НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: **Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну**

Виконала:
студентка IV курсу, групи ТМ-01

КЛЕБАН Дар’я Андріївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
доцент, к.т.н., Любов ПОЛЯГУШКО _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент:

_____ _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *доцент Андрій ДОНЕЦЬ* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2024

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студентці

КЛЕБАН Дар’ї Андріївні
(прізвище, ім’я, по батькові)

1. Тема роботи **Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну**

керівник роботи ***Полягушко Любов Григорівна, к.т.н., доцент***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування JavaScript, бібліотека React, платформа Node.js, фреймворк Express.js, СУБД MySQL, SCSS, середовище розробки VS Code

4. Перелік питань, які потрібно розробити
1) провести аналіз серед наявних аналогів
2) аргументувати вибрані інструменти, технології та алгоритми для реалізації програмного забезпечення
3) спроєктувати базу даних
4) спроєктувати архітектуру програмного забезпечення

5) створити прикладний програмний веб-інтерфейс
6) представити процес застосування програмного забезпечення

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють роботу алгоритмів платформи, архітектуру системи; концептуальна модель бази даних; інтерфейс користувача.

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студент _____ **Дар'я КЛЕБАН**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Любов ПОЛЯГУШКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 93 сторінках, містить 71 ілюстрацію, 2 таблиці, 2 додатки, 25 джерел в переліку посилань.

Мета роботи – створення платформи для вивчення веб-технологій та веб-дизайну.

Методи та засоби: мова програмування JavaScript, бібліотека React, платформа Node.js, фреймворк Express.js, СУБД MySQL, SCSS, React Router, Axios, JSONWebToken, BcryptJS, середовище розробки VS Code. Створення динамічних та інтерактивних компонентів інтерфейсу користувача, побудова односторінкових додатків, реалізація REST API, управління маршрутизацією, забезпечення безпеки через аутентифікацію та авторизацію користувачів, хешування паролів, зберігання даних у реляційній базі даних.

Результат – інтерактивна платформа для вивчення веб-технологій та веб-дизайну для організації та керування навчальними процесами онлайн.

Ключові слова: онлайн платформа, онлайн навчання, React, Node.js, JavaScript, MySQL.

ABSTRACT

The thesis is made on 93 pages, contains 71 illustrations, 2 tables, 2 appendices, 25 sources in the list of references.

Purpose: to create a platform for studying web technologies and web design.

Methods and tools: JavaScript programming language, React library, Node.js platform, Express.js framework, MySQL DBMS, SCSS, React Router, Axios, JSONWebToken, BcryptJS, VS Code development environment. Creating dynamic and interactive user interface components, building single-page applications, implementing REST API, managing routing, ensuring security through user authentication and authorization, hashing passwords, storing data in a relational database.

Result: an interactive platform for learning web technologies and web design for organizing and managing online learning processes.

Keywords: online platform, online training, React, Node.js, JavaScript, MySQL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

AJAX (Asynchronous JavaScript and XML) – підхід до побудови користувацьких інтерфейсів вебзастосунків, за яких вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані.

API (Application Programming Interface) – це набір інструкцій, які дозволяють різним програмам та сервісам взаємодіяти між собою та обмінюватись даними.

CSS (Cascading Style Sheets)– каскадні таблиці стилів, мова для опису стилізації вигляду веб-сторінок.

JSON – текстовий формат обміну даними, заснований на JavaScript.

JWT (JSON Web Token) – відкритий стандарт для створення токенів доступу, заснований у форматі JSON.

LDS (Learning Destination Sites) – сайти навчальних ресурсів

LME (Learning Management Ecosystems) – Екосистеми керування навчанням

LMS (Learning Management System) – Системи управління навчанням

REST (Representational State Transfer) - Архітектурний стиль для створення веб-сервісів, який використовує HTTP протокол для комунікації та зазвичай підтримує операції створення, читання, оновлення, видалення з ресурсами.

SCSS – це скриптова метамова, яка інтерпретується в CSS.

Клієнт – веб-браузер.

Рендеринг – це процес відображення вмісту на екрані користувача. У контексті веб-розробки, рендеринг стосується перетворення коду у видимий і взаємодіючий інтерфейс на веб-сторінці або у веб-додатку.

СУБД – Система управління базами даних.

Футер (Footer) – частина веб-сторінки, розташована внизу, яка зазвичай містить інформацію про авторські права, посилання на контакти, додаткові ресурси тощо.

Хедер (Header) – частина веб-сторінки, розташована вгорі, яка містить заголовок, логотип, меню навігації та інші важливі елементи для користувача.

Хук – спеціалізована функція в React.

ВСТУП

Зі зростанням швидкості технологічного прогресу і зміною потреб суспільства, світ постійно змінюється. Технології відіграють ключову роль у цьому процесі, впливаючи на всі сфери нашого життя, сфера освіти не є виключенням. Сучасне покоління виростає в умовах цифрової ери, де доступ до інформації є легким і миттєвим. Тому, для ефективного навчання і підготовки молоді до викликів сучасного світу, потрібні нові методи та підходи до освіти. Традиційні методи навчання часто виявляються недостатньо ефективними, оскільки не враховують потреб і можливостей сучасних учнів. Тому використання інноваційних освітніх технологій та онлайн платформ стає необхідним кроком у вдосконаленні навчального процесу і підготовці молоді до успішної кар'єри у цифровій епосі.

Після виникнення пандемії COVID-19 різко зросла потреба у використанні онлайн навчання. Обмеження та закриття навчальних закладів змусили багатьох шукати альтернативні шляхи отримання освіти, а онлайн платформи стали відповіддю на цей виклик. Це стало дуже вигідним з ряду причин. По-перше, доступність: будь-яка людина, незалежно від місцезнаходження, може мати доступ до якісного освітнього матеріалу в будь-який час. По-друге, гнучкість: онлайн навчання дозволяє студентам навчатися в зручний для них час та темп. Також воно сприяє розвитку навичок самостійності та володіння інформаційними технологіями, що стає все більш важливим у сучасному світі. Згідно дослідження [1] було виявлено, що під час пандемії COVID-19 кількість студентів онлайн зросла з 30% до 70%. Також враховуючи те, що в Україні триває воєнний стан, онлайн навчання набуло ще більшого значення. Багато навчальних закладів перейшли на дистанційну форму навчання, що дозволяє забезпечити доступ до освіти для тисяч учнів та студентів незважаючи на складні обставини. Онлайн платформи відіграють важливу роль у забезпеченні цього процесу.

Платформа онлайн-навчання – це веб-простір або портал для освітнього контенту та ресурсів. Вона об'єднує в собі різноманітні компоненти, такі як лекції,

навчальні матеріали, можливості для взаємодії між учасниками, та дозволяє вчителям та учням вести моніторинг та контроль за прогресом у навчанні. Дані платформи надають можливість студентам працювати в своєму власному темпі. Це означає, що вони можуть вчитися у зручний для них час, враховуючи їхні індивідуальні розклади та потреби. Зазвичай такі платформи представлені у вигляді веб-сайтів або додатків, які дозволяють зручно та ефективно навчатися, опрацьовуючи різноманітні навчальні матеріали необмежену кількість разів у поєднані з виконанням інтерактивних завдань. Вони також надають можливість викладачам і експертам ділитися знаннями та досвідом зі світом, створюючи широку мережу освітніх ресурсів. Крім того, платформи онлайн-навчання часто надають інструменти для відстеження прогресу студентів. Це дозволяє як учням, так і вчителям відстежувати, як добре студенти розуміють матеріал і які аспекти потребують додаткової уваги. Такий контроль прогресу допомагає студентам у плануванні свого навчання і вчителям у виявленні потреб у додатковій підтримці або поясненнях.

Існують різні типи онлайн-платформ для навчання, основні з них:

1. Сайти навчальних ресурсів (Learning Destination Sites, LDS) – це сайти, на яких багато різних розробників курсів пропонують свої навчальні матеріали. Прикладами таких платформ є Coursera, EdX та Udemy. У студента є можливість вибрати теми та викладачів, та записатись на індивідуальний курс відповідно до уподобань. Програми, які пропонуються через LDS, зазвичай є самостійними.

2. Системи управління навчанням (Learning Management System, LMS) – це веб-сайти або додатоки, які дозволяють організувати, керувати та виконувати навчальні процеси онлайн. Зазвичай дані система використовується всередині конкретної організації чи установи. Кожна школа чи університет може мати різну платформу LMS залежно від типу курсів, які вони пропонують. В даному випадку, на відміну від LDS платформ, створення курсу перебуває під відповідальністю викладачів або інших осіб, які мають академічні повноваження та викладають відповідні дисципліни в даній освітній установі. Прикладами таких платформ є Google Classroom, Moodle.

3. Екосистеми керування навчанням (Learning Management Ecosystems, LME) – це розширений навчальний інструмент, який об'єднує кілька програмних рішень, таких як механізми адаптивного навчання, управління навчальним вмістом, програмне забезпечення для розробки курсів, інструменти для оцінювання, тощо. Такі екосистеми зазвичай використовуються великими організаціями, школами або університетами, які пропонують багато онлайнових програм для отримання ступеня та навчальних ресурсів [2].

Детально будуть розглядатися саме системи управління навчанням (Learning Management System, LMS) тому що вони забезпечують комплексне управління навчальним процесом.

На сьогоднішній день існують багато платформ LMS, які надають різноманітні можливості для навчання онлайн. Багато з них вже дозволяють створювати, редагувати та переглядати навчальний контент, відстежувати прогрес і навіть комунікувати в межах платформи. Проте, не дивлячись на ці функції, існують аспекти, які можуть бути покращені або відсутні на деяких існуючих платформах.

Окремо варто звернути увагу на вивчення веб-технологій та веб-дизайну. Освоєння цих сфер стає дедалі важливішим, оскільки ці знання підвищують конкурентоспроможність на ринку праці та відкривають нові можливості для професійного зростання. Використання інноваційних освітніх платформ, які забезпечують доступ до сучасних інструментів і технологій, є необхідним для підготовки молоді до викликів сучасного світу та створення функціональних і корисних веб-проєктів.

Метою розробки веб-застосунку «Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну» є створення інноваційної та інтерактивної платформи, яка забезпечить доступ до якісних освітніх ресурсів для широкого кола користувачів. Платформа сприятиме підвищенню рівня знань та професійних навичок шляхом надання різноманітних курсів. Вона має забезпечити зручний інтерфейс та широкий ряд функціональних можливостей для комфортного навчання. Основна мета – зробити освіту доступною, зручною та ефективною для всіх, незалежно від місця проживання та рівня попередніх знань.

1 ЗАДАЧІ ПРОГРАМНОГО РІШЕННЯ – СТВОРЕННЯ ПЛАТФОРМИ ОНЛАЙН-КУРСІВ ДЛЯ ВИВЧЕННЯ ВЕБ-ТЕХНОЛОГІЙ ТА ВЕБ-ДИЗАЙНУ

Основною метою платформи для вивчення веб-технологій та веб-дизайну є зробити процес навчання цих дисциплін якомога зручнішим і ефективнішим.

Для досягнення мети треба виконати наступні завдання:

• Аналіз аналогічних систем.

• Аналіз програмних засобів для реалізації платформи.

• Визначення функціональних можливостей платформи, варіантів її використання.

• Розробка клієнтської частини платформи з урахуванням специфіки ролей користувачів.

• Розробка серверної частини платформи для забезпечення виконання логіки застосунку.

Платформи вивчення онлайн курсів має містити наступні функціональні можливості:

• Вивід всіх курсів у вигляді карток, з описом курсу та додаткової інформації про курс.

• Можливість реєстрації, авторизації.

• Можливістю реєстрації на обраний курс.

• Управління профілем: зміна прізвища, ім’я, електронної пошти, фото профілю, паролю.

• Перегляд курсів на які є зареєстрованим користувач.

• Перегляд вмісту курсу: уроків, домашніх завдань.

• Можливість виконання домашніх завдань шляхом надсилання файлів.

• Оцінювання домашніх завдань.

• Написання коментарів до домашнього завдання учнем та учителем.

• Розширене фільтрування даних.

2 ОПИС ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ ОНЛАЙН ПЛАТФОРМ ДЛЯ НАВЧАННЯ

Детально розглянемо та порівняємо LMS платформи, а саме: Google Classroom, Moodle, Westudy та Softbook. Проведемо аналіз даних платформ. Виділимо їхні переваги та недоліки.

2.1 Google Classroom

Google Classroom – це безкоштовна платформа, розроблена компанією Google для освітніх закладів, що дозволяє вчителям створювати, публікувати та оцінювати завдання (рис. 2.1). Вона спрощує процес збору робіт від учнів та взаємодію між учасниками навчального процесу. У Google Classroom можна створювати віртуальні класи, куди вчителі запрошують учнів та співробітників, де можна додавати завдання, матеріали для читання, відео та інші ресурси. Вона також інтегрується з іншими продуктами Google, такими як Google Drive, Google Docs, Google Sheets та іншими, що дозволяє зручно працювати з документами.

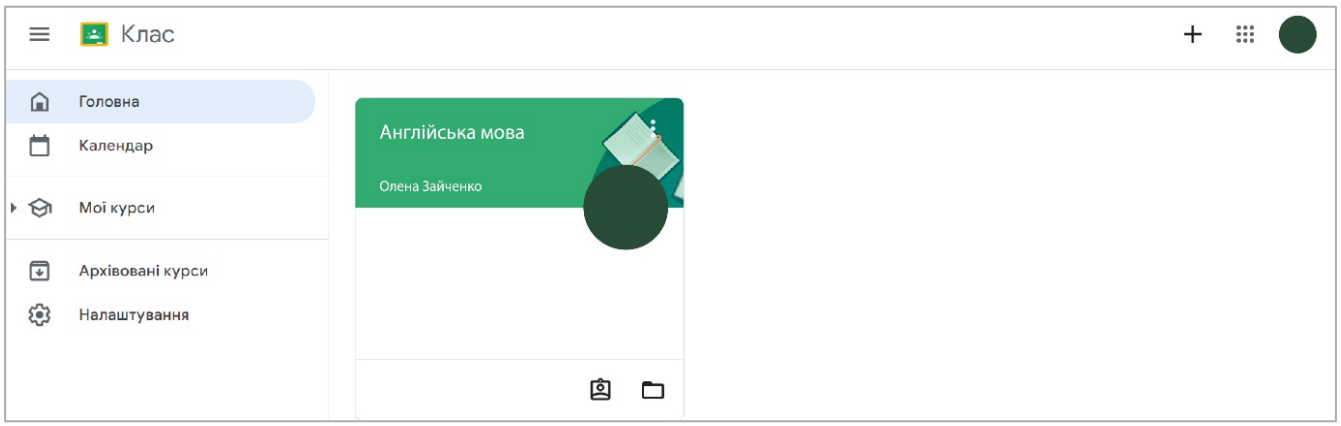


Рисунок 2.1 – Головна сторінка веб-застосунку Google Classroom

Основні переваги даної платформи, які можна виділити:

• Наявність веб-застосунку та додатку для мобільних пристроїв, що забезпечує зручне навчання на будь-якому пристрої.

• Сповіщення про отримання оцінки, нагадування про необхідність здачі роботи до певного терміну.

• Інтеграція з іншими продуктами Google.

• Можливість надсилати запрошень учням.

• Можливість додавати декількох учителів або кураторів до класу.

Але існує і низка недоліків. Основним недоліками даної платформи являються:

• Відсутність фільтрування за датою публікації матеріалу. Задля того щоб переглянути найперший урок курсу доведеться перелискувати всі уроки.

• Розміщення оголошень та завдань усіх разом в одній стрічці. Якщо завдання, які потрібно виконати, можна переглянути на одній сторінці, то переглянути матеріали курсу окремо неможливо. При наявності великої кількості уроків разом з оголошеннями стає важко знайти потрібний запис та зорієнтуватися.

