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 “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Освітній веб-застосунок для репетиторів та учнів

Виконав: студент IV курсу, групи ІО-13
(шифр групи)

Осипенко Костянтин Юрійович
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Каплунов А. В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Рецензент _____
(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

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

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

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Осипенка Костянтина Юрійовича

1. Тема проєкту Освітній веб-застосунок для репетиторів та учнів

керівник проєкту асистент Каплунов А. В.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 05 березня 2025 року №2101-с

2. Термін здачі студентом закінченого проєкту *09.06.2025*

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області

Розділ 2. Розробка веб-застосунку.

Розділ 3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) схема алгоритму роботи системи, структура схеми класів системи, архітектури системи.

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 «31» січня 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*31.01.2025*	
2.	*Вивчення та аналіз завдання*	*03.02.2025-23.02.2025*	
3.	*Розробка архітектури та загальної структури системи*	*24.02.2025-09.03.2025*	
4.	*Розробка структур окремих підсистем*	*10.03.2025-23.03.2025*	
5.	*Програмна реалізація системи*	*24.03.2025-27.04.2025*	
6.	*Оформлення пояснювальної записки*	*28.04.2025-25.05.2025*	
7.	*Захист програмного продукту*	*26.05.2025*	
8.	*Передзахист*	*26.05.2025*	
9.	*Захист*	*16.06.2025*	

Студент-дипломник _____ Костянтин ОСИПЕНКО
(підпис)

Керівник проєкту _____ Артем КАПЛУНОВ
(підпис)

АНОТАЦІЯ

Робота присвячена розробці освітньої онлайн-платформи для організації індивідуального навчання, що забезпечує ефективну взаємодію між репетиторами та учнями.

Розроблено архітектуру застосунку на основі клієнт-серверної парадигми з використанням React.js для клієнтської частини та Django для реалізації серверної частини. Імплементовано комплекс функціональних модулів, що забезпечує управління профілями користувачів, календарне планування занять, створення та структурування навчальних матеріалів.

ANOTATION

The paper focuses on the development of an online educational platform for organising individual learning that ensures effective interaction between tutors and students.

The architecture of the application based on the client-server paradigm was developed using React.js for the client side and Django for the server side. A set of functional modules has been implemented to manage user profiles, schedule classes, create and structure learning materials.

ОПИС АЛЬБОМУ
ДО ДИПЛОМНОГО ПРОЄКТУ

На тему: «Освітній веб-застосунок для репетиторів та учнів»

Київ – 2025

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	*ІАЛЦ.467200.002 ТЗ*	Освітній веб-застосунок для	4		
			репетиторів та учнів			
			Технічне завдання			
	А4	*ІАЛЦ.467200.003 ПЗ*	Освітній веб-застосунок для	64		
			репетиторів та учнів			
			Пояснювальна записка			
	А4	*ІАЛЦ.467200.004 ДА*	Освітній веб-застосунок для	1		
			репетиторів та учнів			
			Схема алгоритму роботи			
			системи			
	А4	*ІАЛЦ.467200.005 ДБ*	Освітній веб-застосунок для	1		
			репетиторів та учнів			
			Структурна схема класів			
			системи			
	А4	*ІАЛЦ.467200.006 ДВ*	Освітній веб-застосунок для	1		
			репетиторів та учнів			
			Архітектура системи			
	А4	*ІАЛЦ.467200.007 ДГ*	Освітній веб-застосунок для	22		
			репетиторів та учнів			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА	
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*		
Розроб		Осипенко К. Ю.			*Освітній веб-застосунок для репетиторів та учнів* Опис альбому	Літ. / Аркуш 1 / Аркушів 1
Перев		Каплунов А. В.				
						НТУУ “КПІ” ФІОТ ІО-13
Н. Контр		Нікольський С.С.				
Затвердив						

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Освітній веб-застосунок для репетиторів та учнів»

Київ – 2025

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Застосунок призначений для організації та підтримки дистанційної взаємодії між репетиторами та учнями в рамках онлайн-освіти.

2. ПРИЧИНИ РОЗРОБКИ

Підставою для розробки освітнього веб-застосунку для репетиторів та учнів є завдання для виконання роботи бакалаврського проєкту за освітньо-професійною програмою «Комп'ютерні системи та мережі» спеціальності 123 «Комп'ютерна інженерія», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут» імені Ігоря Сікорського.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням розробки є створення освітнього веб-застосунку, який забезпечуватиме ефективну взаємодію між викладачами та учнями та спрощення організації навчального процесу.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного проєкту є зовнішні ресурси, такі як офіційні документації, статті та публікації в мережі Інтернет та науково-технічна література.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

- Аутентифікація та авторизація користувачів.
- Керування курсами.
- Формування розкладу.

- Система чату.
- Аналітика.
- Управління профілем.

5.2 Вимоги до програмного забезпечення

- ОС Windows, Mac або Linux.
- Браузер.
- Visual Studio Code.
- Python 3.11.
- Node.js 16.
- PostgreSQL 12.

5.3 Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T
- ROM не менше ніж 64 ГБ
- RAM не менше ніж 4 ГБ.

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	31.01.2025
Вивчення та аналіз завдання	03.02.2025-23.02.2025
Розробка архітектури та загальної структури системи	24.02.2025-09.03.2025
Розробка структур окремих частин системи	10.03.2025-23.03.2025
Програмна реалізація системи	24.03.2025-27.04.2025
Виправлення помилок	28.04.2025-23.05.2025
Оформлення пояснювальної записки	24.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Освітній веб-застосунок для репетиторів та учнів*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СУБД – Система управління базами даних

API – Application Programming Interface

CORS – Cross-Origin Resourse Sharing

CSS – Cascading Style Sheets

CSRF – Cross-Site Request Forgery

DOM – Document Object Model

HTTP – HyperText Transfer Protocol

JWT – JSON Web Token

MVC – Model-View-Controller

MTV – Model-Template-View

ORM – Object-Relational Mapping

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

URI – Uniform Resourse Identifier

UTC – Coordinated Universal Time

XSS – Cross-Site Scripting

ВСТУП

Трансформація освітньої галузі під впливом цифрових технологій набуває нині глобального характеру, що підтверджується як кількісними показниками зростання ринку електронного навчання, так і якісними змінами в методиках передачі знань. Масове впровадження онлайн-освіти стало особливо актуальним на тлі пандемії COVID-19, яка у 2020 році різко змінила уявлення про традиційне навчання. Закриття навчальних закладів і необхідність соціального дистанціювання змусили освітні установи всього світу терміново переходити до дистанційного формату. Цей процес виявив як потенціал цифрових рішень, так і численні виклики, пов’язані з доступом до інтернету, цифровою грамотністю та адаптацією методик викладання.

В Україні трансформація освіти у цифровому напрямку ускладнилася ще й повномасштабною війною. Освітній процес був порушений у багатьох регіонах, частина шкіл і університетів зазнала руйнувань, частина населення була змушена евакуюватися або перейти на навчання з-за кордону. В цих умовах онлайн-освіта стала не просто альтернативою, а критично важливим інструментом для збереження доступу до знань, особливо для школярів і студентів з тимчасово окупованих або прифронтових територій.

На тлі цих подій зростає попит на гнучкі, адаптивні освітні платформи, здатні забезпечити якісну взаємодію між викладачами та учнями незалежно від їхнього місця перебування. Важливими чинниками стають надійність, масштабованість, можливість асинхронного навчання, інтерактивність контенту та ефективні інструменти оцінювання. Окрім того, зростає значення аналітики, що дозволяє відстежувати прогрес учнів і вчасно виявляти труднощі у навчанні.

У цьому контексті актуальною є розробка веб-застосунків для онлайн-освіти, які поєднують технічну стійкість з педагогічною доцільністю. Такий

підхід сприяє не лише збереженню безперервності навчання в умовах криз, але й закладає основу для нової якості освіти в цифрову епоху.

Метою даної роботи є створення інтегрованої онлайн-платформи для організації індивідуального навчання, що забезпечує взаємодію між репетиторами та учнями через реалізацію модульної архітектури з відокремленими клієнтською та серверною частинами. Для досягнення цієї мети встановлено наступні завдання: аналіз сучасних тенденцій у сфері онлайн-освіти та ідентифікація потреб цільової аудиторії; формування функціональних та нефункціональних вимог до застосунку; розробка архітектури системи та обґрунтування технологічного стеку; імплементація функціональних модулів для різних категорій користувачів; створення інтуїтивного користувацького інтерфейсу; забезпечення безпеки та надійності системи; проведення тестування та оптимізації розробленого продукту.

Практична значущість розробки визначається можливістю її застосування для вирішення реальних проблем організації репетиторської діяльності та підвищення ефективності індивідуального навчання через технологічну підтримку комунікації, планування та аналітики навчального процесу.

Дана робота ґрунтується на комплексному підході до розробки програмного забезпечення з використанням сучасних технологій веб-розробки, а саме React.js для клієнтської частини та Django з Django REST Framework для серверної частини застосунку. Методологічний підхід включає аналіз предметної області, проєктування архітектури, імплементацію функціональних компонентів, тестування продукту та оцінку результатів впровадження.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні тенденції онлайн-освіти

У XXI столітті освіта зазнає глибоких трансформацій під впливом цифрових технологій. Онлайн-освіта, яка ще кілька десятиліть тому сприймалась як альтернатива традиційному навчанню, сьогодні стала повноцінною складовою глобальної освітньої екосистеми. Особливо стрімкого розвитку цей напрям набув після пандемії COVID-19, яка змусила навчальні заклади по всьому світу перейти на дистанційний формат.

Мікронавчання утвердилося як домінуючий формат подачі освітнього контенту. Розбиття навчальних матеріалів на короткі модулі тривалістю 5-15 хвилин відповідає когнітивним особливостям сприйняття інформації та узгоджується з ритмом життя сучасної людини [10]. Технологічна реалізація цього підходу відбувається через мобільні застосунки, які забезпечують доступ до навчальних матеріалів у будь-який зручний момент.

Соціальне навчання трансформувалося з додаткового елементу в інтегральну частину онлайн-освіти. Технологічні платформи впроваджують інструменти для взаємодії між учнями, спільного розв'язання завдань та взаємного оцінювання. Практична реалізація цього підходу відбувається через форуми, чати, відеоконференції та спільні проєкти, інтегровані в єдину навчальну екосистему.

Аналітика навчальних даних перетворилася на фундаментальну складову сучасних освітніх платформ [4]. Збір та аналіз даних про взаємодію учнів з навчальними матеріалами, патерни успішності та проблемні аспекти дозволяють оптимізувати навчальний процес на індивідуальному та системному рівнях.

Монетизація онлайн-освіти демонструє диверсифікацію моделей [3]. Спостерігається зміщення від традиційної моделі разової оплати за курс до

підписних моделей, що забезпечують доступ до розширеної бібліотеки освітнього контенту. Freemium-модель, що комбінує базовий безкоштовний доступ з преміальними платними функціями, набуває поширення серед масових освітніх платформ.

Мобільне навчання утвердилося як невід'ємна складова сучасної онлайн-освіти [2]. Більшість учнів регулярно використовують мобільні пристрої для доступу до навчальних матеріалів. Технологічні рішення адаптуються до багатоекранного споживання контенту, забезпечуючи безперешкодний перехід між різними пристроями зі збереженням прогресу навчання. Розробники освітніх платформ оптимізують інтерфейси для мобільних пристроїв, впроваджують offline-режими та зменшують залежність від пропускної здатності інтернет-з'єднання.

Регуляторні аспекти та стандартизація набувають зростаючого значення для онлайн-освіти. Національні та міжнародні органи розробляють стандарти якості для електронного навчання, системи акредитації онлайн-курсів та механізми захисту даних учнів. Європейський Союз впровадив Загальний регламент про захист даних (GDPR), що встановлює суворі вимоги до обробки персональних даних, включаючи освітні платформи [9]. У США Департамент освіти встановлює стандарти для онлайн-програм, що претендують на федеральне фінансування. Ця тенденція до регуляції та стандартизації сприяє підвищенню якості та надійності онлайн-освіти [12].

Сегмент індивідуального репетиторства впевнено зростає в межах ринку онлайн-освіти. Це зростання зумовлене зростаючим попитом на персоналізоване навчання та розвитком цифрових технологій [1]. Онлайн-платформи активно впроваджують інструменти для ефективної взаємодії — інтерактивні дошки, спільний доступ до матеріалів, можливість запису занять та зручні способи оплати. Зростаюча конкуренція стимулює появу вузькоспеціалізованих сервісів, орієнтованих на окремі предмети, вікові групи або особливості національних освітніх систем.

1.2 Аналіз потреб цільової аудиторії

Дослідження потреб основних користувачів освітніх платформ є необхідною умовою для створення функціонального веб-застосунку, який відповідатиме реальним вимогам ринку освітніх послуг. Аналіз потреб репетиторів та учнів демонструє наявність специфічних запитів з обох сторін навчального процесу, що відображає дуалістичний характер взаємодії між суб'єктами освітньої діяльності.

Репетитори стикаються з комплексом організаційних викликів, серед яких питання ефективного управління розкладом занять посідає першочергове місце. Відсутність централізованої системи календарного планування призводить до нераціонального використання робочого часу та ускладнює процес координації з учнями. Тому автоматизація цього процесу дійсно є важливим аспектом, що дозволить перенаправити дані часові ресурси безпосередньо на підготовку до занять та професійний розвиток.

Інша критична потреба репетиторів пов'язана з документуванням навчального процесу та веденням обліку успішності учнів. Збереження даних про прогрес учнів, результати виконаних завдань та статистика успішності є трудомістким процесом, який часто реалізується за допомогою розрізнених інструментів, що призводить до фрагментації інформації та ускладнює аналітичну роботу.

З боку учнів спостерігається запит на доступ до структурованих навчальних матеріалів в електронному форматі. Учні зараз віддають перевагу мультимодальним форматам навчання, які включають текстові, аудіовізуальні та інтерактивні елементи. Потреба в об'єктивній оцінці прогресу навчання є характерною для учнівської аудиторії. Традиційні методи оцінювання, що базуються виключно на суб'єктивному судженні викладача, не забезпечують достатньої транспарентності та вимірюваності навчальних досягнень. Впровадження автоматизованих систем тестування та

оцінювання дозволить забезпечити комплексний аналіз успішності з використанням кількісних метрик.

Комунікаційний аспект взаємодії між репетиторами та учнями також потребує технологічного вдосконалення. Розпорошеність комунікаційних каналів, використання непрофільних месенджерів та соціальних мереж ускладнює збереження історії обговорень, обмін навчальними матеріалами та загалом знижує ефективність навчального діалогу. Інтеграція спеціалізованого комунікаційного модуля в освітній веб-застосунок забезпечить централізацію та структурування комунікаційних процесів.

Аналіз ринку освітніх технологій демонструє недостатнє забезпечення потреб репетиторів та учнів у контексті індивідуалізованого навчання. Існуючі рішення орієнтовані переважно на масові освітні сценарії та не враховують специфіку персоналізованого підходу, що є характерним для репетиторської діяльності. Розробка спеціалізованого веб-застосунку, орієнтованого на підтримку індивідуального навчання, становить актуальне завдання для ринку освітніх технологій.

Окремою категорією потреб є забезпечення доступності освітнього процесу незалежно від географічного розташування учасників. Глобалізація освітнього простору та зростаючий попит на дистанційне навчання формують запит на технологічні рішення, що уможливлюють проведення віртуальних занять з функціоналом, наближеним до очного формату взаємодії.

Потреба в аналітичних інструментах є характерною для обох категорій користувачів. Репетитори зацікавлені в аналізі ефективності власних методик викладання, виявленні проблемних аспектів у засвоєнні матеріалу учнями та оптимізації навчального процесу на основі об'єктивних даних. Учні потребують візуалізації власного прогресу, ідентифікації сильних та слабких сторін у навчанні, формування індивідуальної траєкторії розвитку. Впровадження аналітичного модуля з використанням технологій обробки

даних дозволить задовольнити дані потреби та підвищити ефективність навчального процесу.

Таким чином, комплексний аналіз потреб репетиторів та учнів демонструє необхідність розробки інтегрованого веб-застосунку, який забезпечуватиме автоматизацію організаційних процесів, централізацію комунікації, структурування навчальних матеріалів, об'єктивізацію оцінювання та аналітичну підтримку освітньої діяльності. Врахування виявлених потреб при проектуванні функціональних компонентів системи є критичним фактором для забезпечення практичної цінності розроблюваного програмного продукту.

1.3 Аналіз наявних рішень

З метою кращого розуміння поточного стану ринку онлайн-освіти, у цьому підрозділі буде проаналізовано низку популярних програмних рішень, які призначені для взаємодії між репетиторами та учнями. Розглядатимуться їхні функціональні можливості, переваги, недоліки та ступінь відповідності потребам сучасних користувачів. Це дозволить оцінити та обґрунтувати доцільність створення освітнього веб-застосунку з потрібним функціоналом.

Buki – це популярна онлайн-платформа, що поєднує репетиторів та учнів у сфері приватної освіти. Сервіс працює в Україні та деяких інших країнах Європи, забезпечуючи пошук викладачів за різними предметами, рівнем підготовки та місцем проведення занять [5].

Серед ключових функцій платформи — можливість переглядати особисті профілі репетиторів, які містять відомості про освіту, досвід, вартість занять, доступний розклад і відгуки від попередніх учнів. Завдяки системі рейтингів і відкритим коментарям формується довіра до викладача, що допомагає користувачам ухвалити зважене рішення. Зв’язок між учнем і репетитором відбувається через сайт — потенційний учень залишає заявку,

яку далі передає адміністрація платформи. Репетитори можуть підвищувати видимість свого профілю за допомогою платної підписки, що дає змогу з'являтися в топі результатів пошуку.

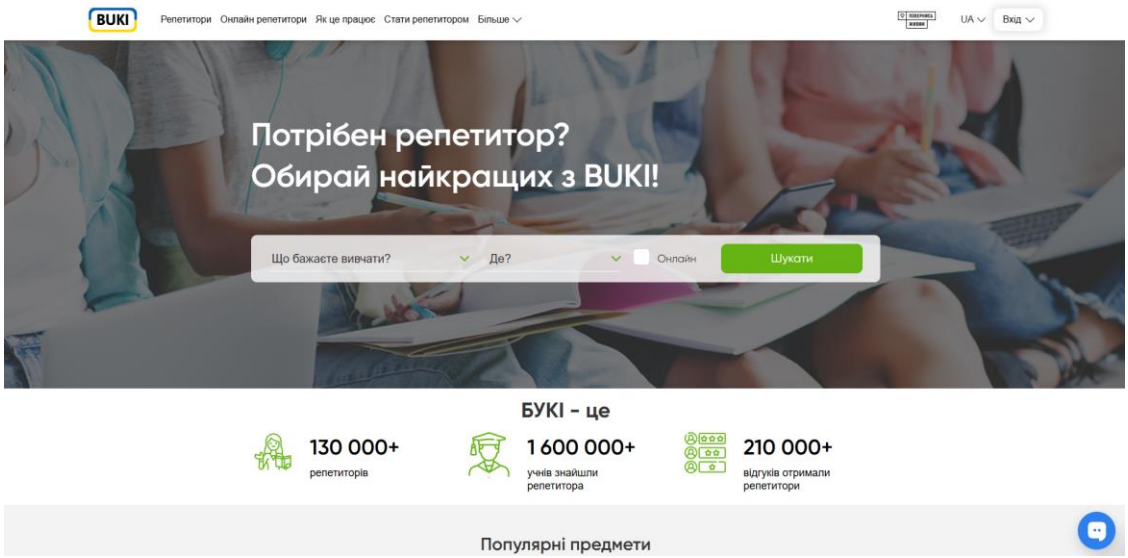


Рисунок 1.1 – Головна сторінка сайту Buki

Однак, незважаючи на зручний інтерфейс і широку базу репетиторів, функціональність Buki обмежується етапом пошуку та початкового контакту. Подальша взаємодія учня з викладачем відбувається поза межами платформи. Сервіс не надає внутрішнього чату, не підтримує інтерактивне ведення розкладу, не має інструментів для оцінювання прогресу або аналітики навчання. Відсутність електронного журналу, завантаження навчальних матеріалів, інтеграції з календарями чи можливості оплати занять через систему створює бар'єри для комплексної організації навчального процесу. Учні, своєю чергою, не мають особистих кабінетів, де могли б бачити історію занять, матеріали або оцінки.

Preply — це глобальна онлайн-платформа, що надає послуги підбору репетиторів для вивчення мов, а також інших навчальних дисциплін. Сервіс орієнтований на інтернаціональну аудиторію і підтримує учнів та викладачів з усього світу. Основна мета платформи — забезпечити безперешкодне онлайн-навчання, що максимально адаптується під індивідуальні потреби користувачів [11].

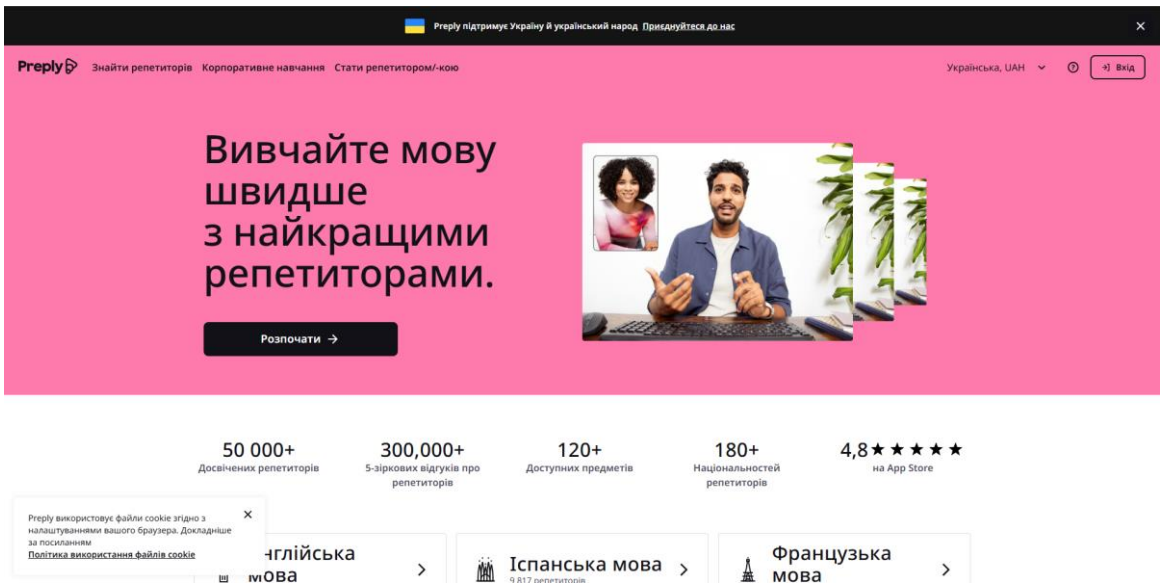


Рисунок 1.2 – Головна сторінка сайту Preply

Користувач має змогу переглядати профілі викладачів, які містять детальну інформацію про їхній досвід, кваліфікацію, відгуки, відеопрезентації та рівень володіння мовами. Вибір викладача супроводжується гнучкою системою фільтрів, а також можливістю забронювати пробне заняття. Після встановлення контакту між учнем і викладачем, уся подальша взаємодія здійснюється всередині системи: є внутрішній чат, відеозв'язок, планувальник занять, розклад, електронна оплата, збереження історії сесій і навчальних матеріалів.

