alert(errorData.error);
      setLoadingLogout(false);
      return;
    }
    logout();
    setLoadingLogout(false);
    router.push('/');
  } catch (error) {
    setLoadingLogout(false);
  }
};

function getProfilePhoto(email: string) {
  return `https://www.gravatar.com/avatar/${md5(email.trim().toLowerCase())}?d=identicon`;
}

const [mobileMenuOpen, setMobileMenuOpen] = useState(false);

return (
  <nav className="bg-white shadow-md">
    <div className="container-wrapper py-4 flex justify-between items-center">
      <div className="flex justify-start items-start space-x-2 text-xl font-bold text-primary">
        <RedCrossIcon />
        <span>MedHelp</span>
      </div>
      <ul className="hidden md:flex flex-1 justify-center space-x-6">
        <li>
          <Link
            href="/"
            className={`font-medium hover:text-primary transition-colors border-b-2 ${pathname ===
  '/' ? 'text-primary font-bold border-primary' : 'border-transparent'}`}
          >
            <TranslatedText text="Головна" />
          </Link>
        </li>
        {user && (
          <>
            <li>
              <Link
                href="/courses"
                className={`font-medium hover:text-primary transition-colors border-b-2
  ${pathname.startsWith('/courses') ? 'text-primary font-bold border-primary' : 'border-
  transparent'}`}
              >
                <TranslatedText text="Курси" />
              </Link>
            </li>
            <li>
              <Link
                href="/simulations"
                className={`font-medium hover:text-primary transition-colors border-b-2
  ${pathname.startsWith('/simulations') ? 'text-primary font-bold border-primary' : 'border-
  transparent'}`}
              >
```

```
            <TranslatedText text="Симуляції" />
          </Link>
        </li>
      </>
    )}
    <li>
      <Link
        href="/about"
        className={`font-medium hover:text-primary transition-colors border-b-2
${pathname.startsWith('/about') ? 'text-primary font-bold border-primary' : 'border-
transparent'}`}
      >
        <TranslatedText text="Про проект" />
      </Link>
    </li>
  </ul>
  <div className="justify-end items-end space-x-3 flex">
    <div className="lang-switcher-block hidden [@media(min-
width:400px)]:block"><LanguageSwitcher /></div>
    {user ? (
      <div className="relative avatar-block" ref={menuRef}>
        <button onClick={() => setMenuOpen(v => !v)} className="flex items-center space-x-2
focus:outline-none">
          <img
            src={getProfilePhoto(user.email)}
            alt="avatar"
            className="w-10 h-10 rounded-full border object-cover"
            style={{ minWidth: 40, minHeight: 40, maxWidth: 48, maxHeight: 48 }}
          />
          <span className="font-medium text-gray-700 hidden lg:block user-email-
span">{user.email}</span>
        </button>
        {menuOpen && (
          <div className="absolute right-0 mt-2 bg-white shadow-lg rounded z-50">
            <div className="lg:hidden block w-full text-left px-4 py-2 text-gray-
700">{user.email}</div>
            {user?.role === 'ROOT' && (
              <Link href="/users" className="block w-full text-left px-4 py-2 hover:bg-gray-
100 font-bold text-blue-600"><TranslatedText text="Користувачі" /></Link>
            )}
            <button className="block w-full text-left px-4 py-2 hover:bg-gray-100 font-bold
text-black-600" onClick={handleLogout}><TranslatedText text="Вийти" /></button>
            <button className="block w-full text-left px-4 py-2 hover:bg-gray-100 font-bold
text-red-600" onClick={() => setShowDeleteModal(true)}><TranslatedText text="Видалити акаунт"
/></button>
          </div>
        )}
      </div>
    ) : (
      <>
        <button onClick={() => setModal('login')} className="px-4 py-2 border border-blue-600
text-blue-600 rounded hover:bg-blue-50 transition hidden sm:block">
          <TranslatedText text="Увійти" />
        </button>
        <button onClick={() => setModal('register')} className="px-4 py-2 border border-gray-
400 text-gray-700 rounded hover:bg-gray-100 transition hidden sm:block">
          <TranslatedText text="Реєстрація" />
        </button>
      </>
    )}

    <div className="md:hidden flex items-center">
      <button className="p-2 text-gray-500" onClick={() => setMobileMenuOpen(v => !v)}>
        <svg className="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M4 6h16M4
12h16m-7 6h7" />
        </svg>
      </button>
      {mobileMenuOpen && (
        <div className="absolute top-16 right-2 bg-white shadow-lg rounded z-50 flex flex-col
p-2 space-y-2 ease-in-out animate-slide-in">
          <Link href="/" className="px-4 py-2 hover:bg-gray-100 rounded"><TranslatedText
text="Головна" /></Link>
          {user && <Link href="/courses" className="px-4 py-2 hover:bg-gray-100
rounded"><TranslatedText text="Курси" /></Link>}
          {user && <Link href="/simulations" className="px-4 py-2 hover:bg-gray-100
rounded"><TranslatedText text="Симуляції" /></Link>}
          <Link href="/about" className="px-4 py-2 hover:bg-gray-100 rounded"><TranslatedText
text="Про проект" /></Link>
          <div className="border-t my-2" />
          <div className="lang-switcher-block hidden [@media(max-width:399px)]:block">
            <LanguageSwitcher />
          </div>
          {!user && (
            <>
              <button onClick={() => setModal('login')} className="px-4 py-2 border border-
blue-600 text-blue-600 rounded hover:bg-blue-50 transition w-full mb-2 block sm:hidden">
                <TranslatedText text="Увійти" />
              </button>
              <button onClick={() => setModal('register')} className="px-4 py-2 border border-
gray-400 text-gray-700 rounded hover:bg-gray-100 transition w-full block sm:hidden">
                <TranslatedText text="Реєстрація" />
              </button>
            </>
          )}
        </div>
```

```
          )}
        </div>
      </div>
    </div>

    <Login
      open={modal === 'login'}
      onClose={() => setModal(null)}
      onForgot={() => setModal('forgot')}
      setLoginEmail={setLoginEmail}
    />
    <ForgotPassword
      open={modal === 'forgot'}
      email={loginEmail}
      lang={lang}
      onClose={() => setModal(null)}
      onReset={() => setModal('reset')}
      onBack={() => setModal('login')}
    />
    <ResetPassword
      open={modal === 'reset'}
      email={loginEmail}
      lang={lang}
      onClose={() => setModal(null)}
      onBack={() => setModal('login')}
      onLogin={() => setModal('login')}
    />
    <Register
      open={modal === 'register'}
      onClose={() => setModal(null)}
      onSuccess={(email, name, password) => {
        setRegisterEmail(email);
        setRegisterName(name);
        setRegisterPassword(password);
        setModal('registerCode');
      }}
    />
    <ConfirmCode
      open={modal === 'registerCode'}
      email={registerEmail}
      name={registerName}
      password={registerPassword}
      lang={lang}
      onClose={() => setModal(null)}
      onBack={() => setModal('register')}
      onSuccess={() => setModal('registerSuccess')}
    />
    <RegisterSuccess
      open={modal === 'registerSuccess'}
      onLogin={() => setModal('login')}
    />
    {loadingLogout && (
      <PreloaderWrapper>
        <div className="flex flex-col items-center justify-center min-h-screen">
          <span className="text-lg font-medium mt-4"><TranslatedText text="Вихід..." /></span>
        </div>
      </PreloaderWrapper>
    )}
    {showDeleteModal && (
      <Modal onClose={() => setShowDeleteModal(false)}>
        <div className="p-6 w-full max-w-md mx-auto">
          <h2 className="text-xl font-bold mb-4 text-center text-red-600">
            <TranslatedText text="Видалити акаунт?" />
          </h2>
          <p className="mb-6 text-center">
            <TranslatedText text="Ви впевнені, що хочете видалити свій акаунт? Цю дію не можна
скасувати." />
          </p>
          <div className="flex justify-end space-x-2">
            <button className="px-4 py-2 bg-gray-300 rounded" onClick={() =>
setShowDeleteModal(false)}>
              <TranslatedText text="Скасувати" />
            </button>
            <button className="px-4 py-2 bg-red-600 text-white rounded"
onClick={handleDeleteAccount}>
              <TranslatedText text="Видалити" />
            </button>
          </div>
        </div>
      </Modal>
    )}
  </nav>
 )
}
```

Footer.tsx

```
'use client'

import { TranslatedText } from './TranslatedText'

export default function Footer() {
  return (
    <footer className="bg-gray-800 text-white">
      <div className="container-wrapper py-4">
        <div className="flex flex-col md:flex-row md:justify-between md:items-start gap-2">
          <div>
```

```
            <div className="text-xl font-bold text-white mb-2">
              MedHelp
            </div>
            <p className="text-gray-300">
              <TranslatedText text="Платформа для навчання навичкам екстреної медичної допомоги" />
            </p>
          </div>
          <div>
            <div className="text-xl font-bold text-white mb-2">
              <TranslatedText text="Контакти" />
            </div>
            <p className="text-gray-300">
              Email: medhelpedu@gmail.com <TranslatedText text="Телефон: +380 63 704 77 31" />
            </p>
          </div>
        </div>
        <div className="mt-4 pt-6 border-t border-gray-700 text-center text-gray-400">
          <p>
            <TranslatedText text="© 2025 MedHelp. Усі права захищені" />
          </p>
        </div>
      </div>
    </footer>
  )
}
```

Notification.tsx

```
import React, { useEffect } from 'react';

const Notification = ({ type, message, onClose }: { type: 'success' | 'error', message: string,
      onClose?: () => void }) => {
  useEffect(() => {
    const timer = setTimeout(() => {
      if (onClose) onClose();
    }, 5000);
    return () => clearTimeout(timer);
  }, [onClose]);

  return (
    <div
      className={`fixed top-4 left-1/2 transform -translate-x-1/2 px-6 py-3 rounded shadow-lg z-50
      text-white text-center ${type === 'success' ? 'bg-green-600' : 'bg-red-600'}`}
    >
      {message}
    </div>
  );
};

export default Notification;
```

Modal.tsx

```
import React from 'react';

interface ModalProps {
  open?: boolean;
  onClose: () => void;
  children: React.ReactNode;
  className?: string;
}

export default function Modal({ open = true, onClose, children, className = '' }: ModalProps) {
  if (!open) return null;

  return (
    <>
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40
      backdrop-blur-sm">
      <div
          className={`bg-white rounded-lg shadow-lg py-6 px-2 w-full relative animate-fadeIn
      overflow-hidden max-w-2xl ${className}`}
        onClick={(e) => e.stopPropagation()}
      >
        <button
            className="absolute top-2 right-4 text-gray-400 hover:text-gray-600 text-xl transition-
      colors"
          onClick={onClose}
        >
          &times;
        </button>
        {children}
      </div>
    </div>
    </>
  );
}
```

LanguageSwitcher.tsx

```
'use client'

import { useState, useEffect } from 'react';

function getInitialLang(): 'ua' | 'en' {
  if (typeof window !== 'undefined') {
```

```
    const saved = localStorage.getItem('preferredLanguage');
    if (saved === 'en' || saved === 'ua') return saved;
  }
  return 'ua';
}

export function LanguageSwitcher() {
  const [language, setLanguage] = useState<'ua' | 'en'>('ua');
  const [mounted, setMounted] = useState(false);

  useEffect(() => {
    setMounted(true);
    const saved = getInitialLang();
    setLanguage(saved);

  }, []);

  const setLang = (lang: 'ua' | 'en') => {
    setLanguage(lang);
    localStorage.setItem('preferredLanguage', lang);
    const event = new CustomEvent('languageChanged', { detail: { language: lang } });
    window.dispatchEvent(event);
  };

  if (!mounted) {
    return null;
  }

  return (
    <div className="flex items-center space-x-2">
      <button
        onClick={() => setLang('ua')}
        className={`px-2 py-1 rounded font-medium ${language === 'ua' ? 'bg-blue-600 text-white' :
      'bg-gray-200 text-gray-700'}`}
      >
        UA
      </button>
      <span className="text-gray-400">|</span>
      <button
        onClick={() => setLang('en')}
        className={`px-2 py-1 rounded font-medium ${language === 'en' ? 'bg-blue-600 text-white' :
      'bg-gray-200 text-gray-700'}`}
      >
        EN
      </button>
    </div>
  );
}
```

PreloaderWrapper.tsx

```
'use client';

import { useEffect, useState, createContext, useContext, useCallback, useRef } from 'react';
import { usePathname } from 'next/navigation';

function getInitialLang(): 'ua' | 'en' {
  if (typeof window !== 'undefined') {
    const saved = localStorage.getItem('preferredLanguage');
    if (saved === 'en' || saved === 'ua') return saved;
  }
  return 'ua';
}

interface TranslationContextType {
  loading: boolean;
  startTranslation: () => void;
  finishTranslation: () => void;
}

const TranslationContext = createContext<TranslationContextType | undefined>(undefined);

export function useTranslationContext() {
  const ctx = useContext(TranslationContext);
  if (!ctx) throw new Error('useTranslationContext must be used within TranslationProvider');
  return ctx;
}

function TranslationProvider({ children }: { children: React.ReactNode }) {
  const [mounted, setMounted] = useState(false);
  const [count, setCount] = useState(0);
  const [initialLang, setInitialLang] = useState<'ua' | 'en'>('ua');
  const failSafeTimer = useRef<NodeJS.Timeout | null>(null);

  useEffect(() => {
    setMounted(true);
    if (typeof window !== 'undefined') {
      const lang = getInitialLang();
      setInitialLang(lang);
      if (lang !== 'ua') setCount(1);
    }
  }, []);

  const startTranslation = useCallback(() => {
    setCount(c => c + 1);
    if (failSafeTimer.current) {
      clearTimeout(failSafeTimer.current);
```

```
      failSafeTimer.current = null;
    }
  }, []);

  const finishTranslation = useCallback(() => setCount(c => Math.max(0, c - 1)), []);

  useEffect(() => {
    if (mounted && count > 0) {
      if (failSafeTimer.current) clearTimeout(failSafeTimer.current);
      failSafeTimer.current = setTimeout(() => setCount(0), 2000);
      return () => {
        if (failSafeTimer.current) clearTimeout(failSafeTimer.current);
      };
    }
  }, [mounted, count]);

  const loading = mounted ? count > 0 : false;

  return (
    <TranslationContext.Provider value={{
      loading,
      startTranslation,
      finishTranslation,
    }}>
      {children}
    </TranslationContext.Provider>
  );
}

function PreloaderContent() {
  return (
    <div style={{
      position: 'fixed',
      top: 0,
      left: 0,
      width: '100vw',
      height: '100vh',
      background: 'white',
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
      zIndex: 9999,
      transition: 'opacity 0.5s',
    }}>
      <svg width="80" height="80" viewBox="0 0 80 80" fill="none" xmlns="http://www.w3.org/2000/svg"
      style={{ animation: 'spin 1.2s linear infinite' }}>
        <circle cx="40" cy="40" r="36" stroke="#E53935" strokeWidth="8" fill="#fff" />
        <rect x="34" y="20" width="12" height="40" rx="3" fill="#E53935" />
        <rect x="20" y="34" width="40" height="12" rx="3" fill="#E53935" />
      </svg>
    </div>
  );
}

export default function PreloaderWrapper({ children }: { children: React.ReactNode }) {
  const [mounted, setMounted] = useState(false);
  const pathname = usePathname();
  const [routeLoading, setRouteLoading] = useState(false);
  const [langLoading, setLangLoading] = useState(false);

  useEffect(() => {
    setMounted(true);
  }, []);

  useEffect(() => {
    if (mounted) {
      setRouteLoading(true);
      const timer = setTimeout(() => setRouteLoading(false), 500); // можно увеличить время, если
      нужно
      return () => clearTimeout(timer);
    }
  }, [pathname, mounted]);

  useEffect(() => {
    if (!mounted) return;
    const handler = () => {
      setLangLoading(true);
      setTimeout(() => setLangLoading(false), 500);
    };
    window.addEventListener('languageChanged', handler);
    return () => window.removeEventListener('languageChanged', handler);
  }, [mounted]);

  if (!mounted) {
    return <PreloaderContent />;
  }

  return (
    <TranslationProvider>
      <PreloaderWrapperInner routeLoading={routeLoading ||
      langLoading}>{children}</PreloaderWrapperInner>
    </TranslationProvider>
  );
}

function PreloaderWrapperInner({ children, routeLoading }: { children: React.ReactNode,
      routeLoading: boolean }) {
```

```
  const { loading } = useTranslationContext();
  const [showPreloader, setShowPreloader] = useState(true);

  useEffect(() => {
    let timer: NodeJS.Timeout | null = null;
    if (!loading && !routeLoading) {
      timer = setTimeout(() => setShowPreloader(false), 1000); // 1 секунда для любого языка
    } else {
      setShowPreloader(true);
    }
    return () => {
      if (timer) clearTimeout(timer);
    };
  }, [loading, routeLoading]);

  return (
    <>
      <div style={{ opacity: showPreloader ? 0 : 1, transition: 'opacity 0.5s' }}>
        {children}
      </div>
      {showPreloader && <PreloaderContent />}
    </>
  );
}
```

Spinner.tsx

```
import React from 'react';

const Spinner: React.FC = () => (
  <div className="flex items-center justify-center w-full h-full py-10">
    <svg width="60" height="60" viewBox="0 0 80 80" fill="none" xmlns="http://www.w3.org/2000/svg"
      className="animate-spin">
      <circle cx="40" cy="40" r="36" stroke="#E53935" strokeWidth="8" fill="#fff" />
      <rect x="34" y="20" width="12" height="40" rx="3" fill="#E53935" />
      <rect x="20" y="34" width="40" height="12" rx="3" fill="#E53935" />
    </svg>
  </div>
);

export default Spinner;
```

TranslatedText.tsx

```
'use client';

import { useState, useEffect } from 'react';

interface TranslatedTextProps {
  text: string;
  className?: string;
}

function getInitialLang(): 'ua' | 'en' {
  if (typeof window !== 'undefined') {
    const saved = localStorage.getItem('preferredLanguage');
    if (saved === 'en' || saved === 'ua') return saved;
  }
  return 'ua';
}

export function TranslatedText({ text, className }: TranslatedTextProps) {
  const [translatedText, setTranslatedText] = useState(text);
  const [isLoading, setIsLoading] = useState(false);
  const [currentLanguage, setCurrentLanguage] = useState<'ua' | 'en'>(getInitialLang);
  const [showSpinner, setShowSpinner] = useState(false);
  const [visible, setVisible] = useState(false);
  const [mounted, setMounted] = useState(false);

  const translateText = async (text: string, targetLang: string) => {
    if (targetLang === 'ua') return text;
    const lang = targetLang === 'en' ? 'en' : 'en';

    try {
      const response = await fetch('/api/translate', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ text, target: lang })
      });
      const data = await response.json();
      return data.translatedText || text;
    } catch (e) {
      return text;
    }
  };

  useEffect(() => {
    setMounted(true);
    setTimeout(() => setVisible(true), 50);
  }, []);

  useEffect(() => {
    let fadeTimeout: NodeJS.Timeout;
    let spinnerTimeout: NodeJS.Timeout;
    let showTextTimeout: NodeJS.Timeout;

    const doTranslation = async () => {
```

```
      setVisible(false);
      spinnerTimeout = setTimeout(() => setShowSpinner(true), 200);
      fadeTimeout = setTimeout(async () => {
        setIsLoading(true);
        const result = await translateText(text, currentLanguage);
        setTranslatedText(result);
        setIsLoading(false);
        setShowSpinner(false);
        showTextTimeout = setTimeout(() => setVisible(true), 100);
      }, 300);
    };

    if (mounted) {
      doTranslation();
    }

    return () => {
      clearTimeout(fadeTimeout);
      clearTimeout(spinnerTimeout);
      clearTimeout(showTextTimeout);
    };
  }, [text, currentLanguage, mounted]);

  useEffect(() => {
    const handleLanguageChange = (event: CustomEvent) => {
      setCurrentLanguage(event.detail.language);
    };
    window.addEventListener('languageChanged' as any, handleLanguageChange);
    return () => {
      window.removeEventListener('languageChanged' as any, handleLanguageChange);
    };
  }, []);

  if (!mounted) return null;

  return (
    <span
      className={className}
      style={{
        opacity: visible && !isLoading ? 1 : 0,
        transition: 'opacity 0.3s'
      }}
    >
      {showSpinner && isLoading ? <span className="spinner" /> : translatedText}
    </span>
  );
}

export async function t(text: string, lang: 'ua' | 'en') {
  if (lang === 'ua') return text;
  try {
    const response = await fetch('/api/translate', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ text, target: lang })
    });
    const data = await response.json();
    return data.translatedText || text;
  } catch (e) {
    return text;
  }
}
```

Image.tsx

```
import React from 'react';
import ImageLogic from './ImageLogic';
import ImageUI from './ImageUI';

interface ImageProps {
  imageUrl: string | null | undefined;
  title: string;
  className?: string;
  placeholderColor?: string;
  showPlaceholderIcon?: boolean;
}

const Image: React.FC<ImageProps> = ({
  imageUrl,
  title,
  className,
  placeholderColor,
  showPlaceholderIcon
}) => {
  return (
    <ImageLogic
      imageUrl={imageUrl}
      title={title}
      className={className}
      placeholderColor={placeholderColor}
      showPlaceholderIcon={showPlaceholderIcon}
    >
      {({
        isError,
        isLoading,
        handleError,
        className,
```

```
        placeholderColor,
        showPlaceholderIcon,
        imageUrl,
        title
      }) => (
        <ImageUI
          isError={isError}
          isLoading={isLoading}
          handleError={handleError}
          className={className}
          placeholderColor={placeholderColor}
          showPlaceholderIcon={showPlaceholderIcon}
          imageUrl={imageUrl}
          title={title}
        />
      )}
    </ImageLogic>
  );
};

export default Image;
```

ImageLogic.tsx

```
import React, { useState, useEffect } from 'react';

interface ImageLogicProps {
  imageUrl: string | null | undefined;
  title: string;
  className?: string;
  placeholderColor?: string;
  showPlaceholderIcon?: boolean;
  children: (props: {
    isError: boolean;
    isLoading: boolean;
    handleError: () => void;
    className: string;
    placeholderColor: string;
    showPlaceholderIcon: boolean;
    imageUrl: string | null | undefined;
    title: string;
  }) => React.ReactNode;
}

const ImageLogic: React.FC<ImageLogicProps> = ({
  imageUrl,
  title,
  className = 'w-full h-full object-cover',
  placeholderColor = 'bg-gray-100',
  showPlaceholderIcon = true,
  children
}) => {
  const [isError, setIsError] = useState(false);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    if (!imageUrl) {
      setIsLoading(false);
      return;
    }

    setIsError(false);
    setIsLoading(true);

    const img = new Image();
    img.src = imageUrl;

    img.onload = () => {
      setIsLoading(false);
    };

    img.onerror = () => {
      setIsError(true);
      setIsLoading(false);
    };
  }, [imageUrl]);

  const handleError = () => {
    setIsError(true);
  };

  return children({
    isError,
    isLoading,
    handleError,
    className,
    placeholderColor,
    showPlaceholderIcon,
    imageUrl,
    title
  });
};

export default ImageLogic;
```

ImageUI.tsx

```
import React from 'react';

interface ImageUIProps {
  isError: boolean;
  isLoading: boolean;
  handleError: () => void;
  className: string;
  placeholderColor: string;
  showPlaceholderIcon: boolean;
  imageUrl: string | null | undefined;
  title: string;
}

const MedicalCrossPlaceholder = ({ className, placeholderColor }: { className: string,
    placeholderColor: string }) => (
  <div className={`${placeholderColor} flex items-center justify-center ${className} transition-
    transform duration-500 hover:scale-110`}>
    <div className="text-red-600 w-full h-full flex items-center justify-center">
      <svg
        viewBox="0 0 100 100"
        className="w-[60%]"
        preserveAspectRatio="xMidYMid meet"
      >
        <circle cx="50" cy="50" r="45" fill="rgba(255, 255, 255, 0.7)" stroke="#E53E3E"
    strokeWidth="4" />
        <rect x="45" y="20" width="10" height="60" fill="#E53E3E" />
        <rect x="20" y="45" width="60" height="10" fill="#E53E3E" />
      </svg>
    </div>
  </div>
);

const ImageUI: React.FC<ImageUIProps> = ({
  isError,
  isLoading,
  handleError,
  className = 'w-full h-full object-cover',
  placeholderColor = 'bg-gray-50',
  showPlaceholderIcon = true,
  imageUrl,
  title
}) => {
  if (!imageUrl || isError) {
    return showPlaceholderIcon ? (
      <MedicalCrossPlaceholder className={className} placeholderColor={placeholderColor} />
    ) : (
      <div className={`${placeholderColor} ${className}`}></div>
    );
  }

  if (isLoading) {
    return (
      <div className={`${placeholderColor} ${className} animate-pulse flex items-center justify-
      center`}>
        <div className="w-10 h-10 border-4 border-blue-200 border-t-blue-500 rounded-full animate-
      spin"></div>
      </div>
    );
  }

  return (
    <div className="relative w-full h-full overflow-hidden">
      <img
        src={imageUrl}
        alt={title}
        className={`${className} transition-opacity duration-300`}
        onError={handleError}
      />
    </div>
  );
};

export default ImageUI;
```

Card.tsx

```
import React, { useState, useEffect } from 'react';
import CardUI from './CardUI';
import CardProgressBar from './CardProgressBar';
import CardFooter from './CardFooter';
import { isRecent } from '../utils/dateUtils';
import { TranslatedText, t } from '../../TranslatedText';

type ButtonType = 'edit' | 'delete' | 'create' | 'add' | null;

interface CardProps {
  item: any;
  onEdit?: (item: any) => void;
  onDelete?: (item: any) => void;
  onClick?: (item: any) => void;
  canEdit?: boolean;
  showProgress?: boolean;
  completed?: boolean;
  hoveredCard?: number | null;
  hoveredButton?: ButtonType;
  onMouseEnter?: (id: number) => void;
  onMouseLeave?: () => void;
```

```
  onButtonHover?: (button: ButtonType) => void;
}

const Card: React.FC<CardProps> = ({
  item,
  onEdit,
  onDelete,
  onClick,
  canEdit = false,
  showProgress = false,
  completed = false,
  hoveredCard,
  hoveredButton,
  onMouseEnter,
  onMouseLeave,
  onButtonHover
}) => {
  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [editPlaceholder, setEditPlaceholder] = useState('Редагувати');
  const [deletePlaceholder, setDeletePlaceholder] = useState('Видалити');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t('Редагувати', lang).then(setEditPlaceholder);
    t('Видалити', lang).then(setDeletePlaceholder);
  }, [lang]);

  const isRecentlyUpdated = item.editedAt && isRecent(item.editedAt);

  const handleEdit = (e: React.MouseEvent) => {
    e.stopPropagation();
    onEdit && onEdit(item);
  };

  const handleDelete = (e: React.MouseEvent) => {
    e.stopPropagation();
    onDelete && onDelete(item);
  };

  const handleClick = () => {
    onClick && onClick(item);
  };

  const renderLeftContent = () => {
    if (completed) {
      return (
        <div className="flex items-center text-green-600 text-sm">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
      fill="currentColor">
            <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0 00-
      1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z"
      clipRule="evenodd" />
          </svg>
          <TranslatedText text="Завершено" />
        </div>
      );
    }
    return null;
  };

  const renderProgressContent = () => {
    if (!showProgress) return null;

    return (
      <div className="px-2 pb-0.5">
        <div className="flex justify-between items-center text-xs mb-0.5">
          <span><TranslatedText text="Прогрес:" /> {item.progress || 0}%</span>
          <span>{item.completedLessonsCount}/{item.totalLessons}</span>
        </div>
        <CardProgressBar progress={item.progress || 0} />
      </div>
    );
  };

  const actionButtons = canEdit ? [
    {
      id: 'edit' as ButtonType,
      icon: (
        <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 text-blue-700" viewBox="0 0 20
      20" fill="currentColor">
          <path d="M13.586 3.586a2 2 0 112.828 2.828l-.793.793-2.828-2.828.793-.793zM11.379 5.793L3
      14.172V17h2.828l8.38-8.379-2.83-2.828z" />
        </svg>
      ),
      tooltip: editPlaceholder,
      onClick: handleEdit,
      hoverColor: 'bg-blue-100',
      onMouseEnter: () => onButtonHover && onButtonHover('edit'),
      onMouseLeave: () => onButtonHover && onButtonHover(null)
    },
    {
      id: 'delete' as ButtonType,
```

```
      icon: (
        <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 text-red-700" viewBox="0 0 20 20"
      fill="currentColor">
          <path fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000 2v10a2 2 0 002
      2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1 1 0 012 0v6a1 1 0 11-2
      0V8zm5-1a1 1 0 00-1 1v6a1 1 0 102 0V8a1 1 0 00-1-1z" clipRule="evenodd" />
        </svg>
      ),
      tooltip: deletePlaceholder,
      onClick: handleDelete,
      hoverColor: 'bg-red-100',
      onMouseEnter: () => onButtonHover && onButtonHover('delete'),
      onMouseLeave: () => onButtonHover && onButtonHover(null)
    }
  ] : [];

  const footer = (
    <>
      {renderProgressContent()}
      <CardFooter
        leftContent={renderLeftContent()}
        actionButtons={actionButtons}
        hoveredButton={hoveredButton}
        hoveredItem={hoveredCard}
      />
    </>
  );

  return (
    <CardUI
      title={item.title}
      description={item.description}
      imageUrl={item.imageUrl}
      isRecent={isRecentlyUpdated}
      version={item.version}
      authorName={item.author?.name}
      footer={footer}
      onClick={handleClick}
      onMouseEnter={() => onMouseEnter && onMouseEnter(item.id)}
      onMouseLeave={onMouseLeave}
    />
  );
};

export default Card;
```

CardFooter.tsx

```
import React from 'react';

type ButtonType = 'edit' | 'delete' | 'create' | 'add' | null;

export interface ActionButton {
  id: ButtonType;
  icon: React.ReactNode;
  tooltip: string;
  onClick: (e: React.MouseEvent) => void;
  color?: string;
  hoverColor?: string;
  onMouseEnter?: () => void;
  onMouseLeave?: () => void;
}

interface CardFooterProps {
  bgColor?: string;
  borderColor?: string;
  leftContent?: React.ReactNode;
  actionButtons?: ActionButton[];
  hoveredButton?: ButtonType;
  hoveredItem?: number | string | null;
  className?: string;
}

const CardFooter: React.FC<CardFooterProps> = ({
  bgColor = 'bg-pink-50',
  borderColor = 'border-pink-100',
  leftContent,
  actionButtons = [],
  hoveredButton,
  hoveredItem,
  className = ''
}) => {
  return (
    <div className={`px-2 py-3 ${bgColor} border-t ${borderColor} flex justify-between items-center
      ${className}`}>
      {leftContent && (
        <div className="flex items-center text-xs">
          {leftContent}
        </div>
      )}

      <div className="flex-grow"></div>

      {actionButtons.length > 0 && (
        <div className="flex space-x-1 ml-auto">
          {actionButtons.map((button) => (
            <div className="relative" key={button.id}>
```

```
              <button
                className={`p-1 rounded-full hover:${button.hoverColor || 'bg-gray-100'} transform
     transition-all`}
                onClick={button.onClick}
                onMouseEnter={button.onMouseEnter}
                onMouseLeave={button.onMouseLeave}
              >
                {button.icon}
              </button>
              {hoveredButton === button.id && hoveredItem !== null && (
                <div className="absolute -top-6 right-0 bg-gray-800 text-white text-xs px-1 py-0.5
     rounded whitespace-nowrap z-20">
                    {button.tooltip}
                </div>
              )}
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default CardFooter;
```

CardProgressBar.tsx

```
import React from 'react';

interface CardProgressBarProps {
  progress: number;
  className?: string;
  barColor?: string;
  bgColor?: string;
}

const CardProgressBar: React.FC<CardProgressBarProps> = ({
  progress,
  className = '',
  barColor = 'bg-blue-500',
  bgColor = 'bg-blue-200'
}) => {
  const normalizedProgress = Math.min(Math.max(progress || 0, 0), 100);

  return (
    <div className={`w-full mt-0.5 ${className}`}>
      <div className={`overflow-hidden h-1 text-xs flex rounded ${bgColor}`}>
        <div
          style={{ width: `${normalizedProgress}%` }}
          className={`shadow-none flex flex-col text-center whitespace-nowrap text-white justify-
     center ${barColor}`}
        ></div>
      </div>
    </div>
  );
};

export default CardProgressBar;
```

CardUI.tsx

```
import React from 'react';
import Image from './image/Image';
import { TranslatedText } from '../../TranslatedText';
import { capitalizeFirstLetter } from '../utils/textUtils';

export interface CardUIProps {
  title: string;
  description?: string;
  imageUrl?: string | null;
  isRecent?: boolean;
  version?: number;
  authorName?: string;
  className?: string;
  footer?: React.ReactNode;
  topRightBadge?: React.ReactNode;
  onClick?: () => void;
  onMouseEnter?: () => void;
  onMouseLeave?: () => void;
}

const CardUI: React.FC<CardUIProps> = ({
  title,
  description,
  imageUrl,
  isRecent,
  version,
  authorName,
  className = '',
  footer,
  topRightBadge,
  onClick,
  onMouseEnter,
  onMouseLeave
}) => {
  return (
    <div
```

```
      className={`relative bg-white rounded-lg shadow-md hover:shadow-lg transition-all transform
      hover:translate-y-[-4px] overflow-hidden h-[420px] flex flex-col cursor-pointer ${isRecent ?
      'border-l-2 border-yellow-400' : ''} ${className}`}
      onClick={onClick}
      onMouseEnter={onMouseEnter}
      onMouseLeave={onMouseLeave}
    >
      <div className="h-72 w-full relative overflow-hidden">
        <Image
          imageUrl={imageUrl}
          title={title}
          className="object-cover w-full h-full"
          placeholderColor="bg-gray-50"
          showPlaceholderIcon={true}
        />
        {authorName && (
          <div className="absolute top-2 left-2 bg-white bg-opacity-90 backdrop-blur-sm px-2 py-1
      rounded shadow-sm text-xs text-gray-800 z-10">
            <TranslatedText text="Автор:" /> {authorName}
          </div>
        )}
        {isRecent && (
          <div className="absolute top-2 right-2 px-2 py-0.5 bg-yellow-100 text-yellow-800 text-xs
      rounded-full z-10 shadow-sm">
            <TranslatedText text="Нещодавно оновлено" />
          </div>
        )}
        {topRightBadge && (
          <div className="absolute top-2 right-2 z-10">
            {topRightBadge}
          </div>
        )}
        {version && version > 1 && (
          <div className="absolute bottom-2 right-2 text-xs">
            <span className="bg-blue-100 bg-opacity-90 backdrop-blur-sm text-blue-700 rounded px-2
      py-0.5 shadow-sm">
              v.{version}
            </span>
          </div>
        )}
      </div>
      <div className="px-4 py-3 flex-1 flex flex-col">
        <h3
          className="text-xl font-semibold mb-3 overflow-hidden"
          style={{
            height: '3.1rem',
            display: '-webkit-box',
            WebkitLineClamp: 2,
            WebkitBoxOrient: 'vertical',
          }}
        >
          <TranslatedText text={capitalizeFirstLetter(title)} />
        </h3>
        <p className="text-gray-600 text-sm mb-auto overflow-hidden whitespace-nowrap text-
      ellipsis">
          <TranslatedText text={capitalizeFirstLetter(description || '')} />
        </p>
      </div>
      <div className="mt-auto">{footer && footer}</div>
    </div>
  );
};

export default CardUI;
```

CardGrid.tsx

```
import React, { ReactNode } from 'react';

interface CardGridProps {
  children: ReactNode;
}

const CardGrid: React.FC<CardGridProps> = ({
  children,
}) => {
  return (
    <div className={"grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6"}>
      {children}
    </div>
  );
};

export default CardGrid;
```

CardList.tsx

```
import React, { useState } from 'react';
import CardGrid from './CardGrid';
import Card from '../card/Card';
import CreateCard from './CreateCard';
import QuickAddButton from './QuickAddButton';
import { TranslatedText } from '../../TranslatedText';

type ButtonType = 'edit' | 'delete' | 'create' | 'add' | null;

interface CardListProps {
```

```
  items: any[];
  user: any;
  showProgress?: boolean;
  completed?: boolean;
  isMine?: boolean;
  onEdit?: (item: any) => void;
  onDelete?: (item: any) => void;
  onStart?: (item: any) => void;
}

const CardList: React.FC<CardListProps> = ({
  items,
  user,
  showProgress,
  completed,
  isMine,
  onEdit,
  onDelete,
  onStart,
}) => {
  const [hoveredCard, setHoveredCard] = useState<number | null>(null);
  const [hoveredButton, setHoveredButton] = useState<ButtonType>(null);

  const handleCreateClick = () => {
    if (user?.role === 'MEDIC' || user?.role === 'ROOT') {
      onStart && onStart({ id: 'create' });
    }
  };

  const isMineAndCanCreate = isMine && (user?.role === 'MEDIC' || user?.role === 'ROOT');

  if ((!items || items.length === 0) && isMineAndCanCreate) {
    return (
      <div className="px-2">
        <CardGrid>
          <CreateCard
            onClick={handleCreateClick}
            onMouseEnter={() => setHoveredButton('create')}
            onMouseLeave={() => setHoveredButton(null)}
            hoveredButton={hoveredButton}
          />
        </CardGrid>
      </div>
    );
  }

  if (!items || items.length === 0) {
    return (
      <div className="min-h-[200px] flex items-center justify-center text-gray-500">
        <TranslatedText text="Нічого не знайдено" />
      </div>
    );
  }

  const canEditItem = (item: any) => {
    if (!user) return false;
    if (user.role === 'ROOT') return true;
    return isMine && user.id === item.authorId;
  };

  const handleCardClick = (item: any) => {
    onStart && onStart(item);
  };

  const handleQuickAddClick = (e: React.MouseEvent) => {
    e.stopPropagation();
    if (user?.role === 'MEDIC' || user?.role === 'ROOT') {
      onStart && onStart({ id: 'create' });
    }
  };

  const handleButtonHover = (button: ButtonType) => {
    setHoveredButton(button);
  };

  return (
    <div className="px-2">
      {isMineAndCanCreate && items.length > 0 && (
        <QuickAddButton
          onClick={handleQuickAddClick}
          onMouseEnter={() => setHoveredButton('add')}
          onMouseLeave={() => setHoveredButton(null)}
          hoveredButton={hoveredButton}
        />
      )}

      <CardGrid>
        {items.map(item => (
          <Card
            key={item.id}
            item={item}
            onEdit={onEdit}
            onDelete={onDelete}
            onClick={handleCardClick}
            canEdit={canEditItem(item)}
            showProgress={showProgress}
            completed={completed}
```

```
            hoveredCard={hoveredCard}
            hoveredButton={hoveredCard === item.id ? hoveredButton : null}
            onMouseEnter={setHoveredCard}
            onMouseLeave={() => setHoveredCard(null)}
            onButtonHover={handleButtonHover}
          />
        ))}

        {isMineAndCanCreate && (
          <CreateCard
            onClick={handleCreateClick}
            onMouseEnter={() => setHoveredButton('create')}
            onMouseLeave={() => setHoveredButton(null)}
            hoveredButton={hoveredButton}
          />
        )}
      </CardGrid>
    </div>
  );
};

export default CardList;
```

CreateCard.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';

type ButtonType = 'edit' | 'delete' | 'create' | 'add' | null;

interface CreateCardProps {
  onClick: () => void;
  onMouseEnter?: () => void;
  onMouseLeave?: () => void;
  hoveredButton: ButtonType;
}

const CreateCard: React.FC<CreateCardProps> = ({
  onClick,
  onMouseEnter,
  onMouseLeave,
  hoveredButton
}) => {
  return (
    <div
      className="relative bg-white rounded-lg border-2 border-dashed border-gray-300 h-[420px] flex
      items-center justify-center cursor-pointer hover:border-blue-500 hover:bg-blue-50 transition-
      all transform hover:translate-y-[-4px]"
      onClick={onClick}
      onMouseEnter={onMouseEnter}
      onMouseLeave={onMouseLeave}
    >
      <div className="text-center">
        <div className="w-16 h-16 rounded-full bg-blue-100 flex items-center justify-center mx-auto
      mb-3 transform hover:scale-110 transition-all">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-8 w-8 text-blue-700" fill="none"
      viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M12 6v6m0 0v6m0-
      6h6m-6 0H6" />
          </svg>
        </div>
        <p className="text-blue-700 font-medium">
          <TranslatedText text="Створити" />
        </p>
      </div>

      {hoveredButton === 'create' && (
        <div className="absolute -top-8 left-1/2 transform -translate-x-1/2 bg-gray-800 text-white
      text-xs px-2 py-1 rounded whitespace-nowrap z-10 opacity-0 animate-fadeIn">
          <TranslatedText text="Створити" />
        </div>
      )}
    </div>
  );
};

export default CreateCard;
```