• Відсутність пошуку за назвою уроку або завдання.

• Відсутність відслідковування прогресу та отримання загальної оцінки по всьому курсу.

• Відсутність можливості створення інтерактивних тестів або завдань.

2.2 Moodle

Moodle – це безкоштовна платформа для управління навчанням, яка надає широкі можливості (рис. 2.2). Завдяки своїй гнучкості, Moodle став популярним інструментом для організації онлайн-курсів у навчальних закладах та корпоративних середовищах. Проте, використання Moodle може вимагати певного рівня підготовки і знань, а також технічної підтримки.

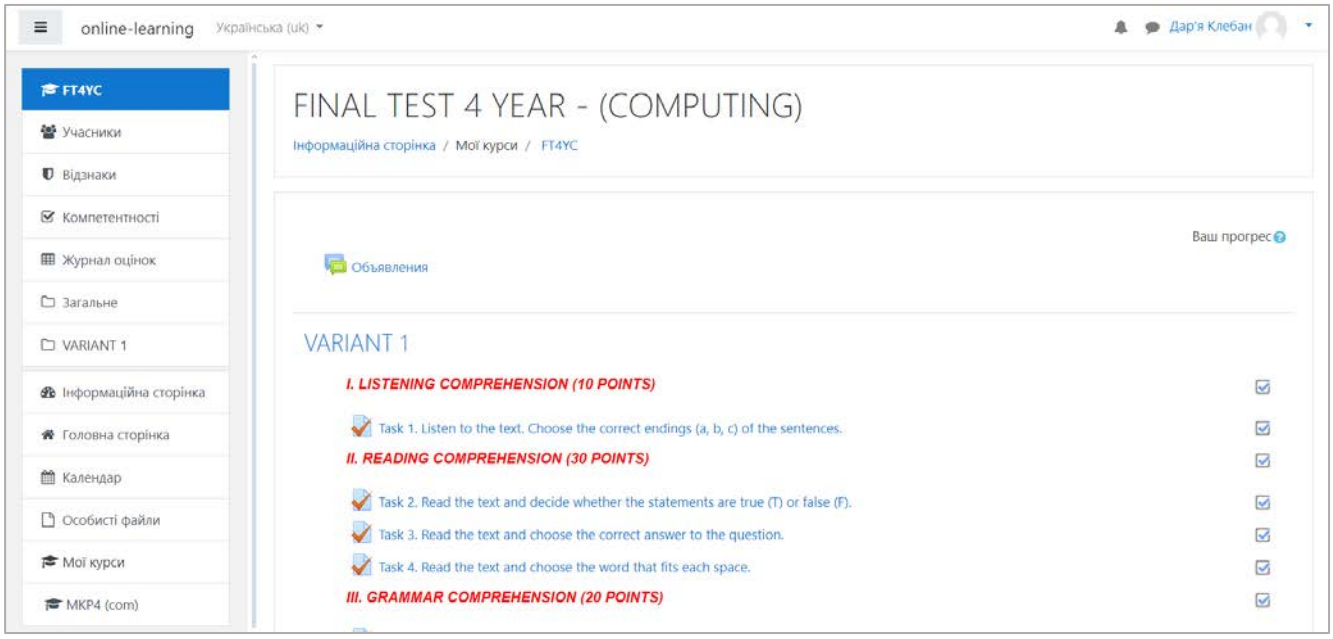


Рисунок 2.2 – Сторінка курсу із завданнями веб-застосунку створеного на основі платформи Moodle

Основні переваги використання, які можна виділити:

• Надає широкі можливості для взаємодії між учасниками навчального процесу, такі як форуми, чати, відеоконференції тощо.

• Гнучкий, що дозволяє налаштовувати робочий простір під індивідуальні вимоги.

• Можливість створювати тести з великою низкою налаштувань.

Недоліками даної платформи є:

• Складність використання для нових користувачів. Для тих, хто тільки починає знайомитися з Moodle, інтерфейс може здатися складним та заплутаним.

• Підтримка та налаштування Moodle може вимагати технічних знань або спеціаліста з IT, щоб забезпечити ефективну роботу платформи.

• Великі вимоги до хостингу. При великій кількості користувачів або обсязі навчального матеріалу, можуть виникнути проблеми з продуктивністю та швидкістю роботи Moodle.

2.3 Westudy

Westudy – це платна українська платформа управління навчанням, призначена для оптимізації освітнього процесу (рис. 2.3). Вона пропонує багато інструментів для створення курсів та дозволяє відстежувати успішність студентів.

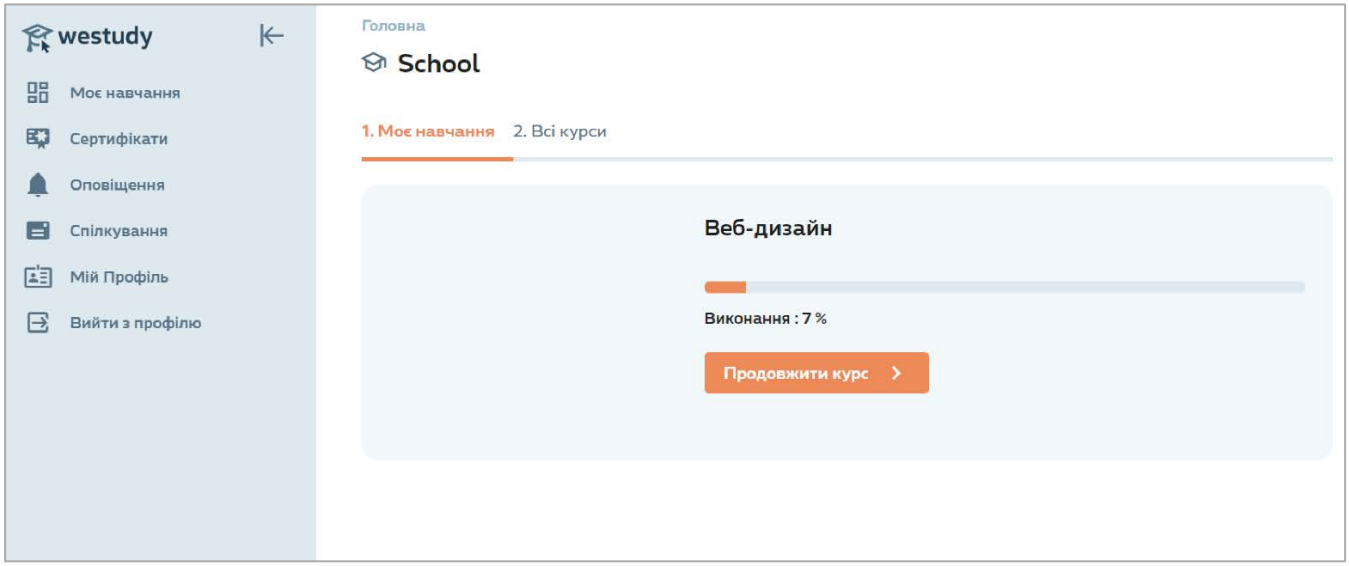


Рисунок 2.3 – Сторінка курсів на які зареєстрований користувач на основі платформи Westudy

Основні переваги даної платформи:

• Можливість створення тестів для студентів з автоматичним підрахунком балів.

• Відслідковування прогресу студента.

• Генерація сертифікатів.

Недоліками даної платформи є:

• Недосконалий інтерфейс. Уроки та домашні завдання розміщенні разом в одній стрічці, що ускладнює процес знаходження потрібного запису.

• Відсутність фільтрацій.

• Відсутність пошуку уроку або завдання.

• Дорогий тарифний план.

2.4 Softbook

Softbook – це платна платформа управління навчанням від українських розробників, яка пропонує широкий спектр функцій для створення онлайн-курсів (рис. 2.4). Платформа містить інструменти для створення курсів, організації навчального процесу та відстеження результатів.

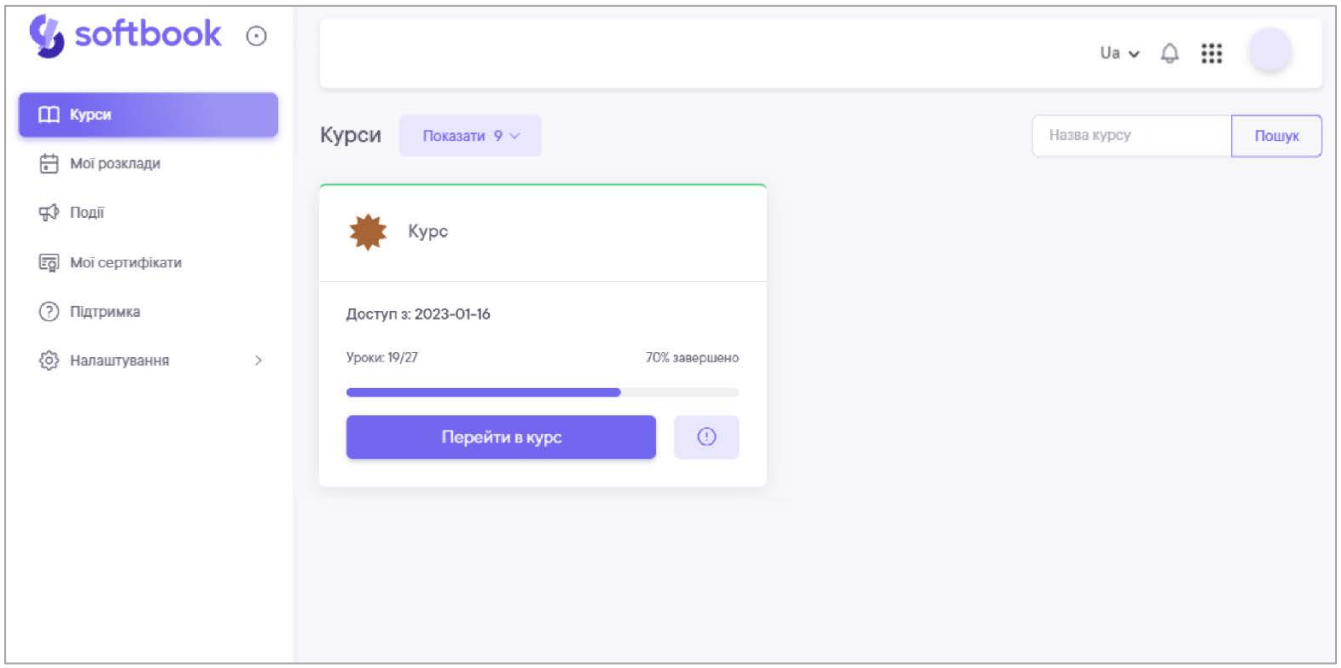


Рисунок 2.4 – Сторінка курсів на які зареєстрований користувач на основі платформи Softbook

Переваги даної платформи:

• Можливість створення тестування.

• Відслідковування прогресу студента.

• Інтеграція Zoom-акаунту з платформою.

Недоліки даної платформи:

• Відсутність фільтрацій.

• Відсутність пошуку уроку або завдання.

• Дорогий тарифний план.

2.5 Порівняння платформ

В результаті аналізу платформ була створена порівняльна таблиця. Усі дані, які були зібрані в ході аналізу, наведено та представлено у таблиці 2.1. У цій таблиці "+" позначає присутність, а "–" відсутність певної функціональної можливості. Крім того, в таблицю додано дані про платформу, яка розроблюється в контексті дипломного проєкту.

Таблиця 2.1 – Порівняння платформ онлайн навчання

Платформи онлайн-навчання / Функціональні можливості	Google Classroom	Moodle	Westudy	Softbook	Дипломний проєкт
Сортування по часу публікації	–	+	–	–	+
Пошук по назві завдання/оголошення	–	+	–	–	+
Підтримка різних типів контенту	+	+	+	+	+
Інтуїтивно зрозумілий інтерфейс	+	–	–	+	+
Подача заявки на курс з подальшим відслідковуванням її стану	–	–	–	–	+
Наявність особистого кабінету та змога його редагувати	–	+	+	–	+
Відстеження прогресу проходження курсу	–	+	+	+	+
Прикріплення файлів домашніх завдань	+	+	–	+	+

Можемо бачити, що різні платформи мають свої сильні та слабкі сторони. Google Classroom відзначається простотою у використанні та безкоштовністю, але має обмежені функціональні можливості. Moodle пропонує широкі функціональні можливості та гнучкість, проте має складний інтерфейс та вимагає технічної підтримки. Westudy надає широкий набір функцій, але не підтримує деякі базові можливості та не є безкоштовним. Softbook пропонує багато функціональних переваг, але його недоліки та висока вартість переважають ці переваги.

3 ЗАСОБИ РЕАЛІЗАЦІЇ

Детально розглянемо технології для створення веб-застосунків. Проаналізуємо особливості використання MySQL, React, Node.js, Express.js, SCSS, React Router, Axios, JSONWebToken та BcryptJS. Виділимо переваги використання даних технологій.

3.1 СУБД MySQL

СУБД MySQL буде використовуватися для управління базою даних платформи.

Система управління базами даних (СУБД) - це комплекс програмно-мовних засобів, що дозволяє створити, контролювати та адмініструвати бази даних [3].

Використання MySQL має численні переваги порівняно з іншими СУБД.

MySQL надійна та безпечна система керування базами даних, що користується популярністю серед відомих компаній завдяки своїм функціям захисту даних від кібератак [4]. Простота використання, продуктивність та оптимальна швидкість є найголовнішими перевагами даної СУБД. Окрім того, ця система також забезпечує масштабованість, що дозволяє ефективно керувати даними від невеликих веб-додатків до великих програм, що обробляють багато даних. Враховуючи всі ці переваги та функціональність, MySQL відповідає вимогам для успішної реалізації платформи.

Бібліотека React

React – це JavaScript бібліотека, призначена для розробки інтерфейсів користувача.

Використання React надає багато переваг. По-перше, React пропонує віртуальний DOM, який дозволяє ефективно оновлювати лише ті елементи сторінки, які змінилися, замість перерендерування всього дерева DOM. Це значно підвищує продуктивність додатка та забезпечує більш плавне відображення змін.

По-друге, React можна використовувати разом з багатьма іншими допоміжними бібліотеками і інструментами, наприклад як Redux для керування станом додатка, React Router для налаштування навігації між сторінками додатку та інші. Ця розширюваність дозволяє швидше виконувати проєкти і полегшує реалізацію.

По-третє, React пропонує компонентний підхід до розробки, що дозволяє створювати повторно використовувані компоненти, що спрощує процес розробки, покращує функціональність веб-застосунків та зменшує кількість повторюваного коду [5].

Використання React для даного веб-додатку є доцільним, оскільки він пропонує широкий набір переваг, таких як скорочення обсягу коду, покращена функціональність та можливість багаторазового використання компонентів.

3.2 Бібліотека React Router

Однією з важливих складових платформи є налаштування маршрутизації між сторінками. Маршрутизація є важливим елементом, оскільки дозволяє користувачам переміщуватися між різними сторінками додатку. Для забезпечення маршрутизації було обрано React Router.

React Router – це JavaScript бібліотека для реалізації динамічної маршрутизації в додатках, створених на основі React. Цей інструмент використовується для створення маршрутизації на стороні клієнта та дозволяє ефективно керувати переходами між різними частинами веб-додатку.

На традиційних веб-сайтах браузер зазвичай завантажує новий документ із сервера щоразу, коли користувач натискає на посилання, і всі елементи знову потрібно обробляти. Маршрутизація на стороні клієнта за допомогою React Router

дозволяє оновлювати URL без повторного запиту до сервера, миттєво змінюючи інтерфейс і отримуючи лише необхідні дані. Це забезпечує швидшу та динамічнішу взаємодію, оскільки не потрібно завантажувати новий документ та повторно обробляти ресурси [6].

3.3 Метамова SCSS

Для оформлення сторінок використовуються каскадні таблиці стилів CSS. Ця мова спеціалізується на заданні зовнішнього вигляду веб-сторінок.