Корисним є те, що даний сервіс активно підтримує навчальний процес. Сервіс надає особисті кабінети як учням, так і викладачам, із доступом до графіку занять, повідомлень, оплати та підтримки. Важливо, що платформа інтегрує оплату безпосередньо в систему, що полегшує фінансові розрахунки. Крім того, Preply пропонує інструменти для довготривалого навчання, з можливістю встановлення цілей та відстеження прогресу, хоча повноцінна аналітика успішності все ще має потенціал для розвитку.

Водночас існують і свої обмеження. Платформа фокусується здебільшого на мовному навчанні, тому інші дисципліни представлені менш широко. Ще одним аспектом є орієнтація на комерційність — сервіс утримує

значну комісію з викладачів і стимулює регулярне бронювання занять через систему, що не завжди зручно для окремих користувачів.

Coursera — це міжнародна освітня онлайн-платформа, заснована у 2012 році, яка пропонує курси, спеціалізації, сертифікаційні програми та повноцінні ступені від провідних університетів і компаній світу. Основна мета платформи — зробити якісну освіту доступною для всіх охочих у зручному онлайн-форматі [6].

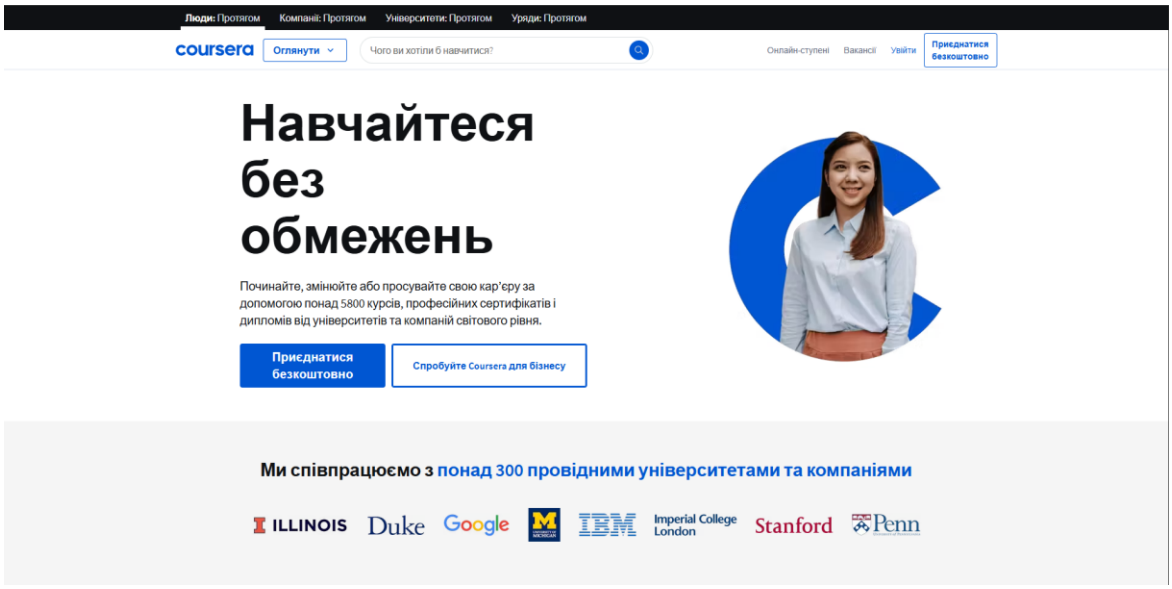


Рисунок 1.3 – Головна сторінка сайту Coursera

Користувачі можуть обирати курси з різноманітних галузей: програмування, бізнесу, гуманітарних наук, дизайну тощо. Усі курси структуровані: містять відеолекції, інтерактивні завдання, тести та форум для обговорення. Навчання проходить у власному темпі або в межах встановленого графіка, а за успішне проходження можна отримати сертифікат (безкоштовно або за оплату залежно від курсу). Деякі програми мають функцію проєктного навчання з рецензіями від інших учасників.

Coursera підтримує особисті кабінети, трекінг прогресу, мобільний доступ, а також має потужну систему рекомендацій. Завдяки партнерству з університетами та компаніями, платформа пропонує визнані сертифікати та дипломи, що підвищують шанси на працевлаштування.

Попри це, Coursera менш орієнтована на персональну взаємодію між викладачем та учнем. У більшості курсів немає прямого зворотного зв’язку з викладачем або індивідуального наставництва. Крім того, платформа більше підходить для самостійного навчання, а не для персоналізованого супроводу з боку репетитора, що обмежує її застосування в контексті приватних занять.

Аналіз наявних рішень засвідчує, що сучасні онлайн-платформи для навчання частково задовольняють потреби користувачів, однак мають певні недоліки.

1.4 Функціональні та нефункціональні вимоги до застосунку

У веб-застосунку має функціонувати комплексний механізм реєстрації та авторизації користувачів із розділенням на ролі «учень» і «викладач», що визначає подальшу диференціацію доступу до функціональних можливостей системи. Реєстрація має відбуваєтися через заповнення форми, де необхідно вказати тип облікового запису, а після успішної реєстрації дані користувача зберігаються в базі даних. Авторизація здійснюється за допомогою email та паролю, після чого користувач потрапляє на відповідний дашбоард.

Для студентів повинен передбачатися персональний кабінет, де відображається аналітична інформація про кількість активних та завершених курсів, розклад уроків і перелік записаних курсів з уроками. Звідси студент зможе переглядати всі свої курси, а також знаходити нові курси через спеціальний розділ із пошуком. Пошуковий функціонал підтримує фільтрацію за предметами, ціною, рівнем складності та іншими критеріями з можливістю комбінування фільтрів і сортування результатів. Детальний опис курсу доступний із можливістю запису на нього, якщо студент ще не є учасником курсу. Студент зможе редагувати свій профіль, змінюючи персональні дані, а також матиме можливість виходу з системи через меню користувача.

Для викладачів надаватиметься розширений функціонал, який дозволятиме створювати та керувати курсами. Вони зможуть додавати нові курси, заповнюючи інформацію про назву, опис, модулі та уроки, а також надавати можливість посилання на додаткові ресурси із навчальними матеріалами. Матеріали структуровані за темами, рівнями складності та категоріями, з можливістю встановлення послідовності їх вивчення. Викладачі зможуть редагувати, дублювати та видаляти курси, а також переглядати список студентів, які записалися на їх курси.

Профіль викладача має містити структуровану інформацію про освіту, досвід роботи з вказанням періодів і місць роботи та предмети викладання. Для комунікації між студентами та викладачами реалізується простий чат із підтримкою текстових повідомлень і сповіщеннями про нові повідомлення безпосередньо у застосунку. Для проведення онлайн-занять викладачі можуть додавати посилання на відео-зустрічі, до яких учні зможуть приєднуватись.

Важливою частиною системи буде функціонал відстеження прогресу навчання. Учні зможуть переглядати детальну інформацію про кількість курсів, на які вони зараховані, завершені курси, найближчі уроки та незавершені завдання. Аналітика для викладачів включає показники кількості студентів, рейтинг курсів та інші дані.

Щодо нефункціональних вимог, веб-застосунок повинен підтримувати роботу у сучасних браузерах з адаптивним інтерфейсом, що коректно відображається на різних пристроях і розмірах екранів. Інтерфейс зокрема має бути побудований з урахуванням принципів зручності та інтуїтивності, забезпечуватиме послідовність візуального оформлення, зрозумілі повідомлення про помилки з рекомендаціями та підказками, а також надавати можливість для зворотнього зв’язку користувачеві з індикаторами прогресу при виконанні тривалих операцій.

Для забезпечення коректної обробки даних система буде враховувати локалізацію користувача, підтримувати різні формати дат і чисел відповідно до локалі, а також зберігає всі часові дані у форматі UTC з конвертацією у локальний час користувача під час відображення.

Окрім основного функціоналу, система буде надавати простір для можливості інтеграції з іншими освітніми платформами та сервісами через відкритий API, реалізований за стандартами REST. API має підтримувати авторизацію за допомогою токенів JWT та детальну документацію з інтерактивними прикладами запитів і відповідей на основі OpenAPI.

ВИСНОВКИ ДО РОЗДІЛУ 1

За останнє десятиліття онлайн-освіта демонструє стабільне зростання та суттєву трансформацію, що відображається як у масштабах ринку, так і в зміні підходів до навчання. Репетитори стикаються з низкою організаційних труднощів, серед яких ефективне планування та управління розкладом є ключовими. Відсутність централізованої системи призводить до значних часових витрат на адміністративні завдання, що негативно впливає на якість підготовки до занять. Автоматизація цих процесів дозволить значно підвищити продуктивність та сфокусуватися на педагогічній діяльності. Учні висловлюють потребу у доступі до структурованих та мультимодальних навчальних матеріалів, що підвищують ефективність засвоєння знань. Використання інтерактивних та різноформатних ресурсів відповідає сучасним освітнім трендам та підтримується дослідженнями.

Аналіз існуючих освітніх платформ і веб-застосунків показує наявність різноманітних функціональних рішень, які забезпечують базові потреби користувачів — від планування занять до організації контенту. Це створює можливість для розробки нового продукту, що поєднає кращі практики галузі з урахуванням специфічних потреб цільової аудиторії.

Тому, розробка освітнього веб-застосунку, який враховує як функціональні, так і нефункціональні вимоги, є актуальним завданням. Такий застосунок повинен відповідати сучасним технологічним стандартам, забезпечувати зручність для користувачів та мати потенціал для подальшого розвитку й масштабування. Чітке формулювання критеріїв якості сприятиме ефективному плануванню розробки, тестуванню та поступовому впровадженню нових функцій з урахуванням зворотного зв’язку.

РОЗДІЛ 2. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

2.1 Вибір та обґрунтування технологій

Для розробки освітнього застосунку було обрано стек технологій, що забезпечує надійність, масштабованість та ефективність функціонування системи. Вибір технологічного стеку базувався на аналізі функціональних вимог до платформи, оцінці продуктивності кожної технології, а також врахуванні сучасних тенденцій в розробці веб-застосунків[14].

Архітектура системи розділена на дві основні частини: серверну (Backend) та клієнтську (Frontend), що забезпечує гнучкість, кращу підтримуваність та спрощує масштабування системи.

Для розробки серверної частини застосуємо фреймворк Django, який є потужною платформою для створення веб-застосунків на мові Python [7]. Django реалізує архітектурний патерн Model-View-Template, що забезпечує чітку структуризацію коду та розділення бізнес-логіки від представлення даних. Важливим фактором вибору Django є наявність вбудованої системи адміністрування, яка значно спрощує керування контентом платформи. Для реалізації RESTful API використаємо Django REST Framework, що надає інструменти для створення, тестування та документування програмних інтерфейсів. Django REST Framework забезпечує серіалізацію даних, валідацію запитів, керування автентифікацією та авторизацією [8].

Для реалізації безпечного механізму автентифікації можна застосувати бібліотеку djangorestframework-simplejwt, яка імплементує стандарт JSON Web Token. Даний підхід забезпечує безпечну передачу інформації про користувача між клієнтом і сервером та ефективно вирішує проблему збереження стану сесії. Для забезпечення безпечної взаємодії між різними доменами використаємо django-cors-headers, що надає можливість конфігурувати Cross-Origin Resource Sharing політики. Цей механізм

дозволяє вебсайтам на одному домені отримувати доступ до ресурсів, розміщених на іншому домені. Його основна мета — забезпечити безпеку браузера, обмежуючи можливість виконання небажаних або шкідливих запитів між різними джерелами, і так клієнтська частина зможе працювати із серверною.

Обробка мультимедійних файлів здійснюється за допомогою бібліотеки Pillow, яка надає функціонал для маніпуляції з зображеннями. Для взаємодії з базою даних PostgreSQL використаємо спеціально драйвер psycopg, що забезпечує високопродуктивний доступ до бази даних з підтримкою асинхронних операцій. Конфігурація середовища виконання буде керуватися за допомогою python-dotenv, що дозволяє зберігати конфіденційні дані поза кодом програми.

Документування API буде реалізовано за допомогою бібліотеки drf-yasg, що автоматично генерує Swagger документацію на основі коду, полегшуючи процес інтеграції з іншими системами та тестування.

Клієнтська частина платформи розроблятиметься з використанням бібліотеки React, що забезпечує компонентний підхід до розробки інтерфейсів [14]. React використовує віртуальний DOM, що дозволяє оптимізувати процес оновлення сторінки та підвищити продуктивність застосунку. Для керування станом застосунку буде використано Redux з бібліотекою @reduxjs/toolkit, що спрощує процес розробки завдяки зменшенню кількості шаблонного коду та надає інструменти для ефективного керування глобальним станом застосунку.

Маршрутизація між сторінками в клієнтському застосунку можна реалізувати за допомогою react-router-dom, що забезпечує навігацію між різними сторінками без перезавантаження. Для стилізації компонентів є сенс використати Material UI (пакети @mui/material, @mui/icons-material, @mui/lab, @mui/x-data-grid), що надає широкий набір готових компонентів, які відповідають принципам Material Design.

Для роботи з формами можна використати бібліотеку Formik разом з Yup 1.3.2 для валідації даних, що значно спрощує процес створення форм та перевірки введених користувачем даних. Взаємодія з серверною частиною здійснюватиметься за допомогою HTTP-клієнта Axios, який забезпечує зручний інтерфейс для здійснення HTTP-запитів.

Для візуалізації даних можна використати бібліотеки Chart.js версії та react-chartjs-2, що надають широкі можливості для створення інтерактивних графіків та діаграм. Для планування та відображення розкладу занять буде використано react-big-calendar, а для візуалізації проєктного планування – frappe-gantt.

Для роботи з текстовими редакторами можна застосувати бібліотеки quill та react-quill, що надають функціонал для форматування тексту. Для роботи з датами буде використано бібліотеки dayjs версії та date-fns, які забезпечують зручні інструменти для обробки та форматування дат.

2.2 Архітектура та структура застосунку

Загальна архітектура розробленого веб-застосунку побудована на основі клієнт-серверної моделі, що відповідає сучасним вимогам до розподілених вебсистем. Вона забезпечує гнучкість, масштабованість та зручність у супроводі й подальшому розширенні функціональності. Повна структурна схема архітектури веб-застосунку, яка ілюструє взаємодію між основними компонентами: клієнтською частиною (Frontend), серверною частиною (Backend), базою даних та зовнішніми сервісами, наведена у відповідному ілюстративному матеріалі (рис. 2.1).

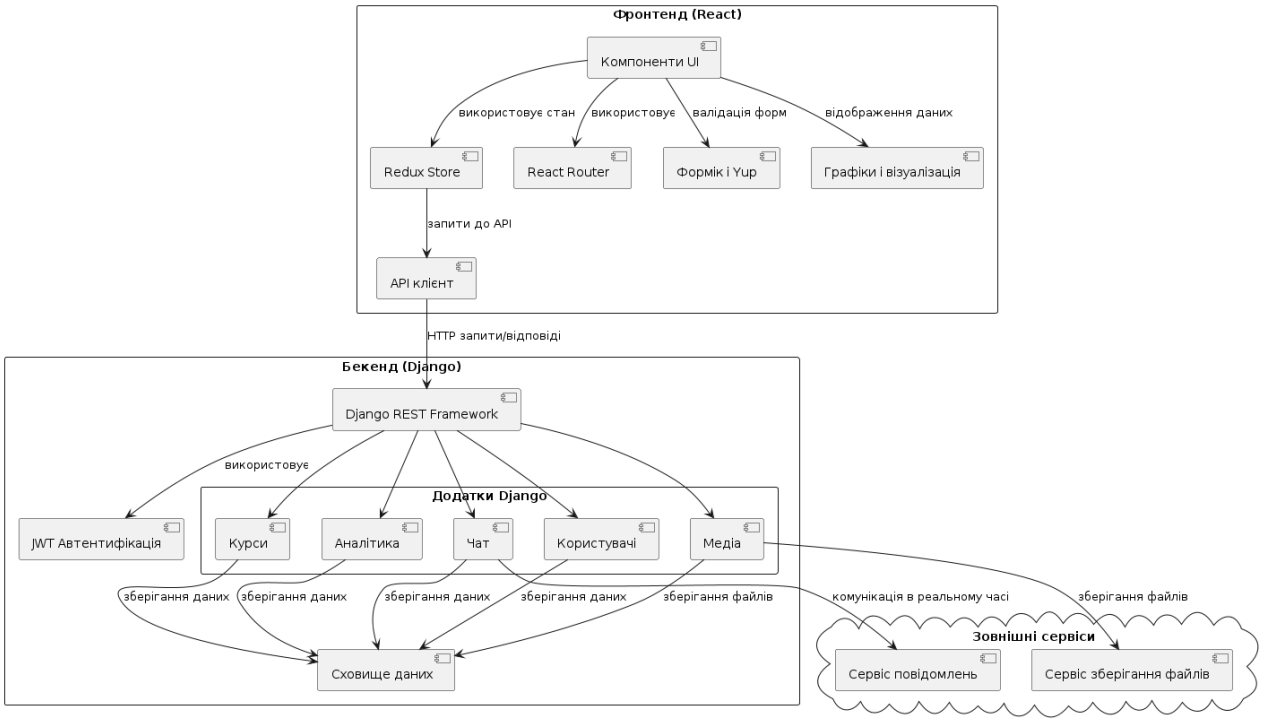


Рисунок 2.1 – Архітектура проєкту

2.2.1 Клієнтська частина

Клієнтська частина реалізована як односторінковий застосунок (Single-Page Application), що функціонує у веб-браузері користувача. Вона відповідає за презентаційну логіку та взаємодію з користувачем, забезпечуючи швидкий та інтерактивний інтерфейс без повного перезавантаження сторінки при навігації між розділами, що суттєво зменшує час відгуку системи та покращує загальну швидкодію застосунку з точки зору користувача. Технологічною основою фронтенд компонента виступає бібліотека React, що використовує декларативний підхід до побудови інтерфейсів та компонентну архітектуру. React забезпечує можливість створення ефективних компонентів інтерфейсу, які можуть бути перевикористані та легко інтегровані у різні частини системи, що значно спрощує процес розробки та підтримки застосунку. React використовує Virtual DOM (віртуальне представлення DOM), що дозволяє мінімізувати

кількість операцій з реальним DOM та підвищити продуктивність застосунку при оновленні інтерфейсу.

Для управління станом застосунку залучено Redux, що реалізує централізоване сховище даних та передбачувані зміни стану через механізм редукторів. Redux забезпечує однонаправлений потік даних, що полегшує відстеження змін стану системи, спрощує налагодження та тестування компонентів. Архітектура Redux включає декілька ключових компонентів:

- Actions - об'єкти, що описують зміни стану;
- Action Creators - функції, що створюють об'єкти дій;
- Reducers - функції, що приймають поточний стан та дію і повертають новий стан;
- Store - об'єкт, що зберігає стан застосунку, надає доступ до нього через метод getState(), дозволяє оновлювати стан через метод dispatch() та реєструє слухачів через метод subscribe().

Така архітектура дозволяє ефективно організувати взаємодію між компонентами застосунку та забезпечити централізоване управління станом.

Візуальне оформлення забезпечується фреймворком React-Bootstrap версії, який надає набір готових компонентів інтерфейсу, адаптованих для використання з React. React-Bootstrap дозволяє створювати адаптивні інтерфейси, що коректно відображаються на різних пристроях та екранах, забезпечуючи високу доступність застосунку для користувачів з різними технічними можливостями. Компоненти React-Bootstrap інтегруються з компонентами React через систему props, що дозволяє гнучко налаштовувати їх зовнішній вигляд та поведінку.

Мережева взаємодія з серверною частиною організована за допомогою бібліотеки Axios, яка абстрагує HTTP-запити та відповіді, надаючи зручний програмний інтерфейс для роботи з RESTful API. Axios підтримує Promise API, що дозволяє використовувати як асинхронні виклики через async/await, так і традиційний підхід з обробниками then/catch. Бібліотека забезпечує

автоматичне перетворення даних у форматі JSON, перехоплення запитів та відповідей, скасування запитів та захист від CSRF атак. Для організації ефективної взаємодії з API розроблено спеціальний сервісний шар, який інкапсулює логіку роботи з Axios та абстрагує деталі HTTP-взаємодії від решти компонентів системи.

Структурна організація клієнтської частини відповідає сучасним практикам розробки React-застосунків та реалізує принцип розділення відповідальності. Директорія src містить основний програмний код, поділений на логічні розділи відповідно до їх функціонального призначення. Компоненти React розміщені в директорії components та структуровані за функціональним призначенням:

- Підпапка auth містить компоненти для автентифікації та авторизації користувачів, включаючи форми входу, реєстрації та компоненти для відображення стану автентифікації.
- Підпапка charts містить компоненти для візуалізації даних у вигляді різноманітних діаграм та графіків, що використовуються для аналітики та моніторингу. Вони реалізовані з використанням бібліотеки Chart.js, яка інтегрована з React через спеціальні обгортки.
- Підпапка dashboard містить компоненти для побудови панелей управління різного призначення, включаючи статистичні віджети, інформаційні картки, панелі навігації та інші елементи, що забезпечують візуалізацію ключових показників та доступ до основних функцій системи.
- Підпапка layout містить компоненти, що відповідають за загальну структуру сторінок, включаючи заголовки, бічні панелі та основні контейнери для контенту.
- Підпапка routes містить компоненти для організації маршрутизації в застосунку, включаючи визначення доступних маршрутів,

захищені маршрути, що вимагають автентифікації, та логіку перенаправлення користувачів. Маршрутизація реалізована за допомогою бібліотеки React Router, яка забезпечує декларативний підхід до навігації в SPA.

- Підпапка ui містить набір базових інтерфейсних компонентів, таких як кнопки, форми, модальні вікна, сповіщення, випадаючі меню, які використовуються в різних частинах застосунку. Ці компоненти забезпечують єдиний стиль інтерфейсу та спрощують розробку нових сторінок та функціональності.

Сторінки застосунку згруповані у директорії pages з подальшим поділом за ролями користувачів та функціональним призначенням:

- Підпапка auth містить сторінки, пов'язані з процесами автентифікації, включаючи вхід та реєстрацію.
- Підпапка dashboard містить сторінки панелей управління для різних ролей користувачів, що надають доступ до ключових функцій системи та відображають важливу інформацію.
- Підпапка public містить публічні сторінки, доступні без автентифікації, включаючи головну сторінку, сторінку з інформацією про систему, контактну інформацію та умови використання.
- Підпапки student та teacher містять сторінки, специфічні для відповідних ролей користувачів. Сторінки студента включають доступ до курсів, матеріалів, результатів навчання та комунікацій з викладачами. Сторінки викладача містять інструменти для створення та управління курсами, матеріалами та комунікації зі студентами.

Управління станом застосунку зосереджено в директорії redux, яка містить файли з визначенням дій (actions), редукторів (reducers), селекторів (selectors) та типів (types). Дії визначають можливі зміни стану системи та

інкапсулюють логіку взаємодії з API. Редуктори визначають, як стан системи має змінюватися у відповідь на дії. Селектори надають доступ до даних зі стану застосунку та виконують необхідні перетворення даних. Типи визначають константи, що використовуються для ідентифікації дій та забезпечують типобезпеку при розробці.

Стилі оформлення розміщені в директорії styles та організовані за допомогою CSS модулів, що забезпечують інкапсуляцію стилів на рівні компонентів та запобігають конфліктам імен селекторів. Використання CSS модулів дозволяє створювати модульні та перевикористовувані компоненти без ризику побічних ефектів при зміні стилів.