QuickAddButton.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';

type ButtonType = 'edit' | 'delete' | 'create' | 'add' | null;

interface QuickAddButtonProps {
  onClick: (e: React.MouseEvent) => void;
  onMouseEnter: () => void;
  onMouseLeave: () => void;
  hoveredButton: ButtonType;
}

const QuickAddButton: React.FC<QuickAddButtonProps> = ({
  onClick,
  onMouseEnter,
  onMouseLeave,
  hoveredButton
```

```
}) => {
  return (
    <div className="flex justify-end mb-4">
      <div
        className="relative w-10 h-10 rounded-full bg-blue-500 flex items-center justify-center
      cursor-pointer hover:bg-blue-600 transform hover:scale-110 transition-all text-white shadow-
      md"
        onClick={onClick}
        onMouseEnter={onMouseEnter}
        onMouseLeave={onMouseLeave}
      >
        <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24"
      stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M12 6v6m0 0v6m0-
      6h6m-6 0H6" />
        </svg>

        {hoveredButton === 'add' && (
          <div className="absolute -top-8 right-0 bg-gray-800 text-white text-xs px-2 py-1 rounded
      whitespace-nowrap z-10 opacity-0 animate-fadeIn">
            <TranslatedText text="Створити" />
          </div>
        )}
      </div>
    </div>
  );
};

export default QuickAddButton;
```

Carousel.tsx

```
import React, { useState, useEffect, useRef } from 'react';
import Card from '../card/Card';

interface CarouselProps {
  items: any[];
  visibleCount?: number;
  loading?: boolean;
  emptyText?: string;
  onCardClick?: (item: any) => void;
}

const Carousel: React.FC<CarouselProps> = ({
  items,
  visibleCount = 4,
  loading = false,
  emptyText,
  onCardClick
}) => {
  const [start, setStart] = useState(0);
  const [count, setCount] = useState(visibleCount);
  const timerRef = useRef<NodeJS.Timeout | null>(null);

  const sm = 640;
  const md = 768;
  const lg = 1024;
  const xl = 1280;

  useEffect(() => {
    timerRef.current = setInterval(() => {
      setStart(prev => (prev + 1) % items.length);
    }, 10000);
    return () => {
      if (timerRef.current) clearInterval(timerRef.current);
    };
  }, [items.length]);

  useEffect(() => {
    function handleResize() {
      if (window.innerWidth < sm) setCount(1);
      else if (window.innerWidth < md) setCount(2);
      else if (window.innerWidth < lg) setCount(3);
      else if (window.innerWidth < xl) setCount(4);
      else setCount(visibleCount);
    }
    handleResize();
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, [visibleCount]);

  if (loading) {
    return (
      <div className="flex justify-center items-center min-h-[180px]">
        <div className="animate-spin rounded-full h-10 w-10 border-b-2 border-blue-500"></div>
      </div>
    );
  }

  if (!items.length) {
    return (
      <div className="text-center text-gray-400 py-8">
        {emptyText}
      </div>
    );
  }
```

```
  const showItems = [];
  for (let i = 0; i < Math.min(count, items.length); i++) {
    showItems.push(items[(start + i) % items.length]);
  }

  const handlePrev = () => {
    setStart(prev => (prev - 1 + items.length) % items.length);
  };
  const handleNext = () => {
    setStart(prev => (prev + 1) % items.length);
  };

  return (
    <div className="relative w-full flex items-center">
      <button
        className="absolute left-0 z-10 bg-white rounded-full shadow p-2 -translate-y-1/2 top-1/2
      disabled:opacity-30"
        onClick={handlePrev}
        disabled={items.length <= count}
        aria-label="Назад"
      >
        <svg width="24" height="24" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M15 19l-7-7 7-7"/></svg>
      </button>
      <div className="flex w-full gap-6 justify-center">
        {showItems.map((item) => (
          <div key={item.id} className="flex-1 min-w-0 max-w-xs">
            <Card
              item={item}
              onClick={onCardClick}
            />
          </div>
        ))}
      </div>
      <button
        className="absolute right-0 z-10 bg-white rounded-full shadow p-2 -translate-y-1/2 top-1/2
      disabled:opacity-30"
        onClick={handleNext}
        disabled={items.length <= count}
        aria-label="Вперед"
      >
        <svg width="24" height="24" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M9 5l7 7-7 7"/></svg>
      </button>
    </div>
  );
};

export default Carousel;
```

dateUtils.ts

```
export const isRecent = (date: string | Date | null | undefined, days: number = 7): boolean => {
  if (!date) return false;

  const now = new Date();
  const dateObj = typeof date === 'string' ? new Date(date) : date;
  const differenceInDays = (now.getTime() - dateObj.getTime()) / (1000 * 3600 * 24);

  return differenceInDays <= days;
};
```

textUtils.ts

```
\export const capitalizeFirstLetter = (text: string): string => {
  if (!text) return '';
  return text.charAt(0).toUpperCase() + text.slice(1);
};
```

LoginUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';
import LoginForm from './LoginForm';

interface LoginUIProps {
  email: string;
  password: string;
  error: string;
  loading: boolean;
  emailPlaceholder: string;
  passwordPlaceholder: string;
  onForgot: () => void;
  handleSubmit: (e: React.FormEvent) => Promise<void>;
  handleClose: () => void;
  handleEmailChange: (value: string) => void;
  handlePasswordChange: (value: string) => void;
}

const LoginUI: React.FC<LoginUIProps> = ({
  email,
  password,
  error,
  loading,
  emailPlaceholder,
  passwordPlaceholder,
```

```
  onForgot,
  handleSubmit,
  handleClose,
  handleEmailChange,
  handlePasswordChange
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <button className="absolute top-2 right-2 text-gray-400 hover:text-gray-600"
      onClick={handleClose}>&times;</button>
        <h2 className="text-xl font-bold mb-4">
          <TranslatedText text="Увійти" />
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <LoginForm
            email={email}
            password={password}
            emailPlaceholder={emailPlaceholder}
            passwordPlaceholder={passwordPlaceholder}
            loading={loading}
            onEmailChange={handleEmailChange}
            onPasswordChange={handlePasswordChange}
          />
          {error && <div className="text-red-500 text-sm">{error}</div>}
          {(error === 'Невірний пароль' || error === 'Incorrect password') && (
            <button
              type="button"
              className="text-blue-600 text-sm underline hover:text-blue-800"
              onClick={onForgot}
            >
              <TranslatedText text="Забули пароль?" />
            </button>
          )}
          <button
            type="submit"
            className="w-full border border-blue-600 text-blue-600 py-2 rounded hover:bg-blue-50
      transition flex items-center justify-center"
            disabled={loading}
          >
            {loading && <span className="spinner mr-2" style={{ width: 18, height: 18 }} />}
            <TranslatedText text="Увійти" />
          </button>
        </form>
      </div>
    </div>
  );
};

export default LoginUI;
```

LoginLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { t } from '../../TranslatedText';
import { useAuth } from '@/hooks/useAuth';

const TEMP_EMAIL_KEY = 'temp_reset_password_email';

interface LoginLogicProps {
  open: boolean;
  onClose: () => void;
  onForgot: () => void;
  setLoginEmail: (email: string) => void;
  children: (props: {
    email: string;
    password: string;
    error: string;
    loading: boolean;
    emailPlaceholder: string;
    passwordPlaceholder: string;
    handleSubmit: (e: React.FormEvent) => Promise<void>;
    handleClose: () => void;
    handleEmailChange: (value: string) => void;
    handlePasswordChange: (value: string) => void;
  }) => React.ReactNode;
}

const LoginLogic: React.FC<LoginLogicProps> = ({
  open,
  onClose,
  setLoginEmail,
  children
}) => {
  const { login } = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const [emailPlaceholder, setEmailPlaceholder] = useState('Пошта');
  const [passwordPlaceholder, setPasswordPlaceholder] = useState('Пароль');
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
```

```
    setLoading(false);

    if (open) {
      const savedEmail = localStorage.getItem(TEMP_EMAIL_KEY);
      if (savedEmail) {
        setEmail(savedEmail);
        setLoginEmail(savedEmail);
      } else {
        setEmail('');
      }
      setPassword('');
      setError('');
    }
  }, [open, setLoginEmail]);

  useEffect(() => {
    t('Пошта', lang).then(setEmailPlaceholder);
    t('Пароль', lang).then(setPasswordPlaceholder);
  }, [lang]);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const lang = localStorage.getItem('preferredLanguage') || 'ua';
      const response = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password, lang })
      });

      const data = await response.json();
      if (!response.ok) {
        setError(data.error || 'Сталася помилка. Спробуйте ще раз');
        setLoading(false);
        return;
      }

      if (data.token) {
        localStorage.setItem('token', data.token);
        login(data.token);
      }
      setLoading(false);
      handleClose();
    } catch (err) {
      t('Сталася помилка з мережею.', lang).then(setError);
      setLoading(false);
    }
  };

  const handleClose = () => {
    localStorage.removeItem(TEMP_EMAIL_KEY);
    setEmail('');
    setPassword('');
    setError('');
    setLoading(false);
    onClose();
  };

  const handleEmailChange = (value: string) => {
    setEmail(value);
    setLoginEmail(value);
  };

  const handlePasswordChange = (value: string) => {
    setPassword(value);
  };

  if (!open) return null;

  return children({
    email,
    password,
    error,
    loading,
    emailPlaceholder,
    passwordPlaceholder,
    handleSubmit,
    handleClose,
    handleEmailChange,
    handlePasswordChange
  });
};

export default LoginLogic;
```

LoginForm.tsx

```
\import React, { useState } from 'react';

interface LoginFormProps {
  email: string;
  password: string;
  emailPlaceholder: string;
  passwordPlaceholder: string;
```

```
  loading: boolean;
  onEmailChange: (value: string) => void;
  onPasswordChange: (value: string) => void;
}

const LoginForm = ({
   email,
   password,
   emailPlaceholder,
   passwordPlaceholder,
   loading,
   onEmailChange,
   onPasswordChange
}: LoginFormProps) => {
  const [showPassword, setShowPassword] = useState(false);
  return (
    <>
      <input
        type="email"
        className="w-full border rounded px-3 py-2"
        placeholder={emailPlaceholder}
        value={email}
        onChange={e => onEmailChange(e.target.value)}
        disabled={loading}
      />
      <div className="relative">
        <input
          type={showPassword ? "text" : "password"}
          className="w-full border rounded px-3 py-2 pr-10"
          placeholder={passwordPlaceholder}
          value={password}
          onChange={e => onPasswordChange(e.target.value)}
          disabled={loading}
        />
        <button
          type="button"
          className="absolute right-3 top-1/2 -translate-y-1/2 text-gray-400"
          onClick={() => setShowPassword(v => !v)}
          tabIndex={-1}
        >
          {showPassword ? (
            <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M1 12s4-7 11-7 11 7 11 7-4 7-11 7S1 12 1 12z"/><circle cx="12" cy="12" r="3"/></svg>
          ) : (
            <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M17.94 17.94A10.94 10.94 0 0 1 12 19c-7 0-11-7-11-7a21.77 21.77 0 0 1 5.06-6.06M1 1l22
      22"/><path d="M9.53 9.53A3 3 0 0 0 12 15a3 3 0 0 0 2.47-5.47"/></svg>
          )}
        </button>
      </div>
    </>
  );
};

export default LoginForm;
```

Login.tsx

```
import React from 'react';
import LoginLogic from './LoginLogic';
import LoginUI from './LoginUI';

interface LoginProps {
  open: boolean;
  onClose: () => void;
  onForgot: () => void;
  setLoginEmail: (email: string) => void;
}

const Login: React.FC<LoginProps> = ({
  open,
  onClose,
  onForgot,
  setLoginEmail
}) => {
  return (
    <LoginLogic
      open={open}
      onClose={onClose}
      onForgot={onForgot}
      setLoginEmail={setLoginEmail}
    >
      {({
        email,
        password,
        error,
        loading,
        emailPlaceholder,
        passwordPlaceholder,
        handleSubmit,
        handleClose,
        handleEmailChange,
        handlePasswordChange
      }) => (
        <LoginUI
          email={email}
          password={password}
```

```
          error={error}
          loading={loading}
          emailPlaceholder={emailPlaceholder}
          passwordPlaceholder={passwordPlaceholder}
          onForgot={onForgot}
          handleSubmit={handleSubmit}
          handleClose={handleClose}
          handleEmailChange={handleEmailChange}
          handlePasswordChange={handlePasswordChange}
        />
      )}
    </LoginLogic>
  );
};

export default Login;
```

ResetPasswordUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';
import ResetPasswordForm from './ResetPasswordForm';

interface ResetPasswordUIProps {
  codeDigits: string[];
  codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
  newPassword: string;
  loading: boolean;
  error: string;
  success: boolean;
  resendTimer: number;
  resendLoading: boolean;
  resendMessage: string;
  handleCodeInput: (idx: number, value: string) => void;
  handleCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
  handlePasswordChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleSubmit: (e: React.FormEvent) => Promise<void>;
  handleResend: () => Promise<void>;
  handleClose: () => void;
  handleBack: () => void;
  handleLogin: () => void;
}

const ResetPasswordUI: React.FC<ResetPasswordUIProps> = ({
  codeDigits,
  codeInputsRef,
  newPassword,
  loading,
  error,
  success,
  resendTimer,
  resendLoading,
  resendMessage,
  handleCodeInput,
  handleCodePaste,
  handlePasswordChange,
  handleSubmit,
  handleResend,
  handleClose,
  handleBack,
  handleLogin
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <button className="absolute top-2 right-2 text-gray-400 hover:text-gray-600"
      onClick={handleClose}>&times;</button>
        <h2 className="text-xl font-bold mb-4">
          <TranslatedText text="Зміна пароля" />
        </h2>

        {success ? (
          <div className="text-center space-y-4">
            <div className="text-green-600 text-lg font-medium">
              <TranslatedText text="Пароль успішно змінено!" />
            </div>
            <button
              className="border border-blue-600 text-blue-600 py-2 px-4 rounded hover:bg-blue-50
      transition"
              onClick={handleLogin}
            >
              <TranslatedText text="Увійти" />
            </button>
          </div>
        ) : (
          <ResetPasswordForm
            codeDigits={codeDigits}
            codeInputsRef={codeInputsRef}
            newPassword={newPassword}
            loading={loading}
            error={error}
            onCodeInput={handleCodeInput}
            onCodePaste={handleCodePaste}
            onPasswordChange={handlePasswordChange}
            onBack={handleBack}
            onSubmit={handleSubmit}
            resendTimer={resendTimer}
```

```
            resendLoading={resendLoading}
            resendMessage={resendMessage}
            onResend={handleResend}
          />
        )}
      </div>
    </div>
  );
};

export default ResetPasswordUI;
```

ResetPasswordLogic.tsx

```
import React, { useState, useRef, useEffect } from 'react';
import { t } from '../../TranslatedText';

const TEMP_EMAIL_KEY = 'temp_reset_password_email';

interface ResetPasswordLogicProps {
  open: boolean;
  email: string;
  lang?: 'ua' | 'en';
  onClose: () => void;
  onBack: () => void;
  onLogin: () => void;
  children: (props: {
    codeDigits: string[];
    codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
    newPassword: string;
    loading: boolean;
    error: string;
    success: boolean;
    resendTimer: number;
    resendLoading: boolean;
    resendMessage: string;
    handleCodeInput: (idx: number, value: string) => void;
    handleCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
    handlePasswordChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleSubmit: (e: React.FormEvent) => Promise<void>;
    handleResend: () => Promise<void>;
    handleClose: () => void;
    handleBack: () => void;
    handleLogin: () => void;
  }) => React.ReactNode;
}

const ResetPasswordLogic: React.FC<ResetPasswordLogicProps> = ({
  open,
  email,
  lang: initialLang = 'ua',
  onClose,
  onBack,
  onLogin,
  children
}) => {
  const [codeDigits, setCodeDigits] = useState(['', '', '', '', '', '']);
  const [newPassword, setNewPassword] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const [lang, setLang] = useState<'ua' | 'en'>(initialLang);
  const codeInputsRef = useRef<(HTMLInputElement | null)[]>([]);
  const [success, setSuccess] = useState(false);
  const [resendTimer, setResendTimer] = useState(0);
  const [resendLoading, setResendLoading] = useState(false);
  const [resendMessage, setResendMessage] = useState('');

  useEffect(() => {
    if (open && email) {
      localStorage.setItem(TEMP_EMAIL_KEY, email);
    }
  }, [open, email]);

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
    setCodeDigits(['', '', '', '', '', '']);
    setNewPassword('');
    setLoading(false);
    setError('');
    setSuccess(false);
    setResendTimer(0);
    setResendLoading(false);
    setResendMessage('');
  }, [open]);

  useEffect(() => {
    if (resendTimer > 0) {
      const timer = setTimeout(() => setResendTimer(resendTimer - 1), 1000);
      return () => clearTimeout(timer);
    }
  }, [resendTimer]);

  if (!open) return null;

  const handleCodeInput = (idx: number, value: string) => {
```

```
  if (!/^[0-9]?$/.test(value)) return;
  const newDigits = [...codeDigits];
  newDigits[idx] = value;
  setCodeDigits(newDigits);

  if (value && idx < 5) {
    codeInputsRef.current[idx + 1]?.focus();
  }

  if (!value && idx > 0) {
    codeInputsRef.current[idx - 1]?.focus();
  }
  setError('');
};

const handleCodePaste = (e: React.ClipboardEvent<HTMLInputElement>) => {
  const paste = e.clipboardData.getData('text').replace(/\D/g, '').slice(0, 6);

  if (paste.length === 6) {
    setCodeDigits(paste.split(''));
    codeInputsRef.current[5]?.focus();
  }
  setError('');
};

const handlePasswordChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  setNewPassword(e.target.value);
  setError('');
};

const handleResend = async () => {
  setResendLoading(true);
  setResendMessage('');

  try {
    const response = await fetch('/api/password/forgot', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, lang })
    });
    const data = await response.json();
    if (!response.ok) {
      t(data.error || 'Сталася помилка. Спробуйте ще раз.', lang).then(setResendMessage);
      setResendLoading(false);
      return;
    }
    t('Код повторно надіслано', lang).then(setResendMessage);
    setResendTimer(60);
    setResendLoading(false);
  } catch (err) {
    t('Сталася помилка з мережею.', lang).then(setResendMessage);
    setResendLoading(false);
  }
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setError('');

  if (codeDigits.some(d => !d) || !newPassword) {
    setError(lang === 'en' ? 'All fields are required' : 'Усі поля обов’язкові');
    return;
  }
  setLoading(true);

  try {
    const lang = localStorage.getItem('preferredLanguage') || 'ua';
    const code = codeDigits.join('');
    const response = await fetch('/api/password/reset', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, code, newPassword, lang })
    });
    const data = await response.json();
    if (!response.ok) {
      setError(data.error || 'Сталася помилка. Спробуйте ще раз');
      setLoading(false);
      return;
    }
    setLoading(false);
    setSuccess(true);

    localStorage.removeItem(TEMP_EMAIL_KEY);
  } catch (err) {
    setError('Сталася помилка з мережею');
    setLoading(false);
  }
};

const handleClose = () => {
  localStorage.removeItem(TEMP_EMAIL_KEY);
  onClose();
};

const handleBack = () => {
  onBack();
};
```

```
  const handleLogin = () => {
    localStorage.removeItem(TEMP_EMAIL_KEY);
    onLogin();
  };

  return children({
    codeDigits,
    codeInputsRef,
    newPassword,
    loading,
    error,
    success,
    resendTimer,
    resendLoading,
    resendMessage,
    handleCodeInput,
    handleCodePaste,
    handlePasswordChange,
    handleSubmit,
    handleResend,
    handleClose,
    handleBack,
    handleLogin
  });
};

export default ResetPasswordLogic;
```

ResetPasswordForm.tsx

```
import React, { useState } from 'react';
import { TranslatedText } from '../../TranslatedText';

interface ResetPasswordFormProps {
  codeDigits: string[];
  codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
  newPassword: string;
  loading: boolean;
  error: string;
  onCodeInput: (idx: number, value: string) => void;
  onCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
  onPasswordChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  onBack: () => void;
  onSubmit: (e: React.FormEvent) => void;
  resendTimer: number;
  resendLoading: boolean;
  resendMessage: string;
  onResend: () => void;
}

const ResetPasswordForm = ({
  codeDigits,
  codeInputsRef,
  newPassword,
  loading,
  error,
  onCodeInput,
  onCodePaste,
  onPasswordChange,
  onBack,
  onSubmit,
  resendTimer,
  resendLoading,
  resendMessage,
  onResend
}: ResetPasswordFormProps) => {
  const [showPassword, setShowPassword] = useState(false);
  return (
    <form onSubmit={onSubmit} className="space-y-4">
      <div>
        <label className="block text-sm font-medium mb-1">
          <TranslatedText text="Введіть код з пошти" />
        </label>
        <div className="flex space-x-2 justify-center mb-2">
          {codeDigits.map((digit, idx) => (
            <input
              key={idx}
              ref={el => { codeInputsRef.current[idx] = el; }}
              type="text"
              inputMode="numeric"
              maxLength={1}
              className="w-10 h-12 text-center border rounded text-xl font-mono focus:ring-2
     focus:ring-blue-400"
              value={digit}
              onChange={e => onCodeInput(idx, e.target.value)}
              onPaste={idx === 0 ? onCodePaste : undefined}
              onKeyDown={e => {
                if (e.key === 'Backspace' && !codeDigits[idx] && idx > 0) {
                  codeInputsRef.current[idx - 1]?.focus();
                }
              }}
            />
          ))}
        </div>
        <button
          type="button"
```

```
          className={`w-full border border-blue-600 text-blue-600 py-2 rounded hover:bg-blue-50
      transition flex items-center justify-center mb-2 ${resendTimer > 0 ? 'bg-gray-100 text-gray-
      400 border-gray-300 cursor-not-allowed' : ''}`}
          onClick={onResend}
          disabled={resendTimer > 0 || resendLoading}
        >
          {resendTimer > 0 ? (
            <span><TranslatedText text="Надіслати ще раз" /> ({resendTimer})</span>
          ) : (
            <TranslatedText text="Надіслати ще раз" />
          )}
        </button>
        {resendMessage && <div className="text-green-600 text-sm text-center mb-
      1">{resendMessage}</div>}
        <label className="block text-sm font-medium mb-1 mt-2">
          <TranslatedText text="Новий пароль" />
        </label>
        <div className="relative">
          <input
            type={showPassword ? "text" : "password"}
            className="w-full border rounded px-3 py-2 pr-10"
            value={newPassword}
            onChange={onPasswordChange}
          />
          <button
            type="button"
            className="absolute right-3 top-1/2 -translate-y-1/2 text-gray-400"
            onClick={() => setShowPassword(v => !v)}
            tabIndex={-1}
          >
            {showPassword ? (
              <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24
      24"><path d="M1 12s4-7 11-7 11 7 11 7-4 7-11 7S1 12 1 12z"/><circle cx="12" cy="12"
      r="3"/></svg>
            ) : (
              <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24
      24"><path d="M17.94 17.94A10.94 10.94 0 0 1 12 19c-7 0-11-7-11-7a21.77 21.77 0 0 1 5.06-6.06M1
      1l22 22"/><path d="M9.53 9.53A3 3 0 0 0 12 15a3 3 0 0 0 2.47-5.47"/></svg>
            )}
          </button>
        </div>
      </div>
      {error && <div className="text-red-500 text-sm">{error}</div>}
      <div className="flex justify-between items-center">
        <button type="button" className="text-gray-500 text-sm underline" onClick={onBack}>
          <TranslatedText text="Назад" />
        </button>
        <button type="submit" className="border border-blue-600 text-blue-600 py-2 px-4 rounded
      hover:bg-blue-50 transition flex items-center justify-center" disabled={loading}>
          {loading && <span className="spinner mr-2" style={{ width: 18, height: 18 }} />}
          <TranslatedText text="Змінити пароль" />
        </button>
      </div>
    </form>
  );
};

export default ResetPasswordForm;
```

ResetPassword.tsx

```
import React from 'react';
import ResetPasswordLogic from './ResetPasswordLogic';
import ResetPasswordUI from './ResetPasswordUI';

interface ResetPasswordProps {
  open: boolean;
  email: string;
  lang?: 'ua' | 'en';
  onClose: () => void;
  onBack: () => void;
  onLogin: () => void;
}

const ResetPassword: React.FC<ResetPasswordProps> = ({
  open,
  email,
  lang,
  onClose,
  onBack,
  onLogin
}) => {
  return (
    <ResetPasswordLogic
      open={open}
      email={email}
      lang={lang}
      onClose={onClose}
      onBack={onBack}
      onLogin={onLogin}
    >
      {({
        codeDigits,
        codeInputsRef,
        newPassword,
        loading,
        error,
```

```
        success,
        resendTimer,
        resendLoading,
        resendMessage,
        handleCodeInput,
        handleCodePaste,
        handlePasswordChange,
        handleSubmit,
        handleResend,
        handleClose,
        handleBack,
        handleLogin
      }) => (
        <ResetPasswordUI
          codeDigits={codeDigits}
          codeInputsRef={codeInputsRef}
          newPassword={newPassword}
          loading={loading}
          error={error}
          success={success}
          resendTimer={resendTimer}
          resendLoading={resendLoading}
          resendMessage={resendMessage}
          handleCodeInput={handleCodeInput}
          handleCodePaste={handleCodePaste}
          handlePasswordChange={handlePasswordChange}
          handleSubmit={handleSubmit}
          handleResend={handleResend}
          handleClose={handleClose}
          handleBack={handleBack}
          handleLogin={handleLogin}
        />
      )}
    </ResetPasswordLogic>
  );
};

export default ResetPassword;
```

ForgotPasswordUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';
import ForgotPasswordForm from './ForgotPasswordForm';

interface ForgotPasswordUIProps {
  email: string;
  loading: boolean;
  message: { text: string; type: 'success' | 'error' } | null;
  handleSend: (e: React.FormEvent) => Promise<void>;
  handleClose: () => void;
  handleBack: () => void;
}

const ForgotPasswordUI: React.FC<ForgotPasswordUIProps> = ({
  email,
  loading,
  message,
  handleSend,
  handleClose,
  handleBack
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <button className="absolute top-2 right-2 text-gray-400 hover:text-gray-600"
      onClick={handleClose}>&times;</button>
        <h2 className="text-xl font-bold mb-4">
          <TranslatedText text="Відновлення пароля" />
        </h2>
        <ForgotPasswordForm
          email={email}
          loading={loading}
          message={message}
          onSend={handleSend}
          onBack={handleBack}
        />
      </div>
    </div>
  );
};

export default ForgotPasswordUI;
```

ForgotPasswordLogic.tsx

```
import React, { useState, useEffect } from 'react';

const TEMP_EMAIL_KEY = 'temp_reset_password_email';

interface ForgotPasswordLogicProps {
  open: boolean;
  email: string;
  lang?: 'ua' | 'en';
  onClose: () => void;
  onReset: () => void;
  onBack: () => void;
```

```
  children: (props: {
    loading: boolean;
    message: { text: string; type: 'success' | 'error' } | null;
    handleSend: (e: React.FormEvent) => Promise<void>;
    handleClose: () => void;
    handleBack: () => void;
  }) => React.ReactNode;
}

const ForgotPasswordLogic: React.FC<ForgotPasswordLogicProps> = ({
  open,
  email,
  lang: initialLang = 'ua',
  onClose,
  onReset,
  onBack,
  children
}) => {
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState<{ text: string; type: 'success' | 'error' } | null>(null);
  const [lang, setLang] = useState<'ua' | 'en'>(initialLang);

  useEffect(() => {
    if (open && email) {
      localStorage.setItem(TEMP_EMAIL_KEY, email);
    }
  }, [open, email]);

  useEffect(() => {
    if (!open) return;

    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
    setLoading(false);
    setMessage(null);
  }, [open]);

  if (!open) return null;

  const handleSend = async (e: React.FormEvent) => {
    e.preventDefault();
    setMessage(null);
    setLoading(true);

    try {
      const response = await fetch('/api/password/forgot', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, lang })
      });
      const data = await response.json();
      if (!response.ok) {
        setMessage({ text: data.error, type: 'error' });
        setLoading(false);
        return;
      }
      setMessage({ text: data.message, type: 'success' });
      setLoading(false);
      setTimeout(onReset, 1000);
    } catch (err) {
      setMessage({ text: 'network Error', type: 'error' });
      setLoading(false);
    }
  };

  const handleClose = () => {
    localStorage.removeItem(TEMP_EMAIL_KEY);
    onClose();
  };

  const handleBack = () => {
    onBack();
  };

  return children({
    loading,
    message,
    handleSend,
    handleClose,
    handleBack
  });
};

export default ForgotPasswordLogic;
```

ForgotPasswordForm.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';

interface ForgotPasswordFormProps {
  email: string;
  loading: boolean;
  message: { text: string; type: 'success' | 'error' } | null;
  onSend: (e: React.FormEvent) => void;
  onBack: () => void;
```

```
}

const ForgotPasswordForm: React.FC<ForgotPasswordFormProps> = ({
  email,
  loading,
  message,
  onSend,
  onBack
}) => {
  return (
    <form onSubmit={onSend} className="space-y-4">
      <div>
        <div className="mb-2">
          <TranslatedText text="Відправити код на пошту" /> <b>{email}</b>
        </div>
      </div>
      {message && (
        <div className={`text-sm ${message.type === 'success' ? 'text-blue-600' : 'text-red-
      500'}`}>{message.text}</div>
      )}
      <div className="flex justify-between items-center">
        <button type="button" className="text-gray-500 text-sm underline" onClick={onBack}>
          <TranslatedText text="Назад" />
        </button>
        <button type="submit" className="border border-blue-600 text-blue-600 py-2 px-4 rounded
      hover:bg-blue-50 transition flex items-center justify-center" disabled={loading}>
          {loading && <span className="spinner mr-2" style={{ width: 18, height: 18 }} />}
          <TranslatedText text="Відправити код" />
        </button>
      </div>
    </form>
  );
};

export default ForgotPasswordForm;
```

ForgotPassword.tsx

```
import React from 'react';
import ForgotPasswordLogic from './ForgotPasswordLogic';
import ForgotPasswordUI from './ForgotPasswordUI';

interface ForgotPasswordProps {
  open: boolean;
  email: string;
  lang?: 'ua' | 'en';
  onClose: () => void;
  onReset: () => void;
  onBack: () => void;
}

const ForgotPassword: React.FC<ForgotPasswordProps> = ({
  open,
  email,
  lang,
  onClose,
  onReset,
  onBack
}) => {
  return (
    <ForgotPasswordLogic
      open={open}
      email={email}
      lang={lang}
      onClose={onClose}
      onReset={onReset}
      onBack={onBack}
    >
      {({
        loading,
        message,
        handleSend,
        handleClose,
        handleBack
      }) => (
        <ForgotPasswordUI
          email={email}
          loading={loading}
          message={message}
          handleSend={handleSend}
          handleClose={handleClose}
          handleBack={handleBack}
        />
      )}
    </ForgotPasswordLogic>
  );
};

export default ForgotPassword;
```

RegisterUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';
import RegisterForm from './RegisterForm';

interface RegisterUIProps {
```

```
  name: string;
  email: string;
  password: string;
  error: string;
  loading: boolean;
  namePlaceholder: string;
  emailPlaceholder: string;
  passwordPlaceholder: string;
  handleSubmit: (e: React.FormEvent) => Promise<void>;
  handleNameChange: (value: string) => void;
  handleEmailChange: (value: string) => void;
  handlePasswordChange: (value: string) => void;
  handleClose: () => void;
}

const RegisterUI: React.FC<RegisterUIProps> = ({
  name,
  email,
  password,
  error,
  loading,
  namePlaceholder,
  emailPlaceholder,
  passwordPlaceholder,
  handleSubmit,
  handleNameChange,
  handleEmailChange,
  handlePasswordChange,
  handleClose
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <button className="absolute top-2 right-2 text-gray-400 hover:text-gray-600"
     onClick={handleClose}>&times;</button>
        <h2 className="text-xl font-bold mb-4">
          <TranslatedText text="Реєстрація" />
        </h2>
        <form onSubmit={handleSubmit} className="space-y-4">
          <RegisterForm
            name={name}
            email={email}
            password={password}
            namePlaceholder={namePlaceholder}
            emailPlaceholder={emailPlaceholder}
            passwordPlaceholder={passwordPlaceholder}
            loading={loading}
            onNameChange={handleNameChange}
            onEmailChange={handleEmailChange}
            onPasswordChange={handlePasswordChange}
          />
          {error && <div className="text-red-500 text-sm">{error}</div>}
          <button
            type="submit"
            className="w-full border border-blue-600 text-blue-600 py-2 rounded hover:bg-blue-50
     transition flex items-center justify-center"
            disabled={loading}
          >
            {loading && <span className="spinner mr-2" style={{ width: 18, height: 18 }} />}
            <TranslatedText text="Зареєструватися" />
          </button>
        </form>
      </div>
    </div>
  );
};

export default RegisterUI;
```

RegisterLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { t } from '../../TranslatedText';

const TEMP_REGISTER_EMAIL_KEY = 'temp_register_email';
const TEMP_REGISTER_NAME_KEY = 'temp_register_name';

interface RegisterLogicProps {
  open: boolean;
  onClose: () => void;
  onSuccess: (email: string, name: string, password: string) => void;
  children: (props: {
    name: string;
    email: string;
    password: string;
    error: string;
    loading: boolean;
    namePlaceholder: string;
    emailPlaceholder: string;
    passwordPlaceholder: string;
    handleSubmit: (e: React.FormEvent) => Promise<void>;
    handleNameChange: (value: string) => void;
    handleEmailChange: (value: string) => void;
    handlePasswordChange: (value: string) => void;
    handleClose: () => void;
  }) => React.ReactNode;
}
```

```
const RegisterLogic: React.FC<RegisterLogicProps> = ({
  open,
  onClose,
  onSuccess,
  children
}) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const [namePlaceholder, setNamePlaceholder] = useState("Прізвище Ім'я");
  const [emailPlaceholder, setEmailPlaceholder] = useState("Пошта");
  const [passwordPlaceholder, setPasswordPlaceholder] = useState("Пароль");
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
    setLoading(false);

    if (open) {
      const savedEmail = localStorage.getItem(TEMP_REGISTER_EMAIL_KEY);
      const savedName = localStorage.getItem(TEMP_REGISTER_NAME_KEY);

      if (savedEmail) {
        setEmail(savedEmail);
      } else {
        setEmail('');
      }

      if (savedName) {
        setName(savedName);
      } else {
        setName('');
      }

      setPassword('');
      setError('');
    }
  }, [open]);

  useEffect(() => {
    t("Прізвище Ім'я", lang).then(setNamePlaceholder);
    t("Пошта", lang).then(setEmailPlaceholder);
    t("Пароль", lang).then(setPasswordPlaceholder);
  }, [lang]);

  if (!open) return null;

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const lang = localStorage.getItem('preferredLanguage') || 'ua';
      const response = await fetch('/api/register/request', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ name, email, password, lang })
      });
      const data = await response.json();
      if (!response.ok) {
        setError(data.error || 'Сталася помилка. Спробуйте ще раз');
        setLoading(false);
        return;
      }
      setLoading(false);
      onSuccess(email, name, password);
    } catch (err) {
      t('Сталася помилка з мережею.', lang).then(setError);
      setLoading(false);
    }
  };

  const handleNameChange = (value: string) => {
    setName(value);
  };

  const handleEmailChange = (value: string) => {
    setEmail(value);
  };

  const handlePasswordChange = (value: string) => {
    setPassword(value);
  };

  const handleClose = () => {
    localStorage.removeItem(TEMP_REGISTER_EMAIL_KEY);
    localStorage.removeItem(TEMP_REGISTER_NAME_KEY);
    onClose();
  };

  return children({
```

```
    name,
    email,
    password,
    error,
    loading,
    namePlaceholder,
    emailPlaceholder,
    passwordPlaceholder,
    handleSubmit,
    handleNameChange,
    handleEmailChange,
    handlePasswordChange,
    handleClose
  });
};

export default RegisterLogic;
```

RegisterForm.tsx

```
import React, { useState } from 'react';

interface RegisterFormProps {
  name: string;
  email: string;
  password: string;
  namePlaceholder: string;
  emailPlaceholder: string;
  passwordPlaceholder: string;
  loading: boolean;
  onNameChange: (value: string) => void;
  onEmailChange: (value: string) => void;
  onPasswordChange: (value: string) => void;
}

const RegisterForm = ({
  name,
  email,
  password,
  namePlaceholder,
  emailPlaceholder,
  passwordPlaceholder,
  loading,
  onNameChange,
  onEmailChange,
  onPasswordChange,
}: RegisterFormProps) => {
  const [showPassword, setShowPassword] = useState(false);
  return (
    <>
      <input
        type="text"
        className="w-full border rounded px-3 py-2"
        placeholder={namePlaceholder}
        value={name}
        onChange={e => onNameChange(e.target.value)}
        disabled={loading}
      />
      <input
        type="email"
        className="w-full border rounded px-3 py-2"
        placeholder={emailPlaceholder}
        value={email}
        onChange={e => onEmailChange(e.target.value)}
        disabled={loading}
      />
      <div className="relative">
        <input
          type={showPassword ? "text" : "password"}
          className="w-full border rounded px-3 py-2 pr-10"
          placeholder={passwordPlaceholder}
          value={password}
          onChange={e => onPasswordChange(e.target.value)}
          disabled={loading}
        />
        <button
          type="button"
          className="absolute right-3 top-1/2 -translate-y-1/2 text-gray-400"
          onClick={() => setShowPassword(v => !v)}
          tabIndex={-1}
        >
          {showPassword ? (
            <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M1 12s4-7 11-7 11 7 11 7-4 7-11 7S1 12 1 12z"/><circle cx="12" cy="12" r="3"/></svg>
          ) : (
            <svg width="20" height="20" fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
      d="M17.94 17.94A10.94 10.94 0 0 1 12 19c-7 0-11-7-11-7a21.77 21.77 0 0 1 5.06-6.06M1 1l22
      22"/><path d="M9.53 9.53A3 3 0 0 0 12 15a3 3 0 0 0 2.47-5.47"/></svg>
          )}
        </button>
      </div>
    </>
  );
};

export default RegisterForm;
```

Register.tsx

```
import React from 'react';
import RegisterLogic from './RegisterLogic';
import RegisterUI from './RegisterUI';

interface RegisterProps {
  open: boolean;
  onClose: () => void;
  onSuccess: (email: string, name: string, password: string) => void;
}

const Register: React.FC<RegisterProps> = ({
  open,
  onClose,
  onSuccess
}) => {
  return (
    <RegisterLogic
      open={open}
      onClose={onClose}
      onSuccess={onSuccess}
    >
      {({
        name,
        email,
        password,
        error,
        loading,
        namePlaceholder,
        emailPlaceholder,
        passwordPlaceholder,
        handleSubmit,
        handleNameChange,
        handleEmailChange,
        handlePasswordChange,
        handleClose
      }) => (
        <RegisterUI
          name={name}
          email={email}
          password={password}
          error={error}
          loading={loading}
          namePlaceholder={namePlaceholder}
          emailPlaceholder={emailPlaceholder}
          passwordPlaceholder={passwordPlaceholder}
          handleSubmit={handleSubmit}
          handleNameChange={handleNameChange}
          handleEmailChange={handleEmailChange}
          handlePasswordChange={handlePasswordChange}
          handleClose={handleClose}
        />
      )}
    </RegisterLogic>
  );
};

export default Register;
```

ConfirmCodeUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';
import ConfirmCodeForm from './ConfirmCodeForm';

interface ConfirmCodeUIProps {
  codeDigits: string[];
  codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
  loading: boolean;
  error: string;
  resendTimer: number;
  resendLoading: boolean;
  resendMessage: string;
  handleCodeInput: (idx: number, value: string) => void;
  handleCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
  handleSubmit: (e: React.FormEvent) => Promise<void>;
  handleResend: () => Promise<void>;
  handleBack: () => void;
  handleClose: () => void;
}

const ConfirmCodeUI: React.FC<ConfirmCodeUIProps> = ({
  codeDigits,
  codeInputsRef,
  loading,
  error,
  resendTimer,
  resendLoading,
  resendMessage,
  handleCodeInput,
  handleCodePaste,
  handleSubmit,
  handleResend,
  handleBack,
  handleClose
}) => {
```

```
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <button className="absolute top-2 right-2 text-gray-400 hover:text-gray-600"
     onClick={handleClose}>&times;</button>
        <h2 className="text-xl font-bold mb-4">
          <TranslatedText text="Підтвердження пошти" />
        </h2>
        <ConfirmCodeForm
          codeDigits={codeDigits}
          codeInputsRef={codeInputsRef}
          loading={loading}
          error={error}
          resendTimer={resendTimer}
          resendLoading={resendLoading}
          resendMessage={resendMessage}
          onCodeInput={handleCodeInput}
          onCodePaste={handleCodePaste}
          onBack={handleBack}
          onSubmit={handleSubmit}
          onResend={handleResend}
        />
      </div>
    </div>
  );
};

export default ConfirmCodeUI;
```

ConfirmCodeLogic.tsx

```
import React, { useRef, useState, useEffect } from 'react';
import { t } from '../../TranslatedText';

const TEMP_REGISTER_EMAIL_KEY = 'temp_register_email';
const TEMP_REGISTER_NAME_KEY = 'temp_register_name';

interface ConfirmCodeLogicProps {
  open: boolean;
  email: string;
  name: string;
  password: string;
  lang?: 'ua' | 'en';
  onBack: () => void;
  onClose: () => void;
  onSuccess: () => void;
  children: (props: {
    codeDigits: string[];
    codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
    loading: boolean;
    error: string;
    resendTimer: number;
    resendLoading: boolean;
    resendMessage: string;
    handleCodeInput: (idx: number, value: string) => void;
    handleCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
    handleSubmit: (e: React.FormEvent) => Promise<void>;
    handleResend: () => Promise<void>;
    handleBack: () => void;
    handleClose: () => void;
  }) => React.ReactNode;
}

const ConfirmCodeLogic: React.FC<ConfirmCodeLogicProps> = ({
  open,
  email,
  name,
  password,
  lang: initialLang = 'ua',
  onBack,
  onClose,
  onSuccess,
  children
}) => {
  const [code, setCode] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const [resendTimer, setResendTimer] = useState(0);
  const [resendLoading, setResendLoading] = useState(false);
  const [resendMessage, setResendMessage] = useState('');
  const [lang, setLang] = useState<'ua' | 'en'>(initialLang);
  const codeInputsRef = useRef<(HTMLInputElement | null)[]>([]);

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
    setCode('');
    setLoading(false);
    setError('');
    setResendTimer(0);
    setResendLoading(false);
    setResendMessage('');

    if (open && email && name) {
      localStorage.setItem(TEMP_REGISTER_EMAIL_KEY, email);
      localStorage.setItem(TEMP_REGISTER_NAME_KEY, name);
```

```
  }
}, [open, email, name]);

useEffect(() => {
  if (resendTimer > 0) {
    const timer = setTimeout(() => setResendTimer(resendTimer - 1), 1000);
    return () => clearTimeout(timer);
  }
}, [resendTimer]);

if (!open) return null;

const handleCodeInput = (idx: number, value: string) => {
  if (!/^[0-9]?$/.test(value)) return;
  const newDigits = code.split('').concat(Array(6 - code.length).fill('')).slice(0, 6);
  newDigits[idx] = value;
  const joined = newDigits.join('').replace(/\D/g, '');
  setCode(joined);

  if (value && idx < 5) {
    codeInputsRef.current[idx + 1]?.focus();
  }

  if (!value && idx > 0) {
    codeInputsRef.current[idx - 1]?.focus();
  }
  setError('');
};

const handleCodePaste = (e: React.ClipboardEvent<HTMLInputElement>) => {
  const paste = e.clipboardData.getData('text').replace(/\D/g, '').slice(0, 6);

  if (paste.length) {
    setCode(paste);
    if (paste.length < 6) {
      codeInputsRef.current[paste.length]?.focus();
    } else {
      codeInputsRef.current[5]?.focus();
    }
  }
  setError('');
};

const handleResend = async () => {
  setResendLoading(true);
  setResendMessage('');

  try {
    const response = await fetch('/api/register/request', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ name, email, password, lang })
    });
    const data = await response.json();
    if (!response.ok) {
      setResendMessage(data.error || 'Сталася помилка. Спробуйте ще раз.');
      setResendLoading(false);
      return;
    }
    t('Код повторно надіслано', lang).then(setResendMessage);
    setResendTimer(60);
    setResendLoading(false);
  } catch (err) {
    t('Сталася помилка з мережею.', lang).then(setResendMessage);
    setResendLoading(false);
  }
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setError('');

  if (code.length < 6) {
    setError(lang === 'en' ? 'All fields are required' : 'Усі поля обов’язкові');
    return;
  }
  setLoading(true);

  try {
    const response = await fetch('/api/register/confirm', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, code, lang })
    });
    const data = await response.json();
    if (!response.ok) {
      setError(data.error || 'Сталася помилка. Спробуйте ще раз');
      setLoading(false);
      return;
    }
    setLoading(false);
    localStorage.removeItem(TEMP_REGISTER_EMAIL_KEY);
    localStorage.removeItem(TEMP_REGISTER_NAME_KEY);

    onSuccess();
  } catch (err) {
    setError('Сталася помилка з мережею');
```

```
      setLoading(false);
    }
  };

  const handleBack = () => {
    onBack();
  };

  const handleClose = () => {
    localStorage.removeItem(TEMP_REGISTER_EMAIL_KEY);
    localStorage.removeItem(TEMP_REGISTER_NAME_KEY);
    onClose();
  };

  const codeDigits = code.split('').concat(Array(6 - code.length).fill('')).slice(0, 6);

  return children({
    codeDigits,
    codeInputsRef,
    loading,
    error,
    resendTimer,
    resendLoading,
    resendMessage,
    handleCodeInput,
    handleCodePaste,
    handleSubmit,
    handleResend,
    handleBack,
    handleClose
  });
};

export default ConfirmCodeLogic;
```

ConfirmCodeForm.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';

interface ConfirmCodeFormProps {
  codeDigits: string[];
  codeInputsRef: React.MutableRefObject<(HTMLInputElement | null)[]>;
  loading: boolean;
  error: string;
  resendTimer: number;
  resendLoading: boolean;
  resendMessage: string;
  onCodeInput: (idx: number, value: string) => void;
  onCodePaste: (e: React.ClipboardEvent<HTMLInputElement>) => void;
  onBack: () => void;
  onSubmit: (e: React.FormEvent) => void;
  onResend: () => void;
}

const ConfirmCodeForm: React.FC<ConfirmCodeFormProps> = ({
  codeDigits,
  codeInputsRef,
  loading,
  error,
  resendTimer,
  resendLoading,
  resendMessage,
  onCodeInput,
  onCodePaste,
  onBack,
  onSubmit,
  onResend
}) => {
  return (
    <form onSubmit={onSubmit} className="space-y-4">
      <div>
        <label className="block text-sm font-medium mb-1">
          <TranslatedText text="Введіть код з пошти" />
        </label>
        <div className="flex space-x-2 justify-center mb-2">
          {codeDigits.map((digit, idx) => (
            <input
              key={idx}
              ref={el => { codeInputsRef.current[idx] = el; }}
              type="text"
              inputMode="numeric"
              maxLength={1}
              className="w-10 h-12 text-center border rounded text-xl font-mono focus:ring-2
     focus:ring-blue-400"
              value={digit}
              onChange={e => onCodeInput(idx, e.target.value)}
              onPaste={idx === 0 ? onCodePaste : undefined}
              onKeyDown={e => {
                if (e.key === 'Backspace' && !codeDigits[idx] && idx > 0) {
                  codeInputsRef.current[idx - 1]?.focus();
                }
              }}
              autoFocus={idx === 0}
              disabled={loading}
            />
          ))}
```

```
        </div>
        <button
          type="button"
          className={`w-full border border-blue-600 text-blue-600 py-2 rounded hover:bg-blue-50
      transition flex items-center justify-center mb-2 ${resendTimer > 0 ? 'bg-gray-100 text-gray-
      400 border-gray-300 cursor-not-allowed' : ''}`}
          onClick={onResend}
          disabled={resendTimer > 0 || resendLoading}
        >
          {resendTimer > 0 ? (
            <span><TranslatedText text="Надіслати ще раз" /> ({resendTimer})</span>
          ) : (
            <TranslatedText text="Надіслати ще раз" />
          )}
        </button>
        {resendMessage && <div className="text-green-600 text-sm text-center mb-
      1">{resendMessage}</div>}
      </div>
      {error && <div className="text-red-500 text-sm">{error}</div>}
      <div className="flex justify-between items-center">
        <button type="button" className="text-gray-500 text-sm underline" onClick={onBack}>
          <TranslatedText text="Назад" />
        </button>
        <button type="submit" className="border border-blue-600 text-blue-600 py-2 px-4 rounded
      hover:bg-blue-50 transition flex items-center justify-center" disabled={loading}>
          {loading && <span className="spinner mr-2" style={{ width: 18, height: 18 }} />}
          <TranslatedText text="Підтвердити" />
        </button>
      </div>
    </form>
  );
};

export default ConfirmCodeForm;
```

ConfirmCode.tsx

```
import React from 'react';
import ConfirmCodeLogic from './ConfirmCodeLogic';
import ConfirmCodeUI from './ConfirmCodeUI';

interface ConfirmCodeProps {
  open: boolean;
  email: string;
  name: string;
  password: string;
  lang?: 'ua' | 'en';
  onBack: () => void;
  onClose: () => void;
  onSuccess: () => void;
}

const ConfirmCode: React.FC<ConfirmCodeProps> = ({
  open,
  email,
  name,
  password,
  lang,
  onBack,
  onClose,
  onSuccess
}) => {
  return (
    <ConfirmCodeLogic
      open={open}
      email={email}
      name={name}
      password={password}
      lang={lang}
      onBack={onBack}
      onClose={onClose}
      onSuccess={onSuccess}
    >
      {({
        codeDigits,
        codeInputsRef,
        loading,
        error,
        resendTimer,
        resendLoading,
        resendMessage,
        handleCodeInput,
        handleCodePaste,
        handleSubmit,
        handleResend,
        handleBack,
        handleClose
      }) => (
        <ConfirmCodeUI
          codeDigits={codeDigits}
          codeInputsRef={codeInputsRef}
          loading={loading}
          error={error}
          resendTimer={resendTimer}
          resendLoading={resendLoading}
          resendMessage={resendMessage}
          handleCodeInput={handleCodeInput}
```

```
          handleCodePaste={handleCodePaste}
          handleSubmit={handleSubmit}
          handleResend={handleResend}
          handleBack={handleBack}
          handleClose={handleClose}
        />
      )}
    </ConfirmCodeLogic>
  );
};

export default ConfirmCode;
```

RegisterSuccessUI.tsx

```
import React from 'react';
import { TranslatedText } from '../../TranslatedText';

interface RegisterSuccessUIProps {
  handleLogin: () => void;
}

const RegisterSuccessUI: React.FC<RegisterSuccessUIProps> = ({
  handleLogin
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
      <div className="bg-white rounded-lg shadow-lg p-6 w-full max-w-sm relative">
        <h2 className="text-xl font-bold mb-4 text-center">
          <TranslatedText text="Реєстрація успішна!" />
        </h2>
        <div className="text-green-600 text-center font-medium mb-4">
          <TranslatedText text="Тепер ви можете увійти до свого акаунта" />
        </div>
        <button className="w-full border border-blue-600 text-blue-600 py-2 rounded hover:bg-blue-50
     transition" onClick={handleLogin}>
          <TranslatedText text="Увійти" />
        </button>
      </div>
    </div>
  );
};

export default RegisterSuccessUI;
```

RegisterSuccessLogic.tsx

```
import React, { useEffect } from 'react';

const TEMP_REGISTER_EMAIL_KEY = 'temp_register_email';
const TEMP_REGISTER_NAME_KEY = 'temp_register_name';

interface RegisterSuccessLogicProps {
  open: boolean;
  onLogin: () => void;
  children: (props: {
    handleLogin: () => void;
  }) => React.ReactNode;
}

const RegisterSuccessLogic: React.FC<RegisterSuccessLogicProps> = ({
  open,
  onLogin,
  children
}) => {
  useEffect(() => {
    if (open) {
      localStorage.removeItem(TEMP_REGISTER_EMAIL_KEY);
      localStorage.removeItem(TEMP_REGISTER_NAME_KEY);
    }
  }, [open]);

  if (!open) return null;

  const handleLogin = () => {
    onLogin();
  };

  return children({
    handleLogin
  });
};

export default RegisterSuccessLogic;
```

RegisterSuccess.tsx

```
import React from 'react';
import RegisterSuccessLogic from './RegisterSuccessLogic';
import RegisterSuccessUI from './RegisterSuccessUI';

interface RegisterSuccessProps {
  open: boolean;
  onLogin: () => void;
}
```

```
const RegisterSuccess: React.FC<RegisterSuccessProps> = ({
  open,
  onLogin
}) => {
  return (
    <RegisterSuccessLogic
      open={open}
      onLogin={onLogin}
    >
      {({
        handleLogin
      }) => (
        <RegisterSuccessUI
          handleLogin={handleLogin}
        />
      )}
    </RegisterSuccessLogic>
  );
};

export default RegisterSuccess;
```

search\bar\BarLogic.tsx

```
import { Simulation } from '@/components/form/types';
import React, { useState, useRef, useEffect } from 'react';

export interface BarLogicProps {
  withVoice?: boolean;
  simulations?: Simulation[];
  onFilteredResults?: (filteredSimulations: Simulation[]) => void;
  children: (props: {
    searchQuery: string;
    isListening: boolean;
    setSearchQuery: (query: string) => void;
    handleSearch: () => void;
    toggleListening: () => void;
    handleClearSearch: () => void;
  }) => React.ReactNode;
}

const BarLogic: React.FC<BarLogicProps> = ({
  withVoice = true,
  simulations = [],
  onFilteredResults,
  children
}) => {
  const [searchQuery, setSearchQuery] = useState('');
  const [isListening, setIsListening] = useState(false);
  const recognitionRef = useRef<any>(null);

  useEffect(() => {
    if (withVoice && typeof window !== 'undefined') {
      const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;
      if (SpeechRecognition) {
        recognitionRef.current = new SpeechRecognition();
        recognitionRef.current.continuous = false;
        recognitionRef.current.lang = 'uk-UA';
        recognitionRef.current.interimResults = false;
        recognitionRef.current.maxAlternatives = 1;

        recognitionRef.current.onresult = (event: any) => {
          const speechResult = event.results[0][0].transcript;
          setSearchQuery(speechResult);
          handleSearchAndFilter(speechResult);
          setIsListening(false);
        };

        recognitionRef.current.onerror = () => {
          setIsListening(false);
        };

        recognitionRef.current.onend = () => {
          setIsListening(false);
        };
      }
    }
  }, [withVoice, simulations, onFilteredResults]);

  const filterSimulations = (query: string, simulationsToFilter: Simulation[]): Simulation[] => {
    if (!query.trim()) return [];
    const lowerQuery = query.toLowerCase();
    const words = lowerQuery.split(/\s+/);

    const scored = simulationsToFilter.map(simulation => {
      let score = 0;
      const title = simulation.title.toLowerCase();
      const description = simulation.description.toLowerCase();
      const keywords = (simulation.keywords || []).map(kw => kw.toLowerCase());

      for (const word of words) {
        if (title.includes(word)) score += 3;
        if (description.includes(word)) score += 2;
        if (keywords.some(kw => kw.includes(word))) score += 4;
      }
      return { simulation, score };
    });
```

```
    const filtered = scored.filter(s => s.score > 0);
    filtered.sort((a, b) => b.score - a.score);
    return filtered.map(s => s.simulation);
  };

  const handleSearchAndFilter = (query: string) => {
    if (simulations.length > 0 && onFilteredResults) {
      const filtered = filterSimulations(query, simulations);
      onFilteredResults(filtered);
    }
  };

  const handleSearch = () => {
    handleSearchAndFilter(searchQuery);
  };

  const handleClearSearch = () => {
    setSearchQuery('');
    handleSearchAndFilter('');
  };

  const toggleListening = () => {
    if (!withVoice) return;
    if (isListening) {
      recognitionRef.current?.stop();
    } else {
      try {
        setSearchQuery('');
        recognitionRef.current?.start();
        setIsListening(true);
      } catch (error) {
        setIsListening(false);
      }
    }
  };

  return children({
    searchQuery,
    isListening,
    setSearchQuery,
    handleSearch,
    toggleListening,
    handleClearSearch
  });
};

export default BarLogic;
```

search\bar\BarUI.tsx

```
import React, { useState, useEffect } from 'react';
import { TranslatedText, t } from '../../TranslatedText';

interface BarUIProps {
  searchQuery: string;
  isListening: boolean;
  placeholder?: string;
  withVoice?: boolean;
  setSearchQuery: (query: string) => void;
  handleSearch: () => void;
  toggleListening: () => void;
  handleClearSearch: () => void;
}

const BarUI: React.FC<BarUIProps> = ({
  searchQuery,
  isListening,
  withVoice = true,
  setSearchQuery,
  handleSearch,
  toggleListening,
  handleClearSearch
}) => {
  const [findText, setFindText] = useState('Знайти');
  const [speakNowText, setSpeakNowText] = useState('Говоріть зараз');
  const [placeholderText, setPlaceholderText] = useState('Опишіть ситуацію, наприклад, «кровотеча в
      області живота»');
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  useEffect(() => {
    const handleLanguageChange = (event: CustomEvent) => {
      setLang(event.detail.language);
    };

    window.addEventListener('languageChanged' as any, handleLanguageChange);

    const savedLang = localStorage.getItem('preferredLanguage');
    if (savedLang === 'en' || savedLang === 'ua') {
      setLang(savedLang as 'ua' | 'en');
    }

    return () => {
      window.removeEventListener('languageChanged' as any, handleLanguageChange);
    };
  }, []);

  useEffect(() => {
```

```
    t('Знайти', lang).then(setFindText);
    t('Говоріть зараз', lang).then(setSpeakNowText);
    t('Опишіть ситуацію, наприклад, «кровотеча в області живота»', lang).then(setPlaceholderText);
  }, [lang]);

  return (
    <div className="w-full px-8">
      <div className="flex items-center mb-2">
        <div className="relative flex-1">
          <input
            type="text"
            value={searchQuery}
            onChange={e => setSearchQuery(e.target.value)}
            onKeyDown={e => e.key === 'Enter' && handleSearch()}
            className="w-full px-5 py-4 rounded-l-lg text-gray-700 focus:outline-none"
            placeholder={placeholderText}
          />
          {searchQuery && (
            <button
              className="absolute right-3 top-1/2 transform -translate-y-1/2 text-gray-500"
              onClick={handleClearSearch}
            >
              ✕
            </button>
          )}
        </div>
        {withVoice && (
          <button
            onClick={toggleListening}
            className={`p-4 ${isListening ? 'bg-red-500' : 'bg-blue-600'} text-white hover:bg-
      opacity-90`}
          >
            🎤
          </button>
        )}
        <button
          onClick={handleSearch}
          className="px-5 py-4 bg-green-600 text-white font-medium rounded-r-lg hover:bg-green-700"
        >
          {findText}
        </button>
      </div>
      {isListening && (
        <div className="text-center text-blue-600 mt-2">
          {speakNowText}
        </div>
      )}
    </div>
  );
};

export default BarUI;
```

src\components\search\bar\Bar.tsx

```
import React from 'react';
import BarLogic from './BarLogic';
import { Simulation } from '@/components/form/types';
import BarUI from './BarUI';

interface BarProps {
  placeholder?: string;
  withVoice?: boolean;
  simulations?: Simulation[];
  onFilteredResults?: (filteredSimulations: Simulation[]) => void;
}

export default function Bar({
  placeholder = '',
  withVoice = true,
  simulations = [],
  onFilteredResults
}: BarProps) {
  return (
    <BarLogic
      withVoice={withVoice}
      simulations={simulations}
      onFilteredResults={onFilteredResults}
    >
      {({
        searchQuery,
        isListening,
        setSearchQuery,
        handleSearch,
        toggleListening,
        handleClearSearch
      }) => (
        <BarUI
          searchQuery={searchQuery}
          isListening={isListening}
          placeholder={placeholder}
          withVoice={withVoice}
          setSearchQuery={setSearchQuery}
          handleSearch={handleSearch}
          toggleListening={toggleListening}
          handleClearSearch={handleClearSearch}
        />
```

```
      )}
    </BarLogic>
  );
}
```

search\filter\FilterLogic.tsx

```
import React, { useState, useEffect } from 'react';

export interface FilterOptions {
  sortBy: 'newest' | 'oldest' | 'titleAZ' | 'titleZA' | 'lessonsUp' | 'lessonsDown' |
      'keywordMatch';
  showOnlyFromRootMedic: boolean;
  recentlyUpdated: boolean;
  searchTerm: string;
  searchBy: 'all' | 'title' | 'keywords' | 'author';
}

interface FilterLogicProps {
  items: { [key: string]: any[] };
  onFiltered: (filtered: { [key: string]: any[] }) => void;
  isUsers?: boolean;
  children: (props: {
    filters: FilterOptions;
    handleSortChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
    handleSearchByChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
    handleRootMedicFilterChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleSearchChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleRecentlyUpdatedChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    isUsers?: boolean;
  }) => React.ReactNode;
}

function filterUsers(users: any[], searchTerm: string) {
  if (!searchTerm) return users;
  const searchLower = searchTerm.toLowerCase();
  return users.filter(user =>
    (user.email && user.email.toLowerCase().includes(searchLower)) ||
    (user.name && user.name.toLowerCase().includes(searchLower))
  );
}

function filterEntities(entities: any[], filters: FilterOptions) {
  let result = [...entities];
  const { searchTerm, searchBy, showOnlyFromRootMedic, sortBy, recentlyUpdated } = filters;

  if (showOnlyFromRootMedic) {
    result = result.filter(entity => {
      const isRootAuthor = entity.author && entity.author.role === 'ROOT';
      return isRootAuthor;
    });
  }

  if (recentlyUpdated) {
    const weekAgo = new Date();
    weekAgo.setDate(weekAgo.getDate() - 7);
    result = result.filter(entity => {
      if (!entity.editedAt) return false;
      return new Date(entity.editedAt) > weekAgo;
    });
  }

  if (searchTerm) {
    const searchLower = searchTerm.toLowerCase();
    result = result.filter(entity => {
      let relevance = 0;
      if (searchBy === 'all' || searchBy === 'title') {
        if (entity.title.toLowerCase().includes(searchLower)) relevance += 3;
      }
      if (searchBy === 'all' && entity.description &&
      entity.description.toLowerCase().includes(searchLower)) relevance += 1;
      if ((searchBy === 'all' || searchBy === 'keywords') && entity.keywords &&
      Array.isArray(entity.keywords)) {
        entity.keywords.forEach((keyword: string) => {
          if (keyword.toLowerCase().includes(searchLower) ||
      searchLower.includes(keyword.toLowerCase())) relevance += 2;
        });
      }
      if ((searchBy === 'all' || searchBy === 'author') && entity.author) {
        if (entity.author.name.toLowerCase().includes(searchLower)) relevance += 2;
      }
      entity.searchRelevance = relevance;
      return relevance > 0;
    });
  }

  switch (sortBy) {
    case 'newest':
      result.sort((a, b) => new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime());
      break;
    case 'oldest':
      result.sort((a, b) => new Date(a.createdAt).getTime() - new Date(b.createdAt).getTime());
      break;
    case 'titleAZ':
      result.sort((a, b) => a.title.localeCompare(b.title));
      break;
    case 'titleZA':
```

```
      result.sort((a, b) => b.title.localeCompare(a.title));
      break;
    case 'lessonsUp':
      result.sort((a, b) => (b.totalLessons || 0) - (a.totalLessons || 0));
      break;
    case 'lessonsDown':
      result.sort((a, b) => (a.totalLessons || 0) - (b.totalLessons || 0));
      break;
    case 'keywordMatch':
      if (searchTerm) {
        result.sort((a, b) => (b.searchRelevance || 0) - (a.searchRelevance || 0));
      }
      break;
  }

  return result;
}

const FilterLogic: React.FC<FilterLogicProps> = ({ items, onFiltered, isUsers, children }) => {
  const [filters, setFilters] = useState<FilterOptions>({
    sortBy: 'newest',
    showOnlyFromRootMedic: false,
    recentlyUpdated: false,
    searchTerm: '',
    searchBy: 'all',
  });

  useEffect(() => {
    const filtered: { [key: string]: any[] } = {};
    if (isUsers) {
      Object.keys(items).forEach(key => {
        filtered[key] = filterUsers(items[key], filters.searchTerm);
      });
    } else {
      Object.keys(items).forEach(key => {
        filtered[key] = filterEntities(items[key], filters);
      });
    }
    onFiltered(filtered);
  }, [filters, items, onFiltered, isUsers]);

  const handleSortChange = (e: React.ChangeEvent<HTMLSelectElement>) => {
    setFilters(prev => ({
      ...prev,
      sortBy: e.target.value as FilterOptions['sortBy'],
    }));
  };

  const handleSearchByChange = (e: React.ChangeEvent<HTMLSelectElement>) => {
    setFilters(prev => ({
      ...prev,
      searchBy: e.target.value as FilterOptions['searchBy'],
    }));
  };

  const handleRootMedicFilterChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFilters(prev => ({
      ...prev,
      showOnlyFromRootMedic: e.target.checked,
    }));
  };

  const handleSearchChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFilters(prev => ({
      ...prev,
      searchTerm: e.target.value,
    }));
  };

  const handleRecentlyUpdatedChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFilters(prev => ({
      ...prev,
      recentlyUpdated: e.target.checked,
    }));
  };

  return children({
    filters,
    handleSortChange,
    handleSearchByChange,
    handleRootMedicFilterChange,
    handleSearchChange,
    handleRecentlyUpdatedChange,
    isUsers
  });
};

export default FilterLogic;
```

search\filter\FilterUI.tsx

```
import React, { useState, useEffect } from 'react';
import { TranslatedText, t } from '../../TranslatedText';
import { FilterOptions } from './FilterLogic';

interface FilterUIProps {
  filters: FilterOptions;
```

```
  total: number;
  handleSortChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
  handleSearchByChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
  handleRootMedicFilterChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleSearchChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleRecentlyUpdatedChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  isUsers?: boolean;
}

const FilterUI: React.FC<FilterUIProps> = ({
  filters,
  total,
  handleSortChange,
  handleSearchByChange,
  handleRootMedicFilterChange,
  handleSearchChange,
  handleRecentlyUpdatedChange,
  isUsers
}) => {
  const [searchPlaceholder, setSearchPlaceholder] = useState('Пошук');
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  useEffect(() => {
    const handleLanguageChange = (event: CustomEvent) => {
      setLang(event.detail.language);
    };
    window.addEventListener('languageChanged' as any, handleLanguageChange);

    const savedLang = localStorage.getItem('preferredLanguage');
    if (savedLang === 'en' || savedLang === 'ua') {
      setLang(savedLang as 'ua' | 'en');
    }

    return () => {
      window.removeEventListener('languageChanged' as any, handleLanguageChange);
    };
  }, []);

  useEffect(() => {
    t('Пошук', lang).then(setSearchPlaceholder);
  }, [lang]);

  if (isUsers) {
    return (
      <div className="filter-container section-spacing">
        <div className="flex flex-col lg:flex-row items-start lg:items-center gap-4">
          <div className="flex-1 w-full">
            <div className="relative">
              <input
                type="text"
                placeholder={searchPlaceholder}
                className="search-input"
                value={filters.searchTerm}
                onChange={handleSearchChange}
              />
              <div className="absolute left-3 top-2.5 text-gray-400">
                <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" fill="none" viewBox="0 0
    24 24" stroke="currentColor">
                  <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M21 21l-6-
    6m2-5a7 7 0 11-14 0 7 7 0 0114 0z" />
                </svg>
              </div>
            </div>
          </div>
        </div>
      </div>
    );
  }

  return (
    <div className="filter-container section-spacing">
      <div className="flex flex-col lg:flex-row items-start lg:items-center gap-4">
        <div className="flex-1 w-full">
          <div className="relative">
            <input
              type="text"
              placeholder={searchPlaceholder}
              className="search-input"
              value={filters.searchTerm}
              onChange={handleSearchChange}
            />
            <div className="absolute left-3 top-2.5 text-gray-400">
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" fill="none" viewBox="0 0
    24 24" stroke="currentColor">
                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M21 21l-6-6m2-
    5a7 7 0 11-14 0 7 7 0 0114 0z" />
              </svg>
            </div>
          </div>
        </div>

        <div className="flex flex-col sm:flex-row items-start sm:items-center gap-4">
          <div className="flex items-center">
            <span className="text-sm text-gray-700 mr-3 whitespace-nowrap">
              <TranslatedText text="Сортувати за:" />
            </span>
            <select
```

```
            className="border border-gray-300 rounded-lg px-3 py-2 focus:outline-none focus:ring-2
    focus:ring-blue-500 min-w-[50px] max-w-xs w-full"
            value={filters.sortBy}
            onChange={handleSortChange}
          >
            <option value="newest">
              <TranslatedText text="Спочатку нові" />
            </option>
            <option value="oldest">
              <TranslatedText text="Спочатку старі" />
            </option>
            <option value="titleAZ">
              <TranslatedText text="По назві (А-Я)" />
            </option>
            <option value="titleZA">
              <TranslatedText text="По назві (Я-А)" />
            </option>
            <option value="lessonsUp">
              <TranslatedText text="За кількістю уроків ↑" />
            </option>
            <option value="lessonsDown">
              <TranslatedText text="За кількістю уроків ↓" />
            </option>
            <option value="keywordMatch">
              <TranslatedText text="За співпадінням з пошуком" />
            </option>
          </select>
        </div>
      </div>
    </div>

    <div className="mt-3 flex flex-col sm:flex-row justify-between items-start sm:items-center
    gap-2">
      <div className="flex items-center flex-wrap">
        <span className="text-sm text-gray-700 mr-3">
          <TranslatedText text="Шукати за:" />
        </span>
        <select
          className="border border-gray-300 rounded-lg px-3 py-1 text-sm focus:outline-none
    focus:ring-2 focus:ring-blue-500 mr-4"
          value={filters.searchBy}
          onChange={handleSearchByChange}
        >
          <option value="all">
            <TranslatedText text="Усі поля" />
          </option>
          <option value="title">
            <TranslatedText text="Назва" />
          </option>
          <option value="keywords">
            <TranslatedText text="Ключові слова" />
          </option>
          <option value="author">
            <TranslatedText text="Автор" />
          </option>
        </select>

        <div className="flex items-center mt-2 sm:mt-0 mr-4">
          <input
            type="checkbox"
            id="showOnlyFromRootMedic"
            className="mr-2 h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded"
            checked={filters.showOnlyFromRootMedic}
            onChange={handleRootMedicFilterChange}
          />
          <label htmlFor="showOnlyFromRootMedic" className="text-sm text-gray-700">
            <TranslatedText text="Тільки від Головного медика" />
          </label>
        </div>

        <div className="flex items-center mt-2 sm:mt-0">
          <input
            type="checkbox"
            id="recentlyUpdated"
            className="mr-2 h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded"
            checked={filters.recentlyUpdated}
            onChange={handleRecentlyUpdatedChange}
          />
          <label htmlFor="recentlyUpdated" className="text-sm text-gray-700">
            <TranslatedText text="Нещодавно оновлено" />
          </label>
        </div>
      </div>

      <div className="text-sm text-gray-500 flex mt-2 sm:mt-0">
        <span>
          <TranslatedText text="Знайдено:" /> <span className="font-medium text-gray-
    700">{total}</span>
        </span>
      </div>
    </div>

    <div className="mt-1 text-xs text-gray-500">
      <TranslatedText text="Часткові співпадіння враховуються" />
    </div>
  </div>
);
```

```
};

export default FilterUI;
```

search\filter\Filter.tsx

```
import React from 'react';
import FilterLogic from './FilterLogic';
import FilterUI from './FilterUI';

interface FilterProps {
  items: { [key: string]: any[] };
  onFiltered: (filtered: { [key: string]: any[] }) => void;
  total?: number;
  isUsers?: boolean;
}

const Filter: React.FC<FilterProps> = ({
  items,
  onFiltered,
  total,
  isUsers
}) => {
  return (
    <FilterLogic items={items} onFiltered={onFiltered} isUsers={isUsers}>
      {({
        filters,
        handleSortChange,
        handleSearchByChange,
        handleRootMedicFilterChange,
        handleRecentlyUpdatedChange,
        handleSearchChange
      }) => (
        <FilterUI
          filters={filters}
          total={total || 0}
          handleSortChange={handleSortChange}
          handleSearchByChange={handleSearchByChange}
          handleRootMedicFilterChange={handleRootMedicFilterChange}
          handleSearchChange={handleSearchChange}
          handleRecentlyUpdatedChange={handleRecentlyUpdatedChange}
          isUsers={isUsers}
        />
      )}
    </FilterLogic>
  );
};

export default Filter;
```

form\types.ts

```
import type { Test } from './test/types';

export interface Course {
  id?: string;
  title: string;
  description: string;
  keywords: string[];
  imageUrl?: string;
  sections?: Section[];
}

export interface Section {
  id?: string;
  title: string;
  lessons: Lesson[];
}

export interface Lesson {
  id?: string;
  title: string;
  type: 'TEXT' | 'VIDEO' | 'DOCUMENT' | 'TEST';
  content?: string;
  videoUrl?: string;
  documentUrl?: string;
  fileName?: string;
  fileUrl?: string;
  uploadError?: string;
  tempFileName?: string;
  testSaved?: boolean;
  test?: Test;
}

export interface Simulation {
  id?: string;
  title: string;
  description: string;
  keywords: string[];
  imageUrl?: string;
  steps: Step[];
}

export interface Step {
  id?: string;
  title: string;
  content: string;
```

```
  videoUrl: string;
  videoPreviewUrl?: string;
}

export interface FormState {
  title: string;
  description: string;
  keywords: string[];
  imageUrl: string;
  sections?: Section[];
  steps?: Step[];
  editingTitle: boolean;
  editingDescription: boolean;
  pendingImage: File | null;
  pendingFiles: { [key: string]: File };
  errors: Record<string, string>;
  imageLoading: boolean;
  fileLoading: { [key: string]: boolean };
  loading: boolean;
  showCancelModal: boolean;
}
```

KeywordsLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { t } from '@/components/TranslatedText';

interface KeywordsLogicProps {
  keywords: string[];
  updateKeywords: (keywords: string[]) => void;
  removeKeyword: (index: number) => void;
  children: (props: {
    keywords: string[];
    updateKeyword: (index: number, value: string) => void;
    removeKeyword: (index: number) => void;
    keywordPlaceholder: string;
    wordPlaceholder: string;
    addKeywordText: string;
    newKeyword: string;
    error: string;
    handleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleAddKeyword: () => void;
    handleKeyDown: (e: React.KeyboardEvent<HTMLInputElement>) => void;
    handleRemoveKeyword: (index: number) => void;
  }) => React.ReactNode;
}

const KeywordsLogic: React.FC<KeywordsLogicProps> = ({
  keywords,
  updateKeywords,
  removeKeyword,
  children
}) => {
  const [keywordPlaceholder, setKeywordPlaceholder] = useState("Додайте ключове слово");
  const [wordPlaceholder, setWordPlaceholder] = useState("Ключове слово");
  const [addKeywordText, setAddKeywordText] = useState("Додати");
  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [newKeyword, setNewKeyword] = useState('');
  const [error, setError] = useState('');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t("Додайте ключове слово", lang).then(setKeywordPlaceholder);
    t("Додати", lang).then(setAddKeywordText);
    t("Ключове слово", lang).then(setWordPlaceholder);
  }, [lang]);

  useEffect(() => {
    setError('');
  }, [keywords]);

  const isValidKeyword = (keyword: string): boolean => {
    const normalized = keyword.trim().toLowerCase();
    if (!normalized) return false;
    if (normalized.includes(' ')) return false;
    if (keywords.some(k => k.trim().toLowerCase() === normalized)) return false;
    return true;
  };

  const updateKeyword = (index: number, value: string) => {
    const trimmedValue = value.trim().toLowerCase();
    if (index >= keywords.length) {
      if (isValidKeyword(trimmedValue)) {
        const newKeywords = [...keywords, trimmedValue];
        updateKeywords(newKeywords);
      }
    } else {
      const newKeywords = [...keywords];
      const isDuplicate = keywords.some((k, i) =>
        i !== index && k.trim().toLowerCase() === trimmedValue
      );
      if (!trimmedValue.includes(' ') && !isDuplicate) {
```

```
        newKeywords[index] = trimmedValue;
        updateKeywords(newKeywords);
      }
    }
  };

  const handleChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
    const value = e.target.value;
    setNewKeyword(value);
    if (value.includes(' ')) {
      const errorText = await t('Введіть лише одне слово без пробілів', lang);
      setError(errorText);
    } else {
      setError('');
    }
  };

  const handleAddKeyword = async () => {
    const normalized = newKeyword.trim().toLowerCase();
    if (!normalized) return;
    if (normalized.includes(' ')) {
      const errorText = await t('Введіть лише одне слово без пробілів', lang);
      setError(errorText);
      return;
    }
    if (keywords.some(k => k.trim().toLowerCase() === normalized)) {
      const errorText = await t('Таке ключове слово вже додано', lang);
      setError(errorText);
      return;
    }
    updateKeyword(keywords.length, normalized);
    setNewKeyword('');
    setError('');
  };

  const handleKeyDown = async (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter') {
      e.preventDefault();
      handleAddKeyword();
    }
  };

  const handleRemoveKeyword = (index: number) => {
    removeKeyword(index);
    setError('');
  };

  return children({
    keywords,
    updateKeyword,
    removeKeyword,
    keywordPlaceholder,
    wordPlaceholder,
    addKeywordText,
    newKeyword,
    error,
    handleChange,
    handleAddKeyword,
    handleKeyDown,
    handleRemoveKeyword
  });
};

export default KeywordsLogic;
```

KeywordsUI.tsx

```
import React from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface KeywordsUIProps {
  keywords: string[];
  updateKeyword: (index: number, value: string) => void;
  removeKeyword: (index: number) => void;
  keywordPlaceholder: string;
  wordPlaceholder: string;
  addKeywordText: string;
  newKeyword: string;
  error: string;
  handleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleAddKeyword: () => void;
  handleKeyDown: (e: React.KeyboardEvent<HTMLInputElement>) => void;
  handleRemoveKeyword: (index: number) => void;
}

const KeywordsUI: React.FC<KeywordsUIProps> = ({
  keywords,
  keywordPlaceholder,
  addKeywordText,
  wordPlaceholder,
  newKeyword,
  error,
  handleChange,
  handleAddKeyword,
  handleKeyDown,
  handleRemoveKeyword
}) => {
```

```
  return (
    <div className="mb-6 w-full max-w-full [@media(max-width:300px)]:p-0.5">
      <label className="block mb-2 text-sm md:text-base font-medium text-gray-700 flex flex-wrap
      items-center gap-x-1">
        <TranslatedText text="Ключові слова" />
        <span className="text-gray-400 text-xs">(<TranslatedText text="необов'язково" />)</span>
      </label>