SCSS – це скриптова метамова, яка інтерпретується в CSS. Ця метамова дозволяє ефективніше керувати стилями великих проєктів. Основна перевага SCSS полягає в тому, що він пропонує більше можливостей, ніж CSS, такі як вкладення, створення функцій через міксини та успадкування. Використання SCSS спрощує процес написання стилів, зменшуючи кількість коду та забезпечуючи можливість повторного використання фрагментів коду.

Переваги використання SCSS включають його зрозумілий синтаксис, особливо для тих, хто вже вивчив CSS. Крім того, SCSS дозволяє ефективно використовувати змінні, функції та міксини для підтримки повторного використання коду, що збільшує продуктивність та зручність у розробці веб-сторінок [7].

3.4 Середовище Node.js та фреймворк Express.js

Node.js – це середовище виконання JavaScript, яке використовується на серверній стороні і дозволяє створювати високопродуктивні та масштабовані веб-застосунки. Використання JavaScript як мови програмування для реалізації серверної логіки забезпечує зручність та єдність у розробці, оскільки та сама мова використовується і на клієнтському, і на серверному боці.

Node.js працює на основі моделі керованих подій та асинхронного введення-виведення. Це дозволяє йому ефективно обробляти велику кількість запитів одночасно, оскільки не потрібно створювати окремий потік для кожного з'єднання. Замість цього, він використовує чергування за пріоритетами для ефективного розподілу ресурсів та обробки запитів. Це робить Node.js ідеальним вибором для розробки платформи [8].

Express.js – це фреймворк для Node.js, який спрощує створення серверної частини додатків та API. Express.js надає широкий набір функцій для обробки запитів та відправки відповідей, а також для маршрутизації, роботи з шаблонами та статичними файлами [9].

3.5 Бібліотека Axios

Axios — це клієнт-серверна JavaScript бібліотека, яка використовується для виконання HTTP-запитів з клієнтської або серверної сторони Node.js. Основна мета Axios полягає у спрощенні відправлення запитів та обробці відповідей сервера. Вона надає можливість відправляти різні типи запитів, такі як GET, POST, PUT, DELETE, PATCH та обробляти їхні відповіді. Крім того, Axios дозволяє налаштовувати перехоплювачі для обробки та модифікації запитів та відповідей, а також надає механізми для скасування запитів та обробки помилок, що робить його зручним і потужним інструментом для розробки веб-застосунків [10].

3.6 Бібліотека JSONWebToken

JSON Web Token (JWT) – це відкритий стандарт для створення токенів доступу, заснований на форматі JSON. Він надає безпечний спосіб передачі інформації між сторонами, оскільки використовує шифрування [11].

JSONWebToken – це JavaScript бібліотека для створення та підтвердження (перевірки) токенів, що використовуються для аутентифікації/авторизації користувачів. Ця бібліотека є реалізацією JSON Web Tokens (JWT) для Node.js.

JWT зазвичай використовується в контексті аутентифікації. Наприклад, після того, як користувач успішно увійшов в систему, сервер призначає йому JWT, який він буде використовувати при кожному запиті до сервера для підтвердження своєї особи. Сервер перевіряє підпис токена та заявку, щоб переконатися, що вона є дійсною, та здійснює відповідні дії згідно з правами користувача.

JWT є популярним стандартом у веб-розробці, оскільки він легко передається через HTTP заголовки та зберігає інформацію про автентифікацію на клієнтському боці, що дозволяє створювати розподілені та масштабовані системи [12].

3.7 Бібліотека BcryptJS

BcryptJS — це JavaScript-бібліотека для хешування та порівняння паролів за допомогою алгоритму bcrypt.

Під час авторизації користувача за допомогою BcryptJS, спочатку відбувається витягування збереженого хешу пароля з бази даних. BcryptJS порівнює введений користувачем пароль із збереженим хешем пароля. Цей процес дозволяє забезпечити безпеку паролів, оскільки фактичний пароль ніколи не зберігається у відкритому вигляді, а порівнюється лише його хешована версія. Такий метод забезпечує безпеку паролів користувачів у веб-застосунках, що є критичним аспектом для забезпечення конфіденційності та захисту особистої інформації [13].

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПЛАТФОРМИ ОНЛАЙН-КУРСІВ ДЛЯ ВИВЧЕННЯ ВЕБ-ТЕХНОЛОГІЙ ТА ВЕБ-ДИЗАЙНУ

Детально розглянемо процес створення платформи, починаючи з визначення типу веб-додатку, вибору оптимальної архітектури та створення діаграми прецедентів для визначення взаємодії користувачів з системою. Розробимо концептуальну модель даних та опишемо процес реалізації клієнтської та серверної частини додатку.

4.1 Тип веб-додатку

Існують різні типи веб-додатків, основними виділяють наступні: одно сторінкові додатки (SPA) та багатосторінкові додатки (MPA). Розглянемо детальніше кожен тип.

SPA (Single Page Application) – це тип веб-додатка, який працює в браузері та взаємодіє з користувачем, не перезавантажуючи повністю сторінку. У SPA всі необхідні ресурси завантажуються разом з першою сторінкою, а подальша навігація відбувається за допомогою AJAX-запитів (Asynchronous JavaScript and XML) на сервер для отримання та відображення нового контенту. Коли користувачі запускають такий додаток, динамічне перезаписування за допомогою JavaScript дозволяє уникнути перезавантаження або завантаження додаткових сторінок. Користувачі бачать весь основний вміст у браузері, а під час прокручування або переходу на інші сторінки необхідні елементи просто завантажуються замість повного перезавантаження додатку.

MPA (Multi Page Application) – це тип веб-додатка, що складається з великої кількості сторінок, які повністю оновлюються кожного разу, коли відбувається запит на їх відображення або на сторінках змінюються дані. Будь-яка зміна даних

або передача даних на сервер призводить до появи нової сторінки, що відображається в браузері. Це класичний підхід до розробки веб-додатків, який вимагає багаторівневої навігації та більшої уваги до швидкості та продуктивності [14].

Платформа онлайн-навчання реалізовано як одно сторінковий додаток – Single Page Application. Порівняємо SPA та MPA та внесемо порівняння в таблицю 4.1.

Таблиця 4.1 – Порівняння SPA та MPA

	SPA	MPA
Продуктивність	Всі ресурси завантажуються протягом одного сеансу, а потім при взаємодії зі сторінкою змінюються лише необхідні дані. Такий підхід значно підвищує продуктивність.	У разі великої кількості запитів і необхідності перезавантаження великої кількості сторінок продуктивність і швидкість неминуче зменшуються. Особливо це стосується проєктів із високим відвідуванням сайту, великою кількістю сторінок та функцій.
Час завантаження	Коли користувач переміщується в межах додатку, сторінки завантажуються швидше, що забезпечує більш плавну та зручну взаємодію. Проте перше завантаження додатку може зайняти більше часу, оскільки весь додаток завантажуються одразу. Це можна покращити завдяки рендерингу на стороні сервера.	Якщо додаток містить велику кількість сторінок та ресурсів, кожен перехід між сторінками може призводити до додаткових затримок завантаження. Кожного разу, коли користувач відкриває нову сторінку, браузер завантажує всю необхідну HTML-структуру, CSS та JavaScript заново.
SEO	Є певні проблеми з SEO, оскільки немає окремих URL-адрес, оптимізованих для пошукових систем, через це пошукові системи не бачать вміст. Проблему може вирішити рендеринг на стороні сервера. Його можна викликати за допомогою додаткових фреймворків.	Налаштовувати SEO досить просто. Завдяки наявності кількох сторінок та окремих URL-адрес для кожної з них, пошукові системи можуть легко бачити та індексувати вміст цих сторінок.

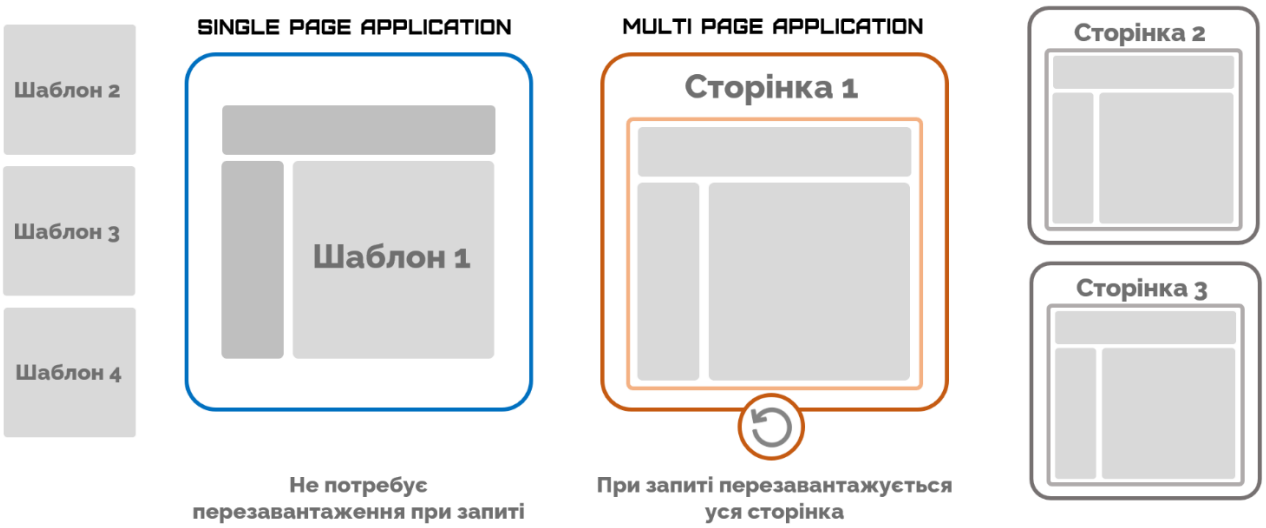


Рисунок 4.1 – Порівняння SPA та MPA

Для створення платформи онлайн-курсів, SPA-додаток є вигідним рішенням, оскільки забезпечує швидку взаємодію, що особливо важливо. Додатково, SPA-додатки спрощують управління станом додатку та дозволяють миттєво оновлювати контент без необхідності перезавантаження сторінок. Згідно порівнянню у таблиці 4.1 та рисунку 4.1, можна побачити переваги SPA над MPA.

4.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення – це структура програмної системи, яка визначає її основні компоненти, їхні взаємозв’язки та принципи взаємодії. Це своєрідний план, який допомагає зрозуміти, як компоненти системи працюють разом і як повинна функціонувати система в цілому [15]. Існують різні типи архітектури програмного забезпечення.

Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну має клієнт-серверну архітектуру.

Клієнт-серверна архітектура – це архітектура програмного забезпечення, в якій задачі обробки даних та управління ресурсами розподілені між клієнтами та

сервером. Основні компоненти цієї архітектури включають клієнта, сервер і базу даних. Клієнт надсилає запити, а сервер обробляє їх та повертає відповіді. Це дозволяє зменшити навантаження на клієнтські машини, делегуючи основну обробку серверу. Клієнт-серверна архітектура передбачає зв'язок між своїми компонентами у вигляді мережного з'єднання. Детальніше розглянемо кожен з компонентів даної архітектури.

• Клієнт – це програма, найчастіше всього браузер, який взаємодіє з сервером для отримання або відправлення даних.

• Сервер – це потужний комп'ютер або програма, що обробляє запити від клієнтів і забезпечує доступ до ресурсів, таких як дані, файли та інші сервіси. Сервер розміщує в базі даних відомості та витягує їх на запит клієнта.

• База даних – це організована колекція даних, яка управляється системою управління базами даних (СУБД). Вона є важливим компонентом клієнт-серверної архітектури, оскільки в ній зберігаються дані, які використовуються як клієнтами, так і сервером.

Опишемо взаємодію компонентів клієнт-серверної архітектури (рис. 4.2):

• Клієнт формує запит (наприклад, на отримання інформації) і відправляє його на сервер.

• Сервер отримує запит, обробляє його за допомогою певної логіки, за потреби звертається до бази даних.

• База даних виконує запит, повертаючи необхідну інформацію серверу.

• Сервер формує відповідь і відправляє її клієнту.

• Клієнт отримує відповідь і відображає її користувачу.

Клієнт-серверна архітектура поділяється на кілька типів – дворівнева та багаторівнева, як окремий тип багаторівневої виділяють трирівневу архітектуру. Розглянемо дворівневу та трирівневу архітектуру більш детальніше.

Дворівнева архітектура містить всього два компоненти: клієнт та сервер. Тут сервер виконує функції додатку та бази даних. Такий підхід використовують для невеликих додатків, де немає великої кількості клієнтів. Цей спосіб є простішим в реалізації, але його надійність низька. Якщо сервер зламають, то зловмисники отримають усі дані.

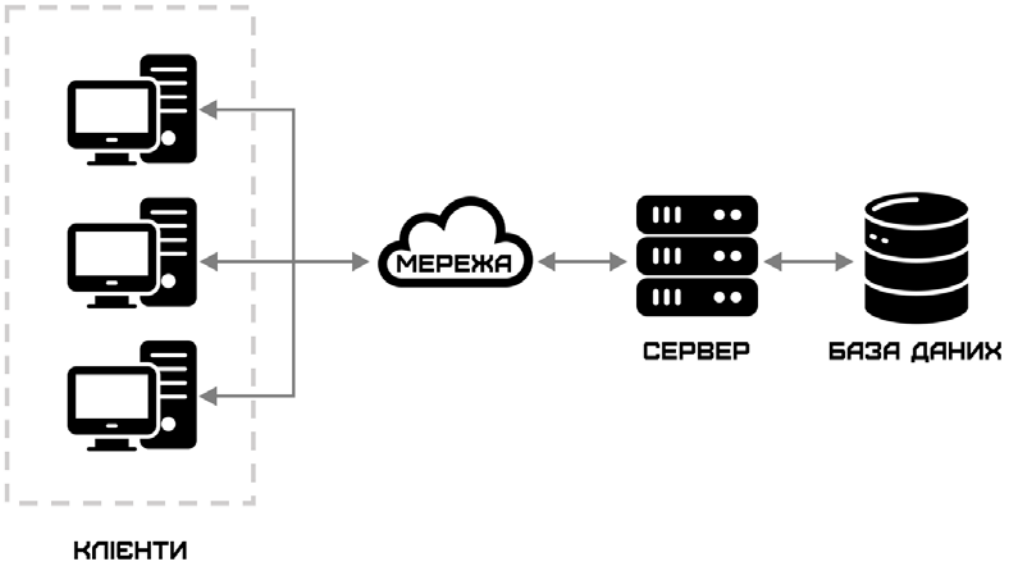


Рисунок 4.2 – Схема взаємодії компонентів клієнт-серверної архітектури

Трирівнева архітектура є набагато надійнішою з точки зору безпеки, аніж дворівнева. На додаток до вже згаданих компонентів архітектури у схемі бере участь база даних, що зберігає інформацію. Вона розташована окремо від інших компонентів. Сервер займається виключно обробкою відомостей, звертаючись до бази даних задля отримання або збереження інформації.

Головною особливістю даної архітектури є роздільний функціонал. Таким чином ефективно розподіляється навантаження між компонентами архітектури, покращується масштабованість і більш точно і швидко відбувається обробка інформації. При цьому користувачу не потрібно бути ознайомленим з внутрішніми процесами та обчисленнями. Для нього існує тільки клієнт зі зручним інтерфейсом, за допомогою якого користувач отримує відповіді на свої запити.

Клієнт-серверна архітектура надійна, адже навантаження розподіляється між серверами, і якщо один з них вийде з ладу, клієнти відразу ж перемикаються на інший, не перериваючи з'єднання.

Сервер централізовано управляє ресурсами та контролює інформацію, що надається клієнтам. Це полегшує управління та дозволяє застосувати єдині політики безпеки та доступу до інформації [16].

Розглянемо переваги та недоліки даної архітектури.