Допоміжні функції та утиліти зібрані в директорії utils та включають функції для форматування дат, валідації форм, обробки помилок, локалізації, роботи з локальним сховищем та інші загальні функції, що використовуються в різних частинах застосунку.

2.2.2 Серверна частина

Серверна частина застосунку побудована на основі Python-фреймворку Django з використанням розширення Django REST Framework для реалізації API. Django реалізує модель MTV, що є варіацією класичної архітектури MVC. Модель (Model) відповідає за структуру даних та взаємодію з базою даних через ORM, що абстрагує SQL-запити та дозволяє працювати з даними на рівні об'єктів Python. Шаблон (Template) відповідає за візуальне представлення даних, хоча в контексті RESTful API його роль мінімізована, оскільки дані передаються у форматі JSON. Представлення (View) обробляє HTTP-запити, взаємодіє з моделями та формує відповіді.

Django REST Framework розширює функціональність Django для ефективної розробки RESTful API. Він надає такі компоненти як:

- серіалізатори (Serializers), які конвертують складні типи даних Python, такі як QuerySets та об'єкти моделей, у нативні типи даних Python, які потім можуть бути легко конвертовані у JSON;
- класи представлень (ViewSets), які об'єднують логіку для набору пов'язаних представлень;
- роутери (Routers), які автоматично генерують URL-патерни для ViewSets;
- механізми автентифікації та авторизації, які забезпечують контроль доступу до API;
- документацію API, яка автоматично генерується на основі коду.

Архітектура серверної частини слідує принципам REST, що передбачає стандартизований підхід до організації API з використанням HTTP методів (GET, POST, PUT, DELETE) та URI для ідентифікації ресурсів. RESTful архітектура забезпечує прозорість системи, можливість кешування, масштабованість та надійність. Кожен ресурс системи, такий як користувач, курс чи урок має унікальний URI та підтримує стандартний набір операцій. Відповіді API формуються у форматі JSON, що забезпечує максимальну сумісність та зручність обробки на стороні клієнта.

База даних системи реалізована за допомогою PostgreSQL версії 13.4, яка забезпечує надійне зберігання даних, підтримку транзакцій, складних запитів та індексів. PostgreSQL обрано через високу продуктивність, відмовостійкість, підтримку JSON та повну відповідність стандарту ACID (Atomicity, Consistency, Isolation, Durability). Взаємодія з базою даних відбувається через ORM Django, який забезпечує абстракцію SQL-запитів та дозволяє використовувати об'єктно-орієнтований підхід до роботи з даними.

Структура серверної частини організована відповідно до архітектурних принципів Django з модульним підходом до організації коду. Кожен функціональний модуль реалізований як окремий Django-застосунок з власними моделями, представленнями та URL-маршрутами:

- Модуль courses відповідає за управління навчальними курсами та включає моделі для курсів, уроків, матеріалів, завдань та прогресу навчання. Він реалізує функціональність створення, редагування, публікації курсів, управління доступом до курсів та відстеження прогресу студентів.
- Модуль users забезпечує управління користувачами системи та включає розширену модель користувача, що доповнює стандартну модель Django додатковими полями та методами. Він реалізує функціональність реєстрації, автентифікації, управління профілями, ролями та дозволами користувачів.
- Модуль analytics відповідає за збір, обробку та візуалізацію аналітичних даних про використання системи. Він включає моделі для зберігання статистики, механізми агрегації даних та API для доступу до аналітичної інформації. Модуль забезпечує моніторинг активності користувачів, аналіз ефективності навчання, визначення проблемних областей та формування рекомендацій для покращення навчального процесу.
- Модуль chat забезпечує функціональність внутрішньої комунікації між користувачами системи. Він включає моделі для зберігання повідомлень, реалізує механізми обміну повідомленнями в реальному часі за допомогою WebSockets та Django Channels, а також надає API для доступу до історії повідомлень.

Завантажені користувачами файли зберігаються в директорії media, яка організована за типами файлів та контекстом їх використання. Django забезпечує безпечне зберігання та доступ до цих файлів, включаючи контроль доступу на основі дозволів користувачів.

Головний модуль проекту osv2 містить загальні налаштування Django (settings.py), включаючи конфігурацію бази даних, статичних файлів, медіа-файлів, додатків, проміжних шарів, автентифікації, кешування та інших

аспектів роботи системи. Він також містить головні URL-маршрути (urls.py), які визначають структуру API та делегують обробку запитів відповідним застосункам.

Управління проектом здійснюється за допомогою скрипту manage.py, який надає інтерфейс командного рядка для взаємодії з проектом Django, включаючи запуск сервера розробки, виконання міграцій бази даних, створення суперкористувача, збір статичних файлів та інші адміністративні функції. Залежності проекту вказані у файлі requirements.txt, що спрощує розгортання системи в різних середовищах та забезпечує відтворюваність конфігурації.

2.3 Опис модулів та функцій застосунку

Модульна структура Backend включає спеціалізовані модулі, кожен з яких відповідає за певний функціональний аспект системи. Модуль courses відповідає за всю функціональність, пов'язану з навчальними курсами, їх структурою та контентом. В ньому реалізовано моделі даних для представлення категорій курсів (Category), самих курсів (Course), модулів курсів (CourseModule), навчальних матеріалів (CourseContent) та системи відгуків (Review). Модель Course містить інформацію про назву, опис, рівень складності, ціну, статус публікації та зображення курсу. Вона пов'язана з моделлю викладача (Teacher) через зовнішній ключ, що дозволяє контролювати права доступу та авторство. CourseModule представляє структурні одиниці курсу з власним описом, порядковим номером та набором навчальних матеріалів. CourseContent відповідає за зберігання різних типів контенту (текст, відео, посилання, файл, завдання) з можливістю вказати тривалість для часового планування. Модель Enrollment фіксує запис студентів на курси (рис. 2.2) з датою реєстрації та статусом проходження.

Модель Event дозволяє планувати події, пов'язані з курсами, такі як вебінари, дедлайни та консультації.

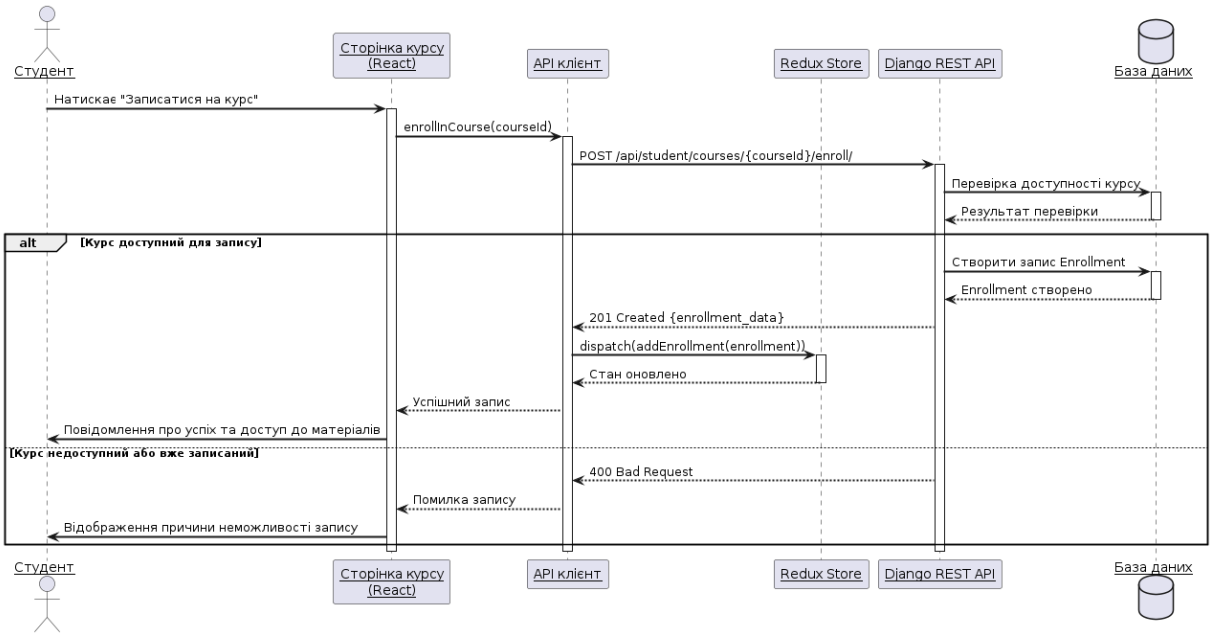


Рисунок 2.2 – Модель запису на курс

Модуль users забезпечує управління користувачами системи. Він реалізує розширену модель користувача (CustomUser), що успадковує базову Django-модель AbstractUser з додаванням специфічних полів, таких як роль користувача (студент, викладач, адміністратор), номер телефону, дата народження та аватар. Цей модуль містить реалізацію серіалізаторів для перетворення складних об'єктів користувачів у формат JSON, необхідний для передачі через API. Функціональні компоненти модуля включають систему реєстрації з валідацією даних, аутентифікацію з використанням JWT та контроль сесій користувачів. В моделі даних реалізовано розширені профілі для ролей «викладач» та «студент», що містять додаткову інформацію, специфічну для цих ролей. Викладацький профіль включає дані про освіту, досвід, спеціалізацію та рейтинг. Студентський профіль містить інформацію про освітній рівень, галузь інтересів та вподобання щодо форматів навчання.

Модуль analytics забезпечує збір, обробку та візуалізацію статистичних даних. Він зберігає в базу даних інформацію про кожну активність

користувачів: перегляди сторінок, час взаємодії з навчальними матеріалами, результати виконання завдань та проходження тестів. Аналітичний функціонал включає алгоритми агрегації даних для формування звітів щодо успішності студентів, популярності курсів, ефективності різних типів контенту та загальної статистики використання платформи. Спеціальні структури даних використовуються для ефективного зберігання часових рядів, що дозволяє будувати графіки активності з різним рівнем деталізації – від погодинної до річної. Серверна частина модуля створює підготовлені аналітичні дані, що передаються до клієнта для візуалізації у вигляді графіків, діаграм та таблиць.

Модуль chat забезпечує комунікаційну інфраструктуру застосунку. Він реалізує модель даних для зберігання чатів (Chat) і повідомлень (Message), а також механізми миттєвого оновлення з використанням технології WebSocket через Django Channels. Модель Chat представляє розмову між двома або більше користувачами з унікальним ідентифікатором, назвою, типом (особистий чи груповий) та учасниками. Message зберігає текст повідомлення, відправника, дату-час, статус прочитання та можливі вкладення. Система підтримує передачу медіафайлів через повідомлення з використанням безпечного механізму зберігання та доступу. Функціонал повідомлень інтегрується з системою сповіщень, забезпечуючи миттєве інформування користувачів про нові повідомлення через візуальні індикатори у браузері.

Структурно Frontend поділяється на ключові директорії згідно з функціональним призначенням. Директорія components містить багаторазові компоненти інтерфейсу, такі як кнопки, форми, модальні вікна, елементи навігації та індикатори стану. Кожен компонент розроблено з урахуванням принципів перевикористання та інкапсуляції, що дозволяє застосовувати їх у різних частинах додатку без дублювання коду. Директорія pages містить комплексні компоненти-сторінки, що агрегують менші компоненти для

представлення повноцінних екранів додатку. Сторінки організовані відповідно до ролей користувачів (студент, викладач) та функціональних розділів (курси, повідомлення, календар, аналітика, профіль).

Найважливішим компонентом архітектури Frontend є система управління станом на основі Redux. Модуль redux реалізує єдине джерело правди для всього додатку, що спрощує контроль за даними та їх синхронізацію між компонентами. Система включає спеціалізовані slices для різних функціональних частин: authSlice управляє станом аутентифікації, notificationsSlice відповідає за сповіщення, coursesSlice керує даними курсів, а analyticsSlice відповідає за аналітичну інформацію. Редьюсери в кожному slice визначають, як зміниться стан у відповідь на різні дії. Redux використовує односпрямований потік даних: компоненти викликають dispatches, що запускають actions, які передаються через reducers, що змінюють state, який потім відображається в компонентах. Асинхронні операції, такі як запити до API, реалізовані через Redux-thunk, що дозволяє виконувати складні послідовності дій та відстежувати їх стан. Наприклад, при завантаженні даних курсу система спочатку встановлює індикатор завантаження, потім виконує HTTP-запит, обробляє результат та оновлює глобальний стан або показує помилку.

Модуль components/layout відповідає за загальну структуру інтерфейсу застосунку. PublicLayout використовується для неаутентифікованих користувачів і включає спрощену навігацію, блок авторизації та інформаційний контент. DashboardLayout застосовується для аутентифікованих користувачів і містить персоналізовану бічну панель навігації, верхню панель з профілем користувача, сповіщеннями та пошуком, а також основний контейнер для відображення контенту. Система маршрутизації на базі React Router визначає доступні шляхи та компоненти, що відповідають їм. Компонент ProtectedRoute обгортає захищені маршрути, перевіряючи стан аутентифікації користувача та його роль, перенаправляючи

на сторінку входу при відсутності авторизації або на повідомлення про помилку при спробі доступу до непризначених для конкретної ролі функцій.

Функціональність створення та управління курсами реалізована через комплекс взаємопов'язаних компонентів. CourseCreate надає інтерфейс для створення нових курсів з багатокроковою формою, що дозволяє поступово заповнювати інформацію, включаючи базові дані, опис, модулі та матеріали. Форма використовує бібліотеку Formik для керування станом і валідацією даних. CourseEdit розширює функціональність CourseCreate, додаючи завантаження існуючих даних курсу для редагування. TeacherCourses надає викладачам панель управління їхніми курсами з можливістю фільтрації за статусом, сортування та пошуку. Компонент TeacherCourseView забезпечує детальний перегляд окремого курсу з можливістю управління студентами, модулями та матеріалами. Цей компонент реалізує каскадний підхід до запитів даних, використовуючи альтернативні API при відмові основного, що підвищує надійність системи. Для кожного курсу також реалізовано систему керування статистикою та успішністю студентів.

Система комунікації представлена компонентами TeacherMessages та StudentMessages, що забезпечують інтерфейс для обміну повідомленнями. Компоненти реалізують двопанельний інтерфейс зі списком чатів та вікном активного чату, підтримують завантаження історії повідомлень, індикацію прочитання та статусу онлайн, а також можливість обміну файлами. Повідомлення оновлюються в реальному часі завдяки використанню WebSocket-підключення, що забезпечує миттєвий обмін без необхідності постійного опитування сервера. У серверній частині цей функціонал забезпечується Django Channels, що розширює стандартний HTTP-підхід Django для підтримки протоколів реального часу.

Аналітична функціональність доступна через компоненти TeacherAnalytics та StudentAnalytics. Перший надає викладачам систему звітності та статистики щодо їхніх курсів, включаючи відвідуваність,

успішність студентів, популярність матеріалів та фінансові показники. Другий показує студентам їхній прогрес, розподіл часу, результати навчання та рекомендації щодо покращення ефективності. Обидва компоненти використовують бібліотеки Recharts та Chart.js для візуалізації даних у вигляді різноманітних діаграм і графіків, що дозволяє ефективно представляти комплексну статистичну інформацію.

Система авторизації користувачів (рис. 2.3) реалізована з використанням JWT (JSON Web Token), що забезпечує безпечний та ефективний механізм аутентифікації без необхідності зберігання стану сесії на сервері. Токени доступу (access token) та оновлення (refresh token) зберігаються в локальному сховищі браузера.

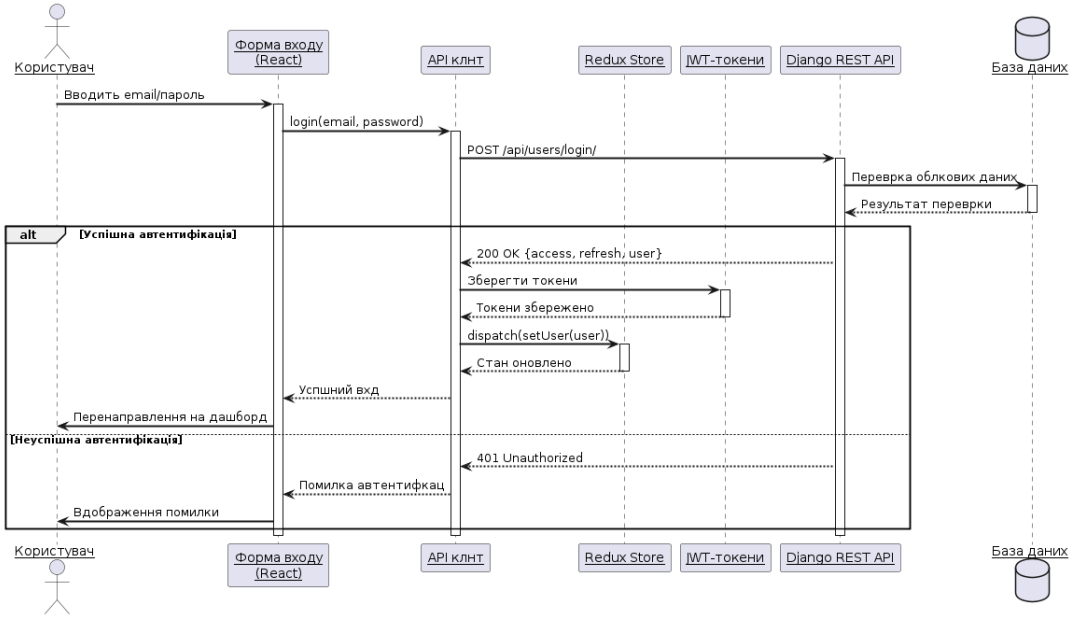


Рисунок 2.3 – Модель авторизації користувача

Інтерцептори в бібліотеці axios налаштовані для автоматичного додавання токена доступу до заголовка Authorization кожного HTTP-запиту, що забезпечує безперервну аутентифікацію без додаткових дій користувача. При закінченні терміну дії токена доступу система автоматично використовує токен оновлення для отримання нової пари токенів, забезпечуючи безперебійну роботу без повторної авторизації. Цей механізм

реалізований через спеціальний інтерцептор відповідей, що перехоплює помилки 401 (Unauthorized) та ініціює процес оновлення токенів.

Модуль middleware.py у Backend частині містить реалізації проміжного програмного забезпечення для аутентифікації, логування, обробки крос-доменних запитів та безпеки. Проміжний шар JWT-аутентифікації перевіряє валідність токенів доступу, декодуючи їх і виконуючи валідацію підпису, терміну дії та змісту. Особливу увагу приділено безпеці токенів – вони зберігаються з відповідними атрибутами безпеки (HTTPOnly, Secure, SameSite) для захисту від XSS та CSRF атак.

Система бази даних в Django використовує ORM для абстрагування від конкретної СУБД та надання зручного програмного інтерфейсу для роботи з даними. Моделі даних визначають структуру бази даних та взаємозв'язки між таблицями. Міграційна система Django забезпечує контроль версій схеми бази даних, що дозволяє безпечно вносити зміни в структуру даних без втрати існуючої інформації. Кожна міграція представляє собою програмну трансформацію схеми, яка може бути застосована або скасована. Це забезпечує можливість точного відстеження та відтворення стану бази даних на будь-якому етапі розробки.

Модуль management/commands містить набір команд командного рядка для адміністративних функцій. Команда create_test_data генерує демонстраційні дані для розробки та тестування, включаючи користувачів з різними ролями, курси з модулями та контентом, записи на курси, повідомлення та аналітичні дані. Команда runserver_with_testdata об'єднує запуск сервера розробки з генерацією тестових даних для швидкого налаштування середовища розробки. Команда reset_test_data дозволяє скинути тестові дані до початкового стану без впливу на реальні дані. Ці інструменти значно спрощують процес розробки та тестування системи.

Компонент CourseProgress відстежує прогрес студента в курсі через візуальний індикатор та детальну статистику по кожному модулю. Він

взаємодіє з аналітичним модулем для збереження даних про завершення уроків, результати тестів та час, витрачений на навчання. Система автоматично визначає поточний статус проходження курсу на основі співвідношення завершеного та загального контенту, враховуючи вагові коефіцієнти для різних типів матеріалів.

Механізм маршрутизації запитів у Django (urls.py) визначає відповідність між URL-шляхами та функціями-обробниками або класами представлень. Кожен модуль містить власний urls.py з локальними маршрутами, які потім включаються до глобальної маршрутизації через механізм include(). Це забезпечує модульність та ізоляцію маршрутів різних функціональних частин системи. На клієнтській частині React Router виконує подібну функцію, визначаючи компоненти, що відповідають різним URL-шляхам, підтримуючи вкладену структуру маршрутів, параметри у шляхах та програмну навігацію.

Система безпеки застосунку реалізує багаторівневий підхід до захисту даних та функціональності. Серверна частина використовує декоратори доступу та класи дозволів Django REST Framework для контролю доступу до API-ендпоінтів на основі ролі та автентифікації користувача. Для кожного ендпоінту визначено набір дозволів, що вимагаються для виконання різних HTTP-методів. Наприклад, GET запити до списку курсів можуть бути доступні всім користувачам, але створення нових курсів (POST) доступне лише викладачам. Клієнтська частина доповнює цю систему через компоненти умовного рендерингу, що показують або приховують елементи інтерфейсу на основі ролі та прав користувача.

2.4 Користувацький інтерфейс та досвід користувача

Користувацький інтерфейс навчального застосунку розроблено з використанням компонентного підходу на базі бібліотеки React та сучасних

фреймворків для побудови інтерфейсів. Архітектура інтерфейсу побудована відповідно до принципу розділення відповідальності, де кожен компонент відповідає за конкретну функціональну частину та має власний життєвий цикл. Такий підхід забезпечує модульність, повторне використання компонентів та спрощує підтримку коду.

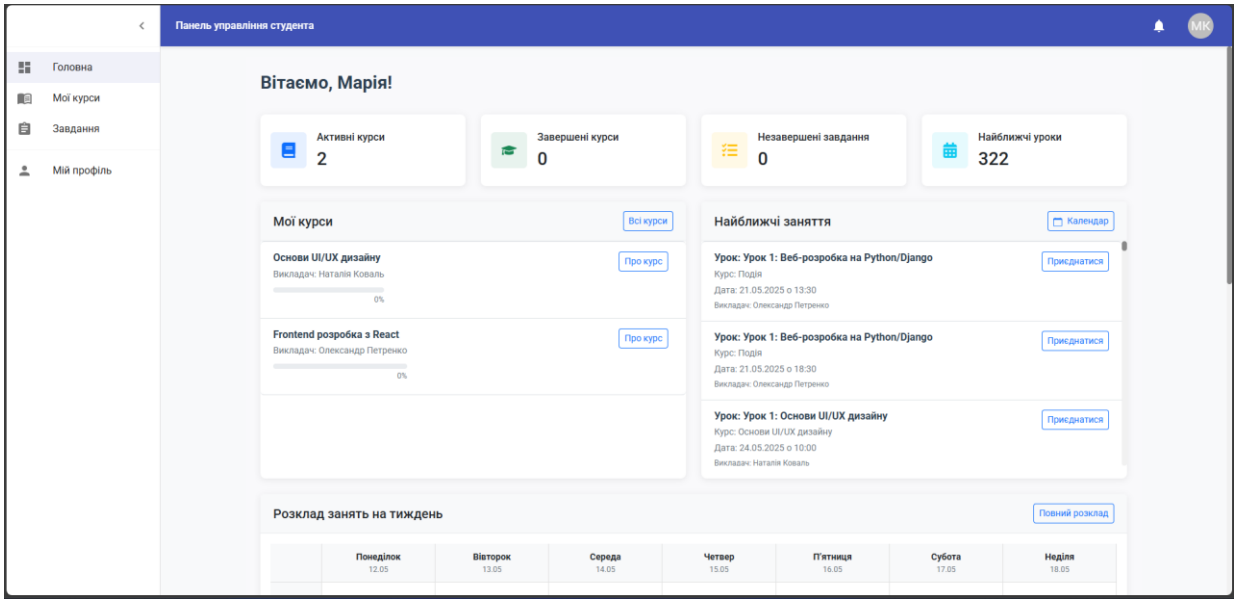


Рисунок 2.4 – Панель керування студента

Основу дизайн-системи застосунку складає бібліотека Material-UI, яка реалізує принципи Material Design від Google. Вибір цієї бібліотеки обумовлений її відповідністю сучасним стандартам проєктування інтерфейсів, широким набором готових компонентів та можливістю гнучкого налаштування відповідно до потреб проєкту. Базові компоненти з бібліотеки @mui/material, такі як Button, Card, TextField, Select, Dialog, забезпечують консистентність інтерфейсу та стандартизовану поведінку елементів управління.