      <p className="text-xs md:text-sm text-gray-500 mb-2">
        <TranslatedText text="Додайте ключові слова для кращого пошуку курсу. Кожне слово додавайте
      окремо." />
      </p>

      <div className="flex flex-col md:flex-row mb-1 space-y-2 md:space-y-0 md:space-x-2 w-full">
        <div className="flex-grow relative w-full">
          <input
            type="text"
            value={newKeyword}
            onChange={handleChange}
            onKeyDown={handleKeyDown}
            className={`w-full border ${error ? 'border-red-300 focus:ring-red-500 focus:border-red-
      500' : 'border-gray-300 focus:ring-blue-500 focus:border-blue-500'} rounded-l-md shadow-sm py-
      2 px-2 md:px-3 focus:outline-none text-sm md:text-base whitespace-pre-line placeholder:text-
      gray-400 placeholder:text-xs md:placeholder:text-base`}
            placeholder={window.innerWidth < 320 ? wordPlaceholder : keywordPlaceholder}
            maxLength={20}
          />
          <div className="absolute right-2 top-1/2 transform -translate-y-1/2 text-xs text-gray-
      400">
            {newKeyword.length}/20
          </div>
        </div>
        <button
          type="button"
          onClick={handleAddKeyword}
          className={`w-full md:w-auto ${error || !newKeyword.trim() ? 'bg-blue-300 cursor-not-
      allowed' : 'bg-blue-400 hover:bg-blue-500'} text-white font-medium py-2 px-4 md:px-6 rounded-
      r-md text-sm md:text-base`}
          disabled={!!error || !newKeyword.trim()}
        >
          <TranslatedText text={addKeywordText} />
        </button>
      </div>

      {error && (
        <p className="text-red-500 text-xs mt-1">{error}</p>
      )}

      {keywords.length > 0 && (
        <div className="flex flex-wrap gap-2 mt-3">
          {keywords.map((keyword, index) => (
            <div
              key={index}
              className="bg-gray-100 text-gray-700 text-xs md:text-sm px-3 py-1 rounded-full group
      hover:bg-gray-200 transition-colors flex items-center"
            >
              {keyword}
              <button
                type="button"
                onClick={() => handleRemoveKeyword(index)}
                className="ml-1 text-gray-400 hover:text-gray-600 inline-flex items-center justify-
      center"
                aria-label="Видалити ключове слово"
              >
                <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" viewBox="0 0 20 20"
      fill="currentColor">
                  <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0
      00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-
      1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />
                </svg>
              </button>
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default KeywordsUI;
```

Keywords.tsx

```
import React from 'react';
import KeywordsLogic from './KeywordsLogic';
import KeywordsUI from './KeywordsUI';

interface KeywordsProps {
  keywords: string[];
  updateKeywords: (keywords: string[]) => void;
  removeKeyword: (index: number) => void;
}

const Keywords: React.FC<KeywordsProps> = ({
  keywords,
  updateKeywords,
  removeKeyword
```

```
}) => {
  return (
    <KeywordsLogic
      keywords={keywords}
      updateKeywords={updateKeywords}
      removeKeyword={removeKeyword}
    >
      {(keywordsProps) => <KeywordsUI {...keywordsProps} />}
    </KeywordsLogic>
  );
};

export default Keywords;
```

SectionLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { Section } from '../types';
import { t } from '@/components/TranslatedText';

interface SectionLogicProps {
  section: Section;
  updateSectionTitle: (value: string) => void;
  addLesson: () => void;
  removeLesson: (lessonIdx: number) => void;
  updateLesson: (lessonIdx: number, field: string, value: any) => void;
  handleFileUpload: (lessonIdx: number, file: File) => void;
  removeSection: () => void;
  moveUp?: () => void;
  moveDown?: () => void;
  moveLessonUp: (lessonIdx: number) => void;
  moveLessonDown: (lessonIdx: number) => void;
  fileLoading: { [key: string]: boolean };
  errors?: Record<string, string>;
  sectionIdx?: number;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
  children: (props: {
    section: Section;
    addLesson: () => void;
    removeLesson: (lessonIdx: number) => void;
    updateLesson: (lessonIdx: number, field: string, value: any) => void;
    handleFileUpload: (lessonIdx: number, file: File) => void;
    removeSection: () => void;
    moveUp?: () => void;
    moveDown?: () => void;
    moveLessonUp: (lessonIdx: number) => void;
    moveLessonDown: (lessonIdx: number) => void;
    fileLoading: { [key: string]: boolean };
    errors?: Record<string, string>;
    sectionError?: string;
    sectionTitlePlaceholder: string;
    addLessonText: string;
    moveUpTitle: string;
    moveDownTitle: string;
    removeSectionTitle: string;
    emptyLessonsText: string;
    handleSectionTitleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    sectionIdx?: number;
    setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message:
      string } | null>>;
  }) => React.ReactNode;
}

const SectionLogic: React.FC<SectionLogicProps> = ({
  section,
  updateSectionTitle,
  addLesson,
  removeLesson,
  updateLesson,
  handleFileUpload,
  removeSection,
  moveUp,
  moveDown,
  moveLessonUp,
  moveLessonDown,
  fileLoading,
  errors,
  sectionIdx,
  setNotification,
  children
}) => {
  const [sectionTitlePlaceholder, setSectionTitlePlaceholder] = useState("Назва розділу");
  const [addLessonText, setAddLessonText] = useState("Додати урок");
  const [moveUpTitle, setMoveUpTitle] = useState("Перемістити розділ вгору");
  const [moveDownTitle, setMoveDownTitle] = useState("Перемістити розділ вниз");
  const [removeSectionTitle, setRemoveSectionTitle] = useState("Видалити розділ");
  const [emptyLessonsText, setEmptyLessonsText] = useState("Додайте уроки до цього розділу");
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  const [sectionError, setSectionError] = useState<string | undefined>(undefined);

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);
```

```
  useEffect(() => {
    t("Назва розділу", lang).then(setSectionTitlePlaceholder);
    t("Додати урок", lang).then(setAddLessonText);
    t("Перемістити розділ вгору", lang).then(setMoveUpTitle);
    t("Перемістити розділ вниз", lang).then(setMoveDownTitle);
    t("Видалити розділ", lang).then(setRemoveSectionTitle);
    t("Додайте уроки до цього розділу", lang).then(setEmptyLessonsText);
  }, [lang]);

  useEffect(() => {
    if (sectionIdx !== undefined && errors) {
      const sectionErrorKey = `section-${sectionIdx}`;

      if (errors[sectionErrorKey]) {
        t(errors[sectionErrorKey], lang).then(setSectionError);
      } else if (errors.section) {
        t(errors.section, lang).then(setSectionError);
      } else {
        setSectionError(undefined);
      }
    } else if (errors && errors.section) {
      t(errors.section, lang).then(setSectionError);
    } else {
      setSectionError(undefined);
    }
  }, [errors, lang, sectionIdx]);

  const handleSectionTitleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    updateSectionTitle(e.target.value);
  };

  return children({
    section,
    addLesson,
    removeLesson,
    updateLesson,
    handleFileUpload,
    removeSection,
    moveUp,
    moveDown,
    moveLessonUp,
    moveLessonDown,
    fileLoading,
    errors,
    sectionError,
    sectionTitlePlaceholder,
    addLessonText,
    moveUpTitle,
    moveDownTitle,
    removeSectionTitle,
    emptyLessonsText,
    handleSectionTitleChange,
    sectionIdx,
    setNotification
  });
};

export default SectionLogic;
```

SectionUI.tsx

```
import React from 'react';
import { Section } from '../types';
import { TranslatedText } from '@/components/TranslatedText';
import Lesson from '../lesson/Lesson';

interface SectionUIProps {
  section: Section;
  addLesson: () => void;
  removeLesson: (lessonIdx: number) => void;
  updateLesson: (lessonIdx: number, field: string, value: any) => void;
  handleFileUpload: (lessonIdx: number, file: File) => void;
  removeSection: () => void;
  moveUp?: () => void;
  moveDown?: () => void;
  moveLessonUp: (lessonIdx: number) => void;
  moveLessonDown: (lessonIdx: number) => void;
  fileLoading: { [key: string]: boolean };
  sectionTitlePlaceholder: string;
  addLessonText: string;
  moveUpTitle: string;
  moveDownTitle: string;
  removeSectionTitle: string;
  emptyLessonsText: string;
  handleSectionTitleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  errors?: Record<string, string>;
  sectionError?: string;
  sectionIdx?: number;
  clearLessonContentError?: (sectionIdx: number, lessonIdx: number) => void;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
}

const SectionUI: React.FC<SectionUIProps> = ({
  section,
  addLesson,
```

```
  removeLesson,
  updateLesson,
  handleFileUpload,
  removeSection,
  moveUp,
  moveDown,
  moveLessonUp,
  moveLessonDown,
  fileLoading,
  sectionTitlePlaceholder,
  addLessonText,
  moveUpTitle,
  moveDownTitle,
  removeSectionTitle,
  emptyLessonsText,
  handleSectionTitleChange,
  errors,
  sectionError,
  sectionIdx,
  clearLessonContentError,
  setNotification,
}) => {
  return (
    <div className="bg-gray-50 border border-gray-200 rounded-lg p-3 md:p-4 mb-4 [@media(max-
      width:300px)]:p-0.5">
       <div className="flex flex-col md:flex-row md:justify-between md:items-center mb-4 space-y-2
      md:space-y-0 md:space-x-2">
         <div className="flex-1">
           <input
             type="text"
             value={section.title}
             onChange={handleSectionTitleChange}
             className={`w-full border ${sectionError ? 'border-red-500' : 'border-gray-300'}
      rounded-md shadow-sm py-2 px-2 md:px-3 focus:outline-none focus:ring-indigo-500 focus:border-
      indigo-500 text-sm md:text-base font-medium`}
             placeholder={sectionTitlePlaceholder}
             maxLength={100}
           />
           <div className="flex justify-between">
             {sectionError && (
               <div className="text-xs text-red-500 mt-1">{sectionError}</div>
             )}
             <div className="text-xs text-gray-500 mt-1 ml-auto">{section.title.length}/100</div>
           </div>
         </div>
         <div className="flex items-center md:ml-3">
           {moveUp && (
             <button
               type="button"
               onClick={moveUp}
               className="text-gray-400 hover:text-gray-600 mx-1 p-1 hover:bg-gray-100 rounded"
               title={moveUpTitle}
             >
               <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
                 <path fillRule="evenodd" d="M5.293 7.707a1 1 0 010-1.414l4-4a1 1 0 011.414 0l4 4a1 1
      0 01-1.414 1.414L11 5.414V17a1 1 0 11-2 0V5.414L6.707 7.707a1 1 0 01-1.414 0z"
      clipRule="evenodd" />
               </svg>
             </button>
           )}
           {moveDown && (
             <button
               type="button"
               onClick={moveDown}
               className="text-gray-400 hover:text-gray-600 mx-1 p-1 hover:bg-gray-100 rounded"
               title={moveDownTitle}
             >
               <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
                 <path fillRule="evenodd" d="M14.707 12.293a1 1 0 010 1.414l-4 4a1 1 0 01-1.414 0l-4-
      4a1 1 0 111.414-1.414L9 14.586V3a1 1 0 012 0v11.586l2.293-2.293a1 1 0 011.414 0z"
      clipRule="evenodd" />
               </svg>
             </button>
           )}
           <button
             type="button"
             onClick={removeSection}
             className="text-red-500 hover:text-red-700 ml-2 p-1 hover:bg-red-50 rounded"
             title={removeSectionTitle}
           >
             <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
               <path fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000 2v10a2 2 0 002
      2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1 1 0 012 0v6a1 1 0 11-2
      0V8zm5-1a1 1 0 00-1 1v6a1 1 0 102 0V8a1 1 0 00-1-1z" clipRule="evenodd" />
             </svg>
           </button>
         </div>
       </div>

       <div className="ml-4 space-y-3">
         {section.lessons.length === 0 && (
           <div className={`p-3 ${errors?.course ? 'bg-red-50 border-red-100 text-red-500' : 'bg-
      gray-50 border-gray-100 text-gray-500'} border rounded-md text-sm`}>
             <div className="flex items-center">
```

```
            <svg xmlns="http://www.w3.org/2000/svg" className={`h-5 w-5 mr-2 ${errors?.course ?
    'text-red-500' : 'text-gray-400'}`} fill="none" viewBox="0 0 24 24" stroke="currentColor">
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={1.5} d="M13 16h-1v-
    4h-1m1-4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z" />
            </svg>
            <span><TranslatedText text={errors?.course || emptyLessonsText} /></span>
          </div>
        </div>
      )}

      {section.lessons.map((lesson, lessonIdx) => (
        <Lesson
          key={lesson.id || `lesson-${sectionIdx}-${lessonIdx}`}
          lesson={lesson}
          updateLesson={(field: string, value: any) => updateLesson(lessonIdx, field, value)}
          removeLesson={() => removeLesson(lessonIdx)}
          handleFileUpload={(file: File) => handleFileUpload(lessonIdx, file)}
          moveUp={lessonIdx > 0 ? () => moveLessonUp(lessonIdx) : undefined}
          moveDown={lessonIdx < section.lessons.length - 1 ? () => moveLessonDown(lessonIdx) :
    undefined}
          isLoading={fileLoading[`${sectionIdx}-${lessonIdx}`]}
          errors={errors}
          lessonIdx={lessonIdx}
          sectionIdx={sectionIdx}
          clearPendingFile={() => {
            if (typeof lessonIdx === 'number' && typeof sectionIdx === 'number') {
              updateLesson(lessonIdx, 'pendingFile', null);
            }
          }}
          clearLessonContentError={clearLessonContentError}
          setNotification={setNotification}
        />
      ))}

      <button
        type="button"
        onClick={addLesson}
        className="mt-3 inline-flex items-center px-3 py-2 border border-transparent text-sm
    leading-4 font-medium rounded-md text-indigo-700 bg-indigo-100 hover:bg-indigo-200
    focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500"
      >
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-2" viewBox="0 0 20 20"
    fill="currentColor">
          <path fillRule="evenodd" d="M10 5a1 1 0 011 1v3h3a1 1 0 110 2h-3v3a1 1 0 11-2 0v-3H6a1 1
    0 110-2h3V6a1 1 0 011-1z" clipRule="evenodd" />
        </svg>
        <TranslatedText text={addLessonText} />
      </button>
    </div>
  </div>
  );
};

export default SectionUI;
```

Section.tsx

```
import React from 'react';
import SectionLogic from './SectionLogic';
import SectionUI from './SectionUI';
import { Section as SectionType } from '../types';

interface SectionProps {
  section: SectionType;
  sectionIdx: number;
  updateSectionTitle: (sectionIdx: number, value: string) => void;
  addLesson: (sectionIdx: number) => void;
  removeLesson: (sectionIdx: number, lessonIdx: number) => void;
  updateLesson: (sectionIdx: number, lessonIdx: number, field: string, value: any) => void;
  handleFileUpload: (sectionIdx: number, lessonIdx: number, file: File) => void;
  removeSection: (sectionIdx: number) => void;
  moveSection: (fromIdx: number, toIdx: number) => void;
  moveLesson: (sectionIdx: number, fromIdx: number, toIdx: number) => void;
  fileLoading: { [key: string]: boolean };
  errors?: Record<string, string>;
  sectionsLength?: number;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
}

const Section: React.FC<SectionProps> = ({
  section,
  sectionIdx,
  updateSectionTitle,
  addLesson,
  removeLesson,
  updateLesson,
  handleFileUpload,
  removeSection,
  moveSection,
  moveLesson,
  fileLoading,
  errors,
  sectionsLength,
  setNotification
}) => {
  const canMoveUp = sectionIdx > 0;
```

```
  const canMoveDown = sectionsLength ? sectionIdx < sectionsLength - 1 : false;

  return (
    <SectionLogic
      section={section}
      updateSectionTitle={(value) => updateSectionTitle(sectionIdx, value)}
      addLesson={() => addLesson(sectionIdx)}
      removeLesson={(lessonIdx) => removeLesson(sectionIdx, lessonIdx)}
      updateLesson={(lessonIdx, field, value) => updateLesson(sectionIdx, lessonIdx, field, value)}
      handleFileUpload={(lessonIdx, file) => handleFileUpload(sectionIdx, lessonIdx, file)}
      removeSection={() => removeSection(sectionIdx)}
      moveUp={canMoveUp ? () => moveSection(sectionIdx, sectionIdx - 1) : undefined}
      moveDown={canMoveDown ? () => moveSection(sectionIdx, sectionIdx + 1) : undefined}
      moveLessonUp={(lessonIdx) => moveLesson(sectionIdx, lessonIdx, lessonIdx - 1)}
      moveLessonDown={(lessonIdx) => moveLesson(sectionIdx, lessonIdx, lessonIdx + 1)}
      fileLoading={fileLoading}
      errors={errors}
      sectionIdx={sectionIdx}
      setNotification={setNotification}
    >
      {(sectionProps) => <SectionUI {...sectionProps} />}
    </SectionLogic>
  );
};

export default Section;
```

form\lesson\lessonUtils.ts

```
const allowedDocumentTypes = [
  'application/pdf',
  'application/msword',
  'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
  'application/vnd.ms-powerpoint',
  'application/vnd.openxmlformats-officedocument.presentationml.presentation',
  'application/vnd.ms-excel',
  'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
  'image/',
  'text/plain'
];

function isAllowedType(file: File, allowedTypes: string[]): boolean {
  return allowedTypes.some(type =>
    type.endsWith('/')
      ? file.type.startsWith(type)
      : file.type === type
  ) || file.name.endsWith('.txt') || file.name.endsWith('.doc') || file.name.endsWith('.docx');
}

function isFileSizeValid(file: File, maxSizeMB: number): boolean {
  return file.size <= maxSizeMB * 1024 * 1024;
}

export const lessonUtils = {
  formatFileName: (fileName: string, maxLength: number = 45): string => {
    if (!fileName) return '';

    if (fileName.length <= maxLength) {
      return fileName;
    }

    const lastDotIndex = fileName.lastIndexOf('.');
    const extension = lastDotIndex !== -1 ? fileName.substring(lastDotIndex) : '';

    const name = fileName.substring(0, lastDotIndex !== -1 ? lastDotIndex : fileName.length);
    const truncatedName = name.substring(0, maxLength - 3 - extension.length) + '...';

    return truncatedName + extension;
  },
  isValidVideoType: (fileType: string): boolean => {
    return fileType.startsWith('video/');
  },
  isValidDocumentType: (fileType: string): boolean =>
    allowedDocumentTypes.some(type => type.endsWith('/') ? fileType.startsWith(type) : fileType ===
      type)
};

export const formatFileName = (fileName: string, maxLength: number = 45): string => {
  return lessonUtils.formatFileName(fileName, maxLength);
};

export const validateVideoFile = (
  file: File,
  invalidFormatErrorText: string,
  sizeExceededErrorText: string
): { valid: boolean; error?: string } => {
  if (!file.type.startsWith('video/')) {
    return { valid: false, error: invalidFormatErrorText };
  }
  if (!isFileSizeValid(file, 200)) {
    return { valid: false, error: sizeExceededErrorText };
  }
  return { valid: true };
};

export const validateDocumentFile = (
  file: File,
```

```
  invalidFormatErrorText: string,
  sizeExceededErrorText: string
): { valid: boolean; error?: string } => {
  if (!isAllowedType(file, allowedDocumentTypes)) {
    return { valid: false, error: invalidFormatErrorText };
  }
  if (!isFileSizeValid(file, 50)) {
    return { valid: false, error: sizeExceededErrorText };
  }
  return { valid: true };
};
```

LessonLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { Lesson } from '../types';
import { t } from '@/components/TranslatedText';
import { formatFileName, validateVideoFile, validateDocumentFile } from './lessonUtils';
import { Test as TestType } from '../test/types';

interface LessonLogicProps {
  lesson: Lesson;
  updateLesson: (field: string, value: any) => void;
  removeLesson: () => void;
  handleFileUpload: (file: File) => void;
  moveUp?: () => void;
  moveDown?: () => void;
  isLoading?: boolean;
  errors?: Record<string, string>;
  clearPendingFile?: () => void;
  lessonIdx?: number;
  sectionIdx?: number;
  clearLessonContentError?: (sectionIdx: number, lessonIdx: number) => void;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
  children: (props: {
    lesson: Lesson;
    updateLesson: (field: string, value: any) => void;
    removeLesson: () => void;
    handleFileUpload: (file: File) => void;
    moveUp?: () => void;
    moveDown?: () => void;
    isLoading?: boolean;
    lessonTitlePlaceholder: string;
    textContentPlaceholder: string;
    typeText: string;
    typeVideo: string;
    typeDocument: string;
    typeTest: string;
    moveUpTitle: string;
    moveDownTitle: string;
    removeLessonTitle: string;
    errors?: Record<string, string>;
    lessonTitleError?: string;
    lessonContentError?: string;
    deleteFileText: string;
    uploadVideoText: string;
    maxVideoSizeText: string;
    videoDescText: string;
    uploadDocText: string;
    supportedDocTypesText: string;
    maxDocSizeText: string;
    docDescText: string;
    optionalText: string;
    handleTitleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleContentChange: (e: React.ChangeEvent<HTMLTextAreaElement>) => void;
    handleVideoUpload: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleDocumentUpload: (e: React.ChangeEvent<HTMLInputElement>) => void;
    handleDeleteFile: () => void;
    handleTypeChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
    handleDescriptionChange: (e: React.ChangeEvent<HTMLTextAreaElement>) => void;
    handleTestChange: (test: TestType) => void;
    formatFileName: (fileName: string) => string;
    clearLessonContentError?: (sectionIdx: number, lessonIdx: number) => void;
    testDraft: TestType;
    setTestDraft: React.Dispatch<React.SetStateAction<TestType>>;
    isTestEditing: boolean;
    setIsTestEditing: React.Dispatch<React.SetStateAction<boolean>>;
    testSaved: boolean;
    editPlaceholder: string;
    deletePlaceholder: string;
    handleEditTest: () => void;
    handleDeleteTest: () => void;
    handleSaveTest: (test: TestType) => void;
  }) => React.ReactNode;
}

const LessonLogic: React.FC<LessonLogicProps> = ({
  lesson,
  updateLesson,
  removeLesson,
  handleFileUpload,
  moveUp,
  moveDown,
  isLoading,
  errors,
  clearPendingFile,
```

```
  lessonIdx,
  sectionIdx,
  clearLessonContentError,
  setNotification,
  children
}) => {
  function clearLessonError() {
    if (typeof clearLessonContentError === 'function' && sectionIdx !== undefined && lessonIdx !==
      undefined) {
      clearLessonContentError(sectionIdx, lessonIdx);
    }
  }

  function resetFileFields(type: 'VIDEO' | 'DOCUMENT') {
    if (type === 'VIDEO') {
      updateLesson('videoUrl', '');
      updateLesson('fileName', '');
      updateLesson('tempFileName', '');
    } else if (type === 'DOCUMENT') {
      updateLesson('documentUrl', '');
      updateLesson('fileName', '');
      updateLesson('tempFileName', '');
    }
  }

  function resetLessonFields() {
    updateLesson('content', '');
    updateLesson('videoUrl', '');
    updateLesson('documentUrl', '');
    updateLesson('fileName', '');
    updateLesson('tempFileName', '');
    updateLesson('uploadError', '');
    if (typeof clearPendingFile === 'function') {
      clearPendingFile();
    }
    const fileInputs = document.querySelectorAll('input[type="file"]');
    fileInputs.forEach(input => {
      const fileInput = input as HTMLInputElement;
      if (fileInput) {
        fileInput.value = '';
      }
    });
  }

  function getFileNameFromUrl(url: string, fallback: string) {
    return url.split('/').pop() || fallback;
  }

  function setTranslation(key: string, setter: (v: string) => void) {
    t(key, lang).then(setter);
  }

  const [lessonTitlePlaceholder, setLessonTitlePlaceholder] = useState("Назва уроку");
  const [textContentPlaceholder, setTextContentPlaceholder] = useState("Вміст уроку");
  const [typeText, setTypeText] = useState("Текст");
  const [typeVideo, setTypeVideo] = useState("Відео");
  const [typeDocument, setTypeDocument] = useState("Документ");
  const [typeTest, setTypeTest] = useState("Тест");
  const [moveUpTitle, setMoveUpTitle] = useState("Перемістити урок вгору");
  const [moveDownTitle, setMoveDownTitle] = useState("Перемістити урок вниз");
  const [removeLessonTitle, setRemoveLessonTitle] = useState("Видалити урок");
  const [lang, setLang] = useState<'ua' | 'en'>('ua');

  const [deleteFileText, setDeleteFileText] = useState("Видалити файл");
  const [uploadVideoText, setUploadVideoText] = useState("Натисніть або перетягніть, щоб завантажити
      відео");
  const [maxVideoSizeText, setMaxVideoSizeText] = useState("Максимальний розмір: 200MB");
  const [videoDescText, setVideoDescText] = useState("Введіть опис відео");
  const [uploadDocText, setUploadDocText] = useState("Натисніть або перетягніть, щоб завантажити
      документ");
  const [supportedDocTypesText, setSupportedDocTypesText] = useState("Підтримуються: PDF, DOC, DOCX,
      XLS, PPT, зображення, текстові файли");
  const [maxDocSizeText, setMaxDocSizeText] = useState("Максимальний розмір: 50MB");
  const [docDescText, setDocDescText] = useState("Введіть опис документа");
  const [optionalText, setOptionalText] = useState("необов'язково");

  const [invalidVideoFormatText, setInvalidVideoFormatText] = useState("Файл повинен бути у відео
      форматі");
  const [videoSizeExceededText, setVideoSizeExceededText] = useState("Розмір файлу перевищує
      максимально допустимий розмір (200MB)");
  const [invalidDocFormatText, setInvalidDocFormatText] = useState("Несумісний формат файлу");
  const [docSizeExceededText, setDocSizeExceededText] = useState("Розмір файлу перевищує максимально
      допустимий розмір (50MB)");
  const [editPlaceholder, setEditPlaceholder] = useState("Редагувати");
  const [deletePlaceholder, setDeletePlaceholder] = useState("Видалити");

  const [lessonTitleError, setLessonTitleError] = useState<string | undefined>(undefined);
  const [lessonContentError, setLessonContentError] = useState<string | undefined>(undefined);

  const [testDraft, setTestDraft] = useState<TestType>(() => {
    if (lesson.type === 'TEST' && lesson.test) {
      return lesson.test;
    }
    return {} as TestType;
  });

  const [testSaved, setTestSaved] = useState<boolean>(lesson.type === 'TEST' && !!lesson.test);
```

```
const [isTestEditing, setIsTestEditing] = useState(
  lesson.type === 'TEST' && (!lesson.test || !Array.isArray(lesson.test.questions) ||
    lesson.test.questions.length === 0)
);

useEffect(() => {
  if (lesson.type === 'TEST' && lesson.testSaved !== testSaved) {
    updateLesson('testSaved', testSaved);
  }
}, [testSaved, lesson.type]);

useEffect(() => {
  if (lesson.type === 'TEST' && lesson.testSaved === undefined) {
    setTestSaved(false);
    updateLesson('testSaved', false);
  }
}, []);

useEffect(() => {
  if (lesson.type === 'TEST' && lesson.test && Array.isArray(lesson.test.questions) &&
    lesson.test.questions.length > 0) {
    setTestSaved(true);
  }
}, []);

useEffect(() => {
  const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
    'ua';
  setLang(currentLang === 'en' ? 'en' : 'ua');
}, []);

useEffect(() => {
  setTranslation("Назва уроку", setLessonTitlePlaceholder);
  setTranslation("Вміст уроку", setTextContentPlaceholder);
  setTranslation("Текст", setTypeText);
  setTranslation("Відео", setTypeVideo);
  setTranslation("Документ", setTypeDocument);
  setTranslation("Тест", setTypeTest);
  setTranslation("Перемістити урок вгору", setMoveUpTitle);
  setTranslation("Перемістити урок вниз", setMoveDownTitle);
  setTranslation("Видалити урок", setRemoveLessonTitle);
  setTranslation("Видалити файл", setDeleteFileText);
  setTranslation("Натисніть або перетягніть, щоб завантажити відео", setUploadVideoText);
  setTranslation("Максимальний розмір: 200MB", setMaxVideoSizeText);
  setTranslation("Введіть опис відео", setVideoDescText);
  setTranslation("Натисніть або перетягніть, щоб завантажити документ", setUploadDocText);
  setTranslation("Підтримуються: PDF, DOC, DOCX, XLS, PPT, зображення, текстові файли",
    setSupportedDocTypesText);
  setTranslation("Максимальний розмір: 50MB", setMaxDocSizeText);
  setTranslation("Введіть опис документа", setDocDescText);
  setTranslation("необов'язково", setOptionalText);
  setTranslation("Файл повинен бути у відео форматі", setInvalidVideoFormatText);
  setTranslation("Розмір файлу перевищує максимально допустимий розмір (200MB)",
    setVideoSizeExceededText);
  setTranslation("Несумісний формат файлу", setInvalidDocFormatText);
  setTranslation("Розмір файлу перевищує максимально допустимий розмір (50MB)",
    setDocSizeExceededText);
  setTranslation("Редагувати", setEditPlaceholder);
  setTranslation("Видалити", setDeletePlaceholder);
}, [lang]);

useEffect(() => {
  if (sectionIdx !== undefined && lessonIdx !== undefined) {
    const titleErrorKey = `lesson-${sectionIdx}-${lessonIdx}`;
    const contentErrorKey = `content-${sectionIdx}-${lessonIdx}`;

    if (errors && errors[titleErrorKey]) {
      t(errors[titleErrorKey], lang).then(setLessonTitleError);
    } else if (errors && errors.lesson) {
      t(errors.lesson, lang).then(setLessonTitleError);
    } else {
      setLessonTitleError(undefined);
    }

    if (errors && errors[contentErrorKey]) {
      t(errors[contentErrorKey], lang).then(setLessonContentError);
    } else if (errors && errors.content) {
      t(errors.content, lang).then(setLessonContentError);
    } else {
      setLessonContentError(undefined);
    }
  } else {
    if (errors && errors.lesson) {
      t(errors.lesson, lang).then(setLessonTitleError);
    } else {
      setLessonTitleError(undefined);
    }

    if (errors && errors.content) {
      t(errors.content, lang).then(setLessonContentError);
    } else {
      setLessonContentError(undefined);
    }
  }
}, [errors, lang, sectionIdx, lessonIdx]);

useEffect(() => {
```

```
  if (!lesson.tempFileName) {
    if (lesson.type === 'VIDEO' && lesson.videoUrl) {
      const name = getFileNameFromUrl(lesson.videoUrl, 'video');
      updateLesson('tempFileName', name);
    } else if (lesson.type === 'DOCUMENT' && lesson.documentUrl) {
      const name = getFileNameFromUrl(lesson.documentUrl, 'document');
      updateLesson('tempFileName', name);
    }
  }
}, [lesson.type, lesson.videoUrl, lesson.documentUrl, lesson.tempFileName, updateLesson]);

const handleTitleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  updateLesson('title', e.target.value);
};

const handleContentChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
  updateLesson('content', e.target.value);
};

const handleVideoUpload = (e: React.ChangeEvent<HTMLInputElement>) => {
  updateLesson('uploadError', '');
  const file = e.target.files?.[0];
  if (!file) return;
  const validation = validateVideoFile(file, invalidVideoFormatText, videoSizeExceededText);
  if (!validation.valid) {
    updateLesson('uploadError', validation.error);
    e.target.value = '';
    return;
  }
  updateLesson('tempFileName', file.name);
  handleFileUpload(file);
};

const handleDocumentUpload = (e: React.ChangeEvent<HTMLInputElement>) => {
  updateLesson('uploadError', '');
  const file = e.target.files?.[0];
  if (!file) return;
  const validation = validateDocumentFile(file, invalidDocFormatText, docSizeExceededText);
  if (!validation.valid) {
    updateLesson('uploadError', validation.error);
    e.target.value = '';
    return;
  }
  updateLesson('tempFileName', file.name);
  handleFileUpload(file);
};

const handleDeleteFile = () => {
  updateLesson('uploadError', '');
  resetFileFields(lesson.type as 'VIDEO' | 'DOCUMENT');
  clearLessonError();
  if (typeof clearPendingFile === 'function') {
    clearPendingFile();
  }
};

const handleTypeChange = (e: React.ChangeEvent<HTMLSelectElement>) => {
  clearLessonError();
  resetLessonFields();
  updateLesson('type', e.target.value);
  if (e.target.value === 'TEST') {
    setIsTestEditing(true);
    setTestSaved(false);
    setTestDraft({} as TestType);
  }
  const event = new CustomEvent('lessonTypeChanged', {
    detail: { lessonId: lesson.id }
  });
  document.dispatchEvent(event);
};

const handleDescriptionChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
  updateLesson('content', e.target.value);
};

const handleTestChange = (test: TestType) => {
  setTestDraft(test);
  setTestSaved(false);
  updateLesson('content', JSON.stringify(test));
  clearLessonError();
};

const handleSaveTest = (test: TestType) => {
  setTestDraft(test);
  updateLesson('test', test);
  setTestSaved(true);
  setIsTestEditing(false);
  if (setNotification) {
    setNotification({ type: 'success', message: 'Тест успішно збережено!' });
  }
  clearLessonError();
};

const handleEditTest = () => {
  setIsTestEditing(true);
  setTestSaved(false);
  clearLessonError();
```

```
  };

  const handleDeleteTest = () => {
    setTestDraft({} as TestType);
    updateLesson('content', '');
    setTestSaved(false);
    setIsTestEditing(true);
    clearLessonError();
    if (typeof sectionIdx === 'number' && typeof lessonIdx === 'number') {
      updateLesson('errors', {
        ...((lesson as any).errors || {}),
        [`content-${sectionIdx}-${lessonIdx}`]: undefined
      });
    }
  };

  const handleRemoveLesson = () => {
    clearLessonError();
    removeLesson();
  };

  return children({
    lesson,
    updateLesson,
    removeLesson: handleRemoveLesson,
    handleFileUpload,
    moveUp,
    moveDown,
    isLoading,
    lessonTitlePlaceholder,
    textContentPlaceholder,
    typeText,
    typeVideo,
    typeDocument,
    typeTest,
    moveUpTitle,
    moveDownTitle,
    removeLessonTitle,
    errors,
    lessonTitleError,
    lessonContentError,
    deleteFileText,
    uploadVideoText,
    maxVideoSizeText,
    videoDescText,
    uploadDocText,
    supportedDocTypesText,
    maxDocSizeText,
    docDescText,
    optionalText,
    handleTitleChange,
    handleContentChange,
    handleVideoUpload,
    handleDocumentUpload,
    handleDeleteFile,
    handleTypeChange,
    handleDescriptionChange,
    handleTestChange,
    formatFileName,
    clearLessonContentError,
    testDraft,
    setTestDraft,
    isTestEditing,
    setIsTestEditing,
    testSaved,
    editPlaceholder,
    deletePlaceholder,
    handleEditTest,
    handleDeleteTest,
    handleSaveTest,
  });
};

export default LessonLogic;
```

LessonUI.tsx

```
import React, { useState, useEffect } from 'react';
import { Lesson } from '../types';
import { TranslatedText } from '@/components/TranslatedText';
import Test from '../test/Test';
import { Test as TestType } from '../test/types';

interface LessonUIProps {
  lesson: Lesson;
  removeLesson: () => void;
  moveUp?: () => void;
  moveDown?: () => void;
  isLoading?: boolean;
  lessonTitlePlaceholder: string;
  textContentPlaceholder: string;
  typeText: string;
  typeVideo: string;
  typeDocument: string;
  typeTest: string;
  moveUpTitle: string;
  moveDownTitle: string;
```

```
  removeLessonTitle: string;
  lessonTitleError?: string;
  lessonContentError?: string;
  deleteFileText: string;
  uploadVideoText: string;
  maxVideoSizeText: string;
  videoDescText: string;
  uploadDocText: string;
  supportedDocTypesText: string;
  maxDocSizeText: string;
  docDescText: string;
  optionalText: string;
  handleTitleChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleContentChange: (e: React.ChangeEvent<HTMLTextAreaElement>) => void;
  handleVideoUpload: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleDocumentUpload: (e: React.ChangeEvent<HTMLInputElement>) => void;
  handleDeleteFile: () => void;
  handleTypeChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
  handleDescriptionChange: (e: React.ChangeEvent<HTMLTextAreaElement>) => void;
  formatFileName: (fileName: string) => string;
  clearLessonContentError?: (sectionIdx: number, lessonIdx: number) => void;
  lessonIdx?: number;
  sectionIdx?: number;
  handleTestChange?: (test: TestType) => void;
}

const LessonUI: React.FC<LessonUIProps & {
  testDraft?: TestType;
  setTestDraft?: React.Dispatch<React.SetStateAction<TestType>>;
  isTestEditing?: boolean;
  editPlaceholder: string;
  deletePlaceholder: string;
  setIsTestEditing?: React.Dispatch<React.SetStateAction<boolean>>;
  testSaved?: boolean;
  handleEditTest?: () => void;
  handleDeleteTest?: () => void;
  handleSaveTest?: (test: TestType) => void;
  validateAndSaveTest?: (test: TestType) => boolean;
  showTestError?: boolean;
  setShowTestError?: React.Dispatch<React.SetStateAction<boolean>>;
}> = ({
  lesson,
  removeLesson,
  moveUp,
  moveDown,
  isLoading,
  lessonTitlePlaceholder,
  textContentPlaceholder,
  typeText,
  typeVideo,
  typeDocument,
  typeTest,
  moveUpTitle,
  moveDownTitle,
  removeLessonTitle,
  lessonTitleError,
  lessonContentError,
  deleteFileText,
  uploadVideoText,
  maxVideoSizeText,
  videoDescText,
  uploadDocText,
  supportedDocTypesText,
  maxDocSizeText,
  docDescText,
  optionalText,
  handleTitleChange,
  handleContentChange,
  handleVideoUpload,
  handleDocumentUpload,
  handleDeleteFile,
  handleTypeChange,
  handleDescriptionChange,
  formatFileName,
  clearLessonContentError,
  lessonIdx,
  sectionIdx,
  testDraft,
  setTestDraft,
  isTestEditing,
  editPlaceholder,
  deletePlaceholder,
  setIsTestEditing,
  testSaved,
  handleEditTest,
  handleDeleteTest,
  handleSaveTest,
}) => {
  const [showDeleteModal, setShowDeleteModal] = useState(false);

  useEffect(() => {
    if (lesson.type !== 'TEST') {
      setIsTestEditing?.(false);
    }
  }, [lesson.type]);

  const handleDeleteClick = (e: React.MouseEvent) => {
```

```
  e.stopPropagation();
  setShowDeleteModal(true);
};

const handleConfirmDelete = () => {
  handleDeleteFile();
  if (typeof sectionIdx === 'number' && typeof lessonIdx === 'number' && clearLessonContentError)
    {
    clearLessonContentError(sectionIdx, lessonIdx);
  }
  setShowDeleteModal(false);
};

const handleCancelDelete = () => {
  setShowDeleteModal(false);
};

return (
  <>
    {showDeleteModal && (
      <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
        <div className="bg-white rounded-lg shadow-lg p-6 max-w-md w-full">
          <div className="mb-4 text-lg font-semibold text-gray-800"><TranslatedText text="Ви
впевнені, що хочете прибрати цей файл?" /></div>
          <div className="flex justify-end gap-2">
            <button onClick={handleCancelDelete} className="px-4 py-2 rounded bg-gray-200
hover:bg-gray-300 text-gray-800"><TranslatedText text="Назад" /></button>
            <button onClick={handleConfirmDelete} className="px-4 py-2 rounded bg-red-500
hover:bg-red-600 text-white"><TranslatedText text="Так" /></button>
          </div>
        </div>
      </div>
    )}
    <div className="p-2 md:p-3 border border-gray-200 rounded-lg shadow-sm mb-3 bg-white w-full
max-w-full [@media(max-width:300px)]:p-0.5">
      <div className="flex flex-col md:flex-row md:justify-between md:items-center mb-2 space-y-2
md:space-y-0 md:space-x-2">
        <div className="flex-1">
          <input
            type="text"
            value={lesson.title}
            onChange={handleTitleChange}
            className={`w-full border ${lessonTitleError ? 'border-red-500' : 'border-gray-300'}
rounded-md shadow-sm py-2 px-2 md:px-3 text-sm md:text-base focus:outline-none focus:ring-
indigo-500 focus:border-indigo-500 `}
            placeholder={lessonTitlePlaceholder}
            maxLength={100}
          />
          <div className="flex justify-between">
            {lessonTitleError && (
              <div className="text-xs text-red-500 mt-1">{lessonTitleError}</div>
            )}
            <div className="text-xs text-gray-500 mt-1 ml-auto">{lesson.title.length}/100</div>
          </div>
        </div>
        <div className="flex items-center md:ml-3">
          {moveUp && (
            <button
              type="button"
              onClick={moveUp}
              className="text-gray-400 hover:text-gray-600 mx-1 p-1 hover:bg-gray-100 rounded"
              title={moveUpTitle}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
fill="currentColor">
                <path fillRule="evenodd" d="M5.293 7.707a1 1 0 010-1.414l4-4a1 1 0 011.414 0l4 4a1
1 0 01-1.414 1.414L11 5.414V17a1 1 0 11-2 0V5.414L6.707 7.707a1 1 0 01-1.414 0z"
clipRule="evenodd" />
              </svg>
            </button>
          )}
          {moveDown && (
            <button
              type="button"
              onClick={moveDown}
              className="text-gray-400 hover:text-gray-600 mx-1 p-1 hover:bg-gray-100 rounded"
              title={moveDownTitle}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
fill="currentColor">
                <path fillRule="evenodd" d="M14.707 12.293a1 1 0 010 1.414l-4 4a1 1 0 01-1.414 0l-
4-4a1 1 0 111.414-1.414L9 14.586V3a1 1 0 012 0v11.586l2.293-2.293a1 1 0 011.414 0z"
clipRule="evenodd" />
              </svg>
            </button>
          )}
          <button
            type="button"
            onClick={removeLesson}
            className="text-red-500 hover:text-red-700 ml-2 p-1 hover:bg-red-50 rounded"
            title={removeLessonTitle}
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
fill="currentColor">
              <path fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000 2v10a2 2 0
002 2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1 1 0 012 0v6a1 1 0
11-2 0V8zm5-1a1 1 0 00-1 1v4a1 1 0 102 0V8a1 1 0 00-1-1z" clipRule="evenodd" />
```