Перевагами являються:

• Масштабованість. Навантаження розподіляється між серверами, кількість яких можна збільшувати. Це підвищує продуктивність системи та дозволяє обробляти більше клієнтських запитів.

• Централізоване керування. Сервер виступає у ролі центрального вузла, здійснюючи контроль над системою.

• Надійність. Вихід з ладу одного з компонентів не впливає на роботу інших, що підвищує стійкість системи до відмов.

• Безпека. Завдяки централізованому управлінню сервер забезпечує контроль доступу та захисту інформації.

• Відсутність дублювання коду. Ця перевага полягає в централізованому зберіганні даних на сервері, що запобігає наявності множинних копій даних і зменшує ризик їх невідповідності. Це забезпечує легше оновлення та підтримку даних, гарантуючи, що всі клієнти мають доступ до актуальної та узгодженої інформації.

• Доступ з будь-якого пристрою. Клієнтом може бути веб-браузер, мобільний або веб-додаток.

• Мінімальні вимоги до користувача. На пристрій достатньо встановити лише програму-клієнт.

• Висока продуктивність. Сервер набагато продуктивніший, ніж звичайний ПК. Він обробляє величезну кількість запитів від великої кількості користувачів.

Але дана архітектура також має недоліки. Основними недоліками являються:

• Повна залежність клієнта від сервера. Якщо сервер недоступний або працює зі збоями, всі клієнти виявляться нефункціональними і не зможуть відобразити дані.

• Залежність від якості мережі. Якщо мережа недоступна або працює зі збоями, система не зможе повноцінно функціонувати.

• Обмеження. Можна одночасно підключити лише певну кількість клієнтів. Також на обробку запитів впливає пропускна спроможність мережі.

• Перевантаження сервера. Сервер може не впоратися з величезною кількістю запитів і відключитися.

Для ліквідації чи зменшення впливу недоліків зазвичай застосовують кластери серверів та балансувальники (рис. 5.3) [17].

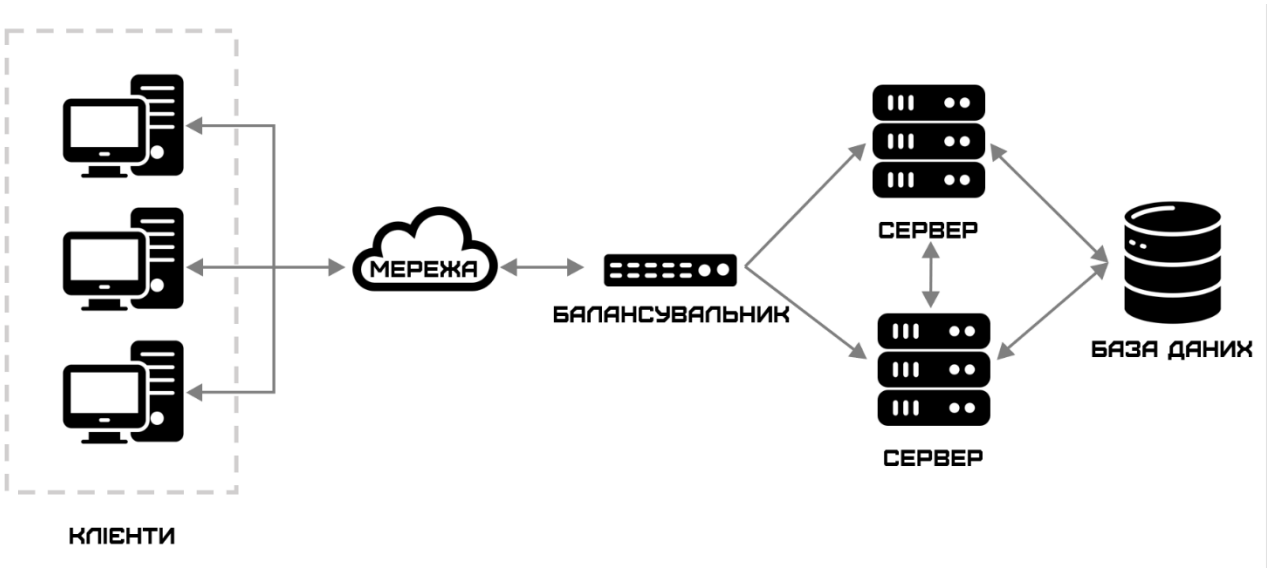


Рисунок 4.3 – Схема взаємодії компонентів клієнт-серверної архітектури з кластером серверів та балансувальником

Кластер серверів – це група серверів, що працюють разом в одній системі, забезпечуючи більш високу доступність та продуктивність. Завдяки цьому ресурси розподіляються між кількома серверами, що дозволяє системі продовжувати функціонувати навіть у випадку виходу з ладу одного з серверів. Це досягається завдяки тому, що інші сервери в кластері автоматично беруть на себе навантаження та виконують завдання, що були призначені для несправного сервера.

Балансувальники навантаження розподіляють вхідний трафік між кількома серверами, щоб уникнути перевантаження одного конкретного сервера і забезпечити рівномірний розподіл запитів. Балансування навантаження і кластери серверів значно підвищують надійність та відмовостійкість системи. Дана архітектура застосовується для високонавантажених додатків, коли обсяг вхідних запитів настільки великий, що один сервер не може з ними впоратися [18].

Розширену архітектуру додатку зображено на рисунку 4.4. На даній схемі відображені основні компоненти архітектури та технології, які використовувалися для її реалізації

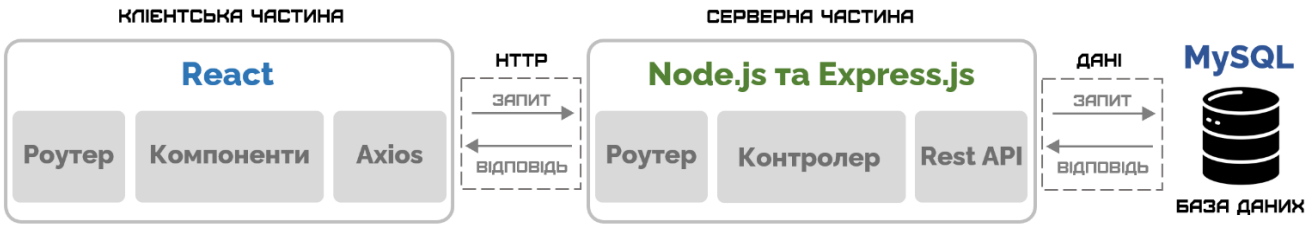


Рисунок 4.4 – Розширена схема взаємодії компонентів архітектури платформи онлайн-курсів

Детально про кожний елемент даної схеми розказано у розділах 4.3 «Концептуальна модель даних», 4.4 «Клієнтська частина системи», 4.5 «Серверна частина системи».

4.3 Діаграма прецедентів

Діаграма прецедентів (Use case diagram) або діаграма варіантів використання –графічне представлення функціональних вимог до системи, шляхом демонстрації взаємодії між користувачами (акторами) та випадками використання (use cases). Діаграма ілюструє, які дії можуть виконувати користувачі та які результати слід очікувати від системи. Діаграми прецедентів використовуються для моделювання системи з точки зору зовнішнього спостерігача [19].

Щоб створити діаграму прецедентів до заданої предметної області, потрібно визначити акторів системи. В даному випадку маємо наступних акторів: студент, викладач. Опишемо дії, які може виконувати кожен з акторів.

Студент може виконувати наступні дії:

- Реєстрація на платформі.
- Реєстрація на курс.
- Перегляд курсів на які зареєстровано користувача.
- Перегляд вмісту курсів: уроків, домашніх завдань.
- Надсилання домашньої роботи.

• Перегляд стану та оцінки домашньої роботи.

Викладач може виконувати наступні дії:

• Додавати: новий курс, уроки до курсу, домашні завдання до уроків.

• Видаляти та редагувати: курси, уроки, домашні завдання.

• Переглядати, приймати або відхиляти: заявки реєстрації студентів на курси.

• Переглядати та оцінювати домашні роботи студентів.

Спільні дії, які доступні як студенту так і викладачу:

• Вихід у систему.

• Вихід із системи.

• Перегляд головної сторінки.

• Перегляд особистого кабінету.

• Перегляд всіх існуючих курсів.

• Управління акаунтом: змінити дані, змінити пароль.

• Писати коментарі до домашньої роботи.

На основі письмового опису була створена діаграма прецедентів. Отриману діаграму можна переглянути на рисунку 4.5.

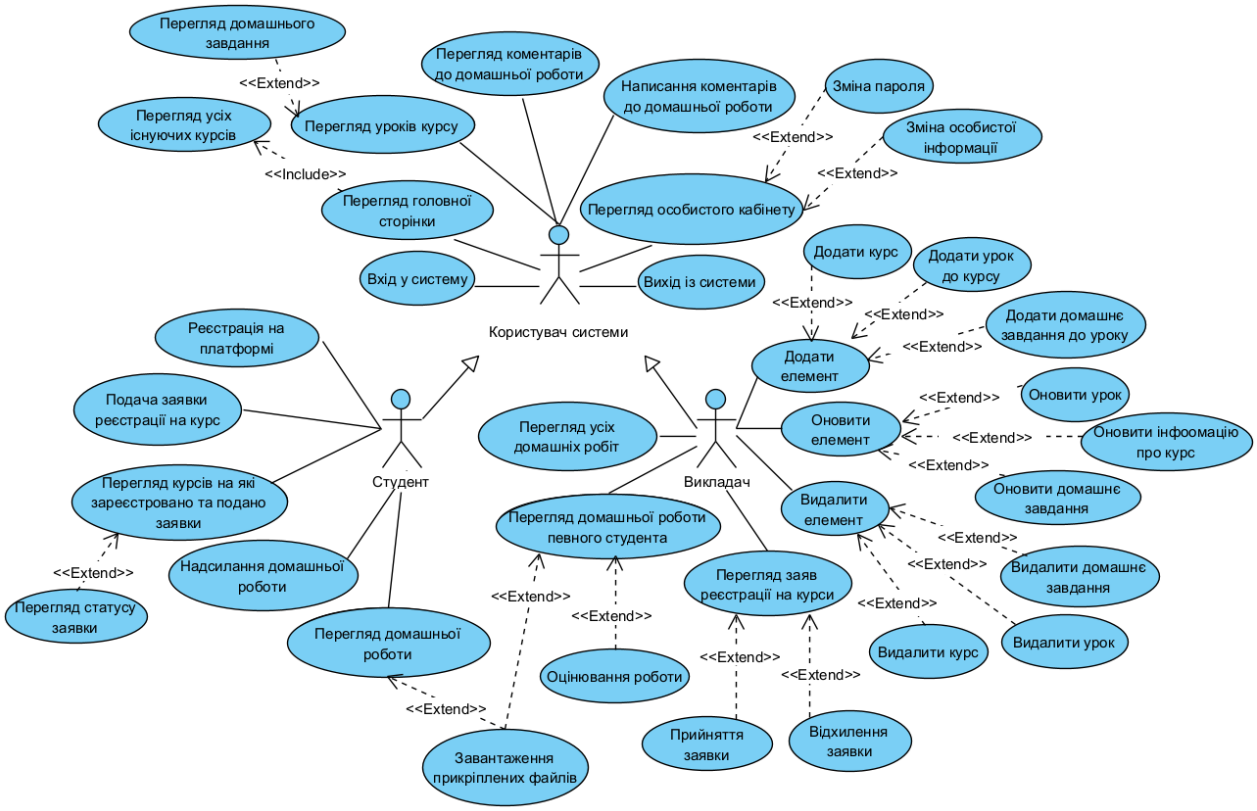


Рисунок 4.5 – Діаграма прецедентів

На основі створеної діаграми прецедентів будуть реалізовані функціональні можливості платформи.

4.4 Концептуальна модель даних

Концептуальна модель даних – це структуроване представлення концепцій бази даних. Основна мета створення концептуальної моделі даних полягає у визначенні сутностей їх атрибутів та зв’язків, які повинні бути представлені в базі даних [20]. Концептуальна модель даних допомагає описати базу даних, яку потрібно створити, та встановлює загальні правила, які слід враховувати при її розробці.

Зв'язки між таблицями формуються шляхом використання зовнішніх ключів (foreign key). Зовнішній ключ представляє собою атрибут або групу атрибутів, які вказують на первинний ключ іншої таблиці. Іншими словами, він є посиланням на конкретний рядок у іншій таблиці.

Зв'язки поділяються на 3 типи:

• Багато до багатьох.

• Один до багатьох.

• Один до одного.

Створимо концептуальну модель відповідно до нашої предметної області, використовуючи наступні таблиці: користувачі (Users), курси (Courses), уроки (Lessons), домашні завдання (Homeworks), домашні роботи (User_Homework), курси на які зареєстровано користувача (User_Course), рівні складності (Levels), статуси домашніх робіт (Homework_statuses), статуси заявок на курси (Application_statuses), статуси користувачів (User_statuses), коментарі (Comments), прогрес студента (Progress).

Концептуальна модель зображена на рисунку 4.6.

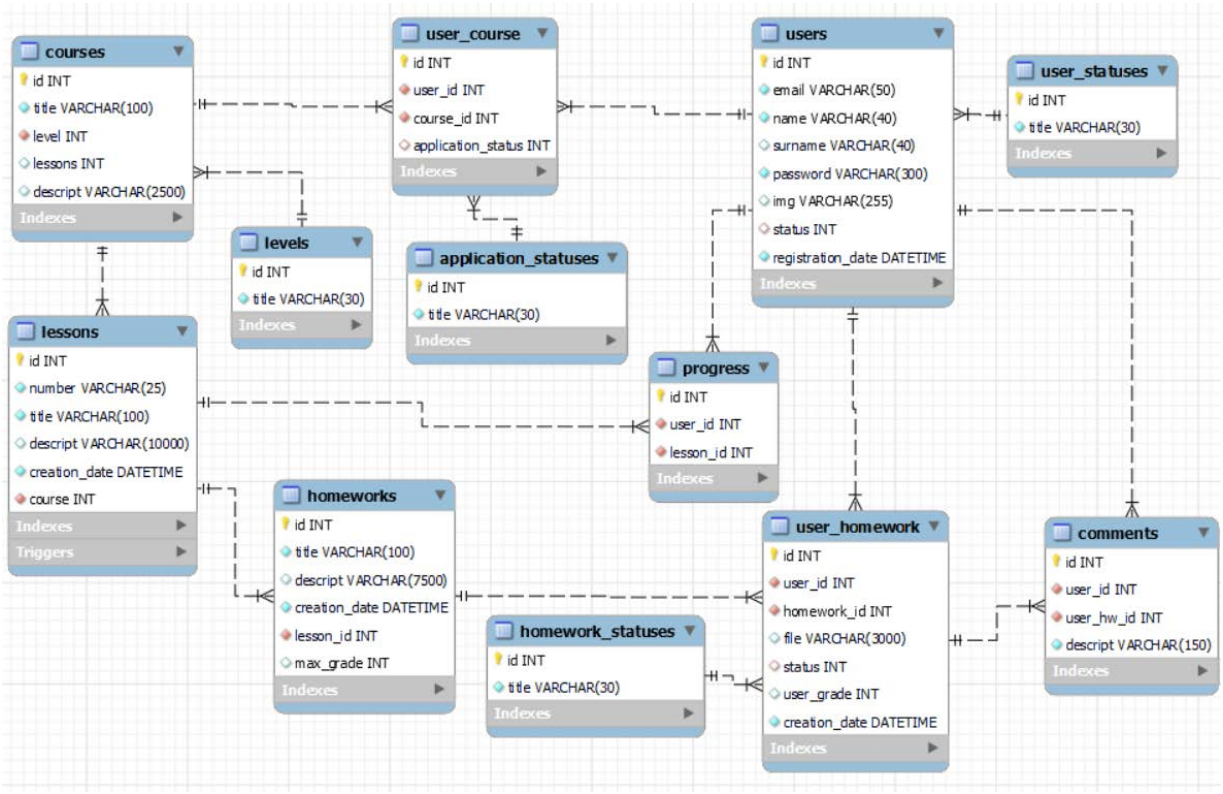


Рисунок 4.6 – Концептуальна модель бази даних для платформи онлайн курсів

Опишемо деякі зв’язки між таблицями:

• Курси (courses) відноситься до уроків (lessons), як багато до одного, оскільки урок може належати тільки одному курсу, але курс може мати багато уроків.