Інтерфейс застосунку побудовано за принципом адаптивного дизайну, що забезпечує коректне відображення на пристроях з різною роздільною здатністю екрану - від мобільних телефонів до настільних комп'ютерів. Реалізація адаптивності здійснюється через використання сітки з Material-UI та медіа-запитів CSS. Для структурування контенту на сторінках

використовуються компоненти Container, Row та Col з бібліотеки react-bootstrap, які забезпечують гнучке розташування елементів залежно від розмірів екрану.

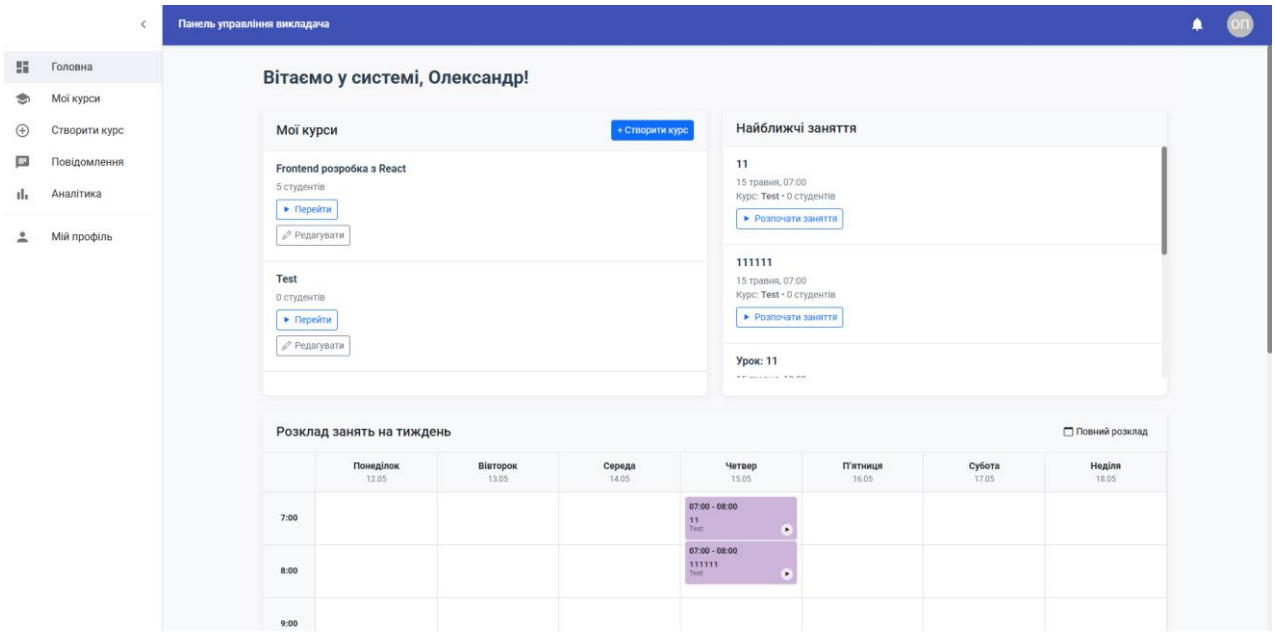


Рисунок 2.5 – Панель керування викладача

Кольорова схема інтерфейсу визначена в файлі theme.js через налаштування теми Material-UI. Палітра кольорів розроблена з урахуванням принципів доступності та містить основні, додаткові та нейтральні кольори. Забезпечено достатній контраст між текстом та фоном для полегшення читання інформації. Система типографіки також визначена в темі та включає ієрархію заголовків, параграфів та інших текстових елементів з відповідними розмірами, інтервалами та товщиною шрифту.

Структура інтерфейсу застосунку складається з кількох ключових компонентів: заголовок (Header), бічна панель навігації (Sidebar), основний контент та нижній колонтитул (Footer). Заголовок містить логотип, навігаційні елементи та панель користувача з доступом до профілю та налаштувань. Бічна панель реалізує ієрархічну навігацію, яка адаптується залежно від ролі користувача (студент, викладач, адміністратор). Основний контент змінюється динамічно відповідно до обраного розділу.

Навігація в застосунку реалізована за допомогою бібліотеки React Router DOM, яка забезпечує клієнтську маршрутизацію без перезавантаження сторінки. Структура маршрутів визначена в компоненті routes та включає публічні, захищені та рольові маршрути. Публічні маршрути доступні всім користувачам, захищені вимагають автентифікації, а рольові перевіряють також роль користувача. Така система забезпечує контроль доступу на рівні інтерфейсу та підвищує безпеку застосунку.

Для відображення даних у табличному форматі використовується компонент DataGrid з бібліотеки @mui/x-data-grid, який забезпечує функціональність сортування, фільтрації, пагінації та експорту даних. Таблиці використовуються для відображення списків студентів, курсів, уроків та інших сутностей системи. Для покращення сприйняття значних обсягів даних застосовано чергування кольорів рядків та виділення активного рядка.

Візуалізація даних реалізована за допомогою бібліотек Chart.js та Recharts, які забезпечують створення різноманітних графіків та діаграм. В компонентах аналітики, таких як StudentsAnalytics та TeacherAnalytics, використовуються лінійні графіки для відображення динаміки показників, стовпчасті діаграми для порівняння значень та кругові діаграми для відображення розподілу. Графіки інтерактивні та забезпечують можливість отримання детальної інформації при наведенні курсору.

Управління формами в застосунку здійснюється за допомогою бібліотеки Formik, яка забезпечує валідацію даних, обробку подій та зручний API для роботи з формами. Схеми валідації визначені за допомогою бібліотеки Yup, яка дозволяє декларативно описувати правила валідації полів. Повідомлення про помилки відображаються безпосередньо під відповідними полями, що покращує зворотний зв'язок для користувача та спрощує виправлення помилок.

Для роботи з датами та часом використовуються компоненти з бібліотеки @mui/x-date-pickers, які забезпечують зручний вибір дати та часу в різних форматах. Базова обробка дат здійснюється за допомогою бібліотек moment та date-fns, які надають широкий функціонал для форматування, аналізу та маніпуляцій з датами. В календарних компонентах, таких як розклад занять, використовується бібліотека react-big-calendar для відображення подій у різних режимах (день, тиждень, місяць).

Для редагування текстового контенту курсів та уроків використовується компонент ReactQuill, який надає функціональний текстовий редактор з можливістю форматування тексту, додавання зображень, таблиць та інших елементів. Цей компонент інтегрований з системою завантаження файлів, що дозволяє викладачам додавати медіа-контент безпосередньо з редактора.

Обробка помилок в інтерфейсі реалізована на кількох рівнях. На рівні компонентів використовуються компоненти Error Boundary з React для локалізації помилок та запобігання краху всього застосунку. На рівні запитів до API реалізована система обробки помилок, яка перехоплює помилки мережі та сервера, відображає відповідні повідомлення користувачу та пропонує варіанти вирішення проблеми. У випадку недоступності основного API застосовується каскадний підхід з використанням альтернативних джерел даних.

Стан застосунку управляється за допомогою Redux з використанням бібліотеки @reduxjs/toolkit, яка спрощує роботу з Redux через автоматичне створення дій, редукторів та селекторів. Такий підхід забезпечує централізоване зберігання стану, прогнозовані оновлення та можливість відстеження змін через Redux DevTools.

Для покращення досвіду користувача реалізовано систему сповіщень та зворотного зв'язку. Сповіщення про успішні операції, помилки та інформаційні повідомлення відображаються у вигляді тостів, які

автоматично зникають через певний час. Критичні дії, такі як видалення даних, вимагають підтвердження через діалогові вікна, що запобігає випадковим операціям.

Навантаження та продуктивність інтерфейсу оптимізовані через використання ледачого завантаження компонентів (lazy loading), віртуалізації списків для відображення великих наборів даних та мемоізації компонентів з використанням React.memo та useMemo. Це забезпечує швидкий відгук інтерфейсу навіть при роботі з великими обсягами даних.

Анімації та переходи в інтерфейсі реалізовані з використанням бібліотеки @emotion/react, яка дозволяє створювати стилізовані компоненти з підтримкою анімацій. Анімації застосовуються помірно для привернення уваги до важливих змін в інтерфейсі, таких як поява нових елементів або зміна стану, при цьому уникаючи надмірного використання ефектів, які можуть відволікати користувача.

Розроблений користувацький інтерфейс та запроваджені практики проєктування користувацького досвіду забезпечують ефективну взаємодію користувачів з системою, спрощують виконання типових задач та сприяють досягненню навчальних цілей платформи. Модульна архітектура та використання стандартизованих компонентів з бібліотек Material-UI та Bootstrap забезпечують консистентність інтерфейсу та можливість його подальшого розвитку.

ВИСНОВКИ ДО РОЗДІЛУ 2

У рамках розділу 2 було здійснено проєктування та розробку веб-застосунку для освітньої платформи з використанням сучасних технологій та архітектурних рішень. Було обрано відповідний технологічний стек, який включає Django з використанням Django REST Framework для розробки серверної частини та React з Redux для клієнтської частини. Застосування такого стеку технологій забезпечує надійність, масштабованість та ефективність функціонування системи, поділяючи архітектуру на незалежні компоненти.

Розроблена архітектура системи базується на клієнт-серверній парадигмі, де взаємодія між цими частинами відбувається через RESTful API з використанням формату JSON, що забезпечує стандартизований механізм обміну даними. Для автентифікації користувачів застосовано механізм JWT, який забезпечує безпечну передачу інформації між клієнтом і сервером без необхідності зберігання стану сесії.

Структурна організація застосунку реалізована за модульним принципом з чітким розподілом відповідальностей. Серверна частина включає спеціалізовані модулі для управління навчальними курсами, користувачами, аналітикою та комунікацією. Клієнтська частина організована у відповідності до функціональних областей системи з використанням компонентного підходу React.

Система включає різні функціональні модулі, що забезпечують повний цикл управління освітньою платформою. Модуль "courses" відповідає за управління навчальними курсами, їх структурою та контентом. Модуль "users" забезпечує управління користувачами, їх автентифікацію та профілями. Аналітичний модуль надає можливості збору, обробки та візуалізації статистичних даних. Модуль комунікації реалізує обмін повідомленнями.

Користувацький інтерфейс розроблено з використанням компонентного підходу на базі бібліотеки React та Material-UI, що забезпечує модульність та відповідність сучасним стандартам дизайну. Реалізовано адаптивний дизайн, який забезпечує коректне відображення на пристроях з різною роздільною здатністю. Для управління станом застосунку використано Redux, що забезпечує централізоване зберігання даних та передбачувані зміни стану.

Система безпеки застосунку реалізує багаторівневий підхід до захисту даних, включаючи контроль доступу на основі ролей, JWT для автентифікації та захист від поширених типів атак. Продуктивність оптимізована через використання ледачого завантаження компонентів, віртуалізації списків та інших технік оптимізації.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Тестування застосунку

Тестування розробленого навчального застосунку відбувалось за кожною роллю користувача в системі.

3.1.1 Тестування ролі студента

1. Реєстрація

Для реєстрації у застосунку як студент потрібно заповнити форму із даними та обрати тип облікового запису «студент» (див. рис.3.3).

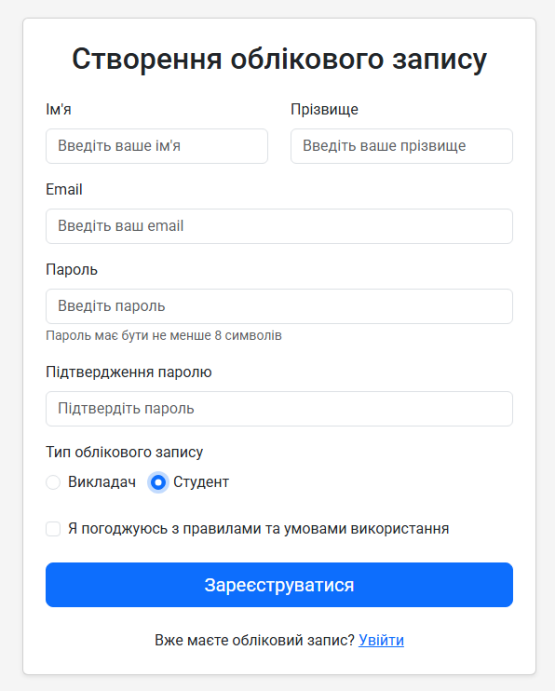


Рисунок 3.1 – Форма реєстрації

Після заповнення всіх полів та натиснення кнопки «Зареєструватись» новий користувач створюється в базі даних і далі можна буде авторизовуватись у своєму профілі.

2. Авторизація

Для авторизації потрібно заповнити форму із полями email та пароль (див. рис.3.2).

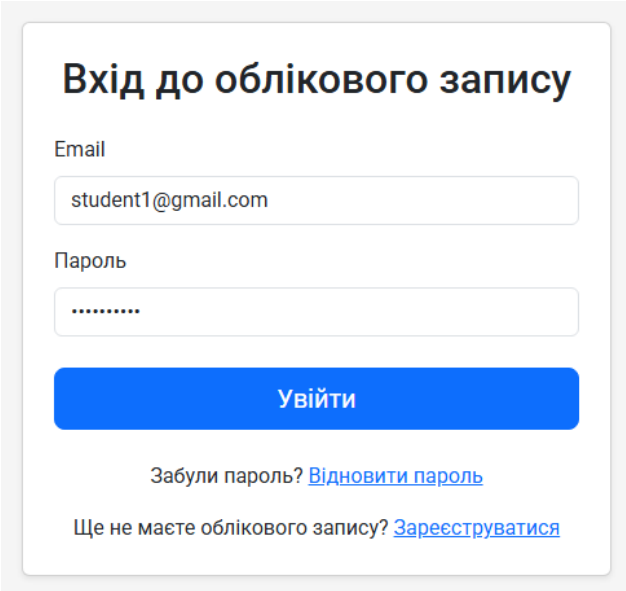


Рисунок 3.2 – Форма авторизації студентського профілю

3. Панель управління

Після авторизації відкривається дашбоард із актальної інформацією про користувача (див. рис.3.3). На дашбоарді виводиться аналітична інформація по студенту, це кількість активних курсів, завершених курсів, на які курси записаний студент, розклад занять, тощо.

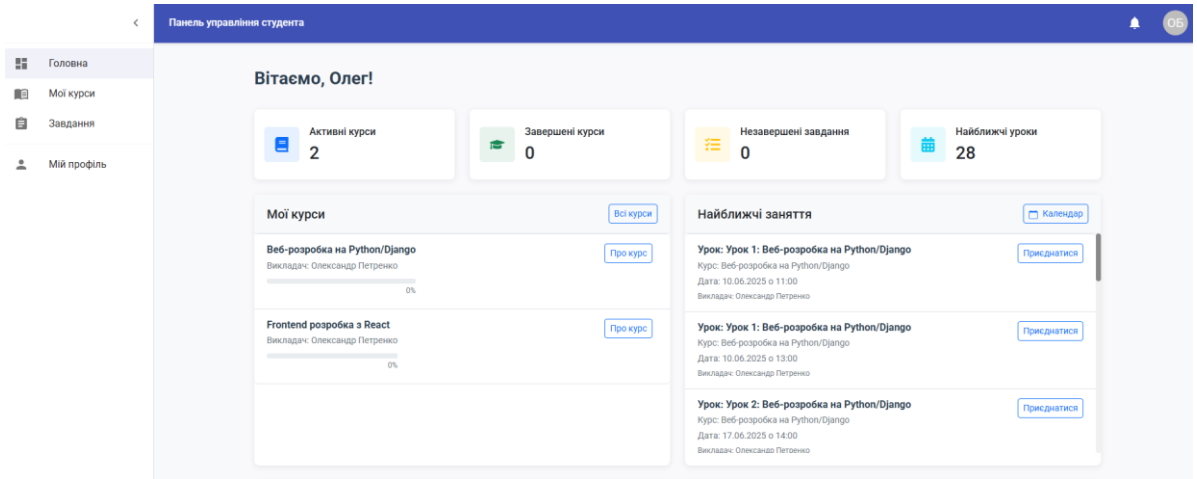


Рисунок 3.3 – Дашбоард студента

Зліва є меню із можливістю передивитись всі курси студента (див. рис.3.4). Якщо потрібно знайти ще курси, для цього можна перейти на повний список курсів через кнопку «Знайти нові курси».

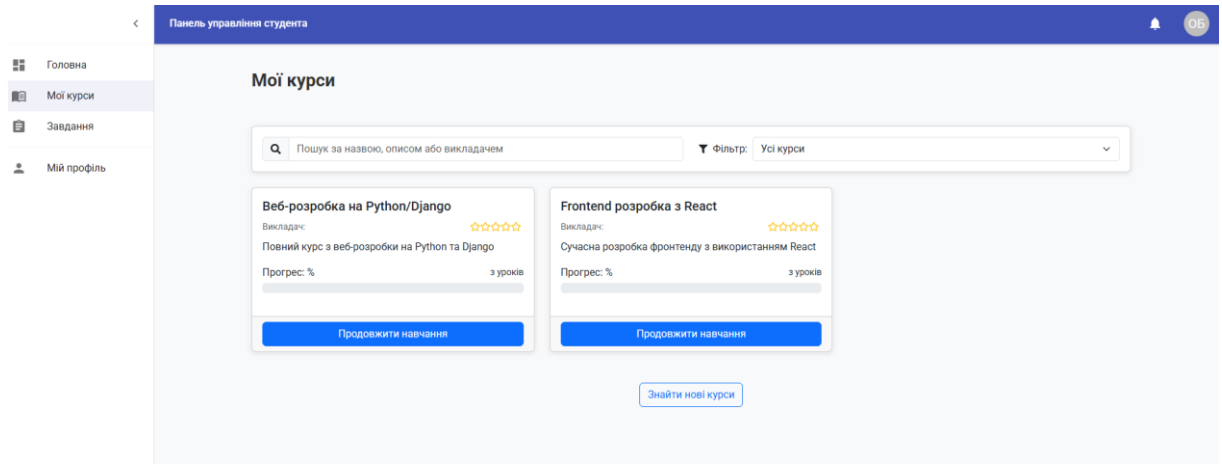


Рисунок 3.4 – Вкладка «Мої курси»

З цієї сторінки можна перейти на детальний опис курсу. Для цього потрібно натиснути кнопку «Детальніше» (див. рис.3.5). Якщо студент вже записаний на курс про це буде запис, якщо ще ні, буде можливість записатись на курс.

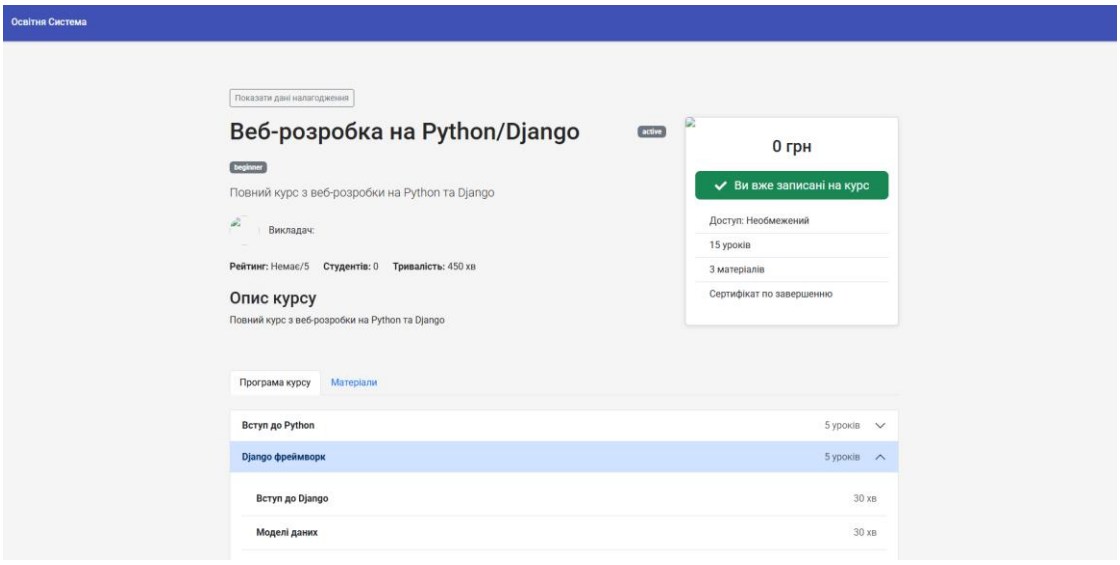


Рисунок 3.5 – Детальний опис курсу із заняттями

Кожен студент може змінювати дані свого профілю (див. рис.3.6), для цього може перейти в профіль через ліву кнопку «Профіль» або через випадаюче меню при натисненні на аватар.

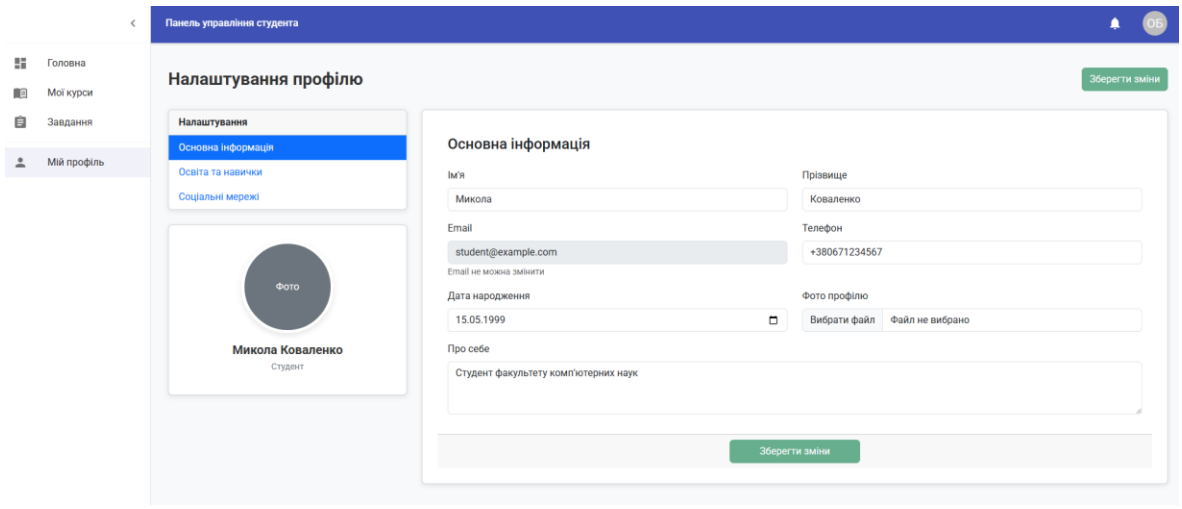


Рисунок 3.6 – Налаштування студентського профілю

Щоб завершити роботу із акаунтом, потрібно натиснути кнопку вийти у випадаючому меню (див. рис.3.7).