```
        </svg>
      </button>
    </div>
  </div>

  <div className="mb-3">
    <select
      value={lesson.type}
      onChange={handleTypeChange}
      className="block w-full mt-1 text-sm border border-gray-300 rounded-md shadow-sm py-2
px-3 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500"
    >
      <option value="TEXT">{typeText}</option>
      <option value="VIDEO">{typeVideo}</option>
      <option value="DOCUMENT">{typeDocument}</option>
      <option value="TEST">{typeTest}</option>
    </select>
  </div>

  {lesson.type === 'TEXT' && (
    <div className="relative">
      <textarea
        value={lesson.content || ''}
        onChange={handleContentChange}
        className={`w-full mt-1 text-sm border ${lesson.type === 'TEXT' && lessonContentError
? 'border-red-500' : 'border-gray-300'} rounded-md shadow-sm py-2 px-3 focus:outline-none
focus:ring-indigo-500 focus:border-indigo-500 overflow-y-auto overscroll-contain`}
        rows={4}
        placeholder={textContentPlaceholder}
        maxLength={5000}
      />
      <div className="flex justify-between">
        {lesson.type === 'TEXT' && lessonContentError && (
          <div className="text-xs text-red-500 mt-1">{lessonContentError}</div>
        )}
        <div className="text-xs text-gray-500 mt-1 ml-auto">
          {(lesson.content?.length || 0)}/5000
        </div>
      </div>
    </div>
  )}

  {lesson.type === 'VIDEO' && (
    <div className="mt-2 p-3 border border-gray-100 rounded-lg bg-gray-50">
      <div className="flex items-center mb-2">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 text-indigo-500 mr-2"
fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M7 4v16M17
4v16M3 8h4m10 0h4M3 12h18M3 16h4m10 0h4M4 20h16a1 1 0 001-1V5a1 1 0 00-1-1H4a1 1 0 00-1 1v14a1
1 0 001 1z" />
        </svg>
        <span className="text-sm font-medium"><TranslatedText text="Відео файл" /></span>
      </div>

      {lesson.tempFileName && !isLoading ? (
        <div className="border border-gray-300 rounded-md p-3 bg-white relative flex items-
center">
          <span className="text-sm text-gray-700 truncate min-w-0 pr-8"
title={lesson.tempFileName || ''}>{formatFileName(lesson.tempFileName || '')}</span>
          <button
            type="button"
            onClick={handleDeleteClick}
            className="absolute right-3 top-1/2 -translate-y-1/2 text-red-500 hover:text-red-
700 bg-white z-10 p-1 rounded-full"
            title={deleteFileText}
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
fill="currentColor">
              <path fillRule="evenodd" d="M4.293 4.293a1 1 0 011.414 0L10 8.586l4.293-4.293a1
1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414 1.414L10 11.414l-4.293 4.293a1 1 0 01-
1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z" clipRule="evenodd" />
            </svg>
          </button>
        </div>
      ) : isLoading && lesson.tempFileName ? (
        <div className="border border-gray-300 rounded-md p-3 bg-white flex items-center
justify-between">
          <div className="flex items-center">
            <span className="text-sm text-gray-700 truncate overflow-hidden whitespace-nowrap
max-w-[120px]" title={lesson.tempFileName || ''}>{formatFileName(lesson.tempFileName ||
'')}</span>
          </div>
          <span className="animate-spin h-5 w-5 mr-2 border-2 border-blue-400 border-t-
transparent rounded-full"></span>
        </div>
      ) : (
        <div className={`border border-dashed ${lessonContentError ? 'border-red-500' :
'border-gray-300'} rounded-md p-6 flex flex-col items-center justify-center bg-white relative
h-full text-center`}>
          <svg xmlns="http://www.w3.org/2000/svg" className="h-10 w-10 text-gray-500 mb-3"
fill="none" viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M7 4v16M17
4v16M3 8h4m10 0h4M3 12h18M3 16h4m10 0h4M4 20h16a1 1 0 001-1V5a1 1 0 00-1-1H4a1 1 0 00-1 1v14a1
1 0 001 1z" />
          </svg>
          <p className="text-sm text-center text-gray-700 mb-1 [@media(max-
width:275px)]:hidden">
```

```
            {uploadVideoText}
          </p>
          <p className="text-xs text-center text-gray-500 mb-3 [@media(max-
width:275px)]:hidden">
            {maxVideoSizeText}
          </p>
          <span className="inline [@media(min-width:276px)]:hidden text-xs text-gray-500 mb-3
flex items-center justify-center">
            <span className="ml-1">&lt;200MB</span>
          </span>
          <input
            type="file"
            accept="video/*"
            onChange={handleVideoUpload}
            className="opacity-0 absolute inset-0 w-full h-full cursor-pointer"
          />
        </div>
      )}
      {isLoading && (
        <div className="mt-2 flex items-center text-blue-500">
          <svg className="animate-spin -ml-1 mr-2 h-4 w-4" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24">
            <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
            <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0
5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
          </svg>
        </div>
      )}
      {!isLoading && lesson.videoUrl && (
        <div className="mt-2 flex items-center text-green-500">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0
00-1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z"
clipRule="evenodd" />
          </svg>
          <span className="text-xs"><TranslatedText text="Файл успішно завантажено" /></span>
        </div>
      )}
      {lesson.type === 'VIDEO' && lessonContentError && !lesson.videoUrl &&
!lesson.tempFileName && !isLoading && (
        <div className="mt-2 text-xs text-red-500 p-2 bg-red-50 rounded-md flex items-center">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1
1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
          </svg>
          <span className="font-medium">{lessonContentError}</span>
        </div>
      )}
      {lesson.uploadError && (
        <div className="mt-2 flex items-center text-red-500 bg-red-50 p-2 rounded-md">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1
1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
          </svg>
          <span className="text-xs font-medium">{lesson.uploadError}</span>
        </div>
      )}

      <div className="mt-3">
        <label className="block text-sm font-medium text-gray-700 mb-1 flex flex-wrap items-
center gap-x-1">
          <TranslatedText text="Опис" />
          <span className="text-gray-400 text-xs">(<TranslatedText text={optionalText}
/>)</span>
        </label>
        <textarea
          value={lesson.content || ''}
          onChange={handleDescriptionChange}
          className={`w-full text-sm border border-gray-300 rounded-md shadow-sm py-2 px-3
focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 overflow-y-auto overscroll-
contain`}
          rows={3}
          placeholder={videoDescText}
          maxLength={500}
        />
        <div className="text-xs text-gray-500 mt-1 text-right">
          {(lesson.content?.length || 0)}/500
        </div>
      </div>
    </div>
  )}

  {lesson.type === 'DOCUMENT' && (
    <div className="mt-2 p-3 border border-gray-100 rounded-md bg-gray-50">
      <div className="flex items-center mb-2">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 text-indigo-500 mr-2"
fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 12h6m-6
4h6m2 5H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2
2 0 01-2 2z" />
        </svg>
        <span className="text-sm font-medium"><TranslatedText text="Документ" /></span>
      </div>
```

```
      {lesson.tempFileName && !isLoading ? (
        <div className="border border-gray-300 rounded-md p-3 bg-white relative flex items-
center">
          <span className="text-sm text-gray-700 truncate min-w-0 pr-8"
title={lesson.tempFileName || ''}>{formatFileName(lesson.tempFileName || '')}</span>
          <button
            type="button"
            onClick={handleDeleteClick}
            className="absolute right-3 top-1/2 -translate-y-1/2 text-red-500 hover:text-red-
700 bg-white z-10 p-1 rounded-full"
            title={deleteFileText}
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
fill="currentColor">
              <path fillRule="evenodd" d="M4.293 4.293a1 1 0 011.414 0L10 8.586l4.293-4.293a1
1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414 1.414L10 11.414l-4.293 4.293a1 1 0 01-
1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z" clipRule="evenodd" />
            </svg>
          </button>
        </div>
      ) : lesson.tempFileName && isLoading ? (
        <div className="border border-gray-300 rounded-md p-3 bg-white flex items-center
justify-between">
          <div className="flex items-center">
            <span className="text-sm text-gray-700 truncate overflow-hidden whitespace-nowrap
max-w-[120px]" title={lesson.tempFileName || ''}>{formatFileName(lesson.tempFileName ||
'')}</span>
          </div>
          <span className="animate-spin h-5 w-5 mr-2 border-2 border-blue-400 border-t-
transparent rounded-full"></span>
        </div>
      ) : (
        <div className={`border border-dashed ${lessonContentError ? 'border-red-500' :
'border-gray-300'} rounded-md p-6 flex flex-col items-center justify-center bg-white
relative `}>
          <svg xmlns="http://www.w3.org/2000/svg" className="h-10 w-10 text-gray-500 mb-3"
fill="none" viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 12h6m-6
4h6m2 5H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2
2 0 01-2 2z" />
          </svg>
          <p className="text-sm text-center text-gray-700 mb-1 [@media(max-
width:280px)]:hidden">
            {uploadDocText}
          </p>
          <p className="text-xs text-center text-gray-500 mb-3 [@media(max-
width:280px)]:hidden">
            {supportedDocTypesText}
          </p>
          <p className="text-xs text-center text-gray-500 mb-3 [@media(max-
width:280px)]:hidden">
            {maxDocSizeText}
          </p>
          <span className="inline [@media(min-width:281px)]:hidden text-xs text-gray-500 mb-3
flex items-center justify-center">
            <span className="ml-1">&lt;50MB</span>
          </span>
          <input
            type="file"
            accept=".pdf,.doc,.docx,.ppt,.pptx,.xls,.xlsx,image/*,.txt"
            onChange={handleDocumentUpload}
            className="opacity-0 absolute inset-0 w-full h-full cursor-pointer"
          />
        </div>
      )}
      {isLoading && (
        <div className="mt-2 flex items-center text-blue-500">
          <svg className="animate-spin -ml-1 mr-2 h-4 w-4" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24">
            <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
            <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0
5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
          </svg>
        </div>
      )}
      {!isLoading && lesson.documentUrl && (
        <div className="mt-2 flex items-center text-green-500">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0
00-1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z"
clipRule="evenodd" />
          </svg>
          <span className="text-xs"><TranslatedText text="Файл успішно завантажено" /></span>
        </div>
      )}
      {lesson.type === 'DOCUMENT' && lessonContentError && !lesson.documentUrl &&
!lesson.tempFileName && !isLoading && (
        <div className="mt-2 text-xs text-red-500 p-2 bg-red-50 rounded-md flex items-center">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1
1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
          </svg>
          <span className="font-medium">{lessonContentError}</span>
```

```
            </div>
          )}
          {lesson.uploadError && (
            <div className="mt-2 flex items-center text-red-500 bg-red-50 p-2 rounded-md">
              <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
                <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1
1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
              </svg>
              <span className="text-xs font-medium">{lesson.uploadError}</span>
            </div>
          )}

          <div className="mt-3">
            <label className="block text-sm font-medium text-gray-700 mb-1 flex flex-wrap items-
center gap-x-1">
              <TranslatedText text="Опис" />
              <span className="text-gray-400 text-xs">(<TranslatedText text={optionalText}
/>)</span>
            </label>
            <textarea
              value={lesson.content || ''}
              onChange={handleDescriptionChange}
              className={`w-full text-sm border border-gray-300 rounded-md shadow-sm py-2 px-3
focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 overflow-y-auto overscroll-
contain`}
              rows={3}
              placeholder={docDescText}
              maxLength={500}
            />
            <div className="text-xs text-gray-500 mt-1 text-right">
              {(lesson.content?.length || 0)}/500
            </div>
          </div>
        </div>
      )}

      {lesson.type === 'TEST' && (
        <>
          {testSaved && !isTestEditing ? (
            <div className="relative flex items-center bg-blue-50 border border-blue-200 rounded
p-2">
              <svg className="h-6 w-6 text-blue-500 mr-2 mb-2 shrink-0" fill="none" viewBox="0 0
24 24" stroke="currentColor">
                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 12h6m-3-
3v6m9 2a9 9 0 11-18 0 9 9 0 0118 0z" />
              </svg>
              <span className="font-medium text-blue-700 truncate min-w-0 pr-14"><TranslatedText
text="Тест збережено" /></span>
              <div className="absolute right-3 top-1/2 -translate-y-1/2 flex gap-1 z-10">
                <button type="button" onClick={handleEditTest} title={editPlaceholder}
className="p-1 text-gray-500 hover:text-blue-600 flex items-center">
                  <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 [@media(max-
width:300px)]:h-4 [@media(max-width:300px)]:w-4" fill="none" viewBox="0 0 24 24"
stroke="currentColor">
                    <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M15.232
5.232l3.536 3.536M9 13h3l8-8a2.828 2.828 0 00-4-4l-8 8v3h3z" />
                  </svg>
                </button>
                <button type="button" onClick={handleDeleteTest} title={deletePlaceholder}
className="p-1 text-red-500 hover:text-red-700 flex items-center">
                  <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 [@media(max-
width:300px)]:h-4 [@media(max-width:300px)]:w-4" fill="none" viewBox="0 0 24 24"
stroke="currentColor">
                    <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M6 18L18
6M6 6l12 12" />
                  </svg>
                </button>
              </div>
            </div>
          ) : null}
          {isTestEditing && testDraft && setTestDraft && handleSaveTest && (
            <div className="bg-white border border-blue-100 rounded p-3 mt-2">
              <Test
                initialTest={testDraft}
                onChange={setTestDraft}
                errors={{}}
                onSaveTest={() => handleSaveTest(testDraft)}
              />
            </div>
          )}
          {lesson.type === 'TEST' && !testSaved && (
            <div className="mt-2 text-xs text-red-500 p-2 bg-red-50 rounded-md flex items-center">
              <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
                <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1
1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
              </svg>
              <span className="font-medium"><TranslatedText text="Тест не збережено" /></span>
            </div>
          )}
        </>
      )}
    </div>
  );
};
```

```
export default LessonUI;
```

Lesson.tsx

```
import React from 'react';
import LessonLogic from './LessonLogic';
import LessonUI from './LessonUI';
import { Lesson as LessonType } from '../types';

interface LessonProps {
  lesson: LessonType;
  updateLesson: (field: string, value: any) => void;
  removeLesson: () => void;
  handleFileUpload: (file: File) => void;
  moveUp?: () => void;
  moveDown?: () => void;
  isLoading?: boolean;
  errors?: Record<string, string>;
  clearPendingFile?: () => void;
  lessonIdx?: number;
  sectionIdx?: number;
  clearLessonContentError?: (sectionIdx: number, lessonIdx: number) => void;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
}

const Lesson: React.FC<LessonProps> = ({
  lesson,
  updateLesson,
  removeLesson,
  handleFileUpload,
  moveUp,
  moveDown,
  isLoading,
  errors,
  clearPendingFile,
  lessonIdx,
  sectionIdx,
  clearLessonContentError,
  setNotification
}) => {
  const handleClearPendingFile = () => {
    if (typeof clearPendingFile === 'function') {
      clearPendingFile();
    }
  };

  return (
    <LessonLogic
      lesson={lesson}
      updateLesson={updateLesson}
      removeLesson={removeLesson}
      handleFileUpload={handleFileUpload}
      moveUp={moveUp}
      moveDown={moveDown}
      isLoading={isLoading}
      errors={errors}
      clearPendingFile={handleClearPendingFile}
      lessonIdx={lessonIdx}
      sectionIdx={sectionIdx}
      clearLessonContentError={clearLessonContentError}
      setNotification={setNotification}
    >
      {(lessonProps) => <LessonUI {...lessonProps} />}
    </LessonLogic>
  );
};

export default Lesson;
```

StepLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { t } from '@/components/TranslatedText';

interface StepLogicProps {
  step: any;
  stepIdx: number;
  updateStep: (idx: number, field: string, value: any) => void;
  removeStep: (idx: number) => void;
  handleFileUpload: (stepIdx: number, file: File) => void;
  moveStep: (fromIdx: number, toIdx: number) => void;
  stepsLength: number;
  errors: Record<string, string>;
  setNotification: (n: any) => void;
  clearStepError?: (key: string) => void;
  fileLoading?: boolean;
  children: (props: {
    videoPreviewUrl: string;
    onVideoChange: (file: File) => void;
    onTitleChange: (v: string) => void;
    onContentChange: (v: string) => void;
    onRemove: () => void;
    onMoveUp: () => void;
    onMoveDown: () => void;
    title: string;
```

```
    content: string;
    titleError?: string;
    contentError?: string;
    videoError?: string;
    canMoveUp: boolean;
    canMoveDown: boolean;
    onRemoveVideo: () => void;
    fileLoading?: boolean;
    step: any;
    titlePlaceholder: string;
    contentPlaceholder: string;
    moveUpTitle: string;
    moveDownTitle: string;
    removeTitle: string;
  }) => React.ReactNode;
}

async function getVideoThumbnail(file: File): Promise<string | null> {
  return new Promise((resolve) => {
    const video = document.createElement('video');
    video.preload = 'metadata';
    video.muted = true;
    video.src = URL.createObjectURL(file);

    video.onloadedmetadata = () => {
      video.currentTime = 0;
    };

    video.onseeked = () => {
      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      const ctx = canvas.getContext('2d');
      if (ctx) {
        ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
        resolve(canvas.toDataURL('image/png'));
      } else {
        resolve(null);
      }
      URL.revokeObjectURL(video.src);
    };

    video.onerror = () => resolve(null);
  });
}

const StepLogic: React.FC<StepLogicProps> = ({
  step,
  stepIdx,
  updateStep,
  removeStep,
  handleFileUpload,
  moveStep,
  stepsLength,
  errors,
  setNotification,
  clearStepError,
  fileLoading = false,
  children
}) => {
  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [titleError, setTitleError] = useState<string | undefined>(undefined);
  const [contentError, setContentError] = useState<string | undefined>(undefined);
  const [videoError, setVideoError] = useState<string | undefined>(undefined);

  const [titlePlaceholder, setTitlePlaceholder] = useState<string>('Введіть назву кроку');
  const [contentPlaceholder, setContentPlaceholder] = useState<string>('Введіть опис кроку');
  const [moveUpTitle, setMoveUpTitle] = useState<string>('Перемістити вгору');
  const [moveDownTitle, setMoveDownTitle] = useState<string>('Перемістити вниз');
  const [removeTitle, setRemoveTitle] = useState<string>('Видалити крок');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t('Введіть назву кроку', lang).then(setTitlePlaceholder);
    t('Введіть опис кроку', lang).then(setContentPlaceholder);
    t('Перемістити вгору', lang).then(setMoveUpTitle);
    t('Перемістити вниз', lang).then(setMoveDownTitle);
    t('Видалити крок', lang).then(setRemoveTitle);
  }, [lang]);

  useEffect(() => {
    if (stepIdx !== undefined && errors) {
      const titleErrorKey = `step-title-${stepIdx}`;
      const contentErrorKey = `step-content-${stepIdx}`;
      const videoErrorKey = `step-videoUrl-${stepIdx}`;

      if (errors[titleErrorKey]) {
        t(errors[titleErrorKey], lang).then(setTitleError);
      } else {
        setTitleError(undefined);
      }
      if (errors[contentErrorKey]) {
```

```
        t(errors[contentErrorKey], lang).then(setContentError);
      } else {
        setContentError(undefined);
      }
      if (errors[videoErrorKey]) {
        t(errors[videoErrorKey], lang).then(setVideoError);
      } else {
        setVideoError(undefined);
      }
    }
  }, [errors, lang, stepIdx]);

  const onTitleChange = (v: string) => {
    updateStep(stepIdx, 'title', v);
    if (titleError) setTitleError(undefined);
    if (clearStepError) clearStepError(`step-title-${stepIdx}`);
  };

  const onContentChange = (v: string) => {
    updateStep(stepIdx, 'content', v);
    if (contentError) setContentError(undefined);
    if (clearStepError) clearStepError(`step-content-${stepIdx}`);
  };

  const onVideoChange = async (file: File) => {
    if (!file.type.startsWith('video/')) {
      setNotification({ type: 'error', message: 'Дозволені лише відеофайли' });
      return;
    }
    if (file.size > 50 * 1024 * 1024) {
      setNotification({ type: 'error', message: 'Максимальний розмір відео — 50MB' });
      return;
    }
    handleFileUpload(stepIdx, file);
    if (clearStepError) clearStepError(`step-videoUrl-${stepIdx}`);
    const thumb = await getVideoThumbnail(file);
    if (thumb) {
      updateStep(stepIdx, 'videoPreviewUrl', thumb);
    }
    if (videoError) setVideoError(undefined);
  };

  const onRemoveVideo = () => {
    updateStep(stepIdx, 'videoPreviewUrl', '');
    updateStep(stepIdx, 'videoUrl', '');
  };

  return children({
    videoPreviewUrl: step.videoPreviewUrl || '',
    onVideoChange,
    onTitleChange,
    onContentChange,
    onRemove: () => removeStep(stepIdx),
    onMoveUp: () => stepIdx > 0 && moveStep(stepIdx, stepIdx - 1),
    onMoveDown: () => stepIdx < stepsLength - 1 && moveStep(stepIdx, stepIdx + 1),
    title: step.title,
    content: step.content,
    titleError,
    contentError,
    videoError,
    canMoveUp: stepIdx > 0,
    canMoveDown: stepIdx < stepsLength - 1,
    onRemoveVideo,
    fileLoading,
    step,
    titlePlaceholder,
    contentPlaceholder,
    moveUpTitle,
    moveDownTitle,
    removeTitle
  });
};

export default StepLogic;
```

StepUI.tsx

```
import React, { useRef, useState } from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface StepUIProps {
  videoPreviewUrl: string;
  onVideoChange: (file: File) => void;
  onTitleChange: (v: string) => void;
  onContentChange: (v: string) => void;
  onRemove: () => void;
  onMoveUp: () => void;
  onMoveDown: () => void;
  title: string;
  content: string;
  canMoveUp?: boolean;
  canMoveDown?: boolean;
  onRemoveVideo?: () => void;
  titleError?: string;
  contentError?: string;
  videoError?: string;
  fileLoading?: boolean;
```

```
  step?: { videoUrl?: string };
  titlePlaceholder: string;
  contentPlaceholder: string;
  moveUpTitle: string;
  moveDownTitle: string;
  removeTitle: string;
}

const StepUI: React.FC<StepUIProps> = ({
  videoPreviewUrl,
  onVideoChange,
  onTitleChange,
  onContentChange,
  onRemove,
  onMoveUp,
  onMoveDown,
  title,
  content,
  canMoveUp = true,
  canMoveDown = true,
  onRemoveVideo,
  titleError,
  contentError,
  videoError,
  fileLoading = false,
  step,
  titlePlaceholder,
  contentPlaceholder,
  moveUpTitle,
  moveDownTitle,
  removeTitle
}) => {
  const fileInputRef = useRef<HTMLInputElement>(null);
  const dropRef = useRef<HTMLDivElement>(null);
  const [showDeleteModal, setShowDeleteModal] = useState(false);

  React.useEffect(() => {
    const dropArea = dropRef.current;
    if (!dropArea) return;
    const handleDragOver = (e: DragEvent) => {
      e.preventDefault();
      dropArea.classList.add('border-blue-500', 'bg-blue-50');
    };
    const handleDragLeave = (e: DragEvent) => {
      e.preventDefault();
      dropArea.classList.remove('border-blue-500', 'bg-blue-50');
    };
    const handleDrop = (e: DragEvent) => {
      e.preventDefault();
      dropArea.classList.remove('border-blue-500', 'bg-blue-50');
      if (e.dataTransfer?.files && e.dataTransfer.files.length > 0) {
        const file = e.dataTransfer.files[0];
        onVideoChange(file);
      }
    };
    dropArea.addEventListener('dragover', handleDragOver);
    dropArea.addEventListener('dragleave', handleDragLeave);
    dropArea.addEventListener('drop', handleDrop);
    return () => {
      dropArea.removeEventListener('dragover', handleDragOver);
      dropArea.removeEventListener('dragleave', handleDragLeave);
      dropArea.removeEventListener('drop', handleDrop);
    };
  }, [onVideoChange]);

  const handleDeleteClick = (e: React.MouseEvent) => {
    e.stopPropagation();
    setShowDeleteModal(true);
  };
  const handleConfirmDelete = () => {
    onRemoveVideo && onRemoveVideo();
    setShowDeleteModal(false);
  };
  const handleCancelDelete = () => {
    setShowDeleteModal(false);
  };

  return (
    <>
      {showDeleteModal && (
        <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-40">
          <div className="bg-white rounded-lg shadow-lg p-6 max-w-md w-full">
            <div className="mb-4 text-lg font-semibold text-gray-800"><TranslatedText text="Ви
впевнені, що хочете прибрати це відео?" /></div>
            <div className="flex justify-end gap-2">
              <button onClick={handleCancelDelete} className="px-4 py-2 rounded bg-gray-200
hover:bg-gray-300 text-gray-800"><TranslatedText text="Назад" /></button>
              <button onClick={handleConfirmDelete} className="px-4 py-2 rounded bg-red-500
hover:bg-red-600 text-white"><TranslatedText text="Так" /></button>
            </div>
          </div>
        </div>
      )}
      <div className="flex flex-col md:flex-row gap-4 p-4 bg-white rounded-lg shadow border border-
gray-200">
        <div className="flex-1 flex flex-col items-center justify-center max-w-[320px] relative min-
h-[130px] w-full mx-auto">
```

```
    <input
      ref={fileInputRef}
      type="file"
      accept="video/*"
      className="hidden"
      onChange={e => {
        if (e.target.files?.[0]) {
          onVideoChange(e.target.files[0]);
        }
      }}
    />
    {fileLoading && (
      <div className="absolute inset-0 flex items-center justify-center bg-white bg-opacity-70
z-10">
        <svg className="animate-spin h-8 w-8 text-blue-500" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24">
          <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
          <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0
5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
        </svg>
      </div>
    )}
    {step?.videoUrl || videoPreviewUrl ? (
      <div className="relative w-full h-32 flex items-center justify-center" onClick={() => {
fileInputRef.current?.click(); }} style={{ cursor: 'pointer' }}>
        <img
          src={videoPreviewUrl}
          alt="preview"
          className="w-full h-full object-cover rounded-lg border"
          style={{ maxHeight: 128, minHeight: 128 }}
        />
        <button
          type="button"
          className="absolute top-1 right-1 bg-white rounded-full p-1 shadow-md hover:bg-red-
100 transition-colors"
          onClick={e => { e.stopPropagation(); handleDeleteClick(e); }}
          title="Видалити відео"
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 text-red-600" viewBox="0
0 20 20" fill="currentColor">
            <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0
00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-
1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />
          </svg>
        </button>
      </div>
    ) : (
      <div
        ref={dropRef}
        className="flex flex-col items-center justify-center w-full h-32 border-2 border-
dashed border-gray-300 rounded-lg hover:border-blue-500 transition-colors cursor-pointer
select-none"
        onClick={() => fileInputRef.current?.click()}
      >
        <svg xmlns="http://www.w3.org/2000/svg" className="h-10 w-10 text-gray-400 mb-2"
fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M4 16v2a2 2 0
002 2h12a2 2 0 002-2v-2M7 10V6a5 5 0 0110 0v4M12 16v-4m0 0l-2 2m2-2l2 2" />
        </svg>
        <span className="text-xs text-gray-500 text-center [@media(max-
width:180px)]:hidden"><TranslatedText text="Завантажити відео" /></span>
        <span className="text-xs text-gray-500 ml-1">&lt;50MB</span>
      </div>
    )}
    {!fileLoading && videoError && (
      <div className="mt-2 flex items-center text-red-500 bg-red-50 p-2 rounded-md">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
          <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1 1
0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
        </svg>
        <span className="text-xs font-medium"><TranslatedText text="Файл не завантажено"
/></span>
      </div>
    )}
    {!fileLoading && step?.videoUrl && !videoError && (
      <div className="mt-2 flex items-center text-green-500">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-1" viewBox="0 0 20 20"
fill="currentColor">
          <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0 00-
1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z"
clipRule="evenodd" />
        </svg>
        <span className="text-xs"><TranslatedText text="Файл завантажено" /></span>
      </div>
    )}
  </div>
  <div className="flex-1 flex flex-col gap-2">
    <input
      type="text"
      value={title}
      onChange={e => onTitleChange(e.target.value)}
      placeholder={titlePlaceholder}
      className="w-full p-2 border rounded text-sm font-semibold border-gray-300"
      maxLength={100}
    />
```

```
          {titleError && <div className="text-red-500 text-xs mt-1">{titleError}</div>}
          <textarea
            value={content}
            onChange={e => onContentChange(e.target.value)}
            placeholder={contentPlaceholder}
            className="w-full p-2 border rounded text-sm border-gray-300 min-h-[60px] overflow-y-
      auto overscroll-contain"
            maxLength={1000}
          />
          {contentError && <div className="text-red-500 text-xs mt-1">{contentError}</div>}
          <div className="flex gap-2 mt-2 items-center">
            <button
              type="button"
              className={`px-2 py-1 bg-gray-200 rounded hover:bg-gray-300 ${!canMoveUp ? 'opacity-50
      cursor-not-allowed' : ''}`}
              onClick={canMoveUp ? onMoveUp : undefined}
              disabled={!canMoveUp}
              title={moveUpTitle}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" fill="none" viewBox="0 0
      24 24" stroke="currentColor">
                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M5 15l7-7 7 7"
      />
              </svg>
            </button>
            <button
              type="button"
              className={`px-2 py-1 bg-gray-200 rounded hover:bg-gray-300 ${!canMoveDown ? 'opacity-
      50 cursor-not-allowed' : ''}`}
              onClick={canMoveDown ? onMoveDown : undefined}
              disabled={!canMoveDown}
              title={moveDownTitle}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" fill="none" viewBox="0 0
      24 24" stroke="currentColor">
                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M19 9l-7 7-7-
      7" />
              </svg>
            </button>
            <button
              type="button"
              className="ml-auto px-2 py-1 bg-red-100 text-red-600 rounded hover:bg-red-200 flex
      items-center group"
              onClick={onRemove}
              title={removeTitle}
            >
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
                <path strokeLinecap="round" fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1
      1 0 000 2v10a2 2 0 002 2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1
      1 0 012 0v6a1 1 0 11-2 0V8zm5-1a1 1 0 00-1 1v6a1 1 0 102 0V8a1 1 0 00-1-1z" clipRule="evenodd"
      />
              </svg>
            </button>
          </div>
        </div>
      </div>
    </>
  );
};

export default StepUI;
```

Step.tsx

```
import React from 'react';
import StepLogic from './StepLogic';
import StepUI from './StepUI';

interface StepProps {
  step: any;
  stepIdx: number;
  updateStep: (idx: number, field: string, value: any) => void;
  removeStep: (idx: number) => void;
  handleFileUpload: (stepIdx: number, file: File) => void;
  moveStep: (fromIdx: number, toIdx: number) => void;
  stepsLength: number;
  errors: Record<string, string>;
  setNotification: (n: any) => void;
  clearStepError: (key: string) => void;
  fileLoading?: boolean;
  videoSuccess?: boolean;
}

const Step: React.FC<StepProps> = (props) => {
  return (
    <StepLogic {...props} fileLoading={props.fileLoading}>
      {(logicProps) => <StepUI {...logicProps} />}
    </StepLogic>
  );
};

export default Step;
```

form\test\types.ts

```
export interface TestQuestion {
```

```
  id: string;
  question: string;
  type: 'SINGLE' | 'MULTIPLE';
  options: TestOption[];
}

export interface TestOption {
  id: string;
  text: string;
  isCorrect: boolean;
}

export interface Test {
  id?: string;
  questions: TestQuestion[];
}

export interface TestFormState {
  questions: TestQuestion[];
  activeQuestionIndex: number;
  errors: Record<string, string>;
}
```

TestLogic.tsx

```
import React, { useState, useEffect } from 'react';
import { Test, TestQuestion, TestOption, TestFormState } from './types';
import { t } from '@/components/TranslatedText';
import { v4 as uuidv4 } from 'uuid';

interface TestLogicProps {
  initialTest?: Test;
  onChange: (test: Test) => void;
  errors?: Record<string, string>;
  children: (props: {
    state: TestFormState;
    addQuestion: () => void;
    removeQuestion: (index: number) => void;
    updateQuestion: (index: number, field: string, value: any) => void;
    addOption: (questionIndex: number) => void;
    removeOption: (questionIndex: number, optionIndex: number) => void;
    updateOption: (questionIndex: number, optionIndex: number, field: string, value: any) => void;
    setActiveQuestion: (index: number) => void;
    moveQuestionUp: (index: number) => void;
    moveQuestionDown: (index: number) => void;
    getSaveButtonStatus: () => boolean;
    translations: {
      questionPlaceholder: string;
      optionPlaceholder: string;
      addQuestionText: string;
      removeQuestionText: string;
      addOptionText: string;
      removeOptionText: string;
      singleChoiceText: string;
      multipleChoiceText: string;
      questionTypeLabel: string;
      correctAnswerText: string;
      moveUpText: string;
      moveDownText: string;
      noQuestionsText: string;
      questionLabelText: string;
    };
    errors: Record<string, string>;
    validateAndSaveTest: () => void;
    showErrors: boolean;
  }) => React.ReactNode;
  onSaveTest: (test: Test) => void;
}

const createEmptyQuestion = (type: 'SINGLE' | 'MULTIPLE' = 'SINGLE'): TestQuestion => ({
  id: uuidv4(),
  question: '',
  type,
  options: type === 'MULTIPLE'
    ? [
        { id: uuidv4(), text: '', isCorrect: true },
        { id: uuidv4(), text: '', isCorrect: true }
      ]
    : [
        { id: uuidv4(), text: '', isCorrect: true },
        { id: uuidv4(), text: '', isCorrect: false }
      ]
});

const TestLogic: React.FC<TestLogicProps> = ({
  initialTest,
  onChange,
  errors = {},
  children,
  onSaveTest
}) => {
  const [state, setState] = useState<TestFormState>({
    questions: initialTest?.questions && initialTest.questions.length > 0
      ? initialTest.questions
      : [createEmptyQuestion()],
    activeQuestionIndex: 0,
    errors: {}
```

```
});

const [translations, setTranslations] = useState({
  questionPlaceholder: 'Введіть питання',
  optionPlaceholder: 'Варіант відповіді',
  addQuestionText: 'Додати питання',
  removeQuestionText: 'Видалити питання',
  addOptionText: 'Додати варіант відповіді',
  removeOptionText: 'Видалити варіант',
  singleChoiceText: 'Одна правильна відповідь',
  multipleChoiceText: 'Декілька правильних відповідей',
  questionTypeLabel: 'Тип питання',
  correctAnswerText: 'Правильна відповідь',
  moveUpText: 'Перемістити вгору',
  moveDownText: 'Перемістити вниз',
  noQuestionsText: 'Немає питань. Додайте перше питання нижче.',
  questionLabelText: 'Питання'
});

const [lang, setLang] = useState<'ua' | 'en'>('ua');
const [showErrors, setShowErrors] = useState(false);

useEffect(() => {
  const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
    'ua';
  setLang(currentLang === 'en' ? 'en' : 'ua');
}, []);

useEffect(() => {
  t('Введіть питання', lang).then(text => setTranslations(prev => ({ ...prev, questionPlaceholder:
    text })));
  t('Варіант відповіді', lang).then(text => setTranslations(prev => ({ ...prev, optionPlaceholder:
    text })));
  t('Додати питання', lang).then(text => setTranslations(prev => ({ ...prev, addQuestionText: text
    })));
  t('Видалити питання', lang).then(text => setTranslations(prev => ({ ...prev, removeQuestionText:
    text })));
  t('Додати варіант відповіді', lang).then(text => setTranslations(prev => ({ ...prev,
    addOptionText: text })));
  t('Видалити варіант', lang).then(text => setTranslations(prev => ({ ...prev, removeOptionText:
    text })));
  t('Одна правильна відповідь', lang).then(text => setTranslations(prev => ({ ...prev,
    singleChoiceText: text })));
  t('Декілька правильних відповідей', lang).then(text => setTranslations(prev => ({ ...prev,
    multipleChoiceText: text })));
  t('Тип питання', lang).then(text => setTranslations(prev => ({ ...prev, questionTypeLabel: text
    })));
  t('Правильна відповідь', lang).then(text => setTranslations(prev => ({ ...prev,
    correctAnswerText: text })));
  t('Перемістити вгору', lang).then(text => setTranslations(prev => ({ ...prev, moveUpText: text
    })));
  t('Перемістити вниз', lang).then(text => setTranslations(prev => ({ ...prev, moveDownText: text
    })));
  t('Немає питань. Додайте перше питання нижче.', lang).then(text => setTranslations(prev => ({
    ...prev, noQuestionsText: text })));
  t('Питання', lang).then(text => setTranslations(prev => ({ ...prev, questionLabelText: text
    })));
}, [lang]);

const addQuestion = () => {
  setState(prev => {
    const type = prev.questions[0]?.type || 'SINGLE';
    const newQuestions = [...prev.questions, createEmptyQuestion(type)];
    const newState = {
      ...prev,
      questions: newQuestions,
      activeQuestionIndex: newQuestions.length - 1
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const removeQuestion = (index: number) => {
  setState(prev => {
    const newQuestions = [...prev.questions];
    newQuestions.splice(index, 1);

    if (newQuestions.length === 0) {
      newQuestions.push(createEmptyQuestion());
    }

    const newState = {
      ...prev,
      questions: newQuestions,
      activeQuestionIndex: Math.min(prev.activeQuestionIndex, newQuestions.length - 1)
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const validateAllQuestions = (questions: TestQuestion[]) => {
  const errors: Record<string, string> = {};
  questions.forEach((q, qIdx) => {
    if (!q.question.trim()) {
      errors[`question-${qIdx}`] = 'Назва питання не може бути порожньою';
```

```
    }
    if (q.options.some(opt => !opt.text.trim())) {
      errors[`option-${qIdx}`] = 'Усі варіанти повинні бути заповнені';
    }
  });
  return errors;
};

const updateQuestion = (index: number, field: string, value: any) => {
  setState(prev => {
    const newQuestions = [...prev.questions];
    newQuestions[index] = {
      ...newQuestions[index],
      [field]: value
    };

    if (field === 'type' && value === 'MULTIPLE') {
      newQuestions[index].options = newQuestions[index].options.map((opt, i) => ({
        ...opt,
        isCorrect: i < 2
      }));
    }

    if (field === 'type' && value === 'SINGLE') {
      const correctIndex = newQuestions[index].options.findIndex(option => option.isCorrect);
      newQuestions[index].options = newQuestions[index].options.map((option, i) => ({
        ...option,
        isCorrect: i === (correctIndex !== -1 ? correctIndex : 0)
      }));
    }

    const errors = showErrors ? validateAllQuestions(newQuestions) : {};
    const newState = {
      ...prev,
      questions: newQuestions,
      errors
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const updateOption = (questionIndex: number, optionIndex: number, field: string, value: any) => {
  setState(prev => {
    const newQuestions = [...prev.questions];
    const options = [...newQuestions[questionIndex].options];
    const qType = newQuestions[questionIndex].type;
    if (field === 'isCorrect' && qType === 'MULTIPLE' && value === false) {
      const checkedCount = options.filter(o => o.isCorrect).length;
      if (checkedCount <= 2 && options[optionIndex].isCorrect) {
        return prev;
      }
    }
    if (field === 'isCorrect' && value === true && qType === 'SINGLE') {
      options.forEach((option, index) => {
        if (index !== optionIndex) {
          options[index] = { ...option, isCorrect: false };
        }
      });
    }
    options[optionIndex] = {
      ...options[optionIndex],
      [field]: value
    };
    newQuestions[questionIndex] = {
      ...newQuestions[questionIndex],
      options
    };

    const errors = showErrors ? validateAllQuestions(newQuestions) : {};
    const newState = {
      ...prev,
      questions: newQuestions,
      errors
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const setActiveQuestion = (index: number) => {
  setState(prev => {
    const errors = showErrors ? validateAllQuestions(prev.questions) : {};
    return {
      ...prev,
      activeQuestionIndex: index,
      errors
    };
  });
};

const validateAndSaveTest = () => {
  const errors = validateAllQuestions(state.questions);
  if (Object.keys(errors).length > 0) {
    setShowErrors(true);
    setState(prev => ({ ...prev, errors }));
    return;
```

```
  }
  setShowErrors(false);
  setState(prev => ({ ...prev, errors: {} }));
  if (typeof onSaveTest === 'function') onSaveTest({ questions: state.questions });
};

const addOption = (questionIndex: number) => {
  setState(prev => {
    const newQuestions = [...prev.questions];
    const newOption: TestOption = {
      id: uuidv4(),
      text: '',
      isCorrect: false
    };

    newQuestions[questionIndex] = {
      ...newQuestions[questionIndex],
      options: [...newQuestions[questionIndex].options, newOption]
    };

    const newState = {
      ...prev,
      questions: newQuestions
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const removeOption = (questionIndex: number, optionIndex: number) => {
  setState(prev => {
    const newQuestions = [...prev.questions];
    const options = [...newQuestions[questionIndex].options];

    if (options.length <= 2) {
      return prev;
    }

    const removingOption = options[optionIndex];
    options.splice(optionIndex, 1);

    if (removingOption.isCorrect && newQuestions[questionIndex].type === 'SINGLE') {
      options[0].isCorrect = true;
    }

    newQuestions[questionIndex] = {
      ...newQuestions[questionIndex],
      options
    };

    const newState = {
      ...prev,
      questions: newQuestions
    };
    onChange({ questions: newQuestions });
    return newState;
  });
};

const moveQuestionUp = (index: number) => {
  if (index === 0) return;

  setState(prev => {
    const newQuestions = [...prev.questions];
    const temp = newQuestions[index];
    newQuestions[index] = newQuestions[index - 1];
    newQuestions[index - 1] = temp;

    return {
      ...prev,
      questions: newQuestions,
      activeQuestionIndex: index - 1
    };
  });
};

const moveQuestionDown = (index: number) => {
  setState(prev => {
    if (index === prev.questions.length - 1) return prev;

    const newQuestions = [...prev.questions];
    const temp = newQuestions[index];
    newQuestions[index] = newQuestions[index + 1];
    newQuestions[index + 1] = temp;

    return {
      ...prev,
      questions: newQuestions,
      activeQuestionIndex: index + 1
    };
  });
};

const getSaveButtonStatus = () => {
  if (state.questions.length === 0) return false;

  for (const question of state.questions) {
```

```
      if (!question.question.trim()) return false;
      if (question.options.length < 2) return false;
      if (question.options.some(option => !option.text.trim())) return false;
      if (!question.options.some(option => option.isCorrect)) return false;
    }

    return true;
  };

  return children({
    state,
    addQuestion,
    removeQuestion,
    updateQuestion,
    addOption,
    removeOption,
    updateOption,
    setActiveQuestion,
    moveQuestionUp,
    moveQuestionDown,
    getSaveButtonStatus,
    translations,
    errors: { ...state.errors, ...errors },
    validateAndSaveTest,
    showErrors
  });
};

export default TestLogic;
```