• Уроки (lessons) відносяться до домашніх завдань (homeworks) як один до одного. Кожен урок може містити тільки одне домашнє завдання, і кожне домашнє завдання належить тільки одному уроку.

4.5 Клієнтська частина системи

Клієнтська частина системи розроблена з використанням бібліотеки React і є ключовим елементом, що забезпечує взаємодію користувачів із платформою. Вона відповідає за відображення інформації, управлінням взаємодією користувача з інтерфейсом і передачу даних між клієнтом і сервером.

Основна роль клієнтської частини – це забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволяє користувачам легко реєструватися на платформі, переглядати доступні курси, ознайомлюватися з навчальними матеріалами та виконувати домашні завдання. Завдяки використанню сучасних технологій, таких як React, система може швидко реагувати на дії користувача, забезпечуючи високу продуктивність і інтерактивність додатку. Використання React дозволяє реалізовувати компонентний підхід до розробки, що сприяє модульності, перевикористанню коду і спрощенню процесу підтримки та оновлення платформи.

Згідно проаналізованих вимог було розроблено структуру інтерфейсу, яка забезпечує різні сценарії відображення для користувачів з різними ролями. Передбачається 2 ролі користувачів у системі: викладач та учень. Кожному користувачу наявні різні інтерфейсні елементи згідно його ролі у системі. Цей підхід гарантує зручність та ефективність використання платформи для всіх категорій користувачів.

До моменту авторизації користувача і визначення його ролі та прав на перегляд та взаємодію зі сторінками, користувач вважається неавторизованим і має обмежені права у системі. Неавторизованому користувачу доступний перегляд наступних сторінок: головна сторінка, сторінка реєстрації, сторінка входу.

Після авторизації користувач отримує роль в системі завчасно визначену для його акаунту. Структуру додатку та наявність сторінок для користувачів з різними ролями можна бачити на рисунку 4.7.

Після визначення структури додатку, для кожної сторінки було створено окремий компонент. Винесення елементів в компоненти дозволяє розділити складний інтерфейс на менші, автономні частини, які можна перевикористовувати та модифікувати незалежно один від одного, що сприяє збереженню чистоти та організації коду.

Рисунок 4.7 – Структура платформи

Не менш важливо створити такі перевикористовувані компоненти, як кнопки, модальні вікна тощо. Наприклад, якщо ми змінимо вигляд або функціональність компонента кнопки, який використовується на різних сторінках додатку, ці зміни автоматично застосуються до всіх інших сторінок, де цей компонент використовується. Це робить процес модифікації простішим, запобігає повторенню коду та забезпечує уніфікований вигляд і поведінку компонентів у всьому додатку. На рисунку 4.8 показано компонент кнопок редагування, а на рисунку 4.9 – приклад використання даного компоненту в контексті іншого компоненту.

```
const EditButtons = ({ editUrl, state, deleteUrl}) => {
    const handleDelete = async () => {
        try {
            await axios.delete(`/api/${deleteUrl}`);
        } catch (err) {
            console.log(err);
        }
    };
    return (
        <div className="editDelete">
            <button onClick={handleDelete}>
                <img src="/bin.svg" alt="" />
            </button>
            <Link to={`${editUrl}`} state={state}>
                <button>
                    <img src="/edit.svg" alt="" />
                </button>
            </Link>
        </div>
    );
};

export default EditButtons;
```

Рисунок 4.8 – Компонент кнопок редагування

```
<EditButtons
    editUrl={`/addcourse?edit=${course.id}`}
    state={course}
    deleteUrl={`courses/${course.id}`}
/>
```

Рисунок 4.9 – Використання компоненту кнопок редагування в іншому компоненті

Для створення інтерфейсу з повторюваними елементами, такими як хедер (header), футер (footer) та бічною панеллю (sidebar), використовуються особливий підхід у порівнянні з перевикористанням таких компонентів, як кнопки. Ці елементи присутні на кожній сторінці, а отже потребують винесення в окремі компоненти та інтегрування у загальний макет (layout). Макет відповідає за організацію розміщення компонентів та включає в себе ті, що повинні бути відображені на кожній сторінці. Такий підхід сприяє підвищенню продуктивності додатку, оскільки при навігації між сторінками ці елементи не потребують повторного рендерингу.

Такий макет створюється за допомогою бібліотеки React Router. Ця бібліотека також забезпечує налаштування маршрутизації сторінок. Layout у React Router – це структурний компонент, що визначає загальний макет сторінки або групи сторінок. Він містить спільні елементи інтерфейсу, такі як хедер, футер та бічну панель, які відображаються на всіх сторінках, що використовують цей макет.

У платформі передбачена певна кількість сторінок, такі як сторінка курсів, уроків, домашніх завдань та інші, кожна з яких містять хедер, бічну панель та футер. Основний контент, що змінюється залежно від сторінки, має відображатися у певному місці. Компонент Outlet є тим місцем, де буде відображатися цей змінний контент.

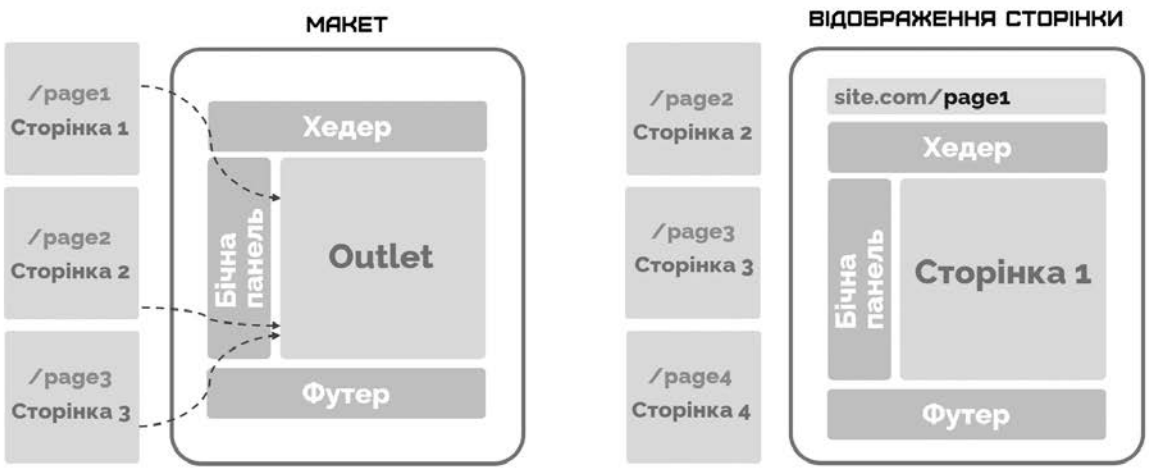


Рисунок 4.10 – Відображення сторінки створеної за допогою макету

При використанні компонента Layout, такі елементи, як хедер, бічна панель та футер, не перерендерюються при кожній зміні маршруту. Це означає, що ці компоненти залишаються стабільними та постійними, навіть коли користувач переходить з однієї сторінки на іншу. Тільки вміст всередині компонента Outlet змінюється відповідно до поточного маршруту. Це означає, що коли користувач переходить на певний шлях, React Router відображає вміст, визначений для цього шляху, у компоненті Outlet (рис. 4.10). Приклад створеного макету зображено на рисунку 4.11.

```
const Layout = () => {
  return (
    <div className="wrapper">
      <Header />
      <div className="content">
        <Sidebar/>
        <Outlet />
      </div>
      <Footer />
    </div>
  );
};
```

Рисунок 4.11 – Макет

Цей підхід дозволяє підтримувати постійну структуру сторінки, одночасно змінюючи лише основний контент залежно від маршруту. Це дозволяє підвищити продуктивність додатку, оскільки при переході на різні сторінки, ці елементи не перерендерюються та уникається дублювання коду.

Після створення всіх сторінок та налаштування макету, потрібно налаштувати маршрутизацію сторінок. Налаштування маршрутизації сторінок виконується за допомогою використання React Router. Це дозволяє керувати навігацією між різними частинами платформи.

Для налаштування маршрутизації потрібно створити маршрути та визначити, які компоненти будуть відображатися для кожного шляху. Для цього використовується createBrowserRouter для створення маршрутизатора з набором маршрутів. На рисунку 4.12 наведено фрагмент коду, який демонструє створення маршрутів. Кожен маршрут визначає шлях (path) та компонент (element), який буде відображено при переході на цей шлях.

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <Layout />,
    children: [
      {
        path: "/courses",
        element: <MainCoursePage />,
      },
      {
        path: "/profile/:id",
        element: <Profile />,
      },
```

Рисунок 4.12 – Фрагмент коду зі створення маршрутів

Розглянемо отримання та надсилання даних до сервера. На рисунку 4.13 відображена функція для отримання даних.

```
useEffect(() => {
    const fetchData = async () => {
        try {
            const response = await axios.get(`/api/lesson/${lessonId}`);
            setLesson(response.data);
        } catch (err) {
            console.log(err);
        }
    };
    fetchData();
}, [lessonId]);
```

Рисунок 4.13 – Отримання даних уроку

Отримати дані можна за допомогою хуку useEffect. useEffect – це хук у React, який дозволяє виконувати побічні ефекти у функціональних компонентах. Це можуть бути такі операції, як виклики API, підписки на веб-сокети, робота з локальним сховищем і т.д. В контексті розробки платформи онлайн-курсів useEffect використовується для отримання даних на відображенні їх у функціональному компоненті. За допомогою axios здійснюється HTTP-запит і в разі успіху отримуються дані.

Використовуючи axios можна також надсилати дані на сервер для їх подальшого запису або зміни. На рисунку 4.14 розміщено фрагмент коду для оновлення вмісти курсу або його створення в залежності від state. State представляє собою поточний стан компонента.

```
const handleClick = async (e) => {
    e.preventDefault();
    try {
        state
            ? await axios.put(`/api/courses/${state.id}`, {
                title,
                level,
                descript,
              })
            : await axios.post(`/api/courses`, {
                title,
                level,
                descript,
              });
        navigate("/courses");
    } catch (err) {
        console.log(err);
    }
};
```

Рисунок 4.14 – Отримання даних уроку

Функція handleClick перевіряє наявність state. Якщо він існує, вона виконує запит PUT на сервер для оновлення курсу з використанням axios.put. Якщо state не існує, тобто це новий курс, вона виконує запит POST для створення нового курсу за допомогою axios.post.

4.6 Серверна частина системи

У сучасних веб-додатках серверна частина відіграє ключову роль, забезпечуючи ефективну обробку запитів клієнтів, виконання логіки додатка та взаємодію з базою даних. Для реалізації серверної частини нашої системи було обрано технології Node.js та Express.js. Це рішення було прийнято з огляду на високу продуктивність, масштабованість та зручність використання даних технологій.

Одним не менш важливим аспектом реалізації є REST API. REST API (Representational State Transfer) – архітектурний стиль або набір правил взаємодії між різними компонентами програмного забезпечення. Він виступає як інтерфейс, через який клієнти можуть звертатися до сервера для отримання даних або виконання операцій. Для доступу до ресурсів та їхнього використання використовується HTTP-запити (рис. 4.15). Такими HTTP-запитами є GET, PUT, POST і DELETE. Вони дають змогу, відповідно, читати, змінювати, створювати й видаляти ресурси [21].

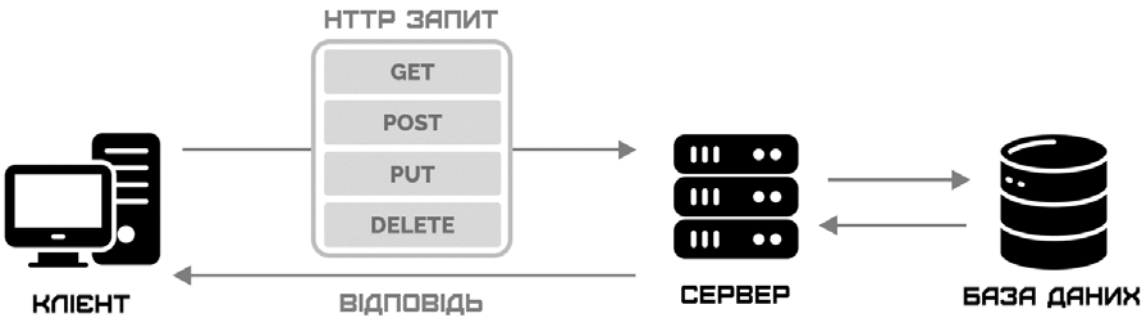


Рисунок 4.15 – Взаємодія елементів згідно REST API

Для створення серверної частини нашого додатка було використано платформу Node.js разом із фреймворком Express.js, що дозволяє ефективно обробляти HTTP-запити (рис 4.16).

```
const app = express()
app.use(express.json())
```

Рисунок 4.16 – Фрагмент коду зі створення серверу

Для налаштування підключення до бази даних було використано модуль mysql (рис 4.17). Підключення до бази даних дозволяє взаємодіяти з даними та виконувати запити.

```
export const db = mysql.createConnection({
    host:"host",
    user:"user",
    password:"password",
    database:"database"
})
```

Рисунок 4.17 – Приклад налаштування з'єднання з базою даних

З метою забезпечення чіткої структури та організації серверної частини додатка використовується маршрутизацію для різних функціональних компонентів. Основні маршрути були розподілені за відповідними логічними частинами додатка: авторизація, управління користувачами, курсами, уроками, заявками та домашніми завданнями (рис. 4.18). Сервер прослуховує запити на різних маршрутах і обробляє їх відповідно до методів HTTP.

```
app.use("/api/auth", authRoutes);
app.use("/api/users", userRoutes);
app.use("/api/courses", courseRoutes);
app.use("/api/lesson", lessonRoutes);
app.use("/api/applications", applicationRoutes);
app.use("/api/homework", homeworkRoutes);
```

Рисунок 4.18 – Маршрути на серверній стороні

Для структурування та організації коду серверної частини використовується концепція роутерів та контролерів. Роутери визначають маршрути запитів, тоді як контролери містять логіку обробки цих запитів. Розподіл між роутерами (routes) і контролерами (controllers) в серверній частині додатка дозволяє створити чітку і організовану структуру.

Роутери відповідають за обробку HTTP-запитів і перенаправлення їх до відповідних контролерів, де вже виконується основна логіка застосунку. Роутери відповідають за визначення шляхів (URL) та відповідних HTTP методів (GET, POST, PUT, DELETE). Вони приймають запити від клієнтів і визначають, який код слід виконати для обробки цих запитів (рис. 4.19).

```
import { register, login, logout } from "../controllers/auth.js";

const router = express.Router()

router.post("/register", register);
router.post("/login", login);
router.post("/logout", logout);

export default router;
```

Рисунок 4.19 – Роутери для реєстрації та аутентифікації користувачів

Контролери містять логіку обробки запитів, що надходять на певні роутери. Вони взаємодіють з базою даних для отримання або надсилання даних та формують відповіді, які відправляються назад клієнту (рис. 4.20).

```
export const addCourse = (req, res) => {
    const q = "INSERT INTO Courses(`title`, `level`,`descript`) VALUES (?)";

    const values = [req.body.title, req.body.level, req.body.descript];

    db.query(q, [values], (err, data) => {
        if (err) return res.status(500).json(err);
        return res.status(200).json("Курс додано.");
    });
};
```

Рисунок 4.20 – Фрагмент коду з контролера для виконання додавання нового курсу

Розглянемо детальніше реєстрацію та вхід користувача.

На рисунку 4.21 показано реалізацію процесу реєстрації нового користувача у системі.