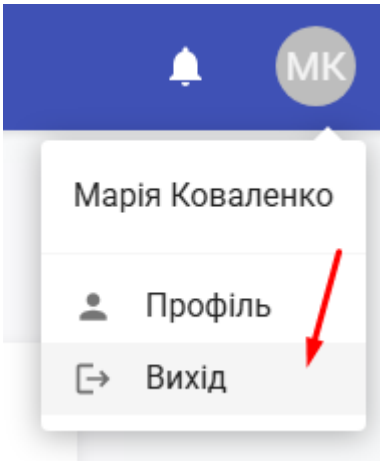


Рисунок 3.7 – Кнопка виходу користувача із профілю

3.1.2 Тестування ролі викладача

1. Реєстрація

Для реєстрації у застосунку як викладач потрібно заповнити форму із даними та обрати тип облікового запису «викладач». Після заповнення всіх полів та натиснення кнопки «Зареєструватись» новий користувач створюється в базі даних і далі можна буде авторизовуватись у своєму профілі.

2. Авторизація

Для авторизації потрібно заповнити форму із полями email та пароль (див. рис.3.8).

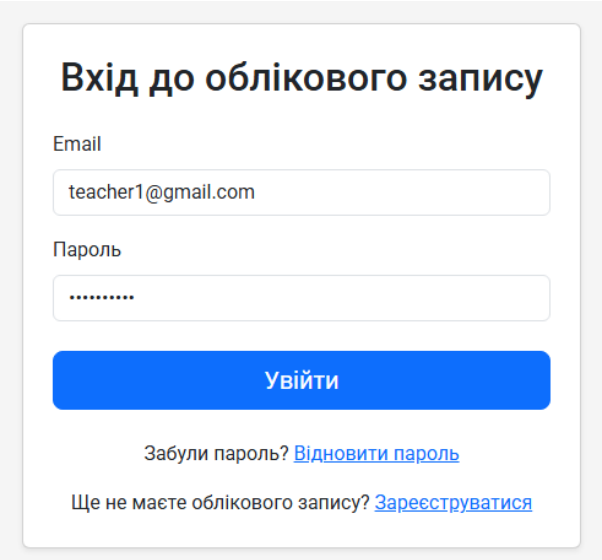


Рисунок 3.8 – Форма авторизації профілю викладача

3. Панель управління викладача

Після авторизації відкривається дашбоард із актульної інформацією про користувача (див. рис.3.9). На дашбоарді виводиться аналітична інформація по викладачу, це кількість курсів, найближчі заняття та розклад, тощо.

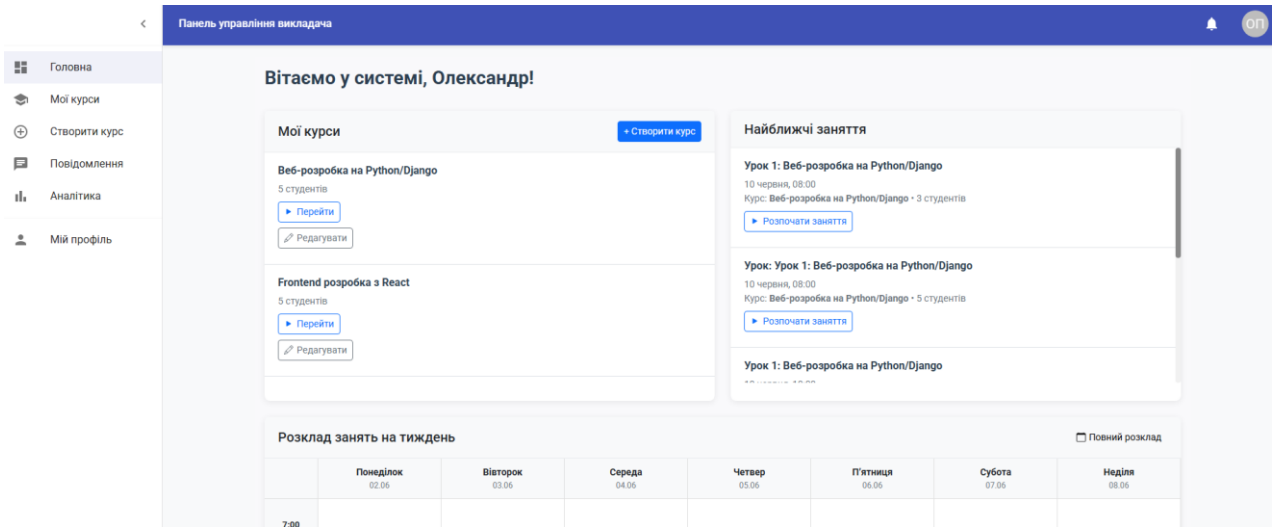


Рисунок 3.9 – Дашбоард викладача

Зліва є меню із можливістю передивитись всі курси створені викладачем (див. рис.3.10). На цій вкладці є можливість відредагувати курс, продивитись курс, подивитись студентів, які записались на курс, дублювати курс чи видалити його.

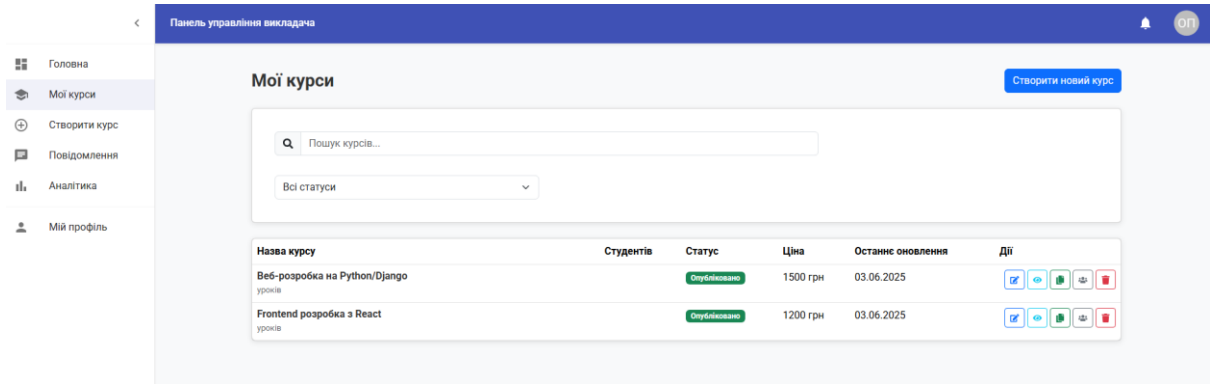


Рисунок 3.10 – Вкладка «Мої курси»

Наступна вкладка дає можливість створити новий курс. Для цього потрібно заповнити необхідну форму вказуючи дані про курс: назва, опис, модулі, уроки та матеріали (див. рис.3.11).

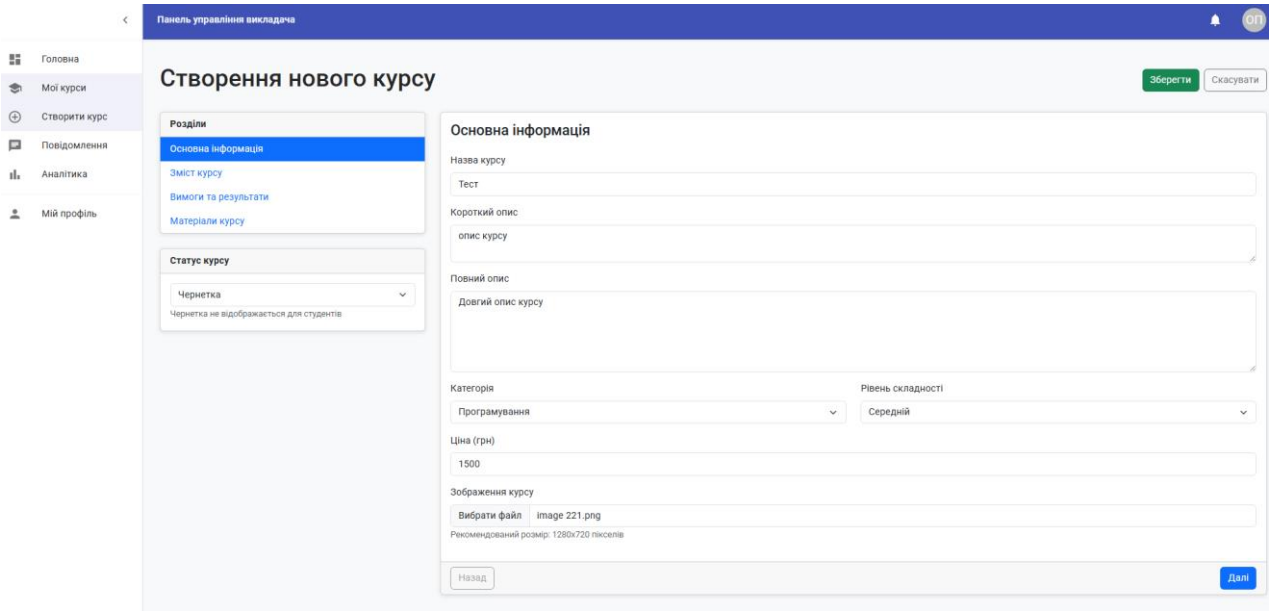


Рисунок 3.11 – Створення нового курсу

Вкладка «Повідомлення» дає можливість продивитись особисте листування із студентами, які записались на курс (див рис.3.12).

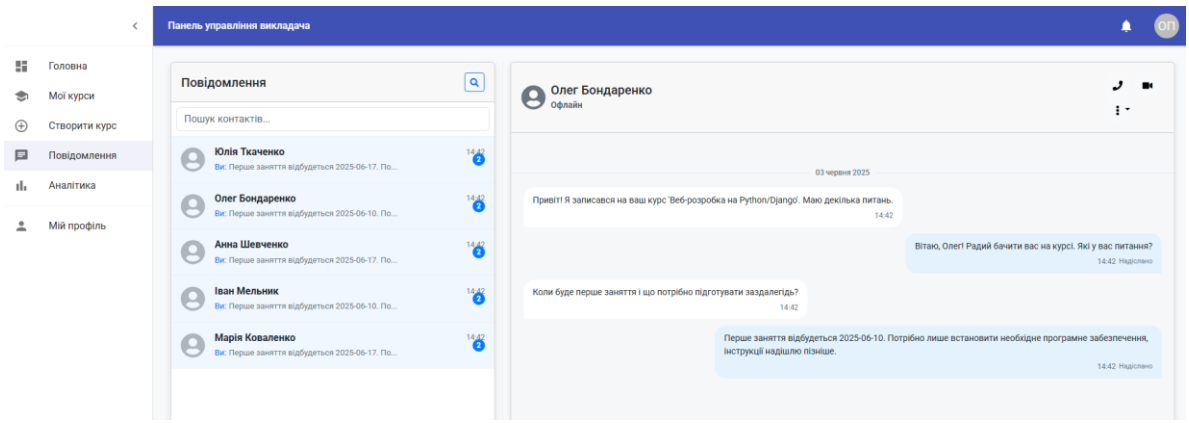


Рисунок 3.12 – Листування із студентом

У вкладці «Аналітика» виводиться аналітика по курсам: кількість студентів, курсів, рейтинг, тощо (див. рис.3.13).

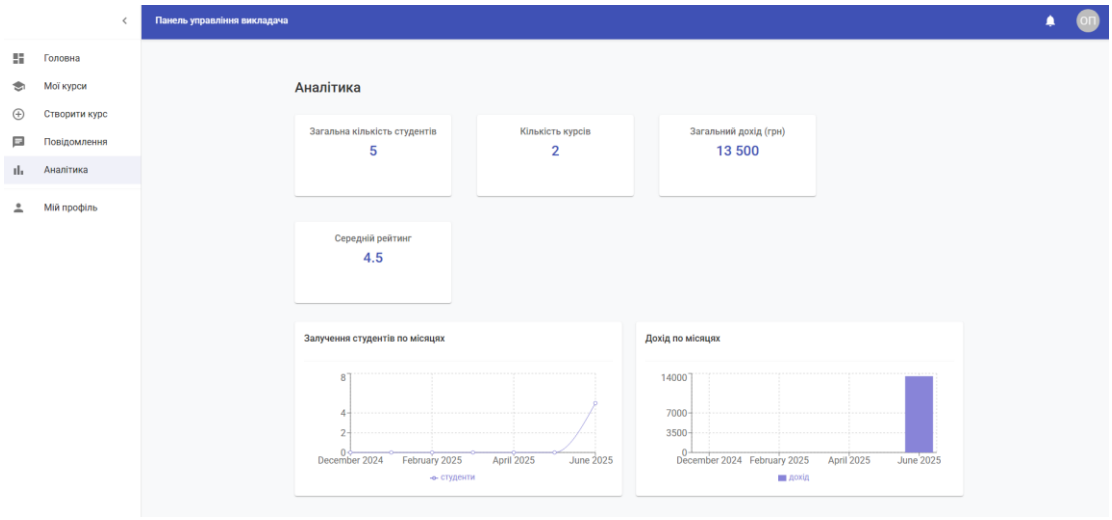


Рисунок 3.13 – Сторінка із аналітикою

Також у викладача є профіль, який можна редагувати (див. рис.3.14).

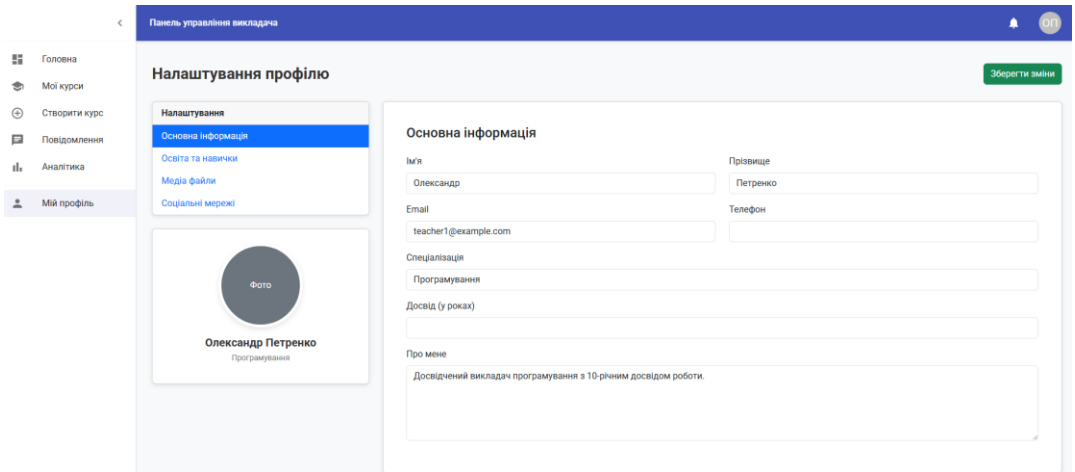


Рисунок 3.14 – Профіль викладача

Для виходу із системи потрібно обрати «Вихід» у випадаючому меню (див. рис.3.15).

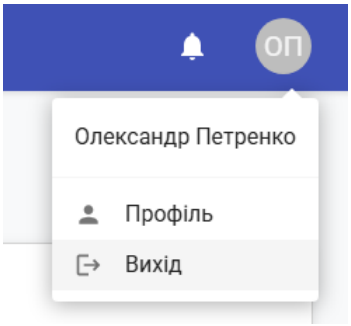


Рисунок 3.15 – Кнопка «Вихід» із профілю

3.2 Оцінка результатів тестування

Нижче наведено оцінку результатів тестування навчального застосунку для ролей «Студент» і «Викладач» у вигляді таблиць. Таблиці містять перелік протестованих функцій, їх опис, результат тестування та зауваження.

Таблиця 3.1 – Результати тестування ролі «Студент»

Функція	Опис	Результат	Зауваження
Реєстрація	Заповнення форми з даними, вибір типу «студен»" (рис.3.1).	Успішно	Форма інтуїтивна, помилок не виявлено.
Авторизація	Введення email та паролю для входу (рис.3.2).	Успішно	Швидкий доступ до дашбоарду.
Дашбоард студента	Відображення аналітики: активні/завершені курси, розклад (рис.3.3).	Успішно	Інтерфейс зручний, дані актуальні.
Вкладка "Мої курси"	Перегляд курсів студента (рис.3.4).	Успішно	Навігація зручна, функції працюють.
Пошук нових курсів	Перехід до списку курсів через кнопку "Знайти нові курси" (рис.3.4).	Успішно	Список курсів доступний, працює коректно.
Детальний опис курсу	Перегляд опису курсу, статус запису, можливість записатися (рис.3.5).	Успішно	Функція запису працює, статус відображається.
Редагування профілю	Зміна даних профілю через вкладку або меню (рис.3.6).	Успішно	Редагування стабільне, помилок немає.
Вихід із системи	Завершення сесії через кнопку "Вихід" (рис.3.7).	Успішно	Сесія завершується коректно.

Усі ключові функції системи було успішно протестовано: реєстрація, авторизація, робота з дашбордом, перегляд і пошук курсів, редагування профілю та вихід із системи. Інтерфейс інтуїтивний, стабільний, без помилок, що забезпечує комфортну та надійну роботу користувача.

Таблиця 3.2 – Результати тестування для ролі «Викладач»

Функція	Опис	Результат	Зауваження
Реєстрація	Заповнення форми з даними, вибір типу "викладач" (рис.3.8).	Успішно	Форма інтуїтивна, помилок не виявлено.
Авторизація	Введення email та паролю для входу (рис.3.8).	Успішно	Швидкий доступ до дашбоарду.
Дашбоард викладача	Відображення аналітики: кількість курсів, розклад, заняття (рис.3.9).	Успішно	Інтерфейс зручний, дані актуальні.
Вкладка "Мої курси"	Перегляд, редагування, дублювання, видалення курсів, список студентів (рис.3.10).	Успішно	Усі функції працюють коректно.
Створення нового курсу	Заповнення даних про курс: назва, опис, модулі, уроки (рис.3.11).	Успішно	Процес створення інтуїтивний.
Повідомлення	Листування зі студентами, записаними на курс (рис.3.12).	Успішно	Листування працює без збоїв.
Аналітика	Відображення даних: кількість студентів, курсів, рейтинг (рис.3.13).	Успішно	Дані відображаються коректно.
Редагування профілю	Зміна даних профілю викладача (рис.3.14).	Успішно	Редагування стабільне, помилок немає.
Вихід із системи	Завершення сесії через кнопку "Вихід" (рис.3.15).	Успішно	Сесія завершується коректно.

Усі основні функції для ролі викладача протестовано успішно. Реєстрація, авторизація, робота з курсами, аналітикою, повідомленнями та профілем відбувається без помилок. Інтерфейс зручний та зрозумілий, функціональність повністю відповідає потребам викладача.

3.3 Безпека та надійність застосунку

Безпека та надійність розробленого навчального застосунку забезпечуються комплексом технічних рішень та організаційних заходів, які впроваджені на всіх рівнях системи. Система автентифікації користувачів

базується на використанні JWT-токенів, що забезпечує надійний механізм перевірки ідентичності користувачів без необхідності зберігання сесій на сервері. Технологія JWT реалізована за допомогою бібліотеки djangorestframework-simplejwt, яка надає дворівневу систему токенів: короткострокові access-токени для доступу до ресурсів та довгострокові refresh-токени для оновлення доступу без повторної автентифікації.

Авторизація в застосунку реалізована на основі рольової моделі, де кожен користувач має одну з двох ролей: студент або викладач. Розмежування прав доступу здійснюється через систему дозволів Django, яка перевіряє повноваження користувача перед виконанням кожної операції. Для API-ендпоінтів використовуються спеціалізовані класи дозволів, які перевіряють роль користувача та його відношення до ресурсів, до яких здійснюється доступ. Такий підхід забезпечує принцип мінімальних привілеїв, коли кожен користувач має доступ лише до тих функцій та даних, які необхідні для виконання його задач.

Захист від SQL-ін'єкцій забезпечується використанням ORM Django, який автоматично екранує параметри запитів до бази даних. Додатково, для запобігання атак типу XSS використовується автоматичне екранування HTML-тегів у шаблонах Django, а на стороні React застосовується принцип React DOM Escaping, який забезпечує автоматичне екранування даних при відображенні інтерфейсу. Захист від CSRF-атак реалізовано через використання CSRF-токенів для всіх модифікуючих запитів до сервера.

Безпека передачі даних забезпечується використанням протоколу HTTPS, який шифрує всі дані, що передаються між клієнтом та сервером. Для налаштування HTTPS використовуються сертифікати SSL/TLS, які забезпечують шифрування трафіку та підтверджують автентичність сервера. Для роботи з чутливими даними, такими як паролі користувачів, застосовується одностороннє хешування з використанням алгоритму PBKDF2 з сіллю, що є стандартним методом зберігання паролів у Django.

Система обробки помилок в застосунку реалізована з урахуванням принципів безпеки. Усі помилки логуються для подальшого аналізу, але користувачам відображаються лише загальні повідомлення про помилки без розкриття внутрішньої інформації про систему. Це реалізовано як на рівні API через обробники виключень Django, так і на рівні інтерфейсу користувача через компоненти обробки помилок React. Аналіз логів помилок здійснюється автоматизовано для виявлення потенційних атак та аномальної поведінки.

Надійність застосунку забезпечується через механізми обробки конкурентних запитів та управління навантаженням. Асинхронна обробка запитів на фронтенді реалізована з використанням Promise API та Redux Toolkit, що дозволяє ефективно керувати станом застосунку навіть при високому навантаженні. Особлива увага приділяється обробці мережевих помилок – застосовано каскадний підхід до запитів, коли у випадку недоступності основного API використовуються альтернативні джерела даних.

Проксіювання запитів між фронтендом та бекендом реалізовано через http-proxy-middleware, що забезпечує єдину точку доступу до API та спрощує налаштування CORS. Це підвищує безпеку застосунку, обмежуючи крос-доменні запити лише дозволеними джерелами. Додатково реалізовано обробку помилок проксі для забезпечення стабільної роботи навіть при тимчасовій недоступності бекенд-сервера.

Моніторинг безпеки та продуктивності застосунку здійснюється через інтегровані інструменти логування та аналізу. Всі критичні операції, такі як автентифікація користувачів, модифікація даних та доступ до чутливої інформації, фіксуються в логах з відповідними метаданими для подальшого аудиту.

Система тестових даних, реалізована в застосунку, забезпечує можливість безпечного тестування функціональності без ризику для

продуктивних даних. Тестові користувачі та дані ізольовані від реальних даних, що дозволяє проводити всебічне тестування системи.

Управління залежностями в проєкті здійснюється через файли package.json для фронтенду та requirements.txt для бекенду, де чітко зафіксовані версії всіх використаних бібліотек. Це забезпечує стабільність середовища та мінімізує ризики, пов'язані з несумісністю версій компонентів.

3.4 Напрямки розвитку застосунку

Подальший розвиток освітньої платформи для викладачів та студентів передбачає комплексний підхід до вдосконалення функціональності та розширення можливостей системи. Аналіз поточної архітектури та функціоналу дозволяє визначити кілька ключових напрямків розвитку, які забезпечать підвищення ефективності навчального процесу та покращення взаємодії між всіма учасниками освітнього процесу.