TestUI.tsx

```
import React from 'react';
import { TestFormState, TestQuestion } from './types';
import { TranslatedText } from '@/components/TranslatedText';

interface TestUIProps {
  state: TestFormState;
  addQuestion: () => void;
  removeQuestion: (index: number) => void;
  updateQuestion: (index: number, field: string, value: any) => void;
  addOption: (questionIndex: number) => void;
  removeOption: (questionIndex: number, optionIndex: number) => void;
  updateOption: (questionIndex: number, optionIndex: number, field: string, value: any) => void;
  setActiveQuestion: (index: number) => void;
  moveQuestionUp: (index: number) => void;
  moveQuestionDown: (index: number) => void;
  translations: {
    questionPlaceholder: string;
    optionPlaceholder: string;
    addQuestionText: string;
    removeQuestionText: string;
    addOptionText: string;
    removeOptionText: string;
    singleChoiceText: string;
    multipleChoiceText: string;
    questionTypeLabel: string;
    correctAnswerText: string;
    moveUpText: string;
    moveDownText: string;
    noQuestionsText: string;
    questionLabelText: string;
  };
  errors: Record<string, string>;
}

const TestUI: React.FC<TestUIProps & { validateAndSaveTest?: () => void, showErrors?: boolean }> =
    ({
  state,
  addQuestion,
  removeQuestion,
  updateQuestion,
  addOption,
  removeOption,
  updateOption,
  setActiveQuestion,
  moveQuestionUp,
  moveQuestionDown,
  translations,
  errors,
  validateAndSaveTest,
  showErrors = false,
}) => {
  const { questions, activeQuestionIndex } = state;
  const activeQuestion = questions[activeQuestionIndex];

  const errorQuestionIndexes = Object.keys(errors)
    .filter(key => (key.startsWith('question-') || key.startsWith('option-')) && errors[key])
    .map(key => parseInt(key.split('-')[1], 10));

  const questionError = errors[`question-${activeQuestionIndex}`];
  const optionError = errors[`option-${activeQuestionIndex}`];
  const allErrors = [];
  if (showErrors && questionError) allErrors.push(questionError);
  if (showErrors && optionError) allErrors.push(optionError);

  const renderQuestionsList = () => {
```

```
  return (
    <div className="min-h-[240px] max-h-[240px] overflow-y-auto overscroll-contain [@media(max-
    width:300px)]:min-h-[160px] [@media(max-width:300px)]:max-h-[160px]">
      {questions.map((question, index) => (
        <div
          key={question.id}
          className={`p-2 mb-2 rounded cursor-pointer h-10 flex items-center justify-between w-
    full
            ${showErrors && errorQuestionIndexes.includes(index) ? 'bg-red-50 border-l-4 border-
    red-500' :
              index === activeQuestionIndex ? 'bg-blue-100 border-l-4 border-blue-500' : 'bg-gray-
    50 hover:bg-gray-100'}
          `}
          onClick={() => setActiveQuestion(index)}
        >
          <div className="w-10 text-center">
            <span className="font-medium">{index + 1}</span>
          </div>
          <div className="flex items-center justify-center flex-grow">
            {index > 0 ? (
              <button type="button" className="p-1 text-gray-500 hover:text-gray-700 mx-1"
    title={translations.moveUpText} onClick={e => { e.stopPropagation(); moveQuestionUp(index);
    }}>
                <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
    fill="currentColor"><path fillRule="evenodd" d="M5.293 9.707a1 1 0 010-1.414l4-4a1 1 0 011.414
    0l4 4a1 1 0 01-1.414 1.414L11 7.414V15a1 1 0 11-2 0V7.414L6.707 9.707a1 1 0 01-1.414 0z"
    clipRule="evenodd" /></svg>
              </button>
            ) : (<div className="w-7 mx-1"></div>)}
            {index < questions.length - 1 ? (
              <button type="button" className="p-1 text-gray-500 hover:text-gray-700 mx-1"
    title={translations.moveDownText} onClick={e => { e.stopPropagation();
    moveQuestionDown(index); }}>
                <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
    fill="currentColor"><path fillRule="evenodd" d="M14.707 10.293a1 1 0 010 1.414l-4 4a1 1 0 01-
    1.414 0l-4-4a1 1 0 111.414-1.414L9 12.586V5a1 1 0 012 0v7.586l2.293-2.293a1 1 0 011.414 0z"
    clipRule="evenodd" /></svg>
              </button>
            ) : (<div className="w-7 mx-1"></div>)}
          </div>
          <div className="flex items-center">
            <button type="button" className="p-1 text-red-500 hover:text-red-700"
    title={translations.removeQuestionText} onClick={e => { e.stopPropagation();
    removeQuestion(index); }} disabled={questions.length <= 1}>
              <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
    fill="currentColor"><path fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000
    2v10a2 2 0 002 2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-1.447A1 1 0 0011 2H9zM7 8a1 1 0 012
    0v6a1 1 0 11-2 0V8zm5-1a1 1 0 00-1 1v6a1 1 0 102 0V8a1 1 0 00-1-1z" clipRule="evenodd"
    /></svg>
            </button>
          </div>
        </div>
      ))}
    </div>
  );
};

const renderQuestionEditor = (question: TestQuestion, questionIndex: number) => {
  return (
    <div className="bg-white p-2 rounded shadow-sm relative">
      <div className="mb-4">
        <label className="block text-sm font-medium text-gray-700 mb-1">
          <TranslatedText text={translations.questionLabelText} />
        </label>
        <input
          type="text"
          value={question.question}
          onChange={(e) => updateQuestion(questionIndex, 'question', e.target.value)}
          placeholder={translations.questionPlaceholder}
          className="w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2
    focus:ring-blue-500"
        />
        {showErrors && errors[`question-${questionIndex}`] && (
          <div className="mt-2 text-xs text-red-500 bg-red-50 border border-red-200 rounded px-2
    py-1 min-w-[120px] text-center">
            {errors[`question-${questionIndex}`]}
          </div>
        )}
      </div>
      <div className="mb-4">
        <label className="block text-sm font-medium text-gray-700 mb-1">
          <TranslatedText text={translations.questionTypeLabel} />
        </label>
        <div className="flex flex-col md:flex-row md:space-x-4 space-y-2 md:space-y-0 items-center
    md:items-start">
          <label className="flex flex-col md:flex-row md:items-center">
            <input
              type="radio"
              checked={question.type === 'SINGLE'}
              onChange={() => updateQuestion(questionIndex, 'type', 'SINGLE')}
              className="h-4 text-blue-600 focus:ring-blue-500 border-gray-300"
            />
            <span className="mt-1 md:mt-0 md:ml-2 text-xs md:text-sm text-center text-gray-700">
              <TranslatedText text={translations.singleChoiceText} />
            </span>
          </label>
          <label className="flex flex-col md:flex-row md:items-center">
```

```
        <input
          type="radio"
          checked={question.type === 'MULTIPLE'}
          onChange={() => updateQuestion(questionIndex, 'type', 'MULTIPLE')}
          className="h-4 text-blue-600 focus:ring-blue-500 border-gray-300"
        />
        <span className="mt-1 md:mt-0 md:ml-2 text-xs md:text-sm text-center text-gray-700">
          <TranslatedText text={translations.multipleChoiceText} />
        </span>
      </label>
    </div>
  </div>
  <div className="mb-0">
    <div className="flex flex-col md:flex-row md:justify-between md:items-center mb-2 space-y-
2 md:space-y-0">
      <label className="block text-sm font-medium text-gray-700 text-start md:text-left">
        <TranslatedText text="Варіанти відповідей" />
      </label>
      <div className="flex justify-center md:justify-end w-full md:w-auto">
        <button
          type="button"
          onClick={() => addOption(questionIndex)}
          className="w-full justify-center inline-flex items-center px-2.5 py-1.5 border
border-transparent text-sm md:text-xs font-medium rounded text-blue-700 bg-blue-100 hover:bg-
blue-200 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500"
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
            <path fillRule="evenodd" d="M10 5a1 1 0 011 1v3h3a1 1 0 110 2h-3v3a1 1 0 11-2 0v-
3H6a1 1 0 110-2h3V6a1 1 0 011-1z" clipRule="evenodd" />
          </svg>
          <TranslatedText text={translations.addOptionText} />
        </button>
      </div>
    </div>
    <div className="space-y-1 min-h-[129px] max-h-[129px] overflow-y-auto overscroll-contain
[@media(max-width:300px)]:min-h-[80px] [@media(max-width:300px)]:max-h-[80px]">
      {question.options.map((option, optionIndex) => (
        <div key={option.id} className="flex items-center py-2 border-b border-gray-200
[@media(max-width:300px)]:py-1">
          <div className="mr-2 mt-3">
            {question.type === 'SINGLE' ? (
              <input
                type="radio"
                checked={option.isCorrect}
                onChange={() => updateOption(questionIndex, optionIndex, 'isCorrect', true)}
                className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300"
              />
            ) : (
              <input
                type="checkbox"
                checked={option.isCorrect}
                onChange={e => {
                  if (!e.target.checked) {
                    const checkedCount = question.options.filter(o => o.isCorrect).length;
                    if (checkedCount <= 2 && option.isCorrect) return;
                  }
                  updateOption(questionIndex, optionIndex, 'isCorrect', e.target.checked);
                }}
                className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded"
              />
            )}
          </div>
          <div className="flex-grow">
            <input
              type="text"
              value={option.text}
              onChange={(e) => updateOption(questionIndex, optionIndex, 'text',
e.target.value)}
              placeholder={translations.optionPlaceholder}
              className="w-full p-2 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500 [@media(max-width:300px)]:p-1 [@media(max-width:300px)]:text-
xs"
            />
          </div>
          <button
            type="button"
            onClick={() => removeOption(questionIndex, optionIndex)}
            disabled={question.options.length <= 2}
            className="ml-1 mt-0 p-1 text-red-500 hover:text-red-700 disabled:opacity-50"
            title={translations.removeOptionText}
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 [@media(max-
width:300px)]:h-4 [@media(max-width:300px)]:w-4" viewBox="0 0 20 20" fill="currentColor">
              <path fillRule="evenodd" d="M4.293 4.293a1 1 0 011.414 0L10 8.586l4.293-4.293a1
1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414 1.414L10 11.414l-4.293 4.293a1 1 0 01-
1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z" clipRule="evenodd" />
            </svg>
          </button>
        </div>
      ))}
    </div>
    {showErrors && errors[`option-${questionIndex}`] && (
      <div className="mt-2 text-xs text-red-500 bg-red-50 border border-red-200 rounded px-2
py-1 min-w-[120px] text-center">
        {errors[`option-${questionIndex}`]}
      </div>
```

```
          )}
        </div>
      </div>
    );
  };

  return (
    <div className="test-editor w-full max-w-full [@media(max-width:300px)]:p-0.5">
      <div className="flex flex-col space-y-2 md:flex-row md:space-y-0 md:space-x-2">
        <div className="md:w-2/5 w-full">
          <div className="bg-white p-1 md:p-1 rounded shadow-sm flex flex-col w-full">
            <div className="mb-4">
              <h3 className="font-medium text-gray-700 mb-2 text-sm md:text-base">
                <TranslatedText text="Питань" /> ({questions.length})
              </h3>
              {questions.length === 0 ? (
                <p className="text-gray-500 italic text-xs md:text-sm">
                  <TranslatedText text={translations.noQuestionsText} />
                </p>
              ) : (
                renderQuestionsList()
              )}
            </div>
            <button
              type="button"
              onClick={addQuestion}
              className="w-full flex justify-center items-center h-10 border border-transparent
      text-sm md:text-base font-medium rounded-md text-white bg-blue-600 hover:bg-blue-700
      focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 mb-2"
            >
              <span className="flex items-center space-x-1">
                <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
                  <path fillRule="evenodd" d="M10 5a1 1 0 011 1v3h3a1 1 0 110 2h-3v3a1 1 0 11-2 0v-
      3H6a1 1 0 110-2h3V6a1 1 0 011-1z" clipRule="evenodd" />
                </svg>
                <span><TranslatedText text={translations.addQuestionText} /></span>
              </span>
            </button>
            <button
              type="button"
              className="w-full flex justify-center items-center h-10 border border-transparent
      text-sm md:text-base font-medium rounded-md text-white bg-green-600 hover:bg-green-700
      focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-green-500 mb-2"
              onClick={validateAndSaveTest}
            >
              <span className="flex items-center space-x-1">
                <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
      fill="currentColor">
                  <path fillRule="evenodd" d="M5 13l4 4L19 7" clipRule="evenodd" />
                </svg>
                <span><TranslatedText text="Зберегти" /></span>
              </span>
            </button>
          </div>
        </div>
        <div className="md:w-3/5 w-full">
          {activeQuestion ? (
            <div className="bg-white p-2 rounded shadow-sm w-full">
              {renderQuestionEditor(activeQuestion, activeQuestionIndex)}
            </div>
          ) : (
            <div className="bg-white p-4 rounded shadow-sm w-full">
              <p className="text-gray-500 italic text-center py-4 text-xs md:text-sm">
                <TranslatedText text="Будь ласка, виберіть питання для редагування або додайте нове"
      />
              </p>
            </div>
          )}
        </div>
      </div>
    </div>
  );
};

export default TestUI;
```

Test.tsx

```
import React from 'react';
import TestLogic from './TestLogic';
import TestUI from './TestUI';
import { Test as TestType } from './types';

interface TestProps {
  initialTest?: TestType;
  onChange: (test: TestType) => void;
  errors?: Record<string, string>;
  onSaveTest?: (test: TestType) => void;
}

const Test: React.FC<TestProps> = ({ initialTest, onChange, errors, onSaveTest }) => {
  return (
    <TestLogic
      initialTest={initialTest}
      onChange={onChange}
      errors={errors}
```

```
      onSaveTest={onSaveTest || (() => {})}
    >
      {props => <TestUI {...props} showErrors={props.showErrors} />}
    </TestLogic>
  );
};

export default Test;
```

form\utils\courseFormUtils.ts

```
export function cleanUndefined(obj: Record<string, any>) {
  const newObj: Record<string, any> = {};
  Object.keys(obj).forEach(key => {
    if (obj[key] !== undefined) newObj[key] = obj[key];
  });
  return newObj;
}

export function swapSectionFiles(obj: Record<string, any>, idxA: number, idxB: number) {
  const newObj: Record<string, any> = {};
  Object.keys(obj).forEach(key => {
    if (key.startsWith(`${idxA}-`)) {
      newObj[key.replace(`${idxA}-`, `${idxB}-`)] = obj[key];
    } else if (key.startsWith(`${idxB}-`)) {
      newObj[key.replace(`${idxB}-`, `${idxA}-`)] = obj[key];
    } else {
      newObj[key] = obj[key];
    }
  });
  return cleanUndefined(newObj);
}

export function swapSectionErrors(errors: Record<string, any>, idxA: number, idxB: number) {
  const newErrors: Record<string, any> = {};
  Object.keys(errors).forEach(key => {
    if (key.startsWith(`section-${idxA}`)) {
      newErrors[key.replace(`section-${idxA}`, `section-${idxB}`)] = errors[key];
    } else if (key.startsWith(`section-${idxB}`)) {
      newErrors[key.replace(`section-${idxB}`, `section-${idxA}`)] = errors[key];
    } else if (key.match(new RegExp(`^(lesson|content)-${idxA}-`))) {
      newErrors[key.replace(`${idxA}-`, `${idxB}-`)] = errors[key];
    } else if (key.match(new RegExp(`^(lesson|content)-${idxB}-`))) {
      newErrors[key.replace(`${idxB}-`, `${idxA}-`)] = errors[key];
    } else {
      newErrors[key] = errors[key];
    }
  });
  return cleanUndefined(newErrors);
}

export function removeSectionPendingFiles(pendingFiles: Record<string, any>, sectionIdx: number) {
  const newPendingFiles = { ...pendingFiles };
  Object.keys(newPendingFiles).forEach(key => {
    if (key.startsWith(`${sectionIdx}-`)) {
      delete newPendingFiles[key];
    }
  });
  return cleanUndefined(newPendingFiles);
}

export function swapLessonFiles(obj: Record<string, any>, sectionIdx: number, idxA: number, idxB:
    number) {
  const newObj = { ...obj };
  const keyA = `${sectionIdx}-${idxA}`;
  const keyB = `${sectionIdx}-${idxB}`;
  const temp = newObj[keyA];
  newObj[keyA] = newObj[keyB];
  newObj[keyB] = temp;
  return cleanUndefined(newObj);
}

export function swapLessonErrors(errors: Record<string, any>, sectionIdx: number, idxA: number,
    idxB: number) {
  const newErrors = { ...errors };
  ['lesson', 'content'].forEach(type => {
    const keyA = `${type}-${sectionIdx}-${idxA}`;
    const keyB = `${type}-${sectionIdx}-${idxB}`;
    const temp = newErrors[keyA];
    newErrors[keyA] = newErrors[keyB];
    newErrors[keyB] = temp;
  });
  return cleanUndefined(newErrors);
}

export function removeLessonPendingFile(pendingFiles: Record<string, any>, sectionIdx: number,
    lessonIdx: number) {
  const newPendingFiles = { ...pendingFiles };
  delete newPendingFiles[`${sectionIdx}-${lessonIdx}`];
  return cleanUndefined(newPendingFiles);
}
```

form\utils\simulationFormUtils.ts

```
export function cleanUndefined(obj: Record<string, any>) {
  const newObj: Record<string, any> = {};
  Object.keys(obj).forEach(key => {
```

```
    if (obj[key] !== undefined) newObj[key] = obj[key];
  });
  return newObj;
}

export function swapStepFiles(obj: Record<string, any>, idxA: number, idxB: number) {
  const newObj = { ...obj };
  const keyA = `${idxA}`;
  const keyB = `${idxB}`;
  const temp = newObj[keyA];
  newObj[keyA] = newObj[keyB];
  newObj[keyB] = temp;
  return cleanUndefined(newObj);
}

export function swapStepErrors(errors: Record<string, any>, idxA: number, idxB: number) {
  const newErrors = { ...errors };
  ['title', 'content', 'videoUrl'].forEach(type => {
    const keyA = `step-${type}-${idxA}`;
    const keyB = `step-${type}-${idxB}`;
    const temp = newErrors[keyA];
    newErrors[keyA] = newErrors[keyB];
    newErrors[keyB] = temp;
  });
  return cleanUndefined(newErrors);
}
```