```
export const register = (req, res) => {
    const q = "SELECT * FROM Users WHERE email = ?";

    db.query(q, [req.body.email], (err, data) => {
        if (err) return res.status(500).json(err);
        if (data.length)
            return res.status(409).json("Користувач вже зареєстрований!");

        const salt = bcrypt.genSaltSync(10);
        const hash = bcrypt.hashSync(req.body.password, salt);

        const q =
            "INSERT INTO Users(`email`, `name`, `surname`, `password`) VALUES (?)";
        const values = [req.body.email, req.body.name, req.body.surname, hash];

        db.query(q, [values], (err, data) => {
            if (err) return res.status(500).json(err);
            return res.status(200).json("Користувач успішно зареєстрований.");
        });
    });
};
```

Рисунок 4.21 – Функція реєстрації користувача

Перш ніж додати нового користувача, виконується перевірка, чи вже існує користувач з вказаною електронною поштою (email). Це забезпечує унікальність облікових записів та уникнення їх дублювання. Якщо користувач з такою електронною поштою вже існує, сервер повертає статус 409 (Conflict) разом з відповідним повідомленням про те, що користувач вже зареєстрований (рис. 5.8).

У випадку, якщо користувач з вказаною електронною поштою ще не існує, пароль, який було надіслано клієнтом, хешується за допомогою алгоритму bcrypt. Це забезпечує безпеку паролю, оскільки зберігається лише його хеш, а не сам пароль (рис. 4.22). Після цього дані про нового користувача (електронну адресу, ім'я, прізвище та хеш паролю) додаються до бази даних. У випадку успішного додавання користувача, сервер повертає статус 200 (OK) разом з повідомленням про успішну реєстрацію.

id	email	name	surname	password
6	svitlanareykiv@gmail.com	Світлана	Рейків	$2a$10$JhoFe0eYoICcWh1FD5ws3OqKNCf4lbB...

Рисунок 4.22 – Приклад збереження хешу паролю в базі даних в полі password

На рисунку 4.23 показано реалізацію процесу автентифікації користувача в системі. Спочатку виконується перевірка, чи існує користувач з вказаною електронною поштою в базі даних. Якщо користувач не знайдений, сервер повертає статус 404 (Not Found) разом з відповідним повідомленням (рис. 5.5).

```
export const login = (req, res) => {
    const q = "SELECT * FROM users WHERE email = ?";

    db.query(q, [req.body.email], (err, data) => {
        if (err) return res.status(500).json(err);
        if (data.length === 0)
            return res.status(404).json("Користувача не знайдено!");

        const isPasswordCorrect = bcrypt.compareSync(
            req.body.password,
            data[0].password
        );

        if (!isPasswordCorrect)
            return res.status(400).json("Неправильний пароль або e-mail!");

        const token = jwt.sign({ id: data[0].id }, "jwtkey");
        const { password, ...other } = data[0];