Першим напрямком є інтеграція розширених аналітичних інструментів для аналізу навчальної активності. Наявна система вже містить базові аналітичні компоненти, однак їх функціональність може бути суттєво розширена шляхом впровадження алгоритмів машинного навчання для прогнозування успішності студентів на основі патернів їхньої активності. Це дозволить викладачам ідентифікувати потенційні проблеми у навчальному процесі та вживати проактивних заходів щодо їх усунення. Впровадження розширеного модуля аналітики потребуватиме вдосконалення структури бази даних та додавання нових ендпоінтів API для обробки та візуалізації комплексних даних.

Другим напрямком розвитку є створення адаптивної системи навчання, що дозволить автоматично налаштовувати навчальний контент відповідно до індивідуальних потреб та рівня знань кожного студента. Поточна архітектура

системи з розділенням на фронтенд та бекенд компоненти забезпечує належну основу для впровадження такого функціоналу. Реалізація адаптивного навчання вимагатиме розробки алгоритмів оцінювання рівня засвоєння матеріалу, створення банку завдань різного рівня складності та модифікації навчальних модулів для підтримки динамічного контенту

Третім напрямком є розширення комунікаційних можливостей системи шляхом впровадження інтегрованих інструментів для синхронного онлайн-навчання. Аналіз існуючої структури чату показує, що система вже має базові комунікаційні можливості, однак їх можна суттєво розширити шляхом додавання підтримки відеоконференцій, спільних дошок для малювання та інтерактивних презентацій. Технічна реалізація цього напрямку потребуватиме інтеграції з протоколами WebRTC для забезпечення передачі аудіо- та відеоданих у режимі реального часу, а також розробки нових компонентів інтерфейсу на фронтенді.

Четвертим напрямком є інтеграція з зовнішніми освітніми ресурсами та системами, що дозволить розширити навчальний контент та забезпечити безперервність освітнього процесу. Реалізація цього напрямку передбачає розробку API-мостів для взаємодії з популярними освітніми платформами, бібліотеками цифрових матеріалів та системами управління навчанням. Технічно це вимагатиме створення абстрактного рівня для уніфікації даних з різних джерел та їх інтеграції в єдину систему.

П’ятим напрямком є впровадження технологій розподіленого навчання для забезпечення масштабованості системи та підтримки великої кількості одночасних користувачів. Це потребуватиме оптимізації архітектури бекенду, впровадження механізмів кешування даних та балансування навантаження між серверами. Важливим компонентом цього напрямку є також оптимізація фронтенду для зменшення навантаження на клієнтські пристрої та забезпечення швидкої роботи системи навіть за умов обмеженого інтернет-з’єднання.

3.5 Можливості масштабування застосунку

Масштабування освітнього веб-застосунку потребує комплексного підходу до архітектури системи та організації процесів розробки. Аналіз поточної структури застосунку, що базується на React у фронтенд частині та Django у бекенд компонентах, демонструє наявність фундаментальних можливостей для горизонтального та вертикального розширення. Розглянемо технічні аспекти масштабування та методологію запуску продукту в контексті збільшення навантаження та розширення функціональності.

Вертикальне масштабування бекенд частини застосунку може бути реалізоване шляхом оптимізації програмного коду та вдосконалення архітектури баз даних. Поточна архітектура використовує Django з традиційним монолітним підходом, що може створювати обмеження при значному збільшенні навантаження. Трансформація архітектури в напрямку мікросервісів дозволить розподілити навантаження між окремими функціональними компонентами системи. Кожен з виділених сервісів – керування користувачами, курсами, аналітикою, повідомленнями – може бути розгорнутий на окремих серверах та масштабований незалежно відповідно до специфічних потреб.

Горизонтальне масштабування передбачає збільшення кількості серверів для обробки зростаючого обсягу запитів. Впровадження балансувальників навантаження, таких як Nginx або HAProxy, забезпечить рівномірний розподіл запитів між серверами. Синхронізація даних між серверами вимагатиме переходу від локального зберігання сесій до централізованих рішень, як-от Redis або Memcached. Для забезпечення надійності збереження даних рекомендується імплементація реплікації бази даних та системи автоматичного резервного копіювання.

Оптимізація фронтенд частини на React потребує впровадження методів кешування та динамічного завантаження компонентів. Аналіз структури package.json демонструє використання значної кількості залежностей, які можуть збільшувати розмір бандлу JavaScript. Впровадження технік розділення коду (code splitting) та ліниве завантаження компонентів (lazy loading) дозволить зменшити початковий обсяг даних, необхідних для завантаження сторінки, що особливо важливо для користувачів з обмеженою швидкістю інтернет-з'єднання.

Запуск продукту в промислову експлуатацію вимагає розробки стратегії поетапного впровадження. Рекомендується використати підхід поступового розгортання (rolling deployment) для мінімізації ризиків при переході від тестового середовища до виробничого. Початковий етап запуску може включати обмежену групу користувачів для валідації функціональності та продуктивності системи в реальних умовах. Процес розгортання має бути автоматизований з використанням інструментів безперервної інтеграції та розгортання (CI/CD), таких як Jenkins, GitHub Actions або GitLab CI.

Контейнеризація застосунку з використанням Docker забезпечить стандартизацію середовища розробки та розгортання, що зменшить вірогідність виникнення помилок, пов'язаних з відмінностями в конфігурації серверів. Використання оркестрації контейнерів через Kubernetes дозволить автоматизувати масштабування, розгортання, балансування навантаження та самовідновлення системи при відмовах.

Моніторинг продуктивності системи після запуску є критичним аспектом забезпечення стабільності функціонування. Імплементація систем моніторингу, таких як Prometheus та Grafana, дозволить відстежувати ключові метрики продуктивності: час відгуку сервера, завантаженість процесора та пам'яті, кількість активних сесій, швидкість мережі. Аналіз цих

даних надасть можливість ідентифікувати вузькі місця системи та приймати обґрунтовані рішення щодо подальшого масштабування.

Резервне копіювання та відновлення даних є невід'ємною частиною стратегії масштабування. Створення регулярних резервних копій бази даних, конфігураційних файлів та користувацького контенту забезпечить можливість швидкого відновлення системи при виникненні критичних ситуацій. Резервні копії мають зберігатися географічно розподілено для захисту від локальних аварій та стихійних лих.

Масштабування системи авторизації та автентифікації потребує особливої уваги, оскільки ці компоненти забезпечують безпеку всієї системи. Використання JWT токенів, як вже впроваджено в поточній архітектурі, має бути доповнено механізмами валідації на стороні бекенду, централізованим управлінням сесіями та моніторингом підозрілої активності. Для забезпечення масштабованості багатокористувацького доступу рекомендується впровадження рольової моделі з гранулярними правами, що дозволить гнучко налаштовувати доступ до різних функцій системи.

ВИСНОВКИ ДО РОЗДІЛУ 3

У рамках третього розділу дипломної роботи було здійснено детальний опис процесу реалізації та тестування освітньої онлайн-платформи для забезпечення ефективного навчального процесу. Розробка проєкту проходила за структурованою методологією, починаючи з аналізу предметної області, де було вивчено потреби студентів та викладачів, та закінчуючи оптимізацією розгортання системи. Розроблено архітектуру з чітким розділенням на клієнтську та серверну компоненти, що забезпечило гнучкість та модульність системи. Клієнтська частина реалізована на базі React.js з використанням Redux для управління станом та Material UI для побудови інтерфейсу. Backend розроблено на Django з Django REST Framework, що забезпечило надійне API для взаємодії з клієнтською частиною. Комплексна система генерації тестових даних дозволила ефективно перевіряти різні сценарії використання платформи без ризику для продуктивних даних.

Тестування програмного забезпечення здійснювалося за допомогою різних методик та інструментів.. Виявлено та усунуто проблеми з тестувальними даними та API зв'язками шляхом створення мосту між API для різних ролей користувачів та впровадження паралельної обробки запитів. Тестування різних ролей системи показало відповідність до поставлених вимог та підтвердило стабільну роботу ключових компонентів системи.

Безпека та надійність застосунку забезпечуються комплексом технічних рішень, включаючи JWT-автентифікацію, рольову модель авторизації та захист від поширених типів атак. Реалізовано резервне копіювання даних на рівні бази даних та файлової системи для забезпечення відновлення після можливих аварійних ситуацій. Масштабованість системи досягнута через модульну архітектуру та чітке розділення функціональних блоків. Впроваджено проксіювання запитів між frontend та backend

компонентами, що забезпечує єдину точку доступу до API та спрощує налаштування CORS.

Визначено п’ять ключових напрямків розвитку застосунку: інтеграція розширених аналітичних інструментів з використанням машинного навчання, створення адаптивної системи навчання, розширення комунікаційних можливостей через інтеграцію відеоконференцій, інтеграція з зовнішніми освітніми ресурсами та впровадження технологій розподіленого навчання. Запропоновано стратегію масштабування системи, яка включає трансформацію архітектури в напрямку мікросервісів, горизонтальне масштабування через збільшення кількості серверів, оптимізацію frontend частини та контейнеризацію застосунку. Рекомендовано поетапне впровадження продукту з автоматизацією процесу розгортання та впровадженням систем моніторингу для відстеження ключових метрик продуктивності.

ВИСНОВКИ

У результаті виконаної роботи розроблено освітній веб-застосунок, що відповідає сучасним тенденціям розвитку електронного навчання та задовольняє потреби ключових користувачів – викладачів та студентів. Проведений аналіз предметної області дозволив ідентифікувати визначальні характеристики сучасного етапу розвитку онлайн-освіти, зокрема персоналізацію навчального процесу, мікронавчання, соціальне навчання, інтеграцію штучного інтелекту та аналітику навчальних даних. Дослідження потреб цільової аудиторії виявило комплекс специфічних запитів з боку репетиторів щодо ефективного управління розкладом занять, документування навчального процесу, обліку оплати послуг та з боку учнів щодо доступу до структурованих навчальних матеріалів, об'єктивної оцінки прогресу та централізованої комунікації з викладачами.

На основі сформульованих функціональних та нефункціональних вимог розроблено архітектурне рішення системи, що базується на клієнт-серверній парадигмі з чітким розподілом функціональних обов'язків між різними компонентами. Клієнтська частина реалізована як односторінковий застосунок на базі React.js з використанням Redux для управління станом та Material UI для побудови користувацького інтерфейсу. Серверна частина імплементована на фреймворку Django з Django REST Framework для створення програмних інтерфейсів та PostgreSQL для зберігання даних. Структурна організація проєкту відповідає принципам модульності та розділення відповідальності, що забезпечує гнучкість розробки та підтримки системи.

Реалізація проєкту проводилась за структурованою методологією з послідовним виконанням етапів аналізу, проєктування, розробки клієнтської та серверної частин, інтеграції компонентів, тестування та оптимізації. Особлива увага приділялась забезпеченню безпеки та надійності застосунку

через впровадження JWT-автентифікації, рольової моделі авторизації та захисту від різних типів атак. Комплексне тестування системи включало автоматизоване тестування компонентів, модульне тестування функціональності, інтеграційне тестування взаємодії підсистем та тестування з використанням згенерованих даних. Результати тестування підтвердили відповідність розробленої системи функціональним та нефункціональним вимогам та безпеки даних.

Реалізований застосунок представляє собою функціональне рішення, що враховує тенденції розвитку освітніх технологій та потреби користувачів. Модульна структура застосунку та використання сучасного стеку технологій забезпечують стабільність функціонування, можливість розширення функціональності та адаптації до змінюваних вимог ринку освітніх послуг. Результати роботи демонструють створення працездатного рішення для організації індивідуального навчання, яке вже зараз може бути корисним для окремих користувачів та викладачів. Водночас система має значний потенціал для подальшого розвитку. Це робить проєкт перспективною основою для подальших ітерацій та вдосконалень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Биков В. Ю. Про наукову і науково-організаційну діяльність Інституту цифровізації освіти НАПН України за 2018-2022 рр. та перпективи його розвитку. Вісник Національної академії педагогічних наук України. 2022. Т. 4, вип. 1.

2. Лютвієва Я. П., Полежаєва О. В. Цифрове навчання : ключові терміни та поняття. Педагогічні науки: реалії та перспективи. 2022. Т. 5, вип. 87.

3. Мельникова О., Олійник Ю. Особливості функціонування ринку онлайн-освіти у світі та Україні. Економічний дискурс. 2020. вип. 3. С. 16–27.

4. Осадчий В. Сучасні тенденції цифровізації управлінських процесів у вищій освіті: аналітика даних, хмарні технології, штучний інтелект. Освітологічний дискурс. 2024. Т. 1, вип. 44.

5. Buki [Електронний ресурс] – Режим доступу до ресурсу: https://buki.com.ua/ (дата звернення 07.05.2025)

6. Coursera [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/ (дата звернення 07.05.2025)

7. Django documentation. Django [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/ (дата звернення: 03.04.2025)

8. Django REST framework. Home - Django REST framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.django-rest-framework.org/ (дата звернення: 03.04.2025)

9. GDPR Compliance in the Education Sector: Protecting Student Data in Learning Environments [Електронний ресурс] // GDPR Advisor – Режим доступу до ресурсу: https://www.gdpr-advisor.com/gdpr-compliance-in-the-education-sector-protecting-student-data-in-learning-environments/ (дата звернення: 06.05.2025)

10. Mohammed G. S., Wakil K., Nawroly S. S. The Effectiveness of Microlearning to Improve Students' Learning Ability. *International Journal of Educational Research Review*. 2018. T. 3 : Issue 1. C. 32–38.

11. Preply [Електронний ресурс] – Режим доступу до ресурсу: https://preply.com/ (дата звернення 07.05.2025)

12. Program Integrity and Institutional Quality: Distance Education and Return of Title IV, HEA Funds. Federal Register [Електронний ресурс] – Режим доступу до ресурсу: https://www.federalregister.gov/documents/2025/01/03/2024-31031/program-integrity-and-institutional-quality-distance-education-and-return-of-title-iv-hea-funds (дата звернення: 06.05.2025)

13. React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/ (дата звернення: 03.04.2025)

14. Technology. 2024 Stack Overflow Developer Survey [Електронний ресурс] – Режим доступу до ресурсу: https://survey.stackoverflow.co/2024/technology#1-web-frameworks-and-technologies (дата звернення: 14.05.2025)

ДОДАТОК А

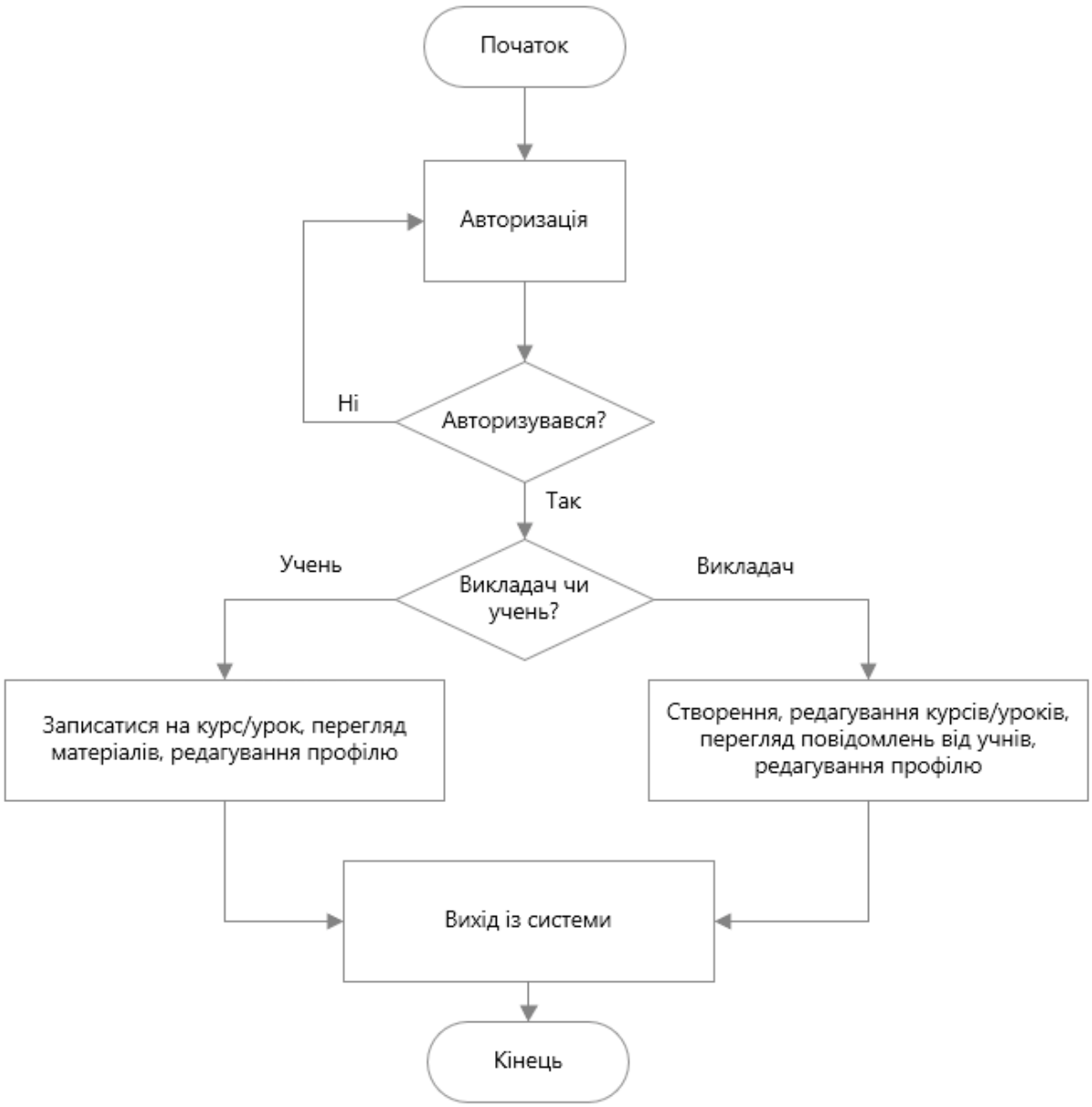
Освітній веб-застосунок для репетиторів та учнів

Схема алгоритму роботи системи

ІАЛЦ.467200.004 ДА

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.004 ДА		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Осипенко К. Ю.			*Освітній веб-застосунок для репетиторів та учнів* Схема алгоритму роботи системи	Літ.	Арк. 1 / Акрушів 1
Перевір.		Каплунов А. В.					
Н. Контр.		Нікольський С.С.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-13	
Затверд.							

ДОДАТОК Б

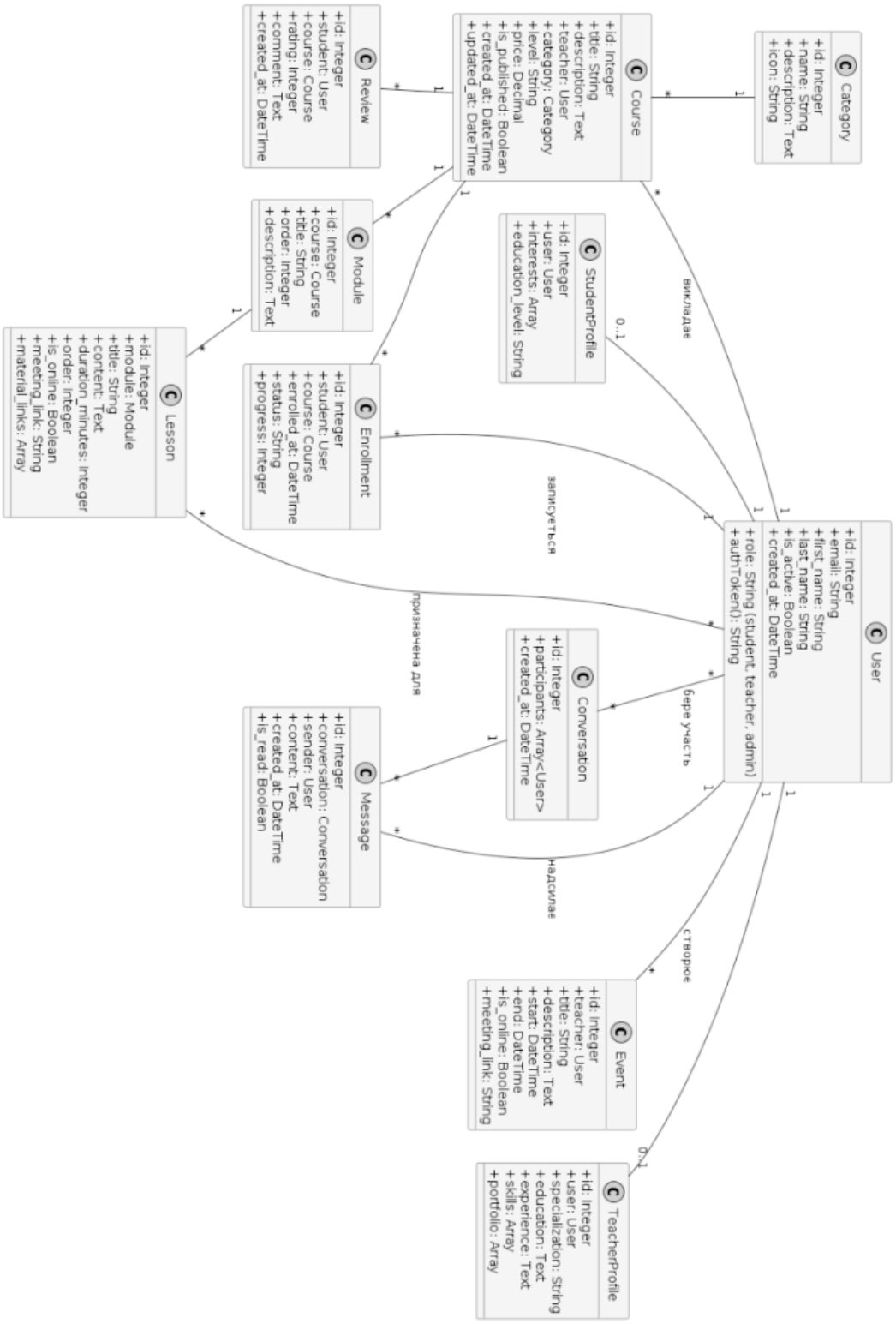
Освітній веб-застосунок для репетиторів та учнів

Структурна схема класів системи

ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.005 ДБ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Осипенко К. Ю.			*Освітній веб-застосунок для репетиторів та учнів* Структурна схема класів системи	Літ. / Арк. 1 / Акрушів 1
Перевір.		Каплунов А. В.				
Н. Контр.		Нікольський С.С.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-13
Затверд.						