FormLogic.tsx

```
import React, { useState, useEffect, useRef } from 'react';
import { Course, Section, Simulation, Step, FormState } from './types';
import { fetchWithAuth } from '@/utils/auth';
import { t } from '../TranslatedText';
import {
  swapSectionFiles,
  swapSectionErrors,
  removeSectionPendingFiles,
  swapLessonFiles,
  swapLessonErrors,
  removeLessonPendingFile
} from './utils/courseFormUtils';
import {
  swapStepFiles,
  swapStepErrors
} from './utils/simulationFormUtils';

interface FormLogicProps {
  type: 'course' | 'simulation';
  initialData?: Course | Simulation;
  initialTitle?: string;
  initialDescription?: string;
  initialImageUrl?: string;
  initialKeywords?: string[];
  initialSections?: Section[];
  initialSteps?: Step[];
  onCancel?: () => void;
  submitText?: string;
  user?: any;
  notification?: { type: 'success' | 'error', message: string } | null;
  setNotification?: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
  onSuccess?: (message: string) => void;
  entityId?: string | number;
  children: (props: {
    type: 'course' | 'simulation';
    state: FormState;
    handleChange: (field: string, value: any) => void;
    handleSubmit: (e: React.FormEvent) => void;
    handleImageUpload: (file: File) => void;
    handleImageClick: () => void;
    handleCancel: () => void;
    confirmCancel: () => void;
    titlePlaceholder: string;
    descriptionPlaceholder: string;
    addSection: () => void;
    removeSection: (idx: number) => void;
    updateSectionTitle: (idx: number, value: string) => void;
    addLesson: (sectionIdx: number) => void;
    removeLesson: (sectionIdx: number, lessonIdx: number) => void;
    updateLesson: (sectionIdx: number, lessonIdx: number, field: string, value: any) => void;
    handleSectionFileUpload: (sectionIdx: number, lessonIdx: number, file: File) => void;
    addStep?: () => void;
    removeStep?: (idx: number) => void;
    updateStep?: (idx: number, field: string, value: any) => void;
    handleStepFileUpload?: (stepIdx: number, file: File) => void;
    moveStep?: (fromIdx: number, toIdx: number) => void;
    moveSection: (fromIdx: number, toIdx: number) => void;
    moveLesson: (sectionIdx: number, fromIdx: number, toIdx: number) => void;
    handleAddKeyword: () => void;
    handleRemoveKeyword: (index: number) => void;
    fileInputRef: React.RefObject<HTMLInputElement>;
    imageDropAreaRef: React.RefObject<HTMLDivElement>;
    submitText: string;
    isSubmitting: boolean;
    notification: { type: 'success' | 'error', message: string } | null;
    setNotification: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message:
      string } | null>>;
```

```
    resetImage: () => void;
    clearStepError: (key: string) => void;
  }) => React.ReactNode;
}

const FormLogic: React.FC<FormLogicProps> = ({
  type,
  initialData,
  initialTitle = '',
  initialDescription = '',
  initialImageUrl = '',
  initialKeywords = [],
  initialSections = [{ title: '', lessons: [] }],
  initialSteps = [{ title: '', content: '', videoUrl: ''}],
  onCancel,
  submitText = 'Створити',
  user,
  notification: externalNotification,
  setNotification: externalSetNotification,
  onSuccess,
  entityId,
  children
}) => {
  const [state, setState] = useState<FormState>({
    title: initialData?.title || initialTitle,
    description: initialData?.description || initialDescription,
    keywords: initialData?.keywords || initialKeywords,
    imageUrl: initialData?.imageUrl || initialImageUrl,
    sections: type === 'course' && initialData && 'sections' in initialData
      ? initialData.sections || initialSections
      : initialSections,
    steps: type === 'simulation' && initialData && 'steps' in initialData
      ? initialData.steps || initialSteps
      : initialSteps,
    editingTitle: false,
    editingDescription: false,
    pendingImage: null,
    pendingFiles: {},
    errors: {},
    imageLoading: false,
    fileLoading: {},
    loading: false,
    showCancelModal: false
  });

  const fileInputRef = useRef<HTMLInputElement>(null);
  const imageDropAreaRef = useRef<HTMLDivElement>(null);
  const [notification, setNotification] = externalSetNotification
    ? [externalNotification, externalSetNotification]
    : useState<{ type: 'success' | 'error', message: string } | null>(null);
  const [showTestError, setShowTestError] = useState(false);

  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [titlePlaceholder, setTitlePlaceholder] = useState("Введіть назву");
  const [descriptionPlaceholder, setDescriptionPlaceholder] = useState("Введіть опис");

  useEffect(() => {
    console.log('STATE CHANGED', state);
  }, [state]);

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t("Введіть назву", lang).then(setTitlePlaceholder);
    t("Введіть опис", lang).then(setDescriptionPlaceholder);
  }, [lang]);

  useEffect(() => {
    const dropArea = imageDropAreaRef.current;
    if (!dropArea) return;

    const handleDragOver = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
      dropArea.classList.add('border-blue-500', 'bg-blue-50');
    };

    const handleDragLeave = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
      dropArea.classList.remove('border-blue-500', 'bg-blue-50');
    };

    const handleDrop = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
      dropArea.classList.remove('border-blue-500', 'bg-blue-50');

      if (e.dataTransfer?.files && e.dataTransfer.files.length > 0) {
        const file = e.dataTransfer.files[0];
        handleImageUpload(file);
      }
    };
```

```
  dropArea.addEventListener('dragover', handleDragOver);
  dropArea.addEventListener('dragleave', handleDragLeave);
  dropArea.addEventListener('drop', handleDrop);

  return () => {
    dropArea.removeEventListener('dragover', handleDragOver);
    dropArea.removeEventListener('dragleave', handleDragLeave);
    dropArea.removeEventListener('drop', handleDrop);
  };
}, []);

useEffect(() => {
  const isDirty = () => {

    const initial = (initialData || {}) as any;
    if ((state.title || '') !== (initial.title || initialTitle || '')) return true;
    if ((state.description || '') !== (initial.description || initialDescription || '')) return
    true;
    if ((state.imageUrl || '') !== (initial.imageUrl || initialImageUrl || '')) return true;
    if (JSON.stringify(state.keywords || []) !== JSON.stringify(initial.keywords ||
    initialKeywords || [])) return true;
    if (state.pendingImage) return true;

    if (type === 'course') {
      const initialSectionsArr = initial.sections || initialSections || [{ title: '', lessons: []
    }];
      if (JSON.stringify(state.sections) !== JSON.stringify(initialSectionsArr)) return true;
    }

    if (type === 'simulation') {
      const initialStepsArr = initial.steps || initialSteps || [{ title: '', content: '',
    videoUrl: ''}];
      if (JSON.stringify(state.steps) !== JSON.stringify(initialStepsArr)) return true;
    }
    return false;
  };
  const handleBeforeUnload = (e: BeforeUnloadEvent) => {
    if (isDirty()) {
      e.preventDefault();
      e.returnValue = '';
      return '';
    }
  };
  window.addEventListener('beforeunload', handleBeforeUnload);
  return () => window.removeEventListener('beforeunload', handleBeforeUnload);
}, [
  state.title, state.description, state.sections, state.steps, state.imageUrl, state.keywords,
    state.pendingImage,
  initialTitle, initialDescription, initialImageUrl, initialKeywords, initialSections,
    initialSteps, initialData, type
]);

const handleChange = (field: string, value: any) => {
  const updatedErrors = { ...state.errors };

  if (field === 'title' && updatedErrors.title) {
    delete updatedErrors.title;
  }

  setState(prev => ({
    ...prev,
    [field]: value,
    errors: updatedErrors
  }));
};

const handleImageClick = () => {
  if (fileInputRef.current) {
    fileInputRef.current.click();
  }
};

const handleImageUpload = (file: File) => {
  if (!file.type.startsWith('image/')) {
    setState(prev => ({
      ...prev,
      errors: { ...prev.errors, image: 'Дозволені лише файли зображень' }
    }));
    return;
  }

  if (file.size > 10 * 1024 * 1024) {
    setState(prev => ({
      ...prev,
      errors: { ...prev.errors, image: 'Максимальний розмір зображення - 10MB' }
    }));
    return;
  }

  setState(prev => ({
    ...prev,
    errors: { ...prev.errors, image: '' },
    pendingImage: file
  }));

  const reader = new FileReader();
```

```
  reader.onload = (event) => {
    const result = event.target?.result;
    if (result) {
      setState(prev => ({
        ...prev,
        imageUrl: result as string
      }));
    }
  };
  reader.readAsDataURL(file);
};

function shiftErrorsAfterRemove(
  errors: Record<string, string>,
  type: 'section' | 'lesson' | 'step',
  sectionIdx: number,
  lessonIdx?: number,
  stepIdx?: number
) {
  const updatedErrors = { ...errors };
  Object.keys(updatedErrors).forEach(key => {
    if (type === 'section' && (key.startsWith(`section-${sectionIdx}`) || key.match(new
    RegExp(`^lesson-${sectionIdx}-`)) || key.match(new RegExp(`^content-${sectionIdx}-`)))) {
      delete updatedErrors[key];
    }
    if (type === 'lesson' && (key === `lesson-${sectionIdx}-${lessonIdx}` || key === `content-
    ${sectionIdx}-${lessonIdx}`)) {
      delete updatedErrors[key];
    }
    if (type === 'step' && (key === `step-title-${stepIdx}` || key === `step-content-${stepIdx}`))
    {
      delete updatedErrors[key];
    }
  });

  const newErrors: Record<string, string> = {};
  Object.keys(updatedErrors).forEach(key => {
    const sectionMatch = key.match(/^section-(\d+)/);
    const lessonMatch = key.match(/^lesson-(\d+)-(\d+)/);
    const contentMatch = key.match(/^content-(\d+)-(\d+)/);
    const stepTitleMatch = key.match(/^step-title-(\d+)/);
    const stepContentMatch = key.match(/^step-content-(\d+)/);
    const stepVideoUrlMatch = key.match(/^step-videoUrl-(\d+)/);

    if (type === 'section' && sectionMatch && Number(sectionMatch[1]) > sectionIdx) {
      newErrors[`section-${Number(sectionMatch[1]) - 1}`] = updatedErrors[key];
    } else if (type === 'section' && lessonMatch && Number(lessonMatch[1]) > sectionIdx) {
      newErrors[`lesson-${Number(lessonMatch[1]) - 1}-${lessonMatch[2]}`] = updatedErrors[key];
    } else if (type === 'section' && contentMatch && Number(contentMatch[1]) > sectionIdx) {
      newErrors[`content-${Number(contentMatch[1]) - 1}-${contentMatch[2]}`] = updatedErrors[key];
    } else if (type === 'lesson' && lessonMatch && Number(lessonMatch[1]) === sectionIdx &&
    Number(lessonMatch[2]) > (lessonIdx ?? -1)) {
      newErrors[`lesson-${sectionIdx}-${Number(lessonMatch[2]) - 1}`] = updatedErrors[key];
    } else if (type === 'lesson' && contentMatch && Number(contentMatch[1]) === sectionIdx &&
    Number(contentMatch[2]) > (lessonIdx ?? -1)) {
      newErrors[`content-${sectionIdx}-${Number(contentMatch[2]) - 1}`] = updatedErrors[key];
    }

    else if (type === 'step' && stepTitleMatch && Number(stepTitleMatch[1]) > sectionIdx) {
      newErrors[`step-title-${Number(stepTitleMatch[1]) - 1}`] = updatedErrors[key];
    } else if (type === 'step' && stepContentMatch && Number(stepContentMatch[1]) > sectionIdx) {
      newErrors[`step-content-${Number(stepContentMatch[1]) - 1}`] = updatedErrors[key];
    } else if (type === 'step' && stepVideoUrlMatch && Number(stepVideoUrlMatch[1]) > sectionIdx)
    {
      newErrors[`step-videoUrl-${Number(stepVideoUrlMatch[1]) - 1}`] = updatedErrors[key];
    }

    else if (!sectionMatch && !lessonMatch && !contentMatch && !stepTitleMatch &&
    !stepContentMatch && !stepVideoUrlMatch) {
      newErrors[key] = updatedErrors[key];
    } else {
      newErrors[key] = updatedErrors[key];
    }
  });
  return newErrors;
}

function moveItem<T>(arr: T[], fromIdx: number, toIdx: number): T[] {
  const copy = [...arr];
  const [removed] = copy.splice(fromIdx, 1);
  copy.splice(toIdx, 0, removed);
  return copy;
}

const addSection = () => {
  setState(prev => ({
    ...prev,
    sections: [...(prev.sections || []), { title: '', lessons: [] }]
  }));
};

const removeSection = (idx: number) => {
  const newErrors = shiftErrorsAfterRemove(state.errors, 'section', idx);
  setState(prev => ({
    ...prev,
    sections: prev.sections?.filter((_, i) => i !== idx) || [],
    errors: newErrors,
```

```
    pendingFiles: removeSectionPendingFiles(prev.pendingFiles, idx)
  }));
};

const updateSectionTitle = (idx: number, value: string) => {
  const updatedErrors = { ...state.errors };
  if (updatedErrors[`section-${idx}`]) {
    delete updatedErrors[`section-${idx}`];
  }

  setState(prev => ({
    ...prev,
    sections: prev.sections?.map((s, i) => i === idx ? { ...s, title: value } : s) || [],
    errors: updatedErrors
  }));
};

const addLesson = (sectionIdx: number) => {
  const updatedErrors = { ...state.errors };
  if (updatedErrors.course) {
    delete updatedErrors.course;
  }

  setState(prev => ({
    ...prev,
    sections: prev.sections?.map((s, i) => i === sectionIdx ?
      { ...s, lessons: [...s.lessons, { title: '', type: 'TEXT', content: '', fileUrl: '' }] } :
    s) || [],
    errors: updatedErrors
  }));
};

const removeLesson = (sectionIdx: number, lessonIdx: number) => {
  const newErrors = shiftErrorsAfterRemove(state.errors, 'lesson', sectionIdx, lessonIdx);
  setState(prev => ({
    ...prev,
    sections: prev.sections?.map((s, i) => i === sectionIdx ?
      { ...s, lessons: s.lessons.filter((_, j) => j !== lessonIdx) } : s) || [],
    errors: newErrors,
    pendingFiles: removeLessonPendingFile(prev.pendingFiles, sectionIdx, lessonIdx)
  }));
};

const updateLesson = (sectionIdx: number, lessonIdx: number, field: string, value: any) => {
  const updatedErrors = { ...state.errors };

  if (field === 'title') {
    delete updatedErrors[`lesson-${sectionIdx}-${lessonIdx}`];
    delete updatedErrors.lesson;
  }

  if (field === 'content' || field === 'type') {
    delete updatedErrors[`content-${sectionIdx}-${lessonIdx}`];
    delete updatedErrors.content;
  }

  if (field === 'type') {
    const currentLesson = state.sections?.[sectionIdx]?.lessons?.[lessonIdx];
    if (currentLesson && (currentLesson.videoUrl || currentLesson.documentUrl ||
    currentLesson.fileName)) {
      if (state.pendingFiles[`${sectionIdx}-${lessonIdx}`]) {
        setState(prev => {
          const newPendingFiles = removeLessonPendingFile(prev.pendingFiles, sectionIdx,
    lessonIdx);
          return {
            ...prev,
            pendingFiles: newPendingFiles,
            errors: updatedErrors,
            sections: prev.sections?.map((s, i) =>
              i === sectionIdx ? {
                ...s,
                lessons: s.lessons.map((l, j) =>
                  j === lessonIdx ? {
                    ...l,
                    [field]: value,
                    fileUrl: '',
                    videoUrl: '',
                    documentUrl: '',
                    fileName: ''
                  } : l
                )
              } : s
            )
          };
        });
        return;
      }
    }
    delete updatedErrors[`content-${sectionIdx}-${lessonIdx}`];
  }

  if ((field === 'videoUrl' || field === 'documentUrl') && value === '') {
    setState(prev => {
      const newPendingFiles = removeLessonPendingFile(prev.pendingFiles, sectionIdx, lessonIdx);
      return {
        ...prev,
        sections: prev.sections?.map((s, i) => i === sectionIdx ?
```

```
          { ...s, lessons: s.lessons.map((l, j) => j === lessonIdx ? { ...l, [field]: value } : l)
    } : s) || [],
        errors: updatedErrors,
        pendingFiles: newPendingFiles
      };
    });
    return;
  }

  setState(prev => ({
    ...prev,
    sections: prev.sections?.map((s, i) => i === sectionIdx ?
      { ...s, lessons: s.lessons.map((l, j) => j === lessonIdx ?
        { ...l, [field]: value } : l) } : s) || [],
    errors: updatedErrors
  }));
};

const handleSectionFileUpload = (sectionIdx: number, lessonIdx: number, file: File) => {
  const updatedErrors = { ...state.errors };
  delete updatedErrors[`content-${sectionIdx}-${lessonIdx}`];

  setState(prev => ({
    ...prev,
    pendingFiles: {
      ...prev.pendingFiles,
      [`${sectionIdx}-${lessonIdx}`]: file
    },
    errors: updatedErrors
  }));
};

const addStep = () => {
  setState(prev => ({
    ...prev,
    steps: [...(prev.steps || []), { title: '', content: '', videoUrl: ''}]
  }));
};

const removeStep = (idx: number) => {
  const newErrors = shiftErrorsAfterRemove(state.errors, 'step', idx);
  const newPendingFiles = { ...state.pendingFiles };

  if (newPendingFiles[`${idx}`]) {
    delete newPendingFiles[`${idx}`];
  }

  const shiftedPendingFiles: Record<string, File> = {};
  Object.keys(newPendingFiles).forEach(key => {
    const keyNum = Number(key);
    if (keyNum > idx) {
      shiftedPendingFiles[`${keyNum - 1}`] = newPendingFiles[key];
    } else if (keyNum < idx) {
      shiftedPendingFiles[key] = newPendingFiles[key];
    }
  });

  setState(prev => ({
    ...prev,
    steps: prev.steps?.filter((_, i) => i !== idx) || [],
    errors: newErrors,
    pendingFiles: shiftedPendingFiles
  }));
};

const updateStep = (stepIdx: number, field: string, value: any) => {
  const updatedErrors = { ...state.errors };
  let newPendingFiles = { ...state.pendingFiles };

  if (field === 'title') {
    delete updatedErrors[`step-title-${stepIdx}`];
  }
  if (field === 'content') {
    delete updatedErrors[`step-content-${stepIdx}`];
    delete updatedErrors.content;
  }
  if (field === 'videoUrl') {
    delete updatedErrors[`step-videoUrl-${stepIdx}`];
    delete updatedErrors.videoUrl;
    delete newPendingFiles[`${stepIdx}`];
  }

  setState(prev => {
    return {
      ...prev,
      steps: prev.steps?.map((s, i) =>
        i === stepIdx ? { ...s, [field]: value } : s
      ) || [],
      errors: updatedErrors,
      pendingFiles: field === 'videoUrl' ? newPendingFiles : prev.pendingFiles
    };
  });
};

const handleStepFileUpload = (stepIdx: number, file: File) => {
  setState(prev => {
    const newErrors = { ...prev.errors };
```

```
    delete newErrors[`step-videoUrl-${stepIdx}`];
    const newPendingFiles = {
      ...prev.pendingFiles,
      [`${stepIdx}`]: file
    };
    return {
      ...prev,
      pendingFiles: newPendingFiles,
      errors: newErrors
    };
  });
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (state.loading) return;

  const validationErrors: Record<string, string> = {};

  if (type === 'course') {
    state.sections?.forEach((section, sectionIdx) => {
      section.lessons.forEach((lesson, lessonIdx) => {
        const pendingKey = `${sectionIdx}-${lessonIdx}`;
        const hasPendingFile = !!state.pendingFiles[pendingKey];
        if (!lesson.title?.trim()) {
          validationErrors[`lesson-${sectionIdx}-${lessonIdx}`] = 'Вкажіть назву уроку';
        }
        if (lesson.type === 'VIDEO' && !lesson.videoUrl && !hasPendingFile) {
          validationErrors[`content-${sectionIdx}-${lessonIdx}`] = 'Завантажте відео для уроку';
        }
        if (lesson.type === 'DOCUMENT' && !lesson.documentUrl && !hasPendingFile) {
          validationErrors[`content-${sectionIdx}-${lessonIdx}`] = 'Завантажте документ для
    уроку';
        }
        if (lesson.type === 'TEXT' && !lesson.content?.trim()) {
          validationErrors[`content-${sectionIdx}-${lessonIdx}`] = 'Вміст уроку не може бути
    порожнім';
        }
        if (lesson.type === 'TEST') {
          if (!lesson.test || !lesson.test.questions || lesson.test.questions.length === 0) {
            validationErrors[`content-${sectionIdx}-${lessonIdx}`] = 'Тест не було збережено';
          }
        }
      });
    });
  }

  if (type === 'simulation') {
    state.steps?.forEach((step, stepIdx) => {
      const pendingKey = `${stepIdx}`;
      const hasPendingFile = !!state.pendingFiles[pendingKey];
      if (!step.title?.trim()) {
        validationErrors[`step-title-${stepIdx}`] = 'Вкажіть назву кроку';
      }
      if (!step.content?.trim()) {
        validationErrors[`step-content-${stepIdx}`] = 'Вкажіть опис кроку';
      }
      if (!step.videoUrl && !hasPendingFile) {
        validationErrors[`step-videoUrl-${stepIdx}`] = 'Завантажте відео для кроку';
      }
    });
  }

  if (!state.title.trim()) {
    validationErrors.title = type === 'course'
      ? 'Назва курсу не може бути порожньою'
      : 'Назва симуляції не може бути порожньою';
  }

  if (type === 'course') {
    let hasLessons = false;
    state.sections?.forEach((section, sIdx) => {
      if (!section.title.trim()) {
        validationErrors[`section-${sIdx}`] = 'Назва розділу не може бути порожньою';
      }
      if (section.lessons.length > 0) {
        hasLessons = true;
      }
    });
    if (!hasLessons) {
      validationErrors.course = 'Додайте хоча б один урок до курсу';
    }
  }

  if (type === 'simulation') {
    if (!state.steps || state.steps.length === 0) {
      validationErrors.simulation = 'Додайте хоча б один крок до симуляції';
    }
  }

  if (Object.keys(validationErrors).length > 0) {
    const hasTestError = Object.keys(validationErrors).some(key => key.startsWith('content-') &&
    validationErrors[key] === 'Тест не було збережено');
    if (hasTestError) setShowTestError(true);
    setState(prev => ({
      ...prev,
      errors: validationErrors
```

```
    }));
    return;
  }

  setState(prev => ({ ...prev, loading: true }));
  setNotification(null);

  const currentLang = localStorage.getItem('preferredLanguage') || 'ua';
    const formDataForCheck = {
      title: state.title,
      lang: currentLang,
    };

    let slugCheckUrl = '';
    let method = entityId ? 'PUT' : 'POST';
    if (type === 'course') {
      slugCheckUrl = entityId
        ? `/api/courses/${entityId}`
        : '/api/courses/control';
    } else {
      slugCheckUrl = entityId
        ? `/api/simulations/${entityId}`
        : '/api/simulations/control';
    }

    const slugCheckRes = await fetchWithAuth(slugCheckUrl, {
      method,
      headers: {
        'Content-Type': 'application/json',
        'X-Only-Check-Slug': 'true'
      },
      body: JSON.stringify(formDataForCheck)
    });

    if (slugCheckRes.status === 409) {
      const errorData = await slugCheckRes.json();
      setNotification({ type: 'error', message: errorData.error });
      setState(prev => ({ ...prev, loading: false }));
      return;
    }

  try {
    let finalImageUrl = state.imageUrl;
    let updatedSections = JSON.parse(JSON.stringify(state.sections));
    let updatedSteps = JSON.parse(JSON.stringify(state.steps));

    const totalLessons = updatedSections.reduce(
      (sum: number, section: Section) => sum + (section.lessons?.length || 0),
      0
    );

    const initialFileLoading: Record<string, boolean> = {};
    Object.keys(state.pendingFiles).forEach(key => {
      initialFileLoading[key] = true;
    });

    if (Object.keys(initialFileLoading).length > 0) {
      setState(prev => ({
        ...prev,
        fileLoading: initialFileLoading
      }));
    }

    await new Promise(resolve => setTimeout(resolve, 100));

    if (state.pendingImage) {
      setState(prev => ({ ...prev, imageLoading: true }));
      const formData = new FormData();
      formData.append('file', state.pendingImage);
      formData.append('fileType', 'images');

      if (initialTitle && type === 'course') {
        formData.append('courseId', String(user?.editingCourseId || ''));
        formData.append('courseTitle', state.title);
      } else if (initialTitle && type === 'simulation') {
        formData.append('simulationId', String(user?.editingSimulationId || ''));
        formData.append('simulationTitle', state.title);
      }

      try {
        const res = await fetchWithAuth('/api/upload', {
          method: 'POST',
          body: formData,
        });

        if (!res.ok) {
          const errorData = await res.json();
          const errorMessage = errorData.error;

          setState(prev => ({
            ...prev,
            imageLoading: false,
            errors: { ...prev.errors, image: errorMessage }
          }));
          return;
        } else {
          const data = await res.json();
```

```
        finalImageUrl = data.url;

        setState(prev => ({
          ...prev,
          imageLoading: false,
          imageUrl: data.url
        }));
      }
    } catch (e) {
      setState(prev => ({
        ...prev,
        imageLoading: false,
        errors: { ...prev.errors, image: 'Сталася помилка при завантаженні зображення' }
      }));
      return;
    }
  }

  let hasFileUploadError = false;
  const remainingPendingFiles = { ...state.pendingFiles };

  for (const key in state.pendingFiles) {
    let sectionIdx = 0, lessonIdx = 0, stepIdx = 0;
    if (type === 'course') [sectionIdx, lessonIdx] = key.split('-').map(Number);
    if (type === 'simulation') [stepIdx] = key.split('-').map(Number);
    const file = state.pendingFiles[key];

    if (file) {
      let lesson = null, step = null;
      if (type === 'course') lesson = updatedSections[sectionIdx].lessons[lessonIdx];
      if (type === 'simulation') step = updatedSteps[stepIdx];
      const formData = new FormData();

      if (type === 'course') formData.append('fileType', lesson.type === 'VIDEO' ? 'videos' :
  'documents');
      if (type === 'simulation') formData.append('fileType', 'videos');
      formData.append('file', file);

      if (initialTitle) {
        formData.append('courseId', String(user?.editingCourseId || ''));
        formData.append('courseTitle', state.title);
      } else if (initialTitle && type === 'simulation') {
        formData.append('simulationId', String(user?.editingSimulationId || ''));
        formData.append('simulationTitle', state.title);
      }

      try {
        const res = await fetchWithAuth('/api/upload', {
          method: 'POST',
          body: formData,
        });

        if (!res.ok) {
          const errorData = await res.json();
          const errorMessage = errorData.error;

          hasFileUploadError = true;

          setState(prev => {
            const updatedSections = JSON.parse(JSON.stringify(prev.sections));
            const updatedSteps = JSON.parse(JSON.stringify(prev.steps));
            if (type === 'course') {
              const lessonToUpdate = updatedSections[sectionIdx]?.lessons[lessonIdx];
              if (lessonToUpdate) {
                lessonToUpdate.uploadError = errorMessage;
              }
            } else if (type === 'simulation') {
              const stepToUpdate = updatedSteps[stepIdx];
              if (stepToUpdate) {
                stepToUpdate.uploadError = errorMessage;
              }
            }

            return {
              ...prev,
              sections: updatedSections,
              steps: updatedSteps,
              errors: {
                ...prev.errors,
                [`content-${sectionIdx}-${lessonIdx}`]: errorMessage,
                [`content-${stepIdx}`]: errorMessage
              },
              fileLoading: {
                ...prev.fileLoading,
                [`${sectionIdx}-${lessonIdx}`]: false,
                [`${stepIdx}`]: false
              }
            };
          });

          continue;
        }

        const data = await res.json();
        if (type === 'course') {
          if (lesson.type === 'VIDEO') lesson.videoUrl = data.url;
          else if (lesson.type === 'DOCUMENT') lesson.documentUrl = data.url;
```

```
          lesson.fileUrl = data.url;
          lesson.fileName = file.name || data.fileName || '';
        } else if (type === 'simulation') {
          step.videoUrl = data.url;
          step.fileUrl = data.url;
          step.fileName = file.name || data.fileName || '';
        }

        setState(prev => {
          const updatedSections = JSON.parse(JSON.stringify(prev.sections));
          const updatedSteps = JSON.parse(JSON.stringify(prev.steps));

          if (type === 'course') {
            const lessonToUpdate = updatedSections[sectionIdx]?.lessons[lessonIdx];
            if (lessonToUpdate) {
              if (lesson.type === 'VIDEO') lessonToUpdate.videoUrl = data.url;
              else if (lesson.type === 'DOCUMENT') lessonToUpdate.documentUrl = data.url;
              lessonToUpdate.fileUrl = data.url;
              lessonToUpdate.fileName = file.name || data.fileName || '';
              lessonToUpdate.uploadError = null;
            }
          } else if (type === 'simulation') {
            const stepToUpdate = updatedSteps[stepIdx];
            if (stepToUpdate) {
              stepToUpdate.videoUrl = data.url;
              stepToUpdate.fileUrl = data.url;
              stepToUpdate.fileName = file.name || data.fileName || '';
              stepToUpdate.uploadError = null;
            }
          }

          return {
            ...prev,
            sections: updatedSections,
            steps: updatedSteps,
            fileLoading: {
              ...prev.fileLoading,
              [`${sectionIdx}-${lessonIdx}`]: false,
              [`${stepIdx}`]: false
            }
          };
        });

        if (type === 'course') {
          if ((lesson.type === 'VIDEO' && lesson.videoUrl) ||
              (lesson.type === 'DOCUMENT' && lesson.documentUrl)) {
            delete remainingPendingFiles[key];
          }
        } else if (type === 'simulation') {
          if (step.videoUrl) {
            delete remainingPendingFiles[key];
          }
        }

      } catch (e) {
        const errorMessage = 'Сталася помилка при завантаженні файлу';

        hasFileUploadError = true;

        setState(prev => {
          const updatedSections = JSON.parse(JSON.stringify(prev.sections));
          const updatedSteps = JSON.parse(JSON.stringify(prev.steps));
          if (type === 'course') {
            const lessonToUpdate = updatedSections[sectionIdx]?.lessons[lessonIdx];
            if (lessonToUpdate) {
              lessonToUpdate.uploadError = errorMessage;
            }
          } else if (type === 'simulation') {
            const stepToUpdate = updatedSteps[stepIdx];
            if (stepToUpdate) {
              stepToUpdate.uploadError = errorMessage;
            }
          }

          return {
            ...prev,
            sections: updatedSections,
            steps: updatedSteps,
            errors: {
              ...prev.errors,
              [`content-${sectionIdx}-${lessonIdx}`]: errorMessage,
              [`content-${stepIdx}`]: errorMessage
            },
            fileLoading: {
              ...prev.fileLoading,
              [`${sectionIdx}-${lessonIdx}`]: false,
              [`${stepIdx}`]: false
            }
          };
        });
      }
    }
  }

  if (hasFileUploadError) {
    setState(prev => ({
      ...prev,
```

```
        loading: false,
        pendingFiles: remainingPendingFiles
      }));
      return;
    }

    const formData = type === 'course' ? {
      title: state.title,
      description: state.description,
      imageUrl: finalImageUrl,
      keywords: state.keywords,
      totalLessons,
      lang: currentLang,
      sections: updatedSections.map((section: Section, order: number) => ({
        title: section.title,
        order,
        lessons: section.lessons.map((lesson: any, lorder: number) => ({
          title: lesson.title,
          type: lesson.type,
          content: lesson.content || '',
          order: lorder,
          test: lesson.type === 'TEST' ? lesson.test : undefined,
          videoUrl: lesson.type === 'VIDEO' ? lesson.videoUrl : undefined,
          documentUrl: lesson.type === 'DOCUMENT' ? lesson.documentUrl : undefined,
          fileName: lesson.fileName || ''
        }))
      }))
    } : {
      title: state.title,
      description: state.description,
      imageUrl: finalImageUrl,
      keywords: state.keywords,
      lang: currentLang,
      steps: updatedSteps.map((step: Step, order: number) => ({
        title: step.title,
        content: step.content,
        videoUrl: step.videoUrl,
        videoPreviewUrl: step.videoPreviewUrl,
        order
      }))
    };

    let res: any = undefined;
    if (entityId && type === 'course') {
      res = await fetchWithAuth(`/api/courses/${entityId}?lang=${currentLang}`, {
        method: 'PUT',
        body: JSON.stringify(formData),
        headers: { 'Content-Type': 'application/json' },
      });
    } else if (entityId && type === 'simulation') {
      res = await fetchWithAuth(`/api/simulations/${entityId}?lang=${currentLang}`, {
        method: 'PUT',
        body: JSON.stringify(formData),
        headers: { 'Content-Type': 'application/json' },
      });
    } else if (type === 'course') {
      res = await fetchWithAuth(`/api/courses/control?lang=${currentLang}`, {
        method: 'POST',
        body: JSON.stringify(formData),
        headers: { 'Content-Type': 'application/json' },
      });
    } else if (type === 'simulation') {
      res = await fetchWithAuth(`/api/simulations/control?lang=${currentLang}`, {
        method: 'POST',
        body: JSON.stringify(formData),
        headers: { 'Content-Type': 'application/json' },
      });
    }

    const data = await res.json();
    if (res.status === 409 || res.status === 401) {
      (setNotification || (() => {}))({ type: 'error', message: data.error });
      setState(prev => ({ ...prev, loading: false }));
      return;
    }
    if (res.ok) {
      (setNotification || (() => {}))({ type: 'success', message: data.message });
      setState(prev => ({ ...prev, loading: false }));
      if (onSuccess) onSuccess(data.message);
      return;
    } else {
      (setNotification || (() => {}))({ type: 'error', message: data.error });
      setState(prev => ({ ...prev, loading: false }));
      return;
    }
  } catch (error: any) {
    (setNotification || (() => {}))({ type: 'error', message: error?.message || 'Сталася помилка'
    });
    setState(prev => ({ ...prev, loading: false }));
  }
};

const handleCancel = () => {
  setState(prev => ({ ...prev, showCancelModal: true }));
};

const confirmCancel = () => {
```

```
  if (onCancel) {
    onCancel();
  }
};

const handleAddKeyword = () => {
};

const handleRemoveKeyword = (index: number) => {
  setState(prev => ({
    ...prev,
    keywords: prev.keywords.filter((_, i) => i !== index)
  }));
};

const moveSection = (fromIdx: number, toIdx: number) => {
  setState(prev => {
    const newSections = moveItem(prev.sections || [], fromIdx, toIdx);
    const newErrors = swapSectionErrors(prev.errors, fromIdx, toIdx);
    const newPendingFiles = swapSectionFiles(prev.pendingFiles, fromIdx, toIdx);
    return {
      ...prev,
      sections: newSections,
      pendingFiles: newPendingFiles,
      errors: newErrors
    };
  });
};

const moveLesson = (sectionIdx: number, fromIdx: number, toIdx: number) => {
  setState(prev => {
    const newSections = [...(prev.sections || [])];
    const section = { ...newSections[sectionIdx] };
    section.lessons = moveItem(section.lessons, fromIdx, toIdx);
    newSections[sectionIdx] = section;
    const newPendingFiles = swapLessonFiles(prev.pendingFiles, sectionIdx, fromIdx, toIdx);
    const newErrors = swapLessonErrors(prev.errors, sectionIdx, fromIdx, toIdx);
    return {
      ...prev,
      sections: newSections,
      pendingFiles: newPendingFiles,
      errors: newErrors
    };
  });
};

const moveStep = (fromIdx: number, toIdx: number) => {
  setState(prev => {
    const newSteps = moveItem(prev.steps || [], fromIdx, toIdx);
    const newPendingFiles = swapStepFiles(prev.pendingFiles, fromIdx, toIdx);
    const newErrors = swapStepErrors(prev.errors, fromIdx, toIdx);
    return {
      ...prev,
      steps: newSteps,
      pendingFiles: newPendingFiles,
      errors: newErrors
    };
  });
};

const resetImage = () => {
  setState(prev => ({
    ...prev,
    imageUrl: '',
    pendingImage: null,
    errors: { ...prev.errors, image: '' },
    imageLoading: false
  }));

  if (fileInputRef.current) {
    fileInputRef.current.value = '';
  }
};

const clearStepError = (key: string) => {
  setState(prev => {
    const newErrors = { ...prev.errors };
    delete newErrors[key];
    return { ...prev, errors: newErrors };
  });
};

return children({
  type,
  state,
  handleChange,
  handleSubmit,
  handleImageUpload,
  handleImageClick,
  handleCancel,
  confirmCancel,
  titlePlaceholder,
  descriptionPlaceholder,
  addSection,
  removeSection,
  updateSectionTitle,
  addLesson,
```

```
    removeLesson,
    updateLesson,
    handleSectionFileUpload,
    addStep,
    removeStep,
    updateStep,
    handleStepFileUpload,
    handleAddKeyword,
    handleRemoveKeyword,
    moveSection,
    moveStep,
    moveLesson,
    fileInputRef,
    imageDropAreaRef,
    submitText,
    isSubmitting: state.loading,
    notification: notification || null,
    setNotification: setNotification as React.Dispatch<React.SetStateAction<{ type: 'success' |
      'error', message: string } | null>>,
    resetImage,
    clearStepError,
  });
};

export default FormLogic;
```

FormUI.tsx

```
import React from 'react';
import { FormState } from './types';
import { TranslatedText } from '@/components/TranslatedText';
import Modal from '@/components/Modal';
import Section from './section/Section';
import Step from './step/Step';
import Keywords from './keywords/Keywords';
import Notification from '../Notification';

interface FormUIProps {
  type: 'course' | 'simulation';
  state: FormState;
  handleChange: (field: string, value: any) => void;
  handleSubmit: (e: React.FormEvent) => void;
  handleImageUpload: (file: File) => void;
  handleImageClick: () => void;
  handleCancel: () => void;
  confirmCancel: () => void;
  titlePlaceholder: string;
  descriptionPlaceholder: string;
  addSection: () => void;
  removeSection: (idx: number) => void;
  updateSectionTitle: (idx: number, value: string) => void;
  addLesson: (sectionIdx: number) => void;
  removeLesson: (sectionIdx: number, lessonIdx: number) => void;
  updateLesson: (sectionIdx: number, lessonIdx: number, field: string, value: any) => void;
  handleSectionFileUpload: (sectionIdx: number, lessonIdx: number, file: File) => void;
  addStep?: () => void;
  removeStep?: (idx: number) => void;
  updateStep?: (idx: number, field: string, value: any) => void;
  handleStepFileUpload?: (stepIdx: number, file: File) => void;
  moveStep?: (fromIdx: number, toIdx: number) => void;
  moveSection: (fromIdx: number, toIdx: number) => void;
  moveLesson: (sectionIdx: number, fromIdx: number, toIdx: number) => void;
  handleAddKeyword: () => void;
  handleRemoveKeyword: (index: number) => void;
  fileInputRef: React.RefObject<HTMLInputElement>;
  imageDropAreaRef: React.RefObject<HTMLDivElement>;
  submitText: string;
  isSubmitting: boolean;
  notification: { type: 'success' | 'error', message: string } | null;
  setNotification: React.Dispatch<React.SetStateAction<{ type: 'success' | 'error', message: string
      } | null>>;
  resetImage: () => void;
  clearStepError: (key: string) => void;
}

const FormUI: React.FC<FormUIProps> = ({
  type,
  state,
  handleChange,
  handleSubmit,
  handleImageUpload,
  handleImageClick,
  handleCancel,
  confirmCancel,
  titlePlaceholder,
  descriptionPlaceholder,
  addSection,
  removeSection,
  updateSectionTitle,
  addLesson,
  removeLesson,
  updateLesson,
  handleSectionFileUpload,
  addStep,
  removeStep,
  updateStep,
  handleStepFileUpload,
```

```
  moveStep,
  handleRemoveKeyword,
  moveSection,
  moveLesson,
  fileInputRef,
  imageDropAreaRef,
  submitText,
  isSubmitting,
  notification,
  setNotification,
  resetImage,
  clearStepError
}) => {

  const getStepErrors = (idx: number) => {
    const errors = { ...state.errors };
    if (type === 'simulation' && state.pendingFiles && state.pendingFiles[idx]) {
      const errKey = `step-videoUrl-${idx}`;
      if (errors[errKey]) delete errors[errKey];
    }
    return errors;
  };

  return (
    <>
      {notification && (
        <Notification type={notification.type} message={notification.message} onClose={() =>
      setNotification(null)} />
      )}
      <form onSubmit={handleSubmit} className="w-full max-w-full p-1 sm:p-2 md:p-4 [@media(max-
      width:300px)]:p-0.5">
        {state.showCancelModal && (
          <Modal onClose={() => handleChange('showCancelModal', false)}>
            <div className="p-4 w-full max-w-md mx-auto">
              <h2 className="text-xl font-bold mb-4 text-center text-red-600"><TranslatedText
      text="Скасувати зміни?" /></h2>
              <p className="mb-6 text-center"><TranslatedText text="Ви впевнені, що хочете скасувати
      всі внесені зміни?" /></p>
              <div className="flex justify-center space-x-3">
                <button className="px-4 py-2 bg-gray-300 rounded hover:bg-gray-400 transition-
      colors" onClick={() => handleChange('showCancelModal', false)}><TranslatedText text="Назад"
      /></button>
                <button className="px-4 py-2 bg-red-600 text-white rounded hover:bg-red-700
      transition-colors" onClick={confirmCancel}><TranslatedText text="Так" /></button>
              </div>
            </div>
          </Modal>
        )}

        <div className="mb-4">
          <label htmlFor="title" className="block mb-2 text-sm md:text-base font-medium">
            <TranslatedText text={type === 'course' ? "Назва курсу" : "Назва симуляції"} />
          </label>
          <input
            type="text"
            id="title"
            value={state.title}
            onChange={e => handleChange('title', e.target.value)}
            placeholder={titlePlaceholder}
            className={`w-full p-2 md:p-3 border rounded ${state.errors.title ? 'border-red-500' :
      'border-gray-300'} text-sm md:text-base`}
            disabled={isSubmitting}
            maxLength={100}
          />
          <div className="flex justify-between">
            {state.errors.title && <p className="text-red-500 text-xs md:text-sm mt-
      1"><TranslatedText text={state.errors.title} /></p>}
            <p className="text-gray-500 text-xs md:text-sm mt-1 ml-
      auto">{state.title.length}/100</p>
          </div>
        </div>

        <div className="mb-4">
          <div className="text-sm font-medium mb-2 flex flex-wrap items-center gap-x-1">
            <TranslatedText text={type === 'course' ? "Зображення курсу" : "Зображення симуляції"}
      />
            <span className="text-gray-400 text-xs">(<TranslatedText text="необов'язково" />)</span>
          </div>
          <div
            ref={imageDropAreaRef}
            className="border-2 border-dashed border-gray-300 rounded-lg p-4 cursor-pointer
      hover:border-blue-500 transition-colors flex flex-col justify-center items-center h-full text-
      center"
            onClick={handleImageClick}
            style={{ minHeight: '160px' }}
          >
            <input
              ref={fileInputRef}
              type="file"
              className="hidden"
              accept="image/*"
              onChange={(e) => e.target.files?.[0] && handleImageUpload(e.target.files[0])}
            />

            {state.imageUrl ? (
              <div className="relative w-full">
                <img
```

```
            src={state.imageUrl}
            className="max-h-48 mx-auto object-contain rounded"
            alt={type === 'course' ? "Зображення курсу" : "Зображення симуляції"}
          />
          <button
            type="button"
            className="absolute top-0 right-0 bg-white rounded-full p-1 shadow-md"
            onClick={(e) => {
              e.stopPropagation();
              resetImage();
            }}
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 text-red-600"
viewBox="0 0 20 20" fill="currentColor">
              <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zM8.707 7.293a1 1 0
00-1.414 1.414L8.586 10l-1.293 1.293a1 1 0 101.414 1.414L10 11.414l1.293 1.293a1 1 0 001.414-
1.414L11.414 10l1.293-1.293a1 1 0 00-1.414-1.414L10 8.586 8.707 7.293z" clipRule="evenodd" />
            </svg>
          </button>
          {state.imageLoading && (
            <div className="absolute inset-0 flex items-center justify-center bg-white bg-
opacity-70">
              <svg className="animate-spin h-8 w-8 text-blue-500"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
                <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
                <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373
0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
              </svg>
            </div>
          )}
        </div>
      ) : (
        <div className="flex flex-col items-center justify-center text-gray-500 h-full">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-12 w-12 mb-2" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={1.5} d="M4
16l4.586-4.586a2 2 0 012.828 0L16 16m-2-2l1.586-1.586a2 2 0 012.828 0L20 14m-6-6h.01M6 20h12a2
2 0 002-2V6a2 2 0 00-2-2H6a2 2 0 00-2 2v12a2 2 0 002 2z" />
          </svg>
          <p className="[@media(max-width:210px)]:hidden"><TranslatedText text="Натисніть або
перетягніть, щоб завантажити зображення" /></p>
          <p className="text-xs mt-1 [@media(max-width:210px)]:hidden"><TranslatedText
text="Підтримуються тільки файли зображень, максимум 10MB" /></p>
          <span className="inline [@media(min-width:211px)]:hidden text-xs text-gray-500 mt-1
flex items-center justify-center">
            <span className="ml-1">&lt;10МБ</span>
          </span>
        </div>
      )}
    </div>
    {state.errors.image && (
      <div className="mt-2 flex items-center text-red-500 bg-red-50 p-2 rounded-md">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4 mr-1" viewBox="0 0 20 20"
fill="currentColor">
          <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1 1
0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd" />
        </svg>
        <span className="text-xs font-medium"><TranslatedText text={state.errors.image}
/></span>
        <button
          type="button"
          className="ml-auto text-red-500 hover:text-red-700"
          onClick={() => {
            handleChange('errors', { ...state.errors, image: '' });
            if (fileInputRef.current) {
              fileInputRef.current.value = '';
            }
          }}
        >
        </button>
      </div>
    )}

    {!state.imageLoading && state.imageUrl && !state.errors.image && !state.pendingImage && (
      <div className="mt-2 flex items-center text-green-500">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-1" viewBox="0 0 20 20"
fill="currentColor">
          <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0 00-
1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z"
clipRule="evenodd" />
        </svg>
        <span className="text-xs"><TranslatedText text="Зображення успішно завантажено"
/></span>
      </div>
    )}

    {state.imageLoading && (
      <div className="mt-2 flex items-center text-blue-500">
        <svg className="animate-spin -ml-1 mr-2 h-4 w-4" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24">
          <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
strokeWidth="4"></circle>
          <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0
5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
        </svg>
        <span className="text-xs"><TranslatedText text="Завантаження зображення..." /></span>
```

```
      </div>
    )}
  </div>

  <div className="mb-4">
    <label htmlFor="description" className="block mb-2 text-sm font-medium flex flex-wrap
items-center gap-x-1">
      <TranslatedText text={type === 'course' ? "Опис курсу" : "Опис симуляції"} />
      <span className="text-gray-400 text-xs">(<TranslatedText text="необов'язково" />)</span>
    </label>
    <textarea
      id="description"
      className="w-full p-3 border rounded border-gray-300 overflow-y-auto overscroll-contain"
      value={state.description}
      onChange={e => handleChange('description', e.target.value)}
      placeholder={descriptionPlaceholder}
      rows={3}
      disabled={isSubmitting}
      maxLength={1000}
    />
    <div className="flex justify-between">
      {state.errors.description && <p className="text-red-500 text-sm mt-
1">{state.errors.description}</p>}
      <p className="text-gray-500 text-xs mt-1 ml-auto">{state.description.length}/1000</p>
    </div>
  </div>

  <Keywords
    keywords={state.keywords}
    updateKeywords={(keywords) => handleChange('keywords', keywords)}
    removeKeyword={handleRemoveKeyword}
  />

  {type === 'course' ? (
    <>
      <div className="mb-4 p-2 md:p-3 bg-blue-50 text-blue-800 border border-blue-200 rounded
text-xs md:text-sm break-words">
        <div className="flex items-center mb-2">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-2" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 16h-1v-
4h-1m1-4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z" />
          </svg>
          <span className="font-medium"><TranslatedText text="Підказка по навігації" /></span>
        </div>
        <ul className="list-disc pl-5 space-y-1">
          <li><TranslatedText text="Ви можете змінювати порядок розділів і уроків,
використовуючи стрілки вгору та вниз" /></li>
          <li><TranslatedText text="Для переміщення розділу використовуйте стрілки біля назви
розділу" /></li>
          <li><TranslatedText text="Для переміщення уроку використовуйте стрілки зліва від
назви уроку" /></li>
        </ul>
      </div>
      <div className="space-y-4">
        {state.sections?.map((section, sIdx) => (
          <Section
            key={sIdx}
            section={section}
            sectionIdx={sIdx}
            updateSectionTitle={updateSectionTitle}
            addLesson={addLesson}
            removeLesson={removeLesson}
            updateLesson={updateLesson}
            handleFileUpload={handleSectionFileUpload}
            removeSection={removeSection}
            moveSection={moveSection}
            moveLesson={moveLesson}
            fileLoading={state.fileLoading}
            errors={state.errors}
            sectionsLength={state.sections?.length || 0}
            setNotification={setNotification}
          />
        ))}
      </div>
      <div className="mt-4">
        <button
          type="button"
          className="px-4 py-2 bg-gray-100 text-gray-700 rounded mr-2 hover:bg-gray-200"
          onClick={addSection}
        >
          <TranslatedText text="+ Додати розділ" />
        </button>
      </div>
      {state.errors.course && (
        <div className="mt-3 text-sm text-red-500"><TranslatedText text={state.errors.course}
/></div>
      )}
    </>
  ) : (
    <>
      <div className="mb-4 p-2 md:p-3 bg-blue-50 text-blue-800 border border-blue-200 rounded
text-xs md:text-sm break-words">
        <div className="flex items-center mb-2">
          <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 mr-2" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
```

```
                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 16h-1v-
    4h-1m1-4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z" />
              </svg>
              <span className="font-medium"><TranslatedText text="Підказка по навігації" /></span>
            </div>
            <ul className="list-disc pl-5 space-y-1">
              <li><TranslatedText text="Ви можете змінювати порядок кроків, використовуючи стрілки
    вгору та вниз" /></li>
            </ul>
          </div>
          <div className="space-y-4">
            {state.steps?.map((step, idx) => (
              <Step
                key={idx}
                step={step}
                stepIdx={idx}
                updateStep={updateStep!}
                removeStep={removeStep!}
                handleFileUpload={handleStepFileUpload!}
                moveStep={moveStep!}
                stepsLength={state.steps?.length || 0}
                errors={getStepErrors(idx)}
                setNotification={setNotification}
                clearStepError={clearStepError}
                fileLoading={state.fileLoading[idx]}
              />
            ))}
          </div>
          <div className="mt-4">
            <button
              type="button"
              className="px-4 py-2 bg-gray-100 text-gray-700 rounded mr-2 hover:bg-gray-200"
              onClick={addStep}
            >
              <TranslatedText text="+ Додати крок" />
            </button>
          </div>
          {state.errors.simulation && (
            <div className="mt-3 text-sm text-red-500"><TranslatedText
    text={state.errors.simulation} /></div>
          )}
        </>
      )}

      <div className="mt-6 flex justify-end space-x-2 [@media(max-width:300px)]:flex-col
    [@media(max-width:300px)]:space-x-0 [@media(max-width:300px)]:space-y-1 [@media(max-
    width:300px)]:w-full">
        <button
          type="button"
          className="px-4 py-2 bg-gray-300 rounded hover:bg-gray-400 w-full"
          onClick={handleCancel}
        >
          <TranslatedText text="Скасувати" />
        </button>
        <button
          type="submit"
          className="px-4 py-2 bg-blue-600 text-white rounded hover:bg-blue-700 disabled:bg-blue-
    300 w-full"
          disabled={isSubmitting}
        >
          {isSubmitting ? <TranslatedText text="Завантаження..." /> : <TranslatedText
    text={submitText} />}
        </button>
      </div>
    </form>
  </>
 );
};

export default FormUI;
```

Form.tsx

```
import React from 'react';
import FormLogic from './FormLogic';
import FormUI from './FormUI';
import { Course, Section, Simulation, Step } from './types';

interface FormProps {
  type: 'course' | 'simulation';
  initialData?: Course | Simulation;
  initialTitle?: string;
  initialDescription?: string;
  initialImageUrl?: string;
  initialKeywords?: string[];
  initialSections?: Section[];
  initialSteps?: Step[];
  onCancel?: () => void;
  submitText?: string;
  user?: any;
  onSuccess?: (message: string) => void;
  entityId?: string | number;
}

const Form: React.FC<FormProps> = ({
  type,
  initialData,
```

```
  initialTitle,
  initialDescription,
  initialImageUrl,
  initialKeywords,
  initialSections,
  initialSteps,
  onCancel,
  submitText,
  user,
  onSuccess,
  entityId
}) => {
  return (
    <FormLogic
      type={type}
      initialData={initialData}
      initialTitle={initialTitle}
      initialDescription={initialDescription}
      initialImageUrl={initialImageUrl}
      initialKeywords={initialKeywords}
      initialSections={initialSections}
      initialSteps={initialSteps}
      onCancel={onCancel}
      submitText={submitText}
      user={user}
      onSuccess={onSuccess}
      entityId={entityId}
    >
      {(logicProps) => <FormUI {...logicProps} />}
    </FormLogic>
  );
};

export default Form;
```

ForbiddenClient.tsx

```
'use client';

import ForbiddenBlock from './PageErrorBlock';

export default function ForbiddenClient() {
  let message = '';

  if (typeof window !== 'undefined') {
    const lang = localStorage.getItem('preferredLanguage') || 'ua';
    if (lang === 'en') {
      message = 'You are not allowed to access this page';
    } else {
      message = 'Ви не маєте доступу до цієї сторінки';
    }
  }
  return <ForbiddenBlock message={message} />;
}
```

NotFoundClient.tsx

```
'use client';

import NotFoundBlock from './PageErrorBlock';

export default function NotFoundClient() {
  let message = '';

  if (typeof window !== 'undefined') {
    const lang = localStorage.getItem('preferredLanguage') || 'ua';
    if (lang === 'en') {
      message = 'Page not found';
    } else {
      message = 'Сторінку не знайдено';
    }
  }
  return <NotFoundBlock message={message} />;
}
```

PageErrorBlock.tsx

```
import React from 'react';

const NotFoundBlock = ({ message }: { message: string }) => (
  <div className="min-h-screen flex items-center justify-center bg-white">
    <div className="flex items-center space-x-6 bg-blue-100 px-8 py-6 rounded-lg shadow">
      <div className="flex-shrink-0">
        <svg width="64" height="64" viewBox="0 0 64 64" fill="none">
          <circle cx="32" cy="20" r="12" fill="#90cdf4"/>
          <rect x="16" y="36" width="32" height="20" rx="10" fill="#90cdf4"/>
          <rect x="28" y="44" width="8" height="12" rx="4" fill="#fff"/>
          <rect x="30" y="48" width="4" height="4" rx="2" fill="#3182ce"/>
          <rect x="24" y="50" width="16" height="4" rx="2" fill="#fff"/>
          <rect x="30" y="52" width="4" height="4" rx="2" fill="#3182ce"/>
          <rect x="36" y="48" width="4" height="4" rx="2" fill="#3182ce"/>
        </svg>
      </div>
      <span className="text-2xl font-semibold text-red-600">{message}</span>
    </div>
  </div>
```

```
);

export default NotFoundBlock;
```

UnauthorizedClient.tsx

```
'use client';

import UnauthorizedBlock from './PageErrorBlock';

export default function UnauthorizedClient() {
  let message = '';

  if (typeof window !== 'undefined') {
    const lang = localStorage.getItem('preferredLanguage') || 'ua';
    if (lang === 'en') {
      message = 'You are not authorized to access this page';
    } else {
      message = 'Ви не авторизовані для доступу до цієї сторінки';
    }
  }
  return <UnauthorizedBlock message={message} />;
}
```