        res.cookie("access_token", token, {
            httpOnly: true,
        })
            .status(200)
            .json(other);
    });
};
```

Рисунок 4.23 – Функція входу користувача у систему

У випадку, якщо користувач існує, тобто його дані містяться у базі даних, порівнюється хеш паролю, збережений у базі даних, із хешем введеного користувачем паролю. Якщо паролі співпадають, сервер створює JWT токен для користувача з використанням його унікального ідентифікатора. Після цього дані про користувача, за винятком паролю, відправляються назад клієнту разом з токеном у вигляді відповіді на успішний запит.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

При відкритті платформи онлайн курсів відображається головна сторінка. На ній відображається логотип, кнопка «Увійти» в навігаційному барі, якщо користувач ще не увійшов до системи, основна інформація про платформу та перелік всіх існуючих курсів на які можна зареєструватися. Перелік курсів зроблений у вигляді карток, які йдуть одна за одною. На кожній картці курсу відображається його назва, короткий опис курсу, кількість уроків, рівень складності та кнопка «Зареєструватися». На рисунку 5.1 та 5.2 продемонстровано головну сторінку платформи.

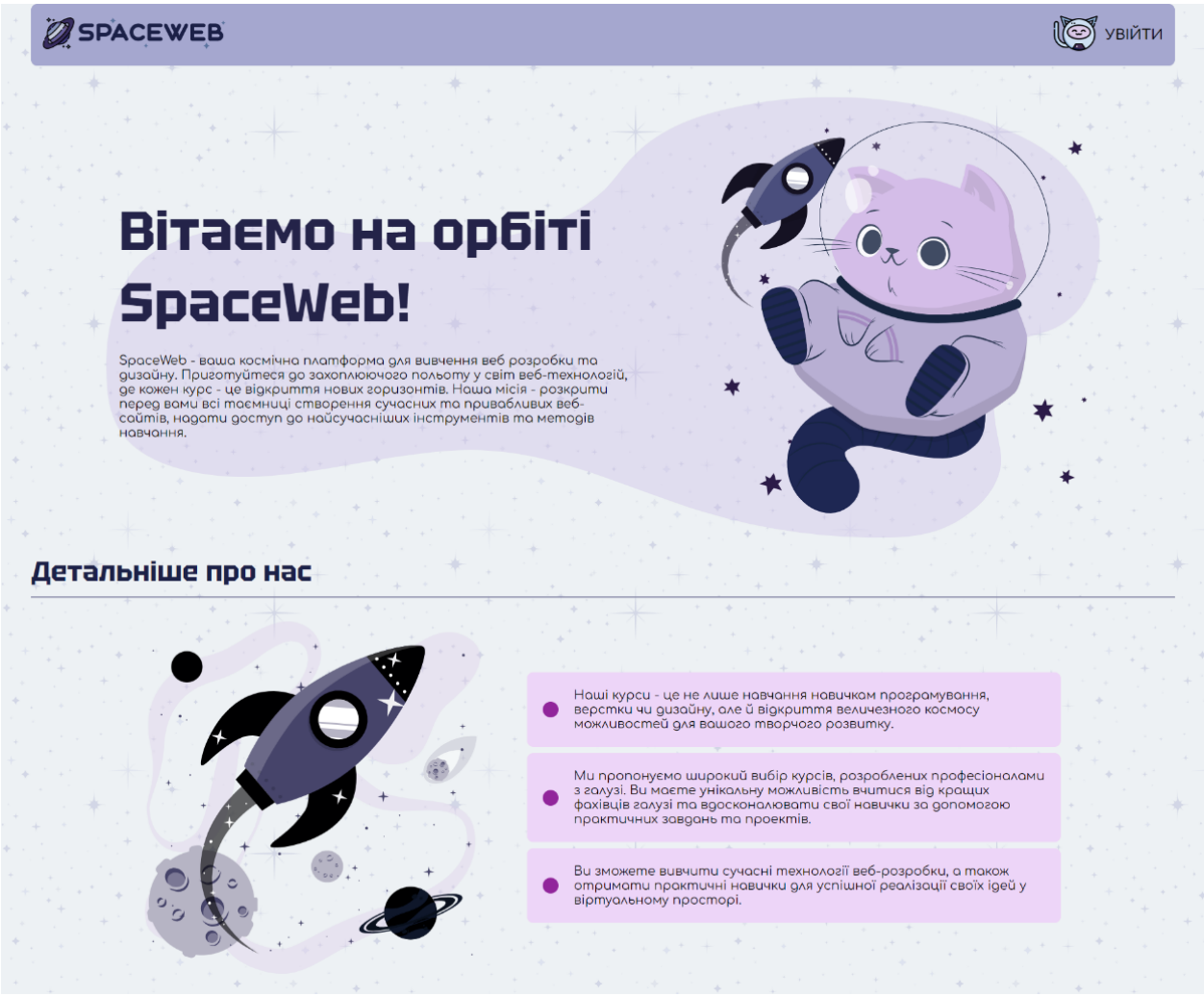


Рисунок 5.1 – Початок головної сторінки платформи

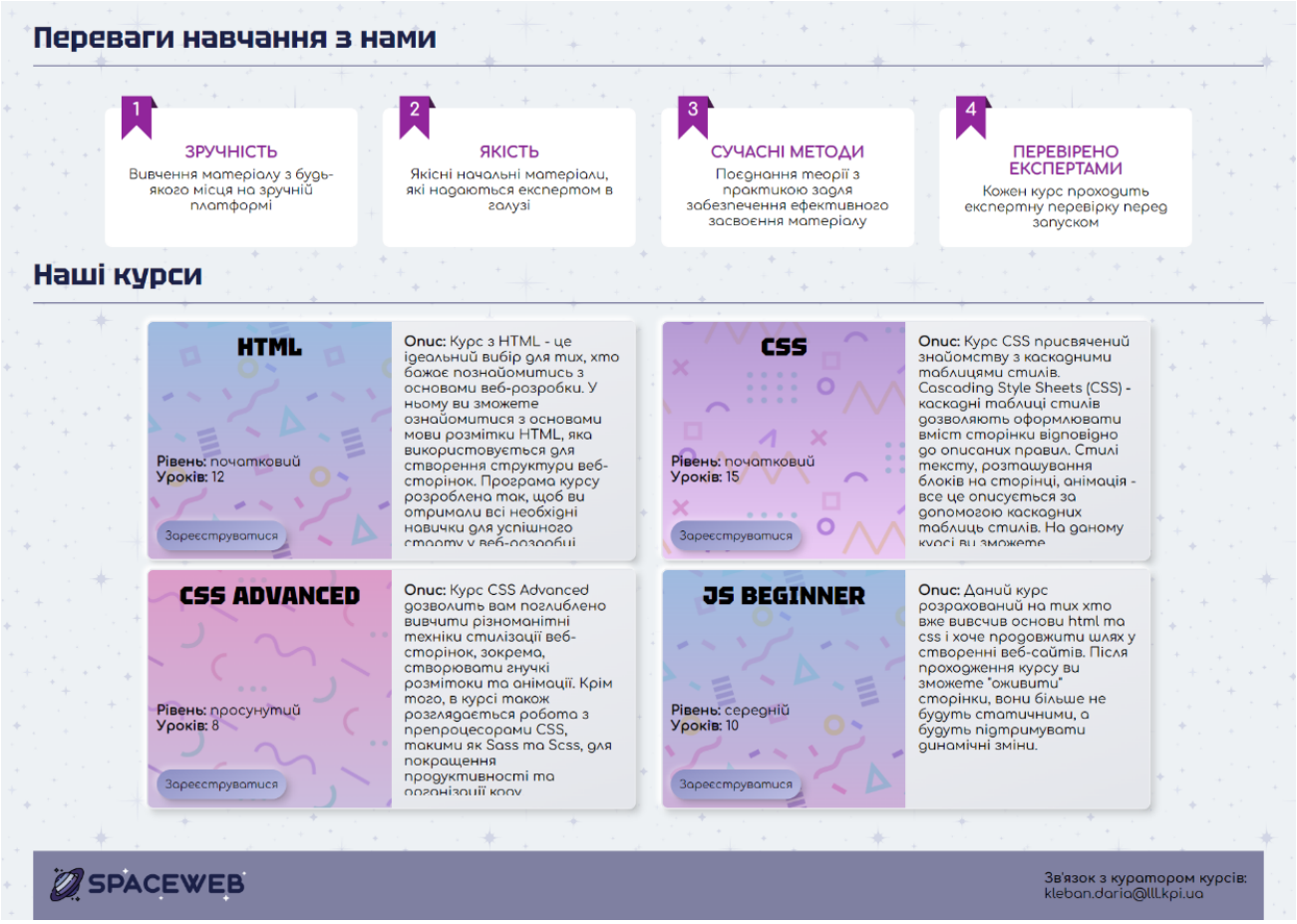


Рисунок 5.2 – Інформація про платформу та перелік курсів головної сторінки платформи

Якщо неавторизований користувач натисне на кнопку «Зареєструватися» він отримає повідомлення про те, що для реєстрації на курс необхідно або бути авторизованим користувачем або ж, у разі відсутності облікового запису, зареєструватися (рис. 5.3).

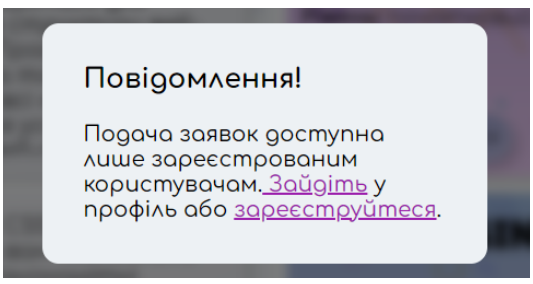
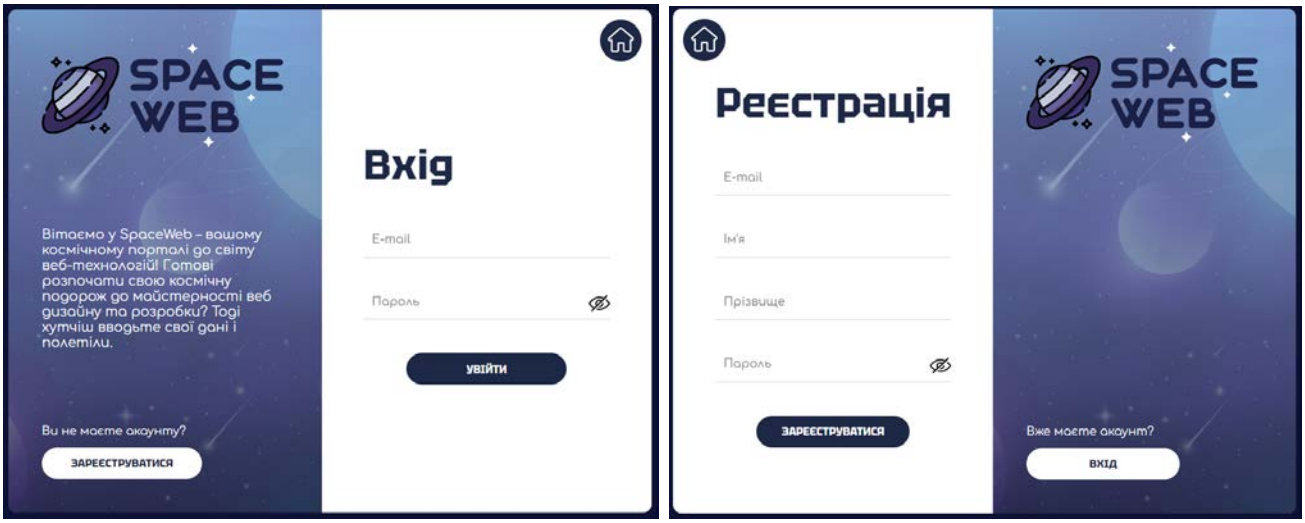


Рисунок 5.3 – Повідомлення при спробі реєстрації на курс неавторизованого користувача

Користувач може увійти до вже створеного акаунта або зареєструватися. Для цього створені спеціальні веб-сторінки з підказками для користувача. На рис. 5.4 показано веб-сторінки для входу та реєстрації у системі. Щоб увійти у систему користувачу потрібно вести свою електронну пошту та пароль. Якщо користувач не зареєстрований, він може перейти на сторінку реєстрації, натиснувши на кнопку «Зареєструватися». Якщо користувач випадково потрапив на цю сторінку і не бажає реєструватися або входити в систему, у правому верхньому куті є кнопка повернення на головну сторінку.



а б

Рисунок 5.4 – Форма входу у систему (а) та форма реєстрації (б)

Платформа використовує систему обробки помилок, яка виводить підказки на екран, якщо користувач зробив щось неправильно, і вказує на можливі проблеми та способи їх вирішення. Якщо користувач вводить дані, які не містяться в базі даних, наприклад, електронну пошту, на яку не зареєстровано акаунт, система виведе спеціальне повідомлення та проінформує про це користувача. На рис. 5.5 показано виведення повідомлення про те, що користувача за введеними даними не знайдено, тобто він не зареєстрований у системі.

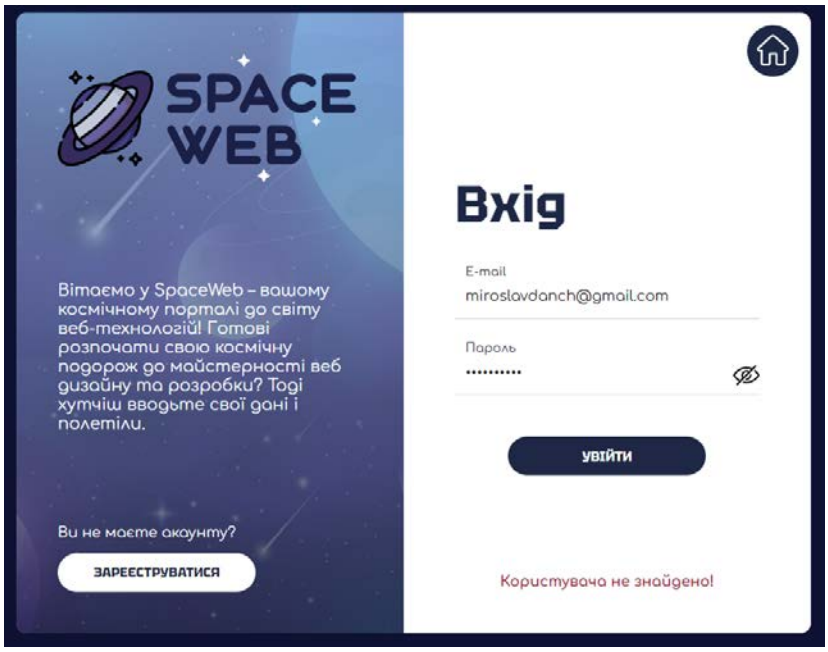


Рисунок 5.5 – Повідомлення про те, що користувач за введеними даними не зареєстрований

Якщо користувач вводить дані свого зареєстрованого акаунта для входу і робить помилку, система виводить відповідне повідомлення, яке зображено на рисунку 5.6.

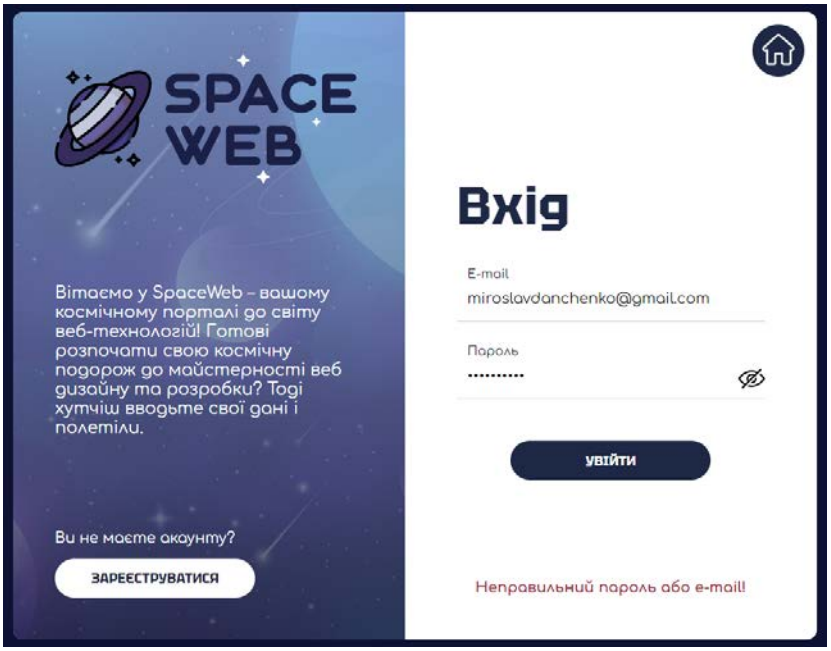
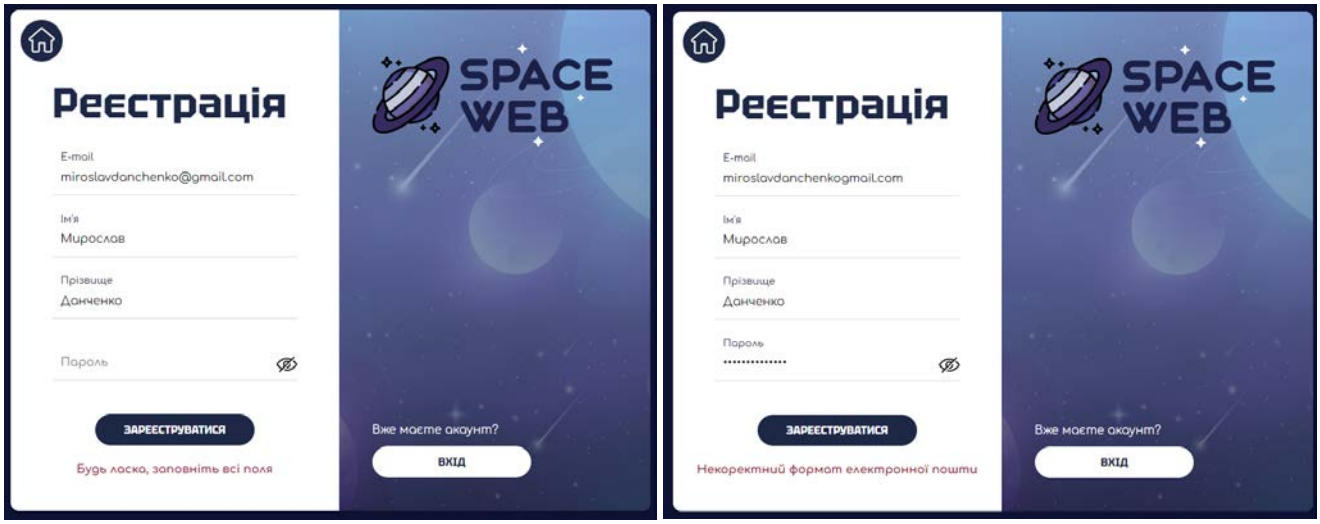


Рисунок 5.6 – Повідомлення про те, що користувач помилився при вводі даних

Для реєстрації необхідно ввести електронну пошту, прізвище, ім'я та пароль. Якщо користувач навмисно не заповнить будь-яке з цих полів, система виведе відповідне повідомлення, вказуючи на відсутні дані, і реєстрація не буде завершена. Також, якщо користувач помилиться при введенні електронної адреси або введе пароль менше чотирьох символів, система повідомить про це (рис. 5.7). Для зручності було додано можливість переглядати пароль, натиснувши на іконку око. Після завершення реєстрації користувач буде перенаправлений на сторінку входу, щоб увійти до свого акаунта. При наявності створеного акаунту користувач може перейти безпосередньо на сторінку входу за допомогою кнопки «Вхід».



а б

Рисунок 5.7 – Обробка помилок при введенні реєстраційних даних: а) відсутні дані в обов'язкових полях; б) некоректний формат електронної пошти.

Якщо користувач вводить електронну пошту, до якої вже прив'язаний акаунт, на сторінці реєстрації, система виведе повідомлення про те, що такий користувач вже зареєстрований і повторна реєстрація неможлива. Відображення цього повідомлення показано на рис. 5.8.

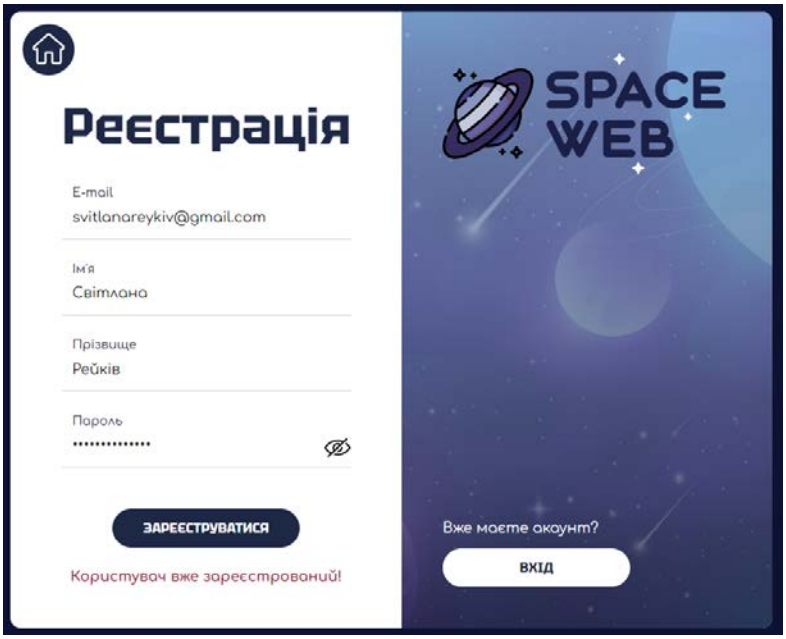


Рисунок 5.8 – Повідомлення про те, що користувач з такою електронною поштою вже зареєстровано