ДОДАТОК В

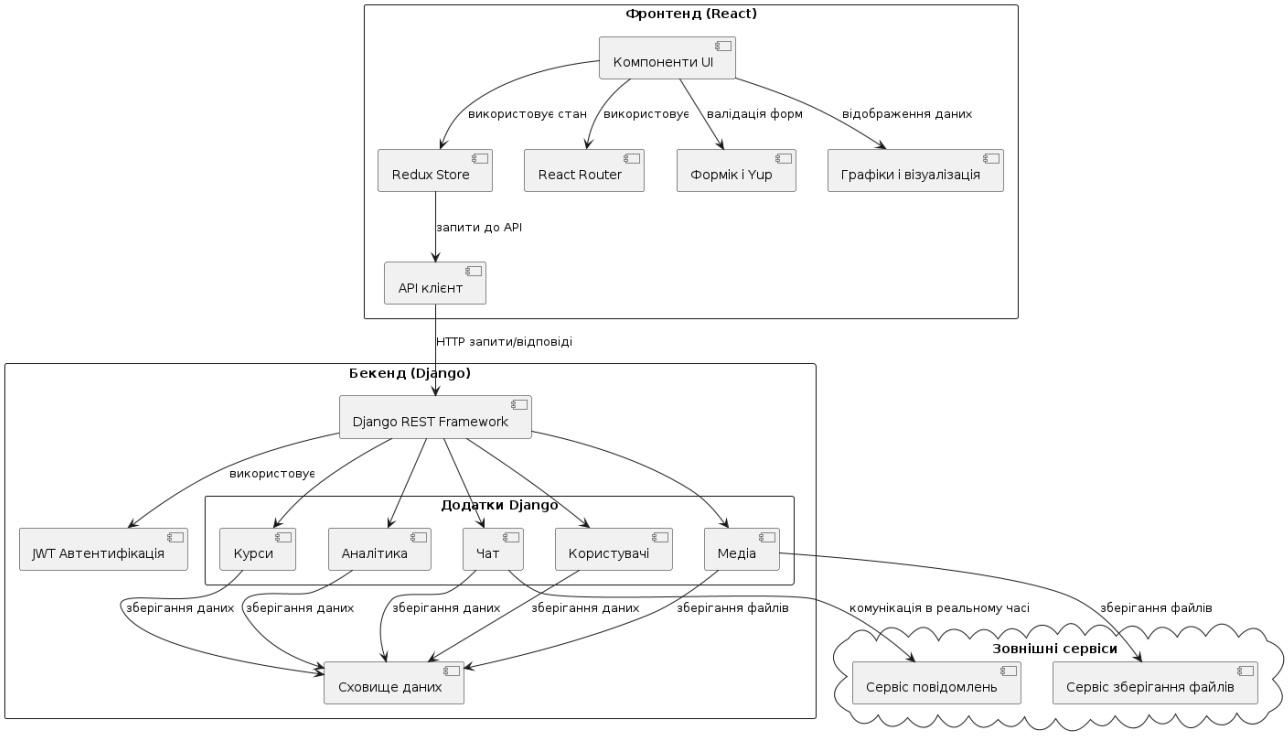
Освітній веб-застосунок для репетиторів та учнів

Архітектура системи

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ 2025 р.



					ІАЛЦ.467200.006 ДВ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Осипенко К. Ю.			*Освітній веб-застосунок для репетиторів та учнів* Архітектура системи	Літ. / Арк. 1 / Акрушів 1
Перевір.		Каплунов А. В.				
Н. Контр.		Нікольський С.С.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-13
Затверд.						

ДОДАТОК Г

Освітній веб-застосунок для репетиторів та учнів

Текст програмного коду

ІАЛЦ.467200.007 ДГ

Аркушів 22

Київ 2025 р.

Backend/analytics/models.py

```
from django.db import models
from django.utils.translation import gettext_lazy as _
from django.contrib.auth import get_user_model
from courses.models import Course

User = get_user_model()

class TeacherStat(models.Model):
    teacher = models.ForeignKey(User, on_delete=models.CASCADE, related_name='teacher_stats')
    date = models.DateField(_('дата'))
    new_students = models.PositiveIntegerField(_('нових студентів'), default=0)
    active_students = models.PositiveIntegerField(_('активних студентів'), default=0)
    course_views = models.PositiveIntegerField(_('переглядів курсів'), default=0)
    profile_views = models.PositiveIntegerField(_('переглядів профілю'), default=0)
    total_revenue = models.DecimalField(_('загальний дохід'), max_digits=10, decimal_places=2, default=0)

    class Meta:
        verbose_name = _('статистика викладача')
        verbose_name_plural = _('статистика викладачів')
        unique_together = ['teacher', 'date']

    def __str__(self):
        return f"Статистика {self.teacher.get_full_name()} за {self.date}"

class CourseStat(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='course_stats')
    date = models.DateField(_('дата'))
    views = models.PositiveIntegerField(_('переглядів'), default=0)
    enrollments = models.PositiveIntegerField(_('записів'), default=0)
    completions = models.PositiveIntegerField(_('завершень'), default=0)
    revenue = models.DecimalField(_('дохід'), max_digits=10, decimal_places=2, default=0)

    class Meta:
        verbose_name = _('статистика курсу')
        verbose_name_plural = _('статистика курсів')
        unique_together = ['course', 'date']

    def __str__(self):
        return f"Статистика курсу {self.course.title} за {self.date}"

class UserActivity(models.Model):
    ACTION_CHOICES = (
        ('view_course', _('Перегляд курсу')),
        ('view_profile', _('Перегляд профілю')),
        ('enroll', _('Запис на курс')),
        ('complete', _('Завершення курсу')),
        ('review', _('Залишення відгуку')),
        ('message', _('Надсилання повідомлення')),
        ('login', _('Вхід в систему')),
        ('logout', _('Вихід з системи')),
    )

    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='activities')
    action = models.CharField(_('дія'), max_length=20, choices=ACTION_CHOICES)
```

```
    object_id = models.PositiveIntegerField(_('ID об\'єкта'), null=True, blank=True)
    object_type = models.CharField(_('тип об\'єкта'), max_length=20, null=True, blank=True)
    timestamp = models.DateTimeField(_('час'), auto_now_add=True)
    ip_address = models.GenericIPAddressField(_('IP-адреса'), null=True, blank=True)
    user_agent = models.TextField(_('user-agent'), null=True, blank=True)

    class Meta:
        verbose_name = _('активність користувача')
        verbose_name_plural = _('активності користувачів')
        ordering = ['-timestamp']

    def __str__(self):
        return f"{self.user.get_full_name()} - {self.get_action_display()} - {self.timestamp}"
```

Backend/chat/models.py

```
from django.db import models
from django.utils.translation import gettext_lazy as _
from django.contrib.auth import get_user_model

User = get_user_model()

class Conversation(models.Model):
    participants = models.ManyToManyField(User, related_name='conversations')
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)
    updated_at = models.DateTimeField(_('оновлено'), auto_now=True)

    class Meta:
        verbose_name = _('розмова')
        verbose_name_plural = _('розмови')

    def __str__(self):
        return f"Розмова {self.id}"

class Message(models.Model):
    conversation = models.ForeignKey(Conversation, on_delete=models.CASCADE, related_name='messages')
    sender = models.ForeignKey(User, on_delete=models.CASCADE, related_name='sent_messages')
    content = models.TextField(_('зміст'))
    file = models.FileField(_('файл'), upload_to='chat_files/', null=True, blank=True)
    is_read = models.BooleanField(_('прочитано'), default=False)
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)

    class Meta:
        verbose_name = _('повідомлення')
        verbose_name_plural = _('повідомлення')
        ordering = ['created_at']

    def __str__(self):
        return f"Повідомлення від {self.sender.get_full_name()} в {self.conversation}"

class Notification(models.Model):
    TYPE_CHOICES = (
        ('message', _('Нове повідомлення')),
        ('course', _('Повідомлення про курс')),
        ('enrollment', _('Запис на курс')),
        ('review', _('Новий відгук')),
        ('system', _('Системне повідомлення')),
    )

    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='notifications')
    type = models.CharField(_('тип'), max_length=20, choices=TYPE_CHOICES)
    message = models.TextField(_('повідомлення'))
    related_object_id = models.PositiveIntegerField(_('ID пов\'язаного об\'єкта'), null=True, blank=True)
```

```
    related_object_type = models.CharField(_('тип пов\'язаного об\'єкта'), max_length=20, null=True, blank=True)
    is_read = models.BooleanField(_('прочитано'), default=False)
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)

    class Meta:
        verbose_name = _('сповіщення')
        verbose_name_plural = _('сповіщення')
        ordering = ['-created_at']

    def __str__(self):
        return f"Сповіщення для {self.user.get_full_name()}: {self.message[:30]}..."
```

Backend/courses/models.py

```
from django.db import models
from django.utils.translation import gettext_lazy as _
from django.contrib.auth import get_user_model

User = get_user_model()

class Category(models.Model):
    name = models.CharField(_('назва'), max_length=100)
    description = models.TextField(_('опис'), blank=True, null=True)

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = _('категорія')
        verbose_name_plural = _('категорії')

class Course(models.Model):
    DIFFICULTY_CHOICES = (
        ('beginner', _('Початковий')),
        ('intermediate', _('Середній')),
        ('advanced', _('Просунутий')),
    )

    STATUS_CHOICES = (
        ('draft', _('Чернетка')),
        ('published', _('Опубліковано')),
        ('archived', _('Архівовано')),
    )

    title = models.CharField(_('назва'), max_length=255)
    description = models.TextField(_('короткий опис'))
    content = models.TextField(_('детальний опис'))
    teacher = models.ForeignKey(User, on_delete=models.CASCADE, related_name='courses')
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, related_name='courses')
    difficulty = models.CharField(_('рівень складності'), max_length=20, choices=DIFFICULTY_CHOICES)
    status = models.CharField(_('статус'), max_length=20, choices=STATUS_CHOICES, default='draft')
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)
    updated_at = models.DateTimeField(_('оновлено'), auto_now=True)
    price = models.DecimalField(_('ціна'), max_digits=10, decimal_places=2, default=0)
    thumbnail = models.ImageField(_('зображення'), upload_to='course_thumbnails/', null=True, blank=True)
    start_date = models.DateField(_('дата початку'), null=True, blank=True)
    duration = models.PositiveIntegerField(_('тривалість (днів)'), null=True, blank=True)
    learning_outcomes = models.TextField(_('результати навчання'), null=True, blank=True)


    def __str__(self):
        return self.title
```

```
    @property
    def average_rating(self):
        reviews = self.reviews.all()
        if reviews:
            total_rating = sum(review.rating for review in reviews)
            return round(total_rating / len(reviews), 1)
        return 0

    @property
    def end_date(self):
        if self.start_date and self.duration:
            from datetime import timedelta
            return self.start_date + timedelta(days=self.duration)
        return None

    class Meta:
        verbose_name = _('курс')
        verbose_name_plural = _('курси')

class CourseModule(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='modules')
    title = models.CharField(_('назва'), max_length=255)
    description = models.TextField(_('опис'), blank=True, null=True)
    order = models.PositiveIntegerField(_('порядок'))

    class Meta:
        ordering = ['order']
        verbose_name = _('модуль курсу')
        verbose_name_plural = _('модулі курсу')

    def __str__(self):
        return f"{self.course.title} - {self.title}"

class CourseContent(models.Model):
    CONTENT_TYPE_CHOICES = (
        ('video', _('Відео')),
        ('text', _('Текст')),
        ('quiz', _('Тест')),
        ('assignment', _('Завдання')),
    )

    module = models.ForeignKey(CourseModule, on_delete=models.CASCADE, related_name='contents')
    title = models.CharField(_('назва'), max_length=255)
    content_type = models.CharField(_('тип контенту'), max_length=20, choices=CONTENT_TYPE_CHOICES)
    content = models.TextField(_('контент'))
    file = models.FileField(_('файл'), upload_to='course_contents/', null=True, blank=True)
    order = models.PositiveIntegerField(_('порядок'))

    class Meta:
        ordering = ['order']
        verbose_name = _('контент курсу')
        verbose_name_plural = _('контент курсу')

    def __str__(self):
        return f"{self.module.title} - {self.title}"

class Review(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='reviews')
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='reviews')
    rating = models.PositiveSmallIntegerField(_('оцінка'), choices=[(i, i) for i in range(1, 6)])
    comment = models.TextField(_('коментар'))
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)
```

```
    class Meta:
        verbose_name = _('відгук')
        verbose_name_plural = _('відгуки')
        unique_together = ['course', 'user']

    def __str__(self):
        return f"{self.user.get_full_name()} - {self.course.title} - {self.rating}"

class Enrollment(models.Model):
    STATUS_CHOICES = (
        ('pending', _('В очікуванні')),
        ('active', _('Активний')),
        ('completed', _('Завершено')),
        ('cancelled', _('Скасовано')),
    )

    course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='enrollments')
    student = models.ForeignKey(User, on_delete=models.CASCADE, related_name='enrollments')
    enrolled_at = models.DateTimeField(_('дата запису'), auto_now_add=True)
    status = models.CharField(_('статус'), max_length=20, choices=STATUS_CHOICES, default='pending')

    class Meta:
        verbose_name = _('запис на курс')
        verbose_name_plural = _('записи на курси')
        unique_together = ['course', 'student']

    def __str__(self):
        return f"{self.student.get_full_name()} - {self.course.title}"

class Event(models.Model):
    teacher = models.ForeignKey(User, on_delete=models.CASCADE, related_name='events')
    title = models.CharField(_('назва'), max_length=255)
    description = models.TextField(_('опис'), blank=True)
    start = models.DateTimeField(_('початок'))
    end = models.DateTimeField(_('кінець'))
    course = models.ForeignKey('Course', on_delete=models.SET_NULL, null=True, blank=True, related_name='events')
    is_online = models.BooleanField(_('онлайн'), default=False)
    meeting_link = models.URLField(_('посилання на зустріч'), blank=True)
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)
    updated_at = models.DateTimeField(_('оновлено'), auto_now=True)

    class Meta:
        verbose_name = _('подія')
        verbose_name_plural = _('події')
        ordering = ['start']

    def __str__(self):
        return self.title

class Lesson(models.Model):
    teacher = models.ForeignKey(User, on_delete=models.CASCADE, related_name='lessons')
    course = models.ForeignKey('Course', on_delete=models.CASCADE, related_name='lessons', null=True, blank=True)
    title = models.CharField(_('назва'), max_length=255)
    description = models.TextField(_('опис'), blank=True)
    start_datetime = models.DateTimeField(_('дата і час початку'))
    duration_minutes = models.IntegerField(_('тривалість (хвилин)'), default=60)
    is_online = models.BooleanField(_('онлайн'), default=False)
    meeting_link = models.URLField(_('посилання на зустріч'), blank=True)
    material_links = models.JSONField(_('посилання на матеріали'), default=list, blank=True)
    students = models.ManyToManyField(User, related_name='scheduled_lessons', blank=True)
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)
    updated_at = models.DateTimeField(_('оновлено'), auto_now=True)
```

```
    class Meta:
        verbose_name = _('урок')
        verbose_name_plural = _('уроки')
        ordering = ['start_datetime']

    def __str__(self):
        return self.title
```

Backend/users/models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser, BaseUserManager
from django.utils.translation import gettext_lazy as _

class UserManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError('Поле email повинно бути вказане')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email, password, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')
        return self._create_user(email, password, **extra_fields)

class User(AbstractUser):
    ROLE_CHOICES = (
        ('student', _('Студент')),
        ('teacher', _('Викладач')),
        ('admin', _('Адміністратор')),
    )

    username = None
    email = models.EmailField(_('email address'), unique=True)
    role = models.CharField(_('роль'), max_length=20, choices=ROLE_CHOICES, default='student')
    avatar = models.ImageField(_('фото профілю'), upload_to='avatars/', null=True, blank=True)
    phone = models.CharField(_('телефон'), max_length=20, blank=True, null=True)
    bio = models.TextField(_('біографія'), blank=True, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = UserManager()

    class Meta:
        verbose_name = _('користувач')
        verbose_name_plural = _('користувачі')
```

```
class TeacherProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='teacher_profile')
    specialization = models.CharField(_('спеціалізація'), max_length=255)
    experience = models.TextField(_('досвід'), blank=True, null=True)
    qualification = models.TextField(_('кваліфікація'), blank=True, null=True)
    languages = models.CharField(_('мови'), max_length=255, blank=True, null=True)
    education = models.JSONField(_('освіта'), default=list, blank=True)
    skills = models.JSONField(_('навички'), default=list, blank=True)
    social_links = models.JSONField(_('соціальні мережі'), default=dict, blank=True)
    rating = models.DecimalField(_('рейтинг'), max_digits=3, decimal_places=2, default=0.0)

    def __str__(self):
        return f"{self.user.get_full_name()} - {self.specialization}"

    class Meta:
        verbose_name = _('профіль викладача')
        verbose_name_plural = _('профілі викладачів')

class TeacherPortfolio(models.Model):
    teacher = models.ForeignKey(TeacherProfile, on_delete=models.CASCADE, related_name='portfolio_items')
    title = models.CharField(_('назва'), max_length=255)
    description = models.TextField(_('опис'), blank=True, null=True)
    file = models.FileField(_('файл'), upload_to='portfolio/')
    file_type = models.CharField(_('тип файлу'), max_length=20,
                                 choices=[('video', _('Відео')), ('pdf', _('PDF')), ('image', _('Зображення'))])
    created_at = models.DateTimeField(_('створено'), auto_now_add=True)

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = _('елемент портфоліо')
        verbose_name_plural = _('елементи портфоліо')
```

Backend/courses/serializers.py

```
from .models import Category, Course, CourseModule, CourseContent, Review, Enrollment, Lesson
from users.serializers import UserSerializer

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ['id', 'name', 'description']

class CourseContentSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseContent
        fields = ['id', 'title', 'content_type', 'content', 'file', 'order']

class CourseModuleSerializer(serializers.ModelSerializer):
    contents = CourseContentSerializer(many=True, read_only=True)

    class Meta:
        model = CourseModule
        fields = ['id', 'title', 'description', 'order', 'contents']

class ReviewSerializer(serializers.ModelSerializer):
    user = UserSerializer(read_only=True)

    class Meta:
        model = Review
        fields = ['id', 'user', 'rating', 'comment', 'created_at']
        read_only_fields = ['id', 'created_at']
```

```
    def create(self, validated_data):
        user = self.context['request'].user
        course = validated_data.get('course')

        if Review.objects.filter(user=user, course=course).exists():
            raise serializers.ValidationError("Ви вже залишили відгук для цього курсу")

        if not Enrollment.objects.filter(student=user, course=course, status='active').exists():
            raise serializers.ValidationError("Ви повинні бути записані на курс, щоб залишити відгук")

        return Review.objects.create(user=user, **validated_data)

class CourseSerializer(serializers.ModelSerializer):
    teacher = UserSerializer(read_only=True)
    category = CategorySerializer(read_only=True)
    modules = CourseModuleSerializer(many=True, read_only=True)
    reviews = ReviewSerializer(many=True, read_only=True)
    avg_rating = serializers.SerializerMethodField()
    reviews_count = serializers.SerializerMethodField()
    students_count = serializers.SerializerMethodField()

    class Meta:
        model = Course
        fields = [
            'id', 'title', 'description', 'content', 'teacher', 'category',
            'difficulty', 'status', 'created_at', 'updated_at', 'price',
            'thumbnail', 'start_date', 'duration', 'modules', 'reviews',
            'avg_rating', 'reviews_count', 'students_count'
        ]
        read_only_fields = ['id', 'created_at', 'updated_at']

    def get_avg_rating(self, obj):
        return obj.reviews.aggregate(Avg('rating'))['rating__avg'] or 0

    def get_reviews_count(self, obj):
        return obj.reviews.count()

    def get_students_count(self, obj):
        return obj.enrollments.filter(status='active').count()

    def create(self, validated_data):
        teacher = self.context['request'].user
        return Course.objects.create(teacher=teacher, **validated_data)

class CourseCreateUpdateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = [
            'title', 'description', 'content', 'category', 'difficulty',
            'status', 'price', 'thumbnail', 'start_date', 'duration'
        ]

class CoursePublicSerializer(serializers.ModelSerializer):
    teacher_name = serializers.SerializerMethodField()
    category_name = serializers.SerializerMethodField()
    avg_rating = serializers.SerializerMethodField()
    reviews_count = serializers.SerializerMethodField()
    students_count = serializers.SerializerMethodField()

    class Meta:
        model = Course
        fields = [
            'id', 'title', 'description', 'teacher_name', 'category_name',
```

```
            'difficulty', 'price', 'thumbnail', 'start_date', 'duration',
            'avg_rating', 'reviews_count', 'students_count'
        ]

    def get_teacher_name(self, obj):
        return obj.teacher.get_full_name()

    def get_category_name(self, obj):
        return obj.category.name if obj.category else None

    def get_avg_rating(self, obj):
        return obj.reviews.aggregate(Avg('rating'))['rating__avg'] or 0

    def get_reviews_count(self, obj):
        return obj.reviews.count()

    def get_students_count(self, obj):
        return obj.enrollments.filter(status='active').count()

class EnrollmentSerializer(serializers.ModelSerializer):
    course = CoursePublicSerializer(read_only=True)
    student = UserSerializer(read_only=True)

    class Meta:
        model = Enrollment
        fields = ['id', 'course', 'student', 'enrolled_at', 'status']
        read_only_fields = ['id', 'enrolled_at']

    def create(self, validated_data):
        student = self.context['request'].user
        course = validated_data.get('course')

        if Enrollment.objects.filter(student=student, course=course).exists():
            raise serializers.ValidationError("Ви вже записані на цей курс")

        return Enrollment.objects.create(student=student, **validated_data)

class LessonSerializer(serializers.ModelSerializer):
    teacher = UserSerializer(read_only=True)
    course = CourseSerializer(read_only=True)

    class Meta:
        model = Lesson
        fields = [
            'id', 'teacher', 'course', 'title', 'description',
            'start_datetime', 'duration_minutes', 'is_online',
            'meeting_link', 'material_links', 'students',
            'created_at', 'updated_at'
        ]
        read_only_fields = ['id', 'created_at', 'updated_at', 'teacher']
```

Backend/users/views.py

```
from rest_framework import viewsets, generics, permissions, status
from rest_framework.response import Response
from rest_framework.decorators import action, api_view, permission_classes, authentication_classes
from rest_framework_simplejwt.views import TokenObtainPairView
from rest_framework_simplejwt.tokens import RefreshToken
from rest_framework.permissions import IsAuthenticated, AllowAny, IsAdminUser
from .models import User, TeacherProfile, TeacherPortfolio
from courses.models import Course, Enrollment
from .serializers import (
    UserSerializer,
    TeacherProfileSerializer,
```

```
    TeacherPortfolioSerializer,
    TeacherPublicProfileSerializer
)
import json
import logging
from django.conf import settings
from django.http import Http404

logger = logging.getLogger(__name__)
@api_view(['POST'])
@permission_classes([permissions.AllowAny])
def register_user(request):
    print("Отримано дані реєстрації:", request.data)
    data = request.data.copy()

    if 'username' not in data or not data['username']:
        if 'email' in data:
            data['username'] = data['email'].split('@')[0]

    serializer = UserSerializer(data=data)
    if serializer.is_valid():
        print("Дані валідні, створюємо користувача")
        user = serializer.save()
        refresh = RefreshToken.for_user(user)
        return Response({
            'user': serializer.data,
            'refresh': str(refresh),
            'access': str(refresh.access_token),
        }, status=status.HTTP_201_CREATED)
    print("Помилки валідації:", serializer.errors)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes([permissions.AllowAny])
def login_user(request):
    print("Отримано дані входу:", request.data)
    email = request.data.get('email')
    password = request.data.get('password')

    if not email or not password:
        return Response(
            {'detail': 'Email та пароль обов\'язкові'},
            status=status.HTTP_400_BAD_REQUEST
        )

    try:
        user = User.objects.get(email=email)
        print("Користувача знайдено:", user.email)

        if not user.check_password(password):
            print("Невірний пароль для користувача:", email)
            return Response(
                {'detail': 'Невірний пароль'},
                status=status.HTTP_400_BAD_REQUEST
            )

        refresh = RefreshToken.for_user(user)
        serializer = UserSerializer(user)

        print("Успішний вхід користувача:", email)
        return Response({
            'user': serializer.data,
            'refresh': str(refresh),
```

```
            'access': str(refresh.access_token),
        })

    except User.DoesNotExist:
        print("Користувача не знайдено:", email)
        return Response(
            {'detail': 'Користувача з таким email не знайдено'},
            status=status.HTTP_404_NOT_FOUND
        )
    except Exception as e:
        print("Помилка при вході:", str(e))
        return Response(
            {'detail': 'Помилка при вході в систему'},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )

@api_view(['GET'])
@permission_classes([permissions.IsAuthenticated])
def get_teacher_profile(request):
    user = request.user

    try:
        if user.role != 'teacher':
            return Response({"detail": "Ви не є викладачем"}, status=status.HTTP_403_FORBIDDEN)

        try:
            profile = TeacherProfile.objects.get(user=user)
            serializer = TeacherProfileSerializer(profile)
            return Response(serializer.data)
        except TeacherProfile.DoesNotExist:
            profile = TeacherProfile.objects.create(
                user=user,
                specialization="Не вказано"  # Обов'язкове поле
            )
            serializer = TeacherProfileSerializer(profile)
            return Response(serializer.data)
    except Exception as e:
        return Response({"detail": f"Помилка при отриманні профілю: {str(e)}"},
                        status=status.HTTP_500_INTERNAL_SERVER_ERROR)

@api_view(['PUT', 'PATCH'])
@permission_classes([permissions.IsAuthenticated])
def update_teacher_profile(request):
    user = request.user

    if user.role != 'teacher':
        return Response({"detail": "Ви не є викладачем"}, status=status.HTTP_403_FORBIDDEN)

    try:
        print(f"Оновлення профілю для {user.username}, дані: {request.data}")

        profile, created = TeacherProfile.objects.get_or_create(
            user=user,
            defaults={'specialization': 'Не вказано'}
        )

        data = request.data.copy()

        user_data = {}
        if 'firstName' in data:
            user_data['first_name'] = data.pop('firstName')
        if 'lastName' in data:
            user_data['last_name'] = data.pop('lastName')
```

```
        if 'email' in data:
            user_data['email'] = data.pop('email')
        if 'phone' in data:
            user_data['phone'] = data.pop('phone')
        if 'bio' in data:
            user_data['bio'] = data.pop('bio')

        if user_data:
            print(f"Оновлення даних користувача: {user_data}")
            for key, value in user_data.items():
                setattr(user, key, value)
            user.save()

        profile_fields = {
            'specialization': 'specialization',
            'experience': 'experience',
            'qualification': 'qualification',
            'languages': 'languages',
            'education': 'education',
            'skills': 'skills',
            'socialLinks': 'social_links'
        }

        for frontend_field, backend_field in profile_fields.items():
            if frontend_field in data:
                value = data[frontend_field]
                if value is None or value == "null" or value == "undefined":
                    if frontend_field in ['education', 'skills']:
                        value = []
                    elif frontend_field == 'socialLinks':
                        value = {}
                    else:
                        value = ""
                elif isinstance(value, str) and frontend_field in ['education', 'skills', 'socialLinks']:
                    try:
                        value = json.loads(value)
                    except json.JSONDecodeError:
                        if frontend_field in ['education', 'skills']:
                            value = []
                        elif frontend_field == 'socialLinks':
                            value = {}
                setattr(profile, backend_field, value)
                print(f"Оновлення поля {backend_field}: {value}")

        profile.save()
        print(f"Профіль оновлено: {profile.__dict__}")

        serializer = TeacherProfileSerializer(profile)
        return Response(serializer.data, status=status.HTTP_200_OK)

    except Exception as e:
        print(f"Помилка при оновленні профілю: {str(e)}")
        return Response(
            {"detail": f"Помилка при оновленні профілю: {str(e)}"},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )

class TeacherProfileViewSet(viewsets.ModelViewSet):
    queryset = TeacherProfile.objects.all()
    serializer_class = TeacherProfileSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
```

```
        user = self.request.user
        if user.is_superuser or user.role == 'admin':
            return TeacherProfile.objects.all()
        if user.role == 'teacher':
            return TeacherProfile.objects.filter(user=user)
        return TeacherProfile.objects.none()

    @action(detail=False, methods=['get'])
    def my_profile(self, request):
        if request.user.role != 'teacher':
            return Response({"detail": "Ви не є викладачем"}, status=status.HTTP_403_FORBIDDEN)

        try:
            profile = TeacherProfile.objects.get(user=request.user)
            serializer = self.get_serializer(profile)
            return Response(serializer.data)
        except TeacherProfile.DoesNotExist:
            return Response({"detail": "Профіль не знайдено"}, status=status.HTTP_404_NOT_FOUND)

class TeacherPortfolioViewSet(viewsets.ModelViewSet):
    queryset = TeacherPortfolio.objects.all()
    serializer_class = TeacherPortfolioSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        user = self.request.user
        if user.is_superuser or user.role == 'admin':
            return TeacherPortfolio.objects.all()
        if user.role == 'teacher':
            try:
                profile = TeacherProfile.objects.get(user=user)
                return TeacherPortfolio.objects.filter(teacher=profile)
            except TeacherProfile.DoesNotExist:
                return TeacherPortfolio.objects.none()
        return TeacherPortfolio.objects.none()

    def perform_create(self, serializer):
        try:
            profile = TeacherProfile.objects.get(user=self.request.user)
            serializer.save(teacher=profile)
        except TeacherProfile.DoesNotExist:
            raise serializers.ValidationError("Профіль викладача не знайдено")

class TeacherDetailView(generics.RetrieveAPIView):
    queryset = TeacherProfile.objects.all()
    serializer_class = TeacherPublicProfileSerializer
    permission_classes = [AllowAny]

    def get_object(self):
        try:
            return TeacherProfile.objects.get(user__id=self.kwargs['pk'])
        except TeacherProfile.DoesNotExist:
            raise Http404("Профіль викладача не знайдено")
```

Backend/courses/views.py

```
class CourseViewSet(viewsets.ModelViewSet):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
    permission_classes = [IsTeacherOrReadOnly]
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ['title', 'description', 'teacher__first_name', 'teacher__last_name']
    ordering_fields = ['created_at', 'title', 'price']
    http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options']
```

```
    def get_queryset(self):
        user = self.request.user

        if self.action in ['list', 'retrieve']:
            return Course.objects.filter(status='published')

        if user.is_authenticated:
            if user.is_superuser or user.role == 'admin':
                return Course.objects.all()
            elif user.role == 'teacher':
                return Course.objects.filter(teacher=user)

        return Course.objects.filter(status='published')

    def get_serializer_class(self):
        if self.action in ['create', 'update', 'partial_update']:
            return CourseCreateUpdateSerializer
        if self.action in ['list', 'retrieve'] and not self.request.user.is_authenticated:
            return CoursePublicSerializer
        return CourseSerializer

    def perform_create(self, serializer):
        serializer.save(teacher=self.request.user)

    @action(detail=True, methods=['post'], permission_classes=[permissions.IsAuthenticated])
    def enroll(self, request, pk=None):
        course = self.get_object()

        if Enrollment.objects.filter(student=request.user, course=course).exists():
            return Response(
                {"detail": "Ви вже записані на цей курс"},
                status=status.HTTP_400_BAD_REQUEST
            )

        enrollment = Enrollment.objects.create(
            student=request.user,
            course=course,
            status='pending'
        )

        serializer = EnrollmentSerializer(enrollment)
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    @action(detail=False, methods=['get'], permission_classes=[permissions.IsAuthenticated])
    def my_courses(self, request):
        if request.user.role != 'teacher':
            return Response(
                {"detail": "Ви не є викладачем"},
                status=status.HTTP_403_FORBIDDEN
            )

        courses = Course.objects.filter(teacher=request.user)
        serializer = self.get_serializer(courses, many=True)
        return Response(serializer.data)

    @action(detail=False, methods=['get'], permission_classes=[permissions.IsAuthenticated])
    def enrolled(self, request):
        enrollments = Enrollment.objects.filter(student=request.user, status='active')
        courses = [enrollment.course for enrollment in enrollments]
        serializer = CoursePublicSerializer(courses, many=True)
        return Response(serializer.data)
```

```
class CourseContentViewSet(viewsets.ModelViewSet):
    queryset = CourseContent.objects.all()
    serializer_class = CourseContentSerializer
    permission_classes = [IsTeacherOrReadOnly]

    def get_queryset(self):
        module_id = self.request.query_params.get('module', None)
        if module_id:
            return CourseContent.objects.filter(module_id=module_id).order_by('order')
        return CourseContent.objects.all()

    def perform_create(self, serializer):
        module_id = self.request.data.get('module')
        module = CourseModule.objects.get(id=module_id)

        if module.course.teacher != self.request.user and not self.request.user.is_superuser:
            self.permission_denied(
                self.request,
                message="Ви не маєте дозволу на додавання контенту до цього модуля"
            )

        serializer.save(module=module)

class EnrollmentViewSet(viewsets.ModelViewSet):
    queryset = Enrollment.objects.all()
    serializer_class = EnrollmentSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        user = self.request.user

        if user.is_superuser or user.role == 'admin':
            return Enrollment.objects.all()
        elif user.role == 'teacher':
            return Enrollment.objects.filter(course__teacher=user)
        else:
            return Enrollment.objects.filter(student=user)

    def perform_create(self, serializer):
        course_id = self.request.data.get('course')
        course = Course.objects.get(id=course_id)

        if Enrollment.objects.filter(student=self.request.user, course=course).exists():
            self.permission_denied(
                self.request,
                message="Ви вже записані на цей курс"
            )

        serializer.save(student=self.request.user, course=course)

class TeacherCoursesView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        if request.user.role != 'teacher':
            return Response(
                {"detail": "Ви не є викладачем"},
                status=status.HTTP_403_FORBIDDEN
            )

        courses = Course.objects.filter(teacher=request.user)
        courses_data = []
        for course in courses:
```

```
            students_count = Enrollment.objects.filter(course=course, status='active').count()
            lessons_count = CourseContent.objects.filter(module__course=course).count()

            course_data = CourseSerializer(course).data
            course_data['studentsCount'] = students_count
            course_data['lessons'] = lessons_count

            courses_data.append(course_data)

        return Response(courses_data)

    def post(self, request):
        if request.user.role != 'teacher':
            return Response(
                {"detail": "Ви не є викладачем"},
                status=status.HTTP_403_FORBIDDEN
            )

        print(f"Дані запиту на створення курсу: {request.data}")

        data = request.data.copy()
        try:
            if 'price' in data:
                try:
                    data['price'] = float(data['price'])
                except (ValueError, TypeError):
                    data['price'] = 0.0

            if 'duration' in data and data['duration']:
                try:
                    data['duration'] = int(data['duration'])
                except (ValueError, TypeError):
                    data['duration'] = 30  # Значення за замовчуванням

            if 'category' in data:
                category_data = data['category']
                print(f"Отримана категорія: {category_data}, тип: {type(category_data)}")

                if isinstance(category_data, dict) and 'id' in category_data:
                    data['category'] = category_data['id']
                elif not isinstance(category_data, (int, float)) and not str(category_data).isdigit():
                    try:
                        category = Category.objects.get(name=category_data)
                        data['category'] = category.id
                    except Category.DoesNotExist:
                        return Response(
                            {"detail": f"Категорія {category_data} не знайдена"},
                            status=status.HTTP_400_BAD_REQUEST
                        )

            required_fields = ['title', 'description', 'content', 'category', 'difficulty']
            missing_fields = [field for field in required_fields if field not in data or not data[field]]

            if missing_fields:
                return Response(
                    {"detail": f"Відсутні обов'язкові поля: {', '.join(missing_fields)}"},
                    status=status.HTTP_400_BAD_REQUEST
                )

        except (ValueError, TypeError) as e:
            error_message = f"Помилка перетворення типу даних: {str(e)}"
            print(error_message)
            return Response({"detail": error_message}, status=status.HTTP_400_BAD_REQUEST)
```

```
        serializer = CourseCreateUpdateSerializer(data=data)
        if serializer.is_valid():
            course = serializer.save(teacher=request.user)

            try:
                if course.start_date:
                    create_course_lessons(course, request.user)
            except Exception as e:
                print(f"Помилка при автоматичному створенні уроків: {str(e)}")

            return Response(
                CourseSerializer(course).data,
                status=status.HTTP_201_CREATED
            )
        print(f"Помилки валідації: {serializer.errors}")
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

class LessonViewSet(viewsets.ModelViewSet):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
    permission_classes = [permissions.IsAuthenticated]
    http_method_names = ['get', 'post', 'put', 'patch', 'delete']

    def get_queryset(self):
        user = self.request.user

        if user.is_superuser or user.role == 'admin':
            return Lesson.objects.all()
        elif user.role == 'teacher':
            return Lesson.objects.filter(teacher=user)
        else:
            return Lesson.objects.filter(students=user)

    def perform_create(self, serializer):
        serializer.save(teacher=self.request.user)

class MaterialViewSet(viewsets.ModelViewSet):
    queryset = None
    serializer_class = None
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        user = self.request.user

        if user.is_superuser or user.role == 'admin':
            return CourseContent.objects.all()
        elif user.role == 'teacher':
            return CourseContent.objects.filter(module__course__teacher=user)
        else:
            return CourseContent.objects.filter(
                module__course__enrollments__student=user,
                module__course__enrollments__status='active'
            )
```

Backend/osv2/urls.py

```
schema_view = get_schema_view(
  openapi.Info(
    title="Викладацька платформа API",
    default_version='v1',
    description="API для системи викладача з публічним профілем та дашбордом",
    contact=openapi.Contact(email="admin@example.com"),
  ),
  public=True,
```

```
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/users/profile/', get_teacher_profile, name='direct-teacher-profile'),
    path('api/users/profile/update/', update_teacher_profile, name='direct-update-teacher-profile'),

    path('api/student/events/', student_events, name='student-events'),
    path('api/student/stats/', student_stats, name='student-stats'),
    path('api/student/courses/', get_student_courses, name='student-courses'),
    path('api/student/courses/<int:pk>/', student_course_detail, name='api-student-course-detail'),
    path('api/student/lessons/upcoming/', student_upcoming_lessons, name='student-upcoming-lessons'),
    path('api/student/schedule/weekly/', student_weekly_schedule, name='student-weekly-schedule'),
    path('api/student/test/', student_test_endpoint, name='student-test'),
    path('api/users/', include('users.urls')),
    path('api/courses/', include('courses.urls')),
    path('api/chat/', include('chat.urls')),
    path('api/analytics/', include('analytics.urls')),
    path('api/teachers/dashboard/stats/', include('analytics.urls')),
    path('api/teachers/dashboard/upcoming-lessons/', include('courses.urls')),
    path('api/teachers/dashboard/activities/', include('analytics.urls')),
    path('api/teachers/courses/', include('courses.urls')),
    path('api/teachers/messages/', include('chat.urls')),
    path('api/teachers/profile/', include('users.urls')),
    path('api/teacher/', include(courses_teacher_urlpatterns)),
    path('api/teacher/', include(chat_teacher_urlpatterns)),
    path('api/teacher/', include(analytics_teacher_urlpatterns)),
    path('api/student/', include(courses_student_urlpatterns)),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
    path('api/reset-test-data/', reset_test_data, name='reset-test-data'),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Frontend/src/App.js

```
import React, { useEffect, useState } from 'react';
import { Routes, Route, Navigate } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { checkAuthStatus } from './redux/slices/authSlice';
import { ThemeProvider, CssBaseline, Box, Typography, CircularProgress, Button } from '@mui/material';
import theme from './theme';

import Login from './pages/auth/Login';
import Register from './pages/auth/Register';
import Dashboard from './pages/Dashboard';
import TeacherDashboard from './pages/dashboard/TeacherDashboard';
import StudentDashboard from './pages/dashboard/StudentDashboard';
import AdminDashboard from './pages/admin/Dashboard';
import PublicLayout from './components/layout/PublicLayout';
import DashboardLayout from './components/layout/DashboardLayout';
import ProtectedRoute from './components/routes/ProtectedRoute';
import Loading from './components/ui/Loading';
import NotFound from './pages/NotFound';

import TeacherMessages from './pages/dashboard/TeacherMessages';
import StudentMessages from './pages/dashboard/StudentMessages';

import CourseCreate from './pages/dashboard/CourseCreate';
import CourseEdit from './pages/dashboard/CourseEdit';
```

```
import TeacherCourses from './pages/dashboard/TeacherCourses';
import TeacherCourseView from './pages/dashboard/TeacherCourseView';

import TeacherAnalytics from './pages/dashboard/TeacherAnalytics';

import TeacherProfile from './pages/dashboard/TeacherProfile';
import TeacherSchedule from './pages/dashboard/TeacherSchedule';
import TeacherCalendar from './pages/dashboard/TeacherCalendar';

import CourseProgress from './components/dashboard/CourseProgress';
import UpcomingLessons from './components/dashboard/UpcomingLessons';
import RecentActivity from './components/dashboard/RecentActivity';
import DashboardStats from './components/dashboard/DashboardStats';

import CourseDetails from './pages/public/CourseDetails';
import TestPage from './pages/public/TestPage';
import CoursesList from './pages/public/CoursesList';

const RoleBasedRedirect = () => {
  const { isAuthenticated, user } = useSelector((state) => state.auth);

  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  if (user?.role === 'admin') {
    return <Navigate to="/admin/dashboard" replace />;
  }

  if (user?.role === 'teacher') {
    return <Navigate to="/teacher/dashboard" replace />;
  }

  if (user?.role === 'student') {
    return <Navigate to="/student/dashboard" replace />;
  }

  return <Navigate to="/dashboard" replace />;
};

const RoleRoute = ({ element, allowedRole }) => {
  const { user } = useSelector((state) => state.auth);

  if (user?.role !== allowedRole) {
    return <Navigate to="/" replace />;
  }

  return element;
};

function App() {
  const dispatch = useDispatch();
  const { isLoading, isAuthenticated } = useSelector((state) => state.auth);
  const [isLoadingTimeout, setIsLoadingTimeout] = useState(true);

  useEffect(() => {
    dispatch(checkAuthStatus());

    const timeoutId = setTimeout(() => {
      setIsLoadingTimeout(false);
    }, 3000);

    return () => clearTimeout(timeoutId);
```

```
}, [dispatch]);

if (isLoading && isLoadingTimeout) {
  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Loading />
    </ThemeProvider>
  );
}

if (isLoading && !isLoadingTimeout) {
  return (
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Box
        sx={{
          display: 'flex',
          flexDirection: 'column',
          justifyContent: 'center',
          alignItems: 'center',
          height: '100vh',
          gap: 2
        }}
      >
        <Typography variant="h6">
          Завантаження триває довше, ніж зазвичай
        </Typography>
        <CircularProgress />
        <Button
          variant="contained"
          color="primary"
          onClick={() => {
            // Примусово перейти на сторінку логіну
            localStorage.removeItem('token');
            localStorage.removeItem('refreshToken');
            window.location.href = '/login';
          }}
        >
          Перейти до сторінки входу
        </Button>
      </Box>
    </ThemeProvider>
  );
}

return (
  <ThemeProvider theme={theme}>
    <CssBaseline />

    <Routes>
      <Route path="/" element={<RoleBasedRedirect />} />
      <Route path="/test" element={<TestPage />} />
      <Route path="/courses" element={<PublicLayout><CoursesList /></PublicLayout>} />
      <Route path="/courses/:id" element={<PublicLayout><CourseDetails /></PublicLayout>} />
      <Route path="/course/:id" element={<PublicLayout><CourseDetails /></PublicLayout>} />
      <Route path="/" element={<PublicLayout />}>
        <Route
          path="login"
          element={isAuthenticated ? <Navigate to="/" replace /> : <Login />}
        />
        <Route
          path="register"
```

```
          element={isAuthenticated ? <Navigate to="/" replace /> : <Register />}
        />
      </Route>
      <Route path="/" element={<ProtectedRoute><DashboardLayout /></ProtectedRoute>}>
        <Route path="dashboard" element={<Dashboard />} />
        <Route path="admin">
          <Route path="dashboard" element={<RoleRoute element={<AdminDashboard />} allowedRole="admin" />} />
          <Route path="users" element={<RoleRoute element={<div>Управління користувачами</div>}
allowedRole="admin" />} />
          <Route path="courses" element={<RoleRoute element={<div>Управління курсами</div>}
allowedRole="admin" />} />
          <Route path="settings" element={<RoleRoute element={<div>Налаштування системи</div>}
allowedRole="admin" />} />
          <Route path="messages" element={<RoleRoute element={<div>Повідомлення адміністратора</div>}
allowedRole="admin" />} />
        </Route>
        <Route path="teacher">
          <Route path="dashboard" element={<RoleRoute element={<TeacherDashboard />} allowedRole="teacher" />}
/>
          <Route path="courses" element={<RoleRoute element={<TeacherCourses />} allowedRole="teacher" />} />
          <Route path="courses/create" element={<RoleRoute element={<CourseCreate />} allowedRole="teacher" />} />
          <Route path="courses/edit/:courseId" element={<RoleRoute element={<CourseEdit />} allowedRole="teacher"
/>} />
          <Route path="courses/:courseId" element={<RoleRoute element={<TeacherCourseView />}
allowedRole="teacher" />} />
          <Route path="students" element={<RoleRoute element={<div>Студенти викладача</div>}
allowedRole="teacher" />} />
          <Route path="schedule" element={<RoleRoute element={<TeacherSchedule />} allowedRole="teacher" />} />
          <Route path="calendar" element={<RoleRoute element={<TeacherCalendar />} allowedRole="teacher" />} />
          <Route path="profile" element={<RoleRoute element={<TeacherProfile />} allowedRole="teacher" />} />
          <Route path="messages" element={<RoleRoute element={<TeacherMessages />} allowedRole="teacher" />} />
          <Route path="analytics" element={<RoleRoute element={<TeacherAnalytics />} allowedRole="teacher" />} />
          <Route path="lessons/:lessonId" element={<RoleRoute element={<div>Урок в процесі</div>}
allowedRole="teacher" />} />
        </Route>
        <Route path="student">
          <Route path="dashboard" element={<RoleRoute element={<StudentDashboard />} allowedRole="student" />}
/>
          <Route path="courses" element={<RoleRoute element={<div>Курси студента</div>} allowedRole="student"
/>} />
          <Route path="courses/:courseId" element={<RoleRoute element={<CourseDetails />} allowedRole="student"
/>} />
          <Route path="schedule" element={<RoleRoute element={<div>Розклад студента</div>}
allowedRole="student" />} />
          <Route path="profile" element={<RoleRoute element={<div>Профіль студента</div>} allowedRole="student"
/>} />
          <Route path="messages" element={<RoleRoute element={<StudentMessages />} allowedRole="student" />} />
        </Route>
      </Route>

      {/* Маршрут 404 */}
      <Route path="*" element={<NotFound />} />
    </Routes>
  </ThemeProvider>
 );
}

export default App;
```

Frontend/src/setupProxy.js

```
const { createProxyMiddleware } = require('http-proxy-middleware');
const path = require('path');
```

```
module.exports = function(app) {
  app.use(
    '/api',
    createProxyMiddleware({
      target: 'http://localhost:8000',
      changeOrigin: true,
      pathRewrite: {
        '^/api': '/api',
      },
      onError: (err, req, res) => {
        console.error('Proxy error:', err);
        res.writeHead(500, {
          'Content-Type': 'text/plain',
        });
        res.end('Помилка при з\'єднанні з сервером API');
      },
    })
  );

  app.use(
    '/media',
    createProxyMiddleware({
      target: 'http://localhost:8000',
      changeOrigin: true,
      onError: (err, req, res, next) => {
        console.error('Media proxy error:', err);
        res.writeHead(404, {
          'Content-Type': 'text/plain',
        });
        res.end('Медіа файл не знайдено');
      },
    })
  );
};
```