UserCard.tsx

```
import React from 'react';
import md5 from 'md5';
import { TranslatedText } from '../TranslatedText';

interface UserCardProps {
  user: { id: number; email: string; name: string; role: string };
  onClick: () => void;
}

const UserCard: React.FC<UserCardProps> = ({
  user,
  onClick
}) => {
  return (
    <div className="relative bg-white rounded shadow p-4 flex flex-col items-center cursor-pointer
      hover:bg-gray-50 transition" onClick={onClick}>
      <img
        src={`https://www.gravatar.com/avatar/${md5(user.email.trim().toLowerCase())}?d=identicon`}
        alt="avatar"
        className="w-16 h-16 rounded-full border mb-2 object-cover"
      />
      <span className="text-sm font-medium break-all">{user.email}</span>
      <span className="text-xs text-gray-500 mb-1"><TranslatedText text={user.name} /></span>
      <span className="absolute top-2 right-2 px-2 py-1 rounded bg-blue-100 text-blue-700 text-xs
      font-bold uppercase">{user.role}</span>
    </div>
  );
};

export default UserCard;
```

UserList.tsx

```
import React from 'react';
import UserCard from './UserCard';

interface UserListProps {
  users: { id: number; email: string; name: string; role: string }[];
  onClick: (user: any) => void;
}

const UserList: React.FC<UserListProps> = ({ users, onClick }) => {
  return (
    <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4 mt-6">
      {users.map(user => (
        <UserCard key={user.id} user={user} onClick={() => onClick(user)} />
      ))}
    </div>
  );
};

export default UserList;
```

CourseLessonIcon.tsx

```
import React from 'react';

interface CourseLessonIconProps {
  type: string;
  documentUrl?: string;
}

const iconMap: Record<string, JSX.Element> = {
  'pdf': <span role="img" aria-label="PDF">📄</span>,
  'excel': <span role="img" aria-label="Excel">📈</span>,
  'image': <span role="img" aria-label="Image">🖼</span>,
  'video': <span role="img" aria-label="Video">🎬</span>,
  'test': <span role="img" aria-label="Test">❓</span>,
  'text': <span role="img" aria-label="Text">📝</span>,
```

```
  'document': <span role="img" aria-label="Document">📄</span>,
  'presentation': <span role="img" aria-label="Presentation">📊</span>,
};

function getIcon(type: string, documentUrl?: string, videoUrl?: string) {
  if (type === 'VIDEO') return iconMap['video'];
  if (type === 'TEST') return iconMap['test'];
  if (type === 'TEXT') return iconMap['text'];

  if (type === 'DOCUMENT') {
    if (documentUrl && /\.(pptx?|ppt)$/i.test(documentUrl)) return iconMap['presentation'];
    if (documentUrl && /\.(jpg|jpeg|png|gif|webp|bmp|svg)$/i.test(documentUrl)) return
      iconMap['image'];
    if (documentUrl && /\.(xls|xlsx)$/i.test(documentUrl)) return iconMap['excel'];
    if (documentUrl && /\.(pdf)$/i.test(documentUrl)) return iconMap['pdf'];
    if (documentUrl && /\.(docx?|doc)$/i.test(documentUrl)) return iconMap['document'];
    if (documentUrl && /\.txt$/i.test(documentUrl)) return iconMap['text'];
    return iconMap['document'];
  }
  return <span role="img" aria-label="Other">🎁</span>;
}

const CourseLessonIcon: React.FC<CourseLessonIconProps> = ({ type, documentUrl }) => {
  const icon = getIcon(type, documentUrl);
  return <span className="text-2xl mr-2">{icon}</span>;
};

export default CourseLessonIcon;
```

CourseSectionList.tsx

```
import React from 'react';
import CourseLessonIcon from './CourseLessonIcon';
import { TranslatedText } from '@/components/TranslatedText';

export interface Lesson {
  id: number;
  title: string;
  type: string;
  completed: boolean;
  locked: boolean;
  documentUrl?: string;
}

export interface Section {
  id: number;
  title: string;
  lessons: Lesson[];
}

interface CourseSectionListProps {
  sections: Section[];
  onLessonClick: (lesson: Lesson) => void;
  locked: boolean;
}

const CourseSectionList: React.FC<CourseSectionListProps> = ({
  sections,
  onLessonClick,
  locked
}) => {
  return (
    <div className="space-y-8">
      {sections.map(section => (
        <div key={section.id}>
          <h2 className="text-lg font-semibold mb-3 text-blue-800 pl-2">
            <TranslatedText text={section.title} />
          </h2>
          <div className="grid gap-4 sm:grid-cols-1 md:grid-cols-2 lg:grid-cols-3">
            {section.lessons.map(lesson => (
              <div
                key={lesson.id}
                className={`relative p-5 rounded-2xl shadow bg-white border flex items-center min-w-
      0 transition-all duration-300 cursor-pointer ${lesson.locked || locked ? 'opacity-50 pointer-
      events-none' : 'hover:shadow-2xl hover:-translate-y-1'} ${lesson.completed ? 'border-green-
      400' : 'border-gray-200'}`}
                onClick={() => onLessonClick(lesson)}
              >
                <CourseLessonIcon type={lesson.type} documentUrl={lesson.documentUrl} />
                <span className="ml-4 font-medium text-base flex-1 truncate">
                  <TranslatedText text={lesson.title} />
                </span>
                {lesson.completed && (
                  <span className="ml-2 text-green-600 text-xl">✓</span>
                )}
                {(lesson.locked || locked) && !lesson.completed && (
                  <span className="ml-2 text-gray-400 text-lg">🔒</span>
                )}
              </div>
            ))}
          </div>
        </div>
      ))}
    </div>
  );
};
```

```
export default CourseSectionList;
```

CourseStartHeader.tsx

```
import React from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface CourseStartHeaderProps {
  started: boolean;
  progress: number;
  onStart: () => void;
  title: string;
  completedCount?: number;
  totalCount?: number;
  description?: string;
}

const CourseStartHeader: React.FC<CourseStartHeaderProps> = ({
  started,
  progress,
  onStart,
  title,
  completedCount,
  totalCount,
  description
}) => {
  return (
    <div className="flex flex-col py-4 border-b mb-6">
      <div className="flex flex-col md:flex-row items-center justify-between w-full">
        <h1 className="text-2xl md:text-3xl font-bold mb-2 md:mb-0">
          <TranslatedText text={title} />
        </h1>
        {!started ? (
          <button
            className="px-6 py-2 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700
     transition mb-2 md:mb-0"
            onClick={onStart}
          >
            <TranslatedText text="Почати проходження курсу" />
          </button>
        ) : (
          <div className="w-full md:w-80 mb-2 md:mb-0">
            <div className="flex items-center justify-between mb-1">
              <span className="text-sm font-medium text-blue-700">
                <TranslatedText text="Прогрес курсу" />
              </span>
              <span className="text-sm font-medium text-blue-700">{progress}%</span>
            </div>
            <div className="w-full bg-blue-200 rounded-full h-3">
              <div
                className="bg-blue-600 h-3 rounded-full transition-all duration-300"
                style={{ width: `${progress}%` }}
              ></div>
            </div>
            {typeof completedCount === 'number' && typeof totalCount === 'number' && (
              <div className="text-xs text-gray-500 mt-1 text-
     right">{completedCount}/{totalCount}</div>
            )}
          </div>
        )}
      </div>
      {description && (
        <div className="mt-3 p-3 bg-gray-50 border border-gray-200 rounded-xl text-gray-700 text-
     base leading-relaxed w-full">
          <TranslatedText text={description} />
        </div>
      )}
    </div>
  );
};

export default CourseStartHeader;
```

LessonModal.tsx

```
import React, { useState } from 'react';
import LessonTextView from '../ui/LessonTextUI';
import LessonVideoView from '../ui/LessonVideoUI';
import LessonDocumentView from '../ui/LessonDocumentUI';
import LessonTestView from '../ui/LessonTestUI';
import ModalWrapper from '@/components/course/modal/ModalWrapper';
import Spinner from '../../Spinner';
import { TranslatedText } from '@/components/TranslatedText';

interface LessonModalProps {
  open: boolean;
  onClose: () => void;
  lesson: any;
  onComplete: () => void;
  completed: boolean;
  loading: boolean;
  result?: any;
}

function getMimeTypeFromUrl(url: string): string {
  if (!url) return '';
```

```
  const ext = url.split('.').pop()?.toLowerCase();
  switch (ext) {
    case 'pdf': return 'application/pdf';
    case 'jpg':
    case 'jpeg':
    case 'png':
    case 'gif':
    case 'webp':
      return `image/${ext === 'jpg' ? 'jpeg' : ext}`;
    case 'doc': return 'application/msword';
    case 'docx': return 'application/vnd.openxmlformats-officedocument.wordprocessingml.document';
    case 'ppt': return 'application/vnd.ms-powerpoint';
    case 'pptx': return 'application/vnd.openxmlformats-officedocument.presentationml.presentation';
    case 'xls': return 'application/vnd.ms-excel';
    case 'xlsx': return 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet';
    case 'txt': return 'text/plain';
    default: return '';
  }
}

const LessonModal: React.FC<LessonModalProps> = ({
  open,
  onClose,
  lesson,
  onComplete,
  completed,
  loading,
  result
}) => {
  const [testFinished, setTestFinished] = useState(false);
  if (!open || !lesson) return null;

  let content = null;
  if (lesson.type === 'TEXT') {
    content = (
      <LessonTextView
        title={lesson.title}
        content={lesson.content}
        onExit={onClose}
        onComplete={onComplete}
        completed={completed}
      />
    );
  } else if (lesson.type === 'VIDEO') {
    content = (
      <LessonVideoView
        title={lesson.title}
        content={lesson.content}
        videoUrl={lesson.videoUrl}
        onExit={onClose}
        onComplete={onComplete}
        completed={completed}
      />
    );
  } else if (lesson.type === 'DOCUMENT') {
    const mimeType = lesson.type || getMimeTypeFromUrl(lesson.documentUrl);
    content = (
      <LessonDocumentView
        title={lesson.title}
        content={lesson.content}
        fileUrl={lesson.documentUrl}
        mimeType={getMimeTypeFromUrl(lesson.documentUrl)}
        onExit={onClose}
        onComplete={onComplete}
        completed={completed}
      />
    );
  } else if (lesson.type === 'TEST') {
    const parsedQuestions = (lesson.test?.questions || []).map((q: any, idx: number) => ({
      id: q.id || idx,
      text: q.question || q.text || '',
      type: (q.type ||   ).toLowerCase(),
      options: (q.options || []).map((opt: any, oidx: number) => ({
        id: opt.id || oidx,
        text: opt.text || '',
      })),
      correctAnswers: (q.options || []).filter((opt: any) => opt.isCorrect).map((opt: any) => opt.id
      || 0),
    }));
    content = (
      <LessonTestView
        title={lesson.title}
        questions={parsedQuestions}
        onExit={onClose}
        onComplete={() => {
          setTestFinished(false);
          onComplete();
        }}
        completed={completed}
        result={result}
        onTestFinished={() => setTestFinished(true)}
      />
    );
  } else {
    content = <div>
      <TranslatedText text="Невідомий тип уроку" />
    </div>;
```

```
  }

  return (
    <ModalWrapper onClose={onClose} hideClose={testFinished}>
      {loading ? (
        <Spinner />
      ) : (
        content
      )}
    </ModalWrapper>
  );
};

export default LessonModal;
```

ModalWrapper.tsx

```
import React from 'react';

interface ModalWrapperProps {
  children: React.ReactNode;
  onClose?: () => void;
  hideClose?: boolean;
}

const ModalWrapper: React.FC<ModalWrapperProps> = ({
  children,
  onClose,
  hideClose
}) => {
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black/30">
      <div className="w-full max-w-3xl h-full max-h-[70vh] flex flex-col items-stretch bg-white
      rounded-2xl shadow-xl p-4 relative border border-blue-100">
        {onClose && !hideClose && (
          <button
            className="absolute top-3 right-8 text-gray-400 hover:text-gray-700 text-2xl font-bold"
            onClick={onClose}
            aria-label="Закрити"
          >
            ×
          </button>
        )}
        {children}
      </div>
    </div>
  );
};

export default ModalWrapper;
```

LessonTextUI.tsx

```
import React from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface LessonTextUIProps {
  title: string;
  content: string;
  onExit: () => void;
  onComplete: () => void;
  completed: boolean;
}

const LessonTextUI: React.FC<LessonTextUIProps> = ({
  title,
  content,
  onExit,
  onComplete,
  completed
}) => {
  const handleDownload = () => {
    const blob = new Blob([`${title}\n\n${content}`], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `${title.replace(/[^a-zA-Z0-9а-яА-ЯёЁіІїЇєЄ\s_-]/g, '') || 'lesson'}.txt`;
    document.body.appendChild(a);
    a.click();
    setTimeout(() => {
      document.body.removeChild(a);
      URL.revokeObjectURL(url);
    }, 100);
  };

  return (
    <div className="flex flex-col h-full overflow-auto overscroll-contain">
      <h2 className="text-2xl font-bold mb-2 text-blue-900 break-words">
        <TranslatedText text={title} />
      </h2>
      <div className="flex-1 text-lg text-gray-800 whitespace-pre-line border rounded p-4 bg-gray-50
      mb-1 overflow-auto overscroll-contain">
        <TranslatedText text={content} />
      </div>
      <div className="flex flex-col w-full gap-2 mt-2 [@media(min-width:370px)]:flex-row
      [@media(min-width:370px)]:justify-end [@media(min-width:370px)]:items-center [@media(min-
      width:370px)]:w-auto [@media(min-width:370px)]:gap-2">
```

```
        <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-gray-200 hover:bg-
      gray-300 rounded transition" onClick={onExit}>
          <TranslatedText text="Вийти" />
        </button>
        <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-blue-600 text-white
      rounded hover:bg-blue-700 transition" onClick={handleDownload}>
          <TranslatedText text="Скачати текст" />
        </button>
        {!completed && <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-
      green-600 text-white rounded hover:bg-green-700 transition" onClick={onComplete}>
          <TranslatedText text="Завершити урок" />
        </button>}
      </div>
    </div>
  );
};

export default LessonTextUI;
```

LessonVideoUI.tsx

```
import React from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface LessonVideoUIProps {
  title: string;
  content?: string;
  videoUrl: string;
  onExit: () => void;
  onComplete: () => void;
  completed: boolean;
}

const LessonVideoUI: React.FC<LessonVideoUIProps> = ({
  title,
  content,
  videoUrl,
  onExit,
  onComplete,
  completed
}) => {
  return (
    <div className="flex flex-col h-full overflow-auto overscroll-contain">
      <h2 className="text-2xl font-bold mb-2 text-blue-900 break-words">
        <TranslatedText text={title} />
      </h2>
      {content && (
        <div className="mb-3 text-gray-700 text-sm border rounded p-2 bg-gray-50 max-h-[60px]
      overflow-auto overscroll-contain">
          <TranslatedText text={content} />
        </div>
      )}
      <div className="flex-1 min-h-0 flex items-start justify-start mb-2 overflow-auto overscroll-
      contain">
        <video controls className="rounded-lg w-full h-full min-w-[250px] min-h-[350px] object-
      contain bg-black">
          <source src={videoUrl} />
          <TranslatedText text="Ваш браузер не підтримує відео!" />
        </video>
      </div>
      <div className="flex flex-col w-full gap-2 mt-2 [@media(min-width:370px)]:flex-row
      [@media(min-width:370px)]:justify-end [@media(min-width:370px)]:items-center [@media(min-
      width:370px)]:w-auto [@media(min-width:370px)]:gap-2">
        <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-gray-200 hover:bg-
      gray-300 rounded transition" onClick={onExit}>
          <TranslatedText text="Вийти" />
        </button>
        {!completed && <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-
      green-600 text-white rounded hover:bg-green-700 transition" onClick={onComplete}>
          <TranslatedText text="Завершити урок" />
        </button>}
      </div>
    </div>
  );
};

export default LessonVideoUI;
```

LessonDocumentUI.tsx

```
import React, { useState, useEffect } from 'react';
import Spinner from '@/components/Spinner';
import { TranslatedText, t } from '@/components/TranslatedText';

interface LessonDocumentUIProps {
  title: string;
  content?: string;
  fileUrl: string;
  mimeType: string;
  onExit: () => void;
  onComplete: () => void;
  completed: boolean;
}

const LessonDocumentUI: React.FC<LessonDocumentUIProps> = ({
  title,
  content,
```

```
  fileUrl,
  mimeType,
  onExit,
  onComplete,
  completed
}) => {
  let format;

  const [showImageModal, setShowImageModal] = useState(false);
  const [downloadError, setDownloadError] = useState(false);
  const [loading, setLoading] = useState(true);
  const [textContent, setTextContent] = useState<string | null>(null);
  const [textError, setTextError] = useState<string | null>(null);

  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [imagePlaceholder, setImagePlaceholder] = useState('Відкрити на весь екран');
  const [alertMessageMistake, setAlertMessageMistake] = useState('Не вдалося завантажити файл!');
  const [alertMessageSuggestion, setAlertMessageSuggestion] = useState('Спробуйте відкрити у новій
      вкладці');
  const [closePlaceholder, setClosePlaceholder] = useState('Закрити');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t('Відкрити на весь екран', lang).then(setImagePlaceholder);
    t('Не вдалося завантажити файл!', lang).then(setAlertMessageMistake);
    t('Спробуйте відкрити у новій вкладці', lang).then(setAlertMessageSuggestion);
    t('Закрити', lang).then(setClosePlaceholder);
  }, [lang]);

  useEffect(() => {
    if (!loading) return;
    const timeout = setTimeout(() => setLoading(false), 1500);
    return () => clearTimeout(timeout);
  }, [loading]);

  if (mimeType === 'application/pdf') {
    format = <iframe src={fileUrl} className="h-full min-h-0 w-full object-contain rounded border
      bg-white" title="PDF" onLoad={() => setLoading(false)} />;
  } else if (mimeType.startsWith('image/')) {
    format = (
      <div className="relative h-full w-full flex-1 min-h-0">
        <img src={fileUrl} alt={title} className="h-full min-h-0 w-full object-contain rounded
      border bg-white" onLoad={() => setLoading(false)} />
        <button
          type="button"
          className="absolute bottom-2 right-2 bg-white/80 hover:bg-white text-blue-700 border
      border-blue-200 rounded-full p-2 shadow transition"
          title={imagePlaceholder}
          onClick={() => setShowImageModal(true)}
        >
          <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" viewBox="0 0 24
      24"><path stroke="currentColor" strokeLinecap="round" strokeLinejoin="round" strokeWidth="2"
      d="M9 3H5a2 2 0 0 0-2 2v4m14-6h4a2 2 0 0 1 2 2v4M3 15v4a2 2 0 0 0 2 2h4m6 0h4a2 2 0 0 0 2-2v-
      4"/></svg>
        </button>
      </div>
    );
  } else if (mimeType === 'text/plain') {
    useEffect(() => {
      setLoading(true);
      setTextError(null);
      setTextContent(null);

      fetch(fileUrl)
        .then(res => {
          if (!res.ok) throw new Error('Помилка завантаження тексту');
          return res.text();
        })
        .then(text => {
          setTextContent(text);
          setLoading(false);
        })
        .catch(() => {
          setTextError('Не вдалося завантажити текстовий файл!');
          setLoading(false);
        });
    }, [fileUrl]);

    format = (
      <div className="w-full h-full min-h-0 bg-white rounded border p-2 overflow-auto text-sm font-
      mono whitespace-pre-wrap">
        {textError ? (
          <div className="text-red-600 text-center py-4">
            <TranslatedText text={textError} />
          </div>
        ) : (
          <div className="text-black-500">
            <TranslatedText text={textContent || ''} />
          </div>
        )}
      </div>
    );
```

```
} else if ([
  'application/msword',
  'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
  'application/vnd.ms-powerpoint',
  'application/vnd.openxmlformats-officedocument.presentationml.presentation',
  'application/vnd.ms-excel',
  'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
].includes(mimeType)) {
  format = (
    <iframe
      src={`https://docs.google.com/gview?url=${encodeURIComponent(fileUrl)}&embedded=true`}
      className="h-full min-h-0 w-full object-contain rounded border bg-white overflow-auto
    overscroll-contain"
      title="Document"
      onLoad={() => setLoading(false)}
    />
  );
} else {
  format = <div className="text-gray-500">
    <TranslatedText text="Невідомий формат документа" />
  </div>;
}

const getCleanFileName = (url: string, title: string, mimeType: string): string => {
  try {
    const urlParts = url.split('/');
    let fileName = urlParts[urlParts.length - 1];
    fileName = fileName.replace(/^[^-]+-/, '');

    if (mimeType === 'text/plain') {
      return (title || 'document') + '.txt';
    }
    return fileName;
  } catch {
    return title || 'document';
  }
}

const handleDownload = async () => {
  try {
    setDownloadError(false);
    const response = await fetch(fileUrl, { mode: 'cors' });

    if (!response.ok) throw new Error('Error downloading file');
    const blob = await response.blob();
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');

    a.href = url;
    a.download = getCleanFileName(fileUrl, title, mimeType);
    document.body.appendChild(a);
    a.click();
    setTimeout(() => {
      window.URL.revokeObjectURL(url);
      a.remove();
    }, 100);
  } catch (e) {
    setDownloadError(true);
    alert(alertMessageMistake + ' ' + alertMessageSuggestion);
  }
};

return (
  <>
    {showImageModal && (
      <div className="fixed inset-0 z-[9999] flex items-center justify-center bg-black/80"
    onClick={() => setShowImageModal(false)}>
        <img src={fileUrl} alt={title} className="max-w-full max-h-full object-contain shadow-2xl
    rounded" />
        <button
          type="button"
          className="absolute top-4 right-8 text-white text-3xl font-bold bg-black/40 rounded-full
    px-3 py-1 hover:bg-black/70 transition"
          onClick={e => { e.stopPropagation(); setShowImageModal(false); }}
          title={closePlaceholder}
        >×</button>
      </div>
    )}
    <div className="flex flex-col h-full overflow-auto overscroll-contain">
      <h2 className="text-2xl font-bold mb-2 text-blue-900 break-words">
        <TranslatedText text={title} />
      </h2>
      {content && (
        <div className="mb-3 text-gray-700 text-sm border rounded p-2 bg-gray-50 max-h-[60px]
    overflow-auto overscroll-contain">
          <TranslatedText text={content} />
        </div>
      )}
      <div className="flex-1 min-h-0 flex flex-col">
        {loading ? <Spinner /> : format}
      </div>
      <div className="flex flex-col w-full gap-2 mt-2 [@media(min-width:530px)]:flex-row
    [@media(min-width:530px)]:justify-end [@media(min-width:530px)]:items-center [@media(min-
    width:530px)]:w-auto [@media(min-width:530px)]:gap-2">
        <button className="w-full [@media(min-width:530px)]:w-auto px-4 py-2 bg-gray-200 hover:bg-
    gray-300 rounded transition" onClick={onExit}>
          <TranslatedText text="Вийти" />
```

```
          </button>
          {(mimeType && fileUrl) && (
            <>
              <button
                type="button"
                onClick={handleDownload}
                className="w-full [@media(min-width:530px)]:w-auto px-4 py-2 bg-blue-600 text-white
     rounded hover:bg-blue-700 transition text-center"
              >
                <TranslatedText text="Завантажити файл" />
              </button>
              {downloadError && (
                <a
                  href={fileUrl}
                  target="_blank"
                  rel="noopener noreferrer"
                    className="w-full [@media(min-width:530px)]:w-auto px-4 py-2 bg-yellow-500 text-
     white rounded hover:bg-yellow-600 transition text-center"
                  style={{ marginTop: 4 }}
                >
                  <TranslatedText text="Відкрити у новій вкладці" />
                </a>
              )}
            </>
          )}
          {!completed && (
            <button
              className="w-full [@media(min-width:530px)]:w-auto px-4 py-2 bg-green-600 text-white
     rounded hover:bg-green-700 transition"
              onClick={onComplete}
            >
              <TranslatedText text="Завершити урок" />
            </button>
          )}
        </div>
      </div>
    </>
  );
};

export default LessonDocumentUI;
```

LessonTestUI.tsx

```
import React, { useState } from 'react';
import { TranslatedText } from '@/components/TranslatedText';

interface Question {
  id: number;
  text: string;
  type: 'single' | 'multiple';
  options: { id: number; text: string }[];
  correctAnswers: number[];
}

interface LessonTestUIProps {
  title: string;
  questions: Question[];
  onExit: () => void;
  onComplete: (result: any) => void;
  completed: boolean;
  result?: any;
  onTestFinished?: () => void;
}

const getAnswerStatus = (user: number[], correct: number[], type: 'single' | 'multiple') => {
  if (type === 'single') {
    if (user.length === 0) return null;
    return user[0] === correct[0] ? 'correct' : 'wrong';
  } else {
    if (user.length === 0) return null;
    const correctSet = new Set(correct);
    const userSet = new Set(user);
    const intersection = [...userSet].filter(x => correctSet.has(x));
    if (intersection.length === correct.length && user.length === correct.length) return 'correct';
    if (intersection.length > 0) return 'partial';
    return 'wrong';
  }
};

const getStatusLabel = (status: string | null) => {
  if (status === 'correct') return 'Правильна відповідь';
  if (status === 'wrong') return 'Неправильна відповідь';
  if (status === 'partial') return 'Частково правильна відповідь';
  return '';
};

const getStatusColor = (status: string | null) => {
  if (status === 'correct') return 'bg-green-100 border-green-400';
  if (status === 'wrong') return 'bg-red-100 border-red-400';
  if (status === 'partial') return 'bg-orange-100 border-orange-400';
  return 'bg-white border-gray-200';
};

const LessonTestUI: React.FC<LessonTestUIProps> = ({
  title,
  questions,
```

```
  onExit,
  onComplete,
  completed,
  result,
  onTestFinished
}) => {
  const [userAnswers, setUserAnswers] = useState<{ [qid: number]: number[] }>({});
  const [showResult, setShowResult] = useState(false);
  const [error, setError] = useState('');
  const [testResult, setTestResult] = useState<any>(null);

  const handleOptionChange = (qid: number, oid: number, type: 'single' | 'multiple') => {
    setUserAnswers(prev => {
      if (type === 'single') {
        return { ...prev, [qid]: [oid] };
      } else {
        const prevArr = prev[qid] || [];
        if (prevArr.includes(oid)) {
          return { ...prev, [qid]: prevArr.filter(id => id !== oid) };
        } else {
          return { ...prev, [qid]: [...prevArr, oid] };
        }
      }
    });
  };

  const handleSubmit = () => {
    for (const q of questions) {
      if (!userAnswers[q.id] || userAnswers[q.id].length === 0) {
        setError('Усі тестові завдання обов'язкові');
        return;
      }
    }
    setError('');
    let correct = 0;
    let partial = 0;
    questions.forEach(q => {
      const status = getAnswerStatus(userAnswers[q.id], q.correctAnswers, q.type);
      if (status === 'correct') correct++;
      else if (status === 'partial') partial++;
    });
    setTestResult({ total: questions.length, correct, partial });
    setShowResult(true);
    if (typeof onTestFinished === 'function') {
      onTestFinished();
    }
  };

  const isDone = completed || showResult;

  return (
    <div className="flex flex-col h-full overflow-auto overscroll-contain">
      <h2 className="text-2xl font-bold mb-2 text-blue-900 break-words">
        <TranslatedText text={title} />
      </h2>
      <div className="flex-1 mb-6 space-y-6 min-h-[100px] overflow-auto pr-2 overscroll-contain">
        {isDone && (testResult || result) && (
          <div className="mb-4 p-3 rounded border bg-blue-50 text-blue-700 font-semibold text-
      center">
            <TranslatedText text="Результат" />: {((result || testResult).correct ?? 0)}/{(result ||
      testResult).total} <TranslatedText text="правильних" />, {(result || testResult).partial ?? 0}
      <TranslatedText text="частково правильних" />
          </div>
        )}
        {questions.map(q => {
          const status = isDone ? getAnswerStatus(userAnswers[q.id] || [], q.correctAnswers, q.type)
      : null;
          return (
            <div key={q.id} className={`p-3 rounded-xl border ${getStatusColor(status)} mb-2
      transition-all duration-300 w-full `}>
              <div className="font-semibold mb-2 break-words text-base sm:text-lg">
                <TranslatedText text={q.text} />
              </div>
              {isDone && status && (
                <div className={`mb-1 text-sm font-semibold ${
                  status === 'correct' ? 'text-green-700' : status === 'wrong' ? 'text-red-700' :
      'text-orange-700'
                } `}>
                  <TranslatedText text={getStatusLabel(status)} />
                </div>
              )}
              <div className={`flex flex-col w-full space-y-2${(showResult || completed) ? ' mt-2' :
      ''} ${(showResult || completed) ? 'pl-0 ml-0' : ''}`}>
                {q.options.map(opt => {
                  const checked = (userAnswers[q.id] || []).includes(opt.id);
                  let optColor = '';
                  if (isDone) {
                    if (q.type === 'single') {
                      if (opt.id === q.correctAnswers[0] && checked) optColor = 'bg-green-200';
                      else if (checked && opt.id !== q.correctAnswers[0]) optColor = 'bg-red-200';
                    } else {
                      if (q.correctAnswers.includes(opt.id) && checked) optColor = 'bg-green-200';
                      else if (checked && !q.correctAnswers.includes(opt.id)) optColor = 'bg-orange-
      200';
                    }
                  }
                  return (
```

```
                <label key={opt.id} className={`flex items-center ${showResult || completed ?
    'px-0 ml-0 pl-0' : 'px-2'} py-1 rounded w-full break-words ${optColor} transition-all
    duration-300`}>
                  {(!showResult && !completed) ? (
                    <input
                      type={q.type === 'single' ? 'radio' : 'checkbox'}
                      name={`q_${q.id}`}
                      value={opt.id}
                      checked={checked}
                      disabled={isDone}
                      onChange={() => handleOptionChange(q.id, opt.id, q.type)}
                      className="mr-2"
                    />
                  ) : null}
                  <span className="break-words">
                    <TranslatedText text={opt.text} />
                  </span>
                </label>
              );
            })}
          </div>
          {isDone && status !== 'correct' && (
            <div className="mt-2 text-xs text-gray-700 break-words">
              <span className="font-semibold">
                <TranslatedText text="Правильна відповідь:" />{' '}
              </span>
              <TranslatedText text={q.options.filter(opt =>
    q.correctAnswers.includes(opt.id)).map(opt => opt.text).join(', ')} />
            </div>
          )}
        </div>
      );
    })}
    {error && <div className="text-red-600 text-center mt-2">
      <TranslatedText text={error} />
    </div>}
  </div>
  {showResult ? (
    <div className="flex flex-col w-full gap-2 mt-2">
      <button className="w-full px-4 py-2 bg-green-600 text-white rounded hover:bg-green-700
    transition" onClick={() => onComplete(testResult || result)}>
        <TranslatedText text="Завершити урок" />
      </button>
    </div>
  ) : (
    <div className="flex flex-col w-full gap-2 mt-2 [@media(min-width:370px)]:flex-row
    [@media(min-width:370px)]:justify-end [@media(min-width:370px)]:items-center [@media(min-
    width:370px)]:w-auto [@media(min-width:370px)]:gap-2">
      <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-gray-200 hover:bg-
    gray-300 rounded transition" onClick={onExit}>
        <TranslatedText text="Вийти" />
      </button>
      {!isDone && <button className="w-full [@media(min-width:370px)]:w-auto px-4 py-2 bg-green-
    600 text-white rounded hover:bg-green-700 transition" onClick={handleSubmit}>
        <TranslatedText text="Завершити тест" />
      </button>}
    </div>
  )}
  </div>
);
};

export default LessonTestUI;
```

SimulationModal.tsx

```
import React, { useState, useEffect } from 'react';
import Modal from '../Modal';
import SimulationVideo from './SimulationVideoUI';
import SimulationStepContent from './SimulationStepContentUI';
import { TranslatedText } from '../TranslatedText';

export interface SimulationStep {
  title: string;
  content: string;
  videoUrl: string;
}

interface SimulationModalProps {
  steps: SimulationStep[];
  initialStep?: number;
  onClose: () => void;
  title?: string;
  description?: string;
}

const MIN_SPEED = 0.5;
const MAX_SPEED = 2.0;
const STEP = 0.1;
const MARKS = [0.5, 1, 1.5, 2];

const SimulationModal: React.FC<SimulationModalProps> = ({ steps, initialStep = 0, onClose, title,
    description }) => {
  const [currentStep, setCurrentStep] = useState(initialStep);
  const [playbackRate, setPlaybackRate] = useState(1);

  const goNext = () => {
```

```
    if (currentStep < steps.length - 1) setCurrentStep(currentStep + 1);
  };
  const goPrev = () => {
    if (currentStep > 0) setCurrentStep(currentStep - 1);
  };
  const finish = () => {
    onClose();
  };

  return (
    <Modal onClose={onClose} className="w-full h-full max-w-none max-h-none md:w-[80vw] md:max-w-
      [1000px] md:h-[70vh]">
      <div className="flex flex-col w-full h-full bg-white shadow-lg overflow-auto overscroll-
      contain md:rounded-lg">
        {title && (
          <div className="px-1 pb-2">
            <h2 className="text-2xl font-bold mb-2"><TranslatedText text={title} /></h2>
            {description && (
              <div className="bg-gray-50 rounded p-1 text-sm max-h-[60px] overflow-auto overscroll-
      contain border mb-2">
                <TranslatedText text={description} />
              </div>
            )}
          </div>
        )}
        <div className="flex flex-1 w-full h-0 flex-col md:flex-row">
          <div className="w-full md:w-[65%] min-w-0 flex flex-col h-full">
            <div className="flex-1 flex items-center justify-center bg-black">
              <div className="w-full aspect-[16/9] flex items-center justify-center">
                <SimulationVideo
                  videoUrl={steps[currentStep].videoUrl}
                  stepIndex={currentStep}
                  playbackRate={playbackRate}
                />
              </div>
            </div>
            <div className="w-full mt-4 mb-2">
              <div className="relative w-full flex items-center">
                <input
                  type="range"
                  min={MIN_SPEED}
                  max={MAX_SPEED}
                  step={STEP}
                  value={playbackRate}
                  onChange={e => setPlaybackRate(parseFloat(e.target.value))}
                  className="w-full accent-blue-600 h-2 bg-gray-200 rounded-lg appearance-none
      cursor-pointer"
                  style={{ zIndex: 2 }}
                />
                <div className="absolute left-0 right-0 top-1/2 -translate-y-1/2 flex justify-
      between w-full z-1 pointer-events-none">
                  {Array.from({ length: Math.round((MAX_SPEED - MIN_SPEED) / 0.1) + 1 }).map((_,
      idx) => {
                    const val = MIN_SPEED + idx * 0.1;
                    const isMark = MARKS.includes(Number(val.toFixed(1)));
                    return (
                      <div key={val} className={`h-3 ${isMark ? 'w-0.5 bg-blue-600' : 'w-0.5 bg-
      gray-400'} rounded`} style={{ height: isMark ? 16 : 10 }} />
                    );
                  })}
                </div>
              </div>
              <div className="flex justify-end w-full mt-1">
                <span className="text-xs text-gray-700 font-
      semibold">{playbackRate.toFixed(1)}x</span>
              </div>
            </div>
          </div>
          <div className="w-full md:w-[35%] min-w-0 max-w-full flex flex-col h-full pl-2 pb-2 pr-1
      bg-white">
            <SimulationStepContent
              step={steps[currentStep]}
              stepIndex={currentStep}
              stepsCount={steps.length}
              onNext={goNext}
              onPrev={goPrev}
              onFinish={finish}
            />
          </div>
        </div>
      </div>
    </Modal>
  );
};

export default SimulationModal;
```

SimulationVideoUI.tsx

```
import React, { useRef, useEffect, useState } from 'react';
import { t } from '../TranslatedText';

interface SimulationVideoUIProps {
  videoUrl: string;
  stepIndex: number;
  playbackRate: number;
}
```

```
const SimulationVideoUI: React.FC<SimulationVideoUIProps> = ({
  videoUrl,
  stepIndex,
  playbackRate
}) => {
  const videoRef = useRef<HTMLVideoElement>(null);
  const [loading, setLoading] = useState(true);
  const [lang, setLang] = useState<'ua' | 'en'>('ua');
  const [downloadTitle, setDownloadTitle] = useState('Завантажити відео');

  useEffect(() => {
    const currentLang = typeof window !== 'undefined' ? localStorage.getItem('preferredLanguage') :
      'ua';
    setLang(currentLang === 'en' ? 'en' : 'ua');
  }, []);

  useEffect(() => {
    t('Завантажити відео', lang).then(setDownloadTitle);
  }, [lang]);

  useEffect(() => {
    if (videoRef.current) {
      videoRef.current.playbackRate = playbackRate;
    }
  }, [videoUrl, playbackRate]);

  useEffect(() => {
    setLoading(true);
  }, [videoUrl]);

  const handleLoadedData = () => {
    setLoading(false);
    if (videoRef.current) {
      videoRef.current.play();
    }
  };

  const handleDownload = async (e: React.MouseEvent) => {
    e.stopPropagation();
    try {
      const response = await fetch(videoUrl);
      const blob = await response.blob();
      const url = window.URL.createObjectURL(blob);
      const a = document.createElement('a');

      a.href = url;
      a.download = `step-${stepIndex + 1}-video.mp4`;
      document.body.appendChild(a);
      a.click();
      setTimeout(() => {
        document.body.removeChild(a);
        window.URL.revokeObjectURL(url);
      }, 100);
    } catch (error) {
      console.error('Error downloading video:', error);
    }
  };

  return (
    <div className="relative w-full h-full flex items-center justify-center">
      <video
        ref={videoRef}
        src={videoUrl}
        className="w-full h-full object-contain bg-black object-contain"
        loop
        muted
        playsInline
        onLoadedData={handleLoadedData}
        controls={false}
        style={{ pointerEvents: 'none', maxHeight: '100%', maxWidth: '100%' }}
      />
      <button
        className="absolute top-2 right-2 bg-white bg-opacity-80 rounded-full p-2 shadow"
        onClick={handleDownload}
        style={{ pointerEvents: 'auto' }}
        title={downloadTitle}
      >
        <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5 text-blue-600" fill="none"
      viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M4 16v2a2 2 0 002
      2h12a2 2 0 002-2v-2M7 10l5 5m0 0l5-5m-5 5V4" />
        </svg>
      </button>
      {loading && (
        <div className="absolute bottom-2 right-2 bg-white bg-opacity-80 rounded-full p-1">
          <svg className="animate-spin h-5 w-5 text-blue-500" xmlns="http://www.w3.org/2000/svg"
      fill="none" viewBox="0 0 24 24">
            <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor"
      strokeWidth="4"></circle>
            <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373
      0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path>
          </svg>
        </div>
      )}
    </div>
  );
```

```
};

export default SimulationVideoUI;
```

SimulationStepContentUI.tsx

```
import React from 'react';
import { TranslatedText } from '../TranslatedText';

interface SimulationStepContentUIProps {
  step: { title: string; content: string };
  stepIndex: number;
  stepsCount: number;
  onNext: () => void;
  onPrev: () => void;
  onFinish: () => void;
}

const SimulationStepContentUI: React.FC<SimulationStepContentUIProps> = ({
  step,
  stepIndex,
  stepsCount,
  onNext,
  onPrev,
  onFinish
}) => {
  return (
    <div className="flex flex-col h-full">
      <div className="mb-4">
        <h3 className="text-lg font-bold mb-2"><TranslatedText text={`Крок ${stepIndex + 1}:`} />
      <br /> <TranslatedText text={step.title} /></h3>
      </div>
      <div className="flex-1 overflow-auto min-h-[100px] bg-gray-50 rounded pt-0 pb-0 p-3 text-sm
      overscroll-contain">
        <TranslatedText text={step.content} />
      </div>
      <div className="flex justify-between mt-2 ml-4 mr-4">
        {stepIndex > 0 && (
          <button className="px-3 py-2 bg-gray-200 rounded hover:bg-gray-300" onClick={onPrev}>
            <TranslatedText text="Назад" />
          </button>
        )}
        {stepIndex < stepsCount - 1 ? (
          <button className="ml-auto px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-700"
      onClick={onNext}>
            <TranslatedText text="Далі" />
          </button>
        ) : (
          <button className="ml-auto px-3 py-2 bg-green-600 text-white rounded hover:bg-green-700"
      onClick={onFinish}>
            <TranslatedText text="Завершити" />
          </button>
        )}
      </div>
    </div>
  );
};

export default SimulationStepContentUI;
```