Коли користувач входить у систему, у навігаційному барі відбуваються певні зміни. Замість кнопки «Увійти» відображатися кнопка «Вихід», а поруч з нею розташовується ім’я користувача. Зміни у навігаційному барі показані на рис. 5.9.

Рисунок 5.9 – Навігаційний бар після входу користувача у систему

Авторизований користувач, натиснувши на іконку котика біля свого імені в навігаційному барі, буде перенаправлений до особистого кабінету. В особистому кабінеті користувачу буде доступна бічна панель, де містяться аватар користувача, його ім’я та вкладки «Профіль» і «Курси». За замовчуванням, при вході в особистий кабінет, користувач потрапляє у вкладку «Курси». На рисунку 5.10

зображено особистий кабінет щойно зареєстрованого користувача. Оскільки, даний користувач був нещодавно зареєстрований і ще не має курсів, на сторінці «Мої курси» буде міститися повідомлення про те, що користувач не зареєстрований на жоден курс.

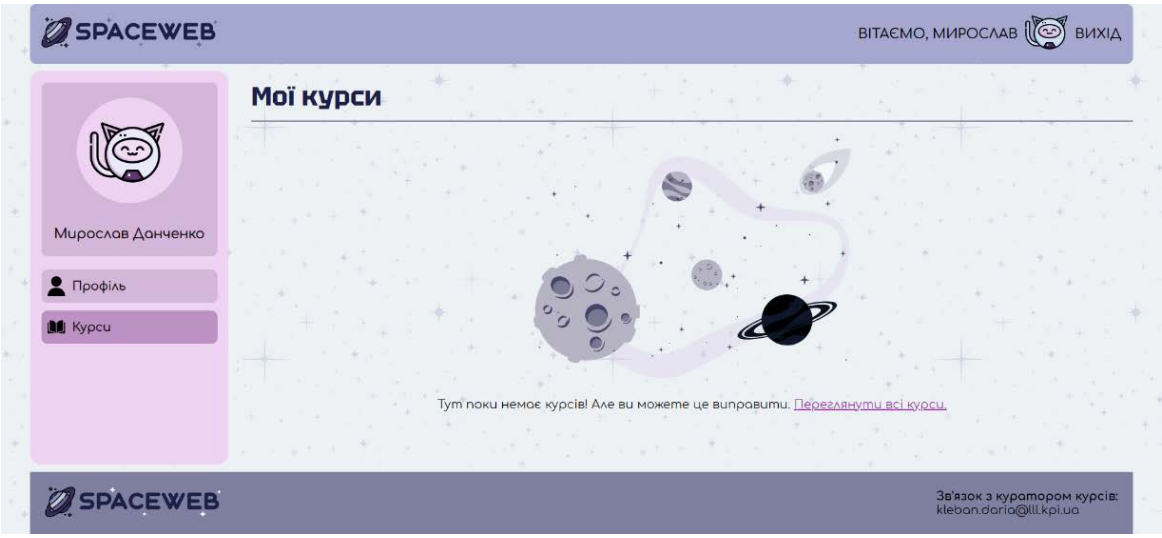


Рисунок 5.10 – Особистий кабінет нового зареєстрованого користувача

Натиснувши на вкладку «Профіль» користувач буде перенаправлений на сторінку свого профілю (рис 5.11). На даній сторінці користувач може переглянути свої дані та змінити їх.

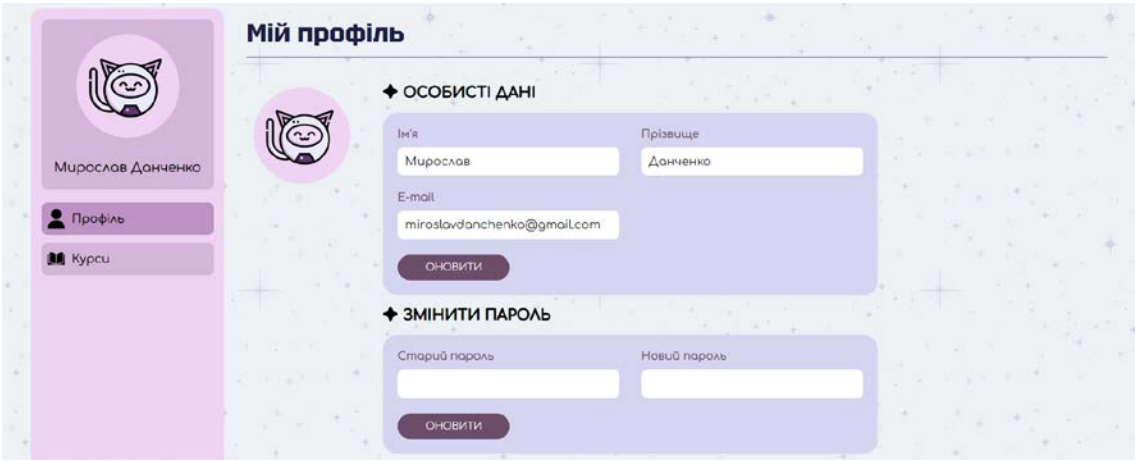


Рисунок 5.11 – Сторінка профілю користувача

Користувач може переглядати та змінювати такі дані: електронну пошту, ім’я, прізвище, фото профілю та пароль. Ці значення містяться в елементі input, тому їх можна легко видалити та ввести нові. Для оновлення фото профілю потрібно навести на аватар і натиснути кнопку «Фото профілю», після чого система запропонує обрати фотографію з пристрою користувача. Щоб зберегти зміни, необхідно натиснути кнопку «Оновити». Лише після цього дані буде змінено і оновлено в базі даних. На рисунках 5.12 – 5.14 показано алгоритм зміни даних профілю користувача.

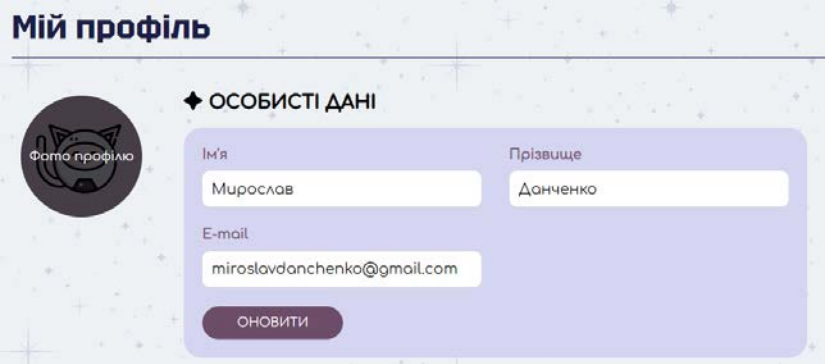


Рисунок 5.12 – Кнопка «Фото профілю»

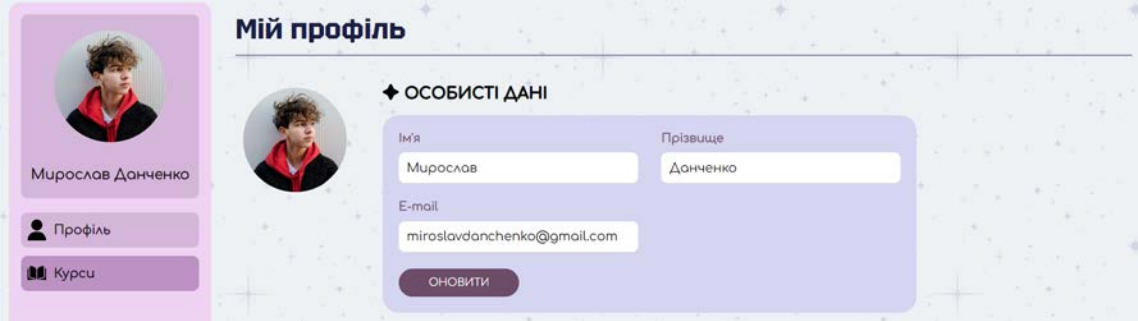


Рисунок 5.13 – Результат заміни фотографії у профілі

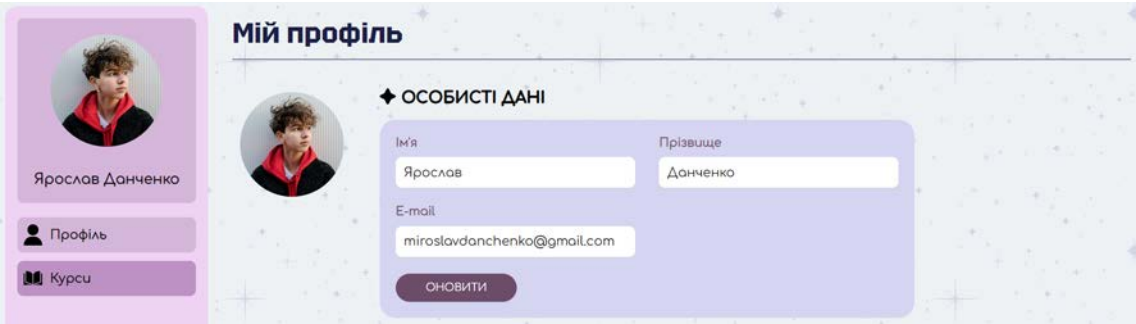


Рисунок 5.14 – Результат заміни імені у профілі

Після авторизації в системі користувач може подавати заявки на курси. Для цього потрібно вибрати курс на головній сторінці та натиснути кнопку «Зареєструватися». Далі система перенаправить користувача до особистого кабінету, де можна переглянути курси, на які користувач зареєстрований, а також подані заявки (рис. 5.15).

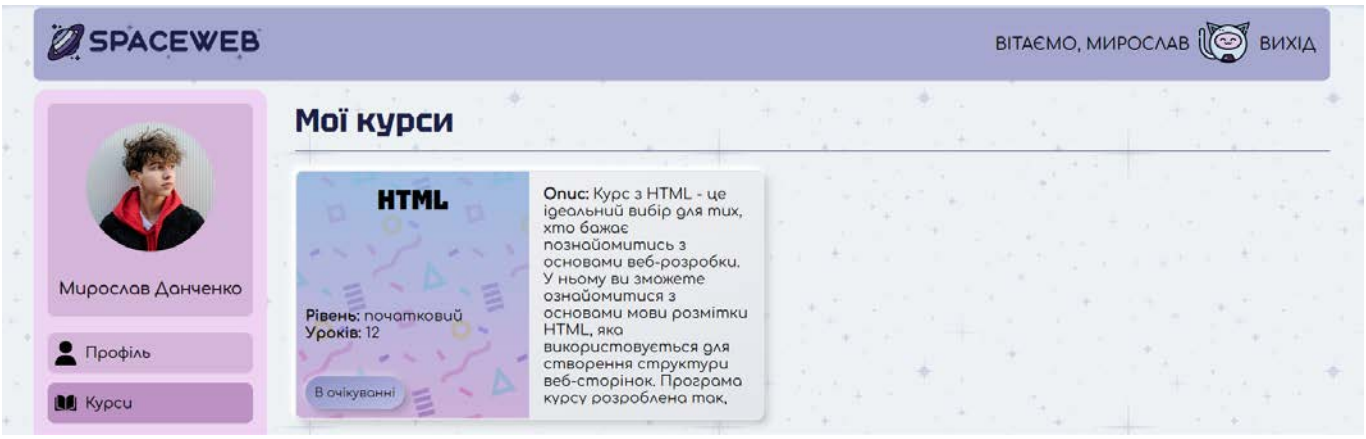


Рисунок 5.15 – Курс зі статусам очікування схвалення від викладача

Якщо користувач натисне на кнопку курсу, який перебуває у стані очікування схвалення від викладача («В очікуванні»), він отримає повідомлення про те, що необхідно зачекати, поки викладач схвалить заявку (рис. 5.16).

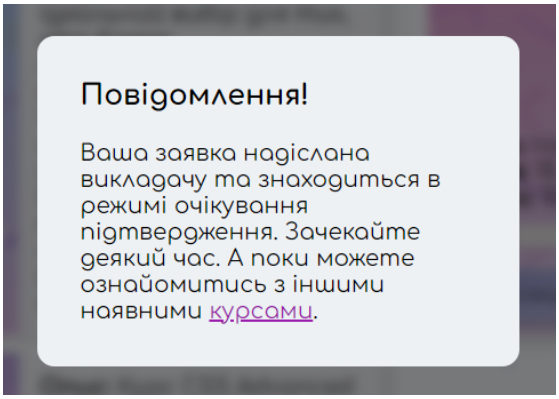


Рисунок 5.16 – Повідомлення про потребу в очікуванні схвалення заявки від викладача

Якщо викладач схвалить проходження курсу, то студент отримує доступ до вмісту курсу та кнопка з надписом «В очікуванні» зміниться на кнопку «Переглянути курс». Також на картці курсу з’являться поля із загальною оцінкою за курс та прогрес бар в якому буде відображатися прогрес проходження курсу (рис. 5.17).

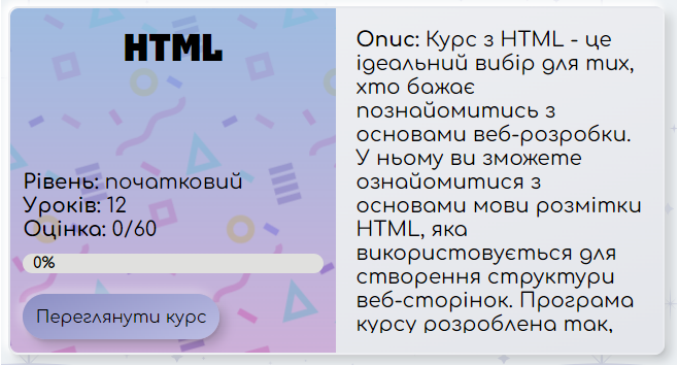


Рисунок 5.17 – Змінена картка курсу після схвалення заявки викладачем

На рисунку 5.18 наведено особистий кабінет студента, який зареєстрований на декілька курсів та деякі з них знаходяться в процесі проходження.

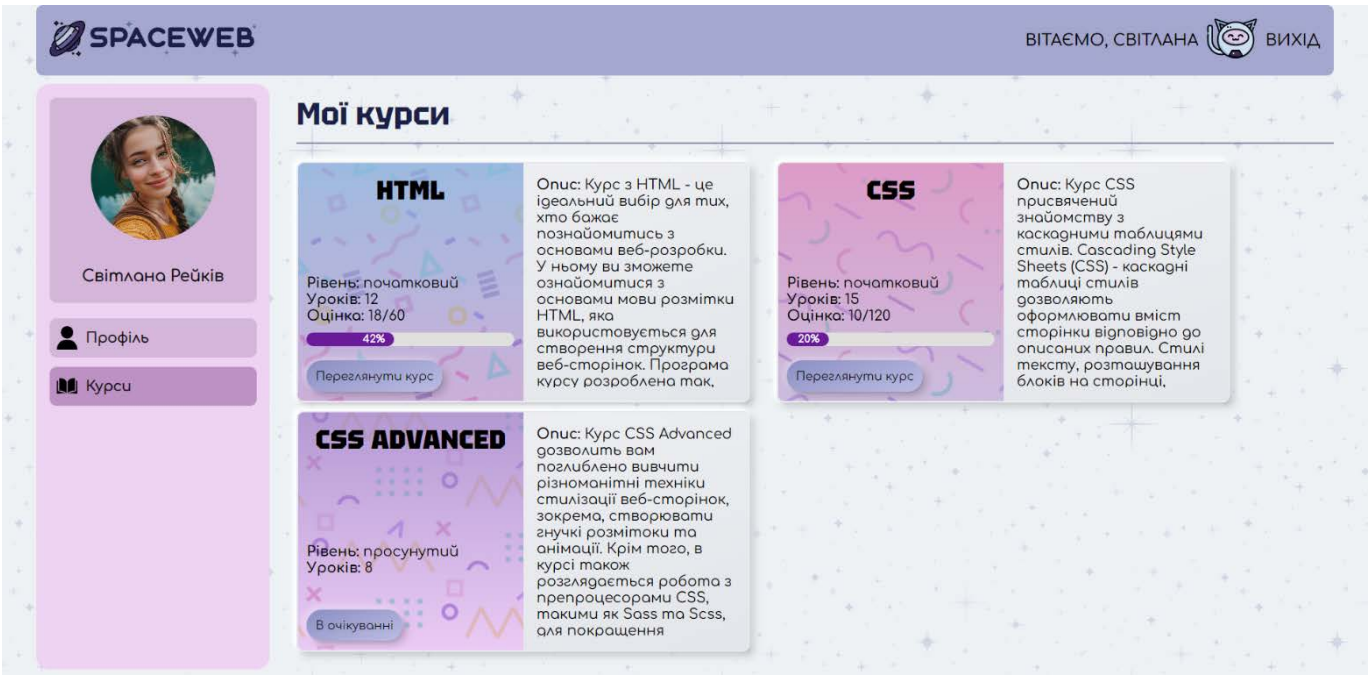


Рисунок 5.18 – Сторінка курсів, які знаходяться на етапі проходження студентом

Користувач може переглядати вміст курсів, на які він зареєстрований і заявка на них схвалена. Для перегляду вмісту необхідно натиснути на кнопку «Переглянути курс», після чого користувача буде перенаправлено на сторінку з вмістом курсу (рис. 5.19). Якщо до уроку міститься домашнє завдання, то на картці уроку буде відображатися кнопка зі знаком шпильки, натиснувши на котру користувач може перейти до вмісту домашнього завдання. Також на картці кожного уроку містяться чекбокси, які дозволяють відмічати користувачу пройдені уроки.

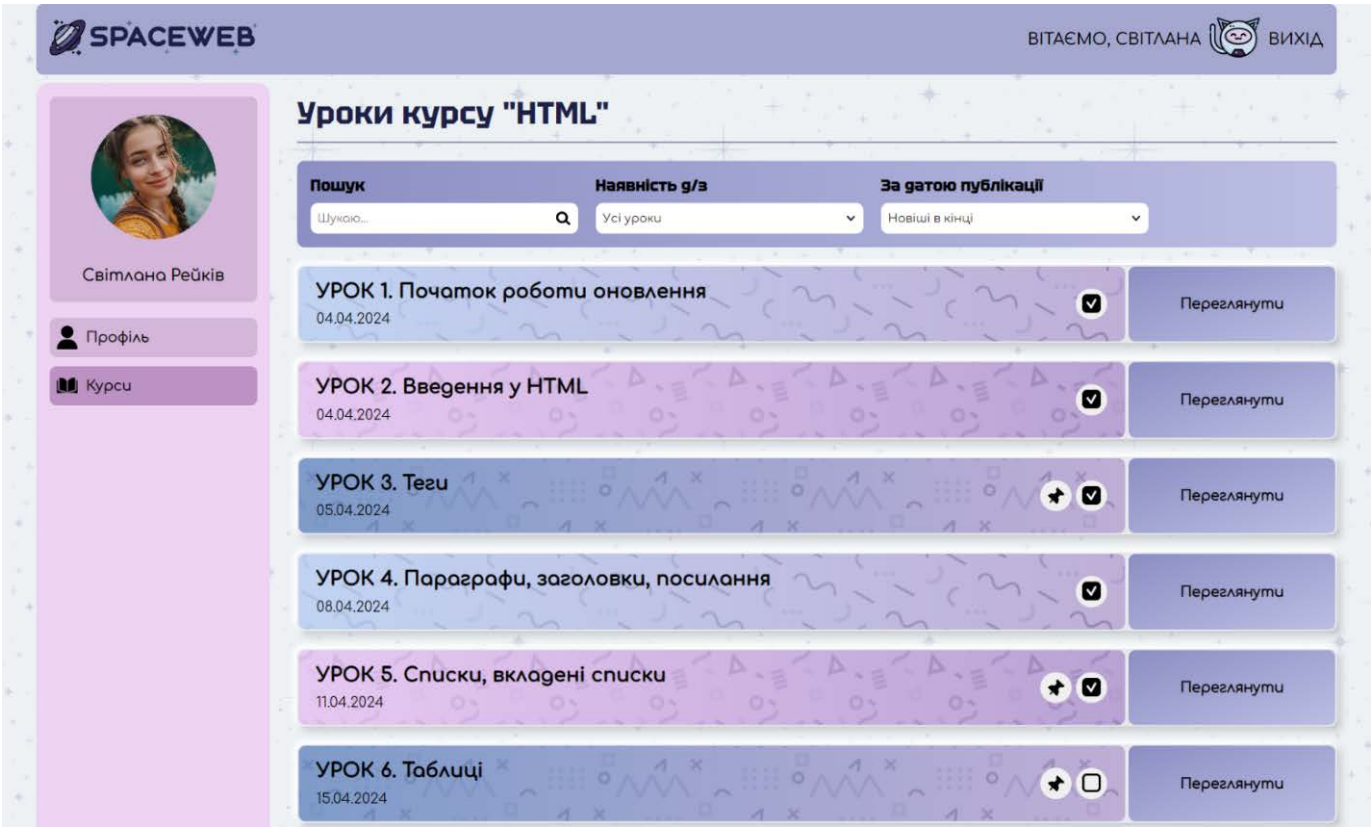


Рисунок 5.19 – Перелік уроків курсу «HTML»

Для користувачів доступні різноманітні фільтри для вибору та пошуку потрібних уроків. Серед цих фільтрів - пошук, наявність д/з та фільтрація за датою публікації. Пошук дозволяє знаходити уроки за їх назвою та номером. Демонстрацію роботи пошуку можна побачити на рисунку 5.20.

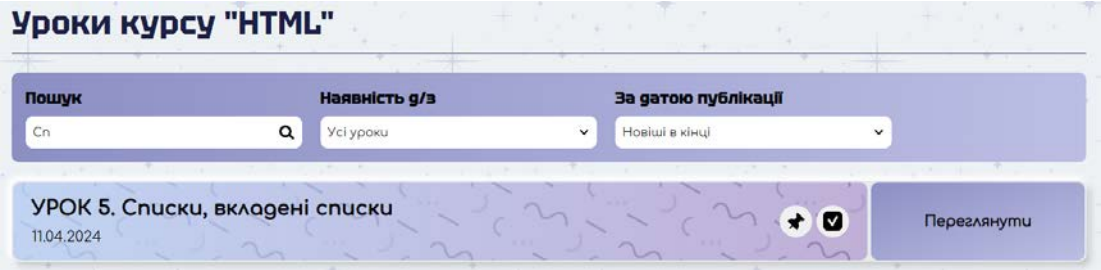


Рисунок 5.20 – Результати за пошуковим запитом користувача

Фільтр «Наявність д/з» дозволяє обрати уроки за наступними критеріями: урок з домашнім завданням, урок без домашнього завдання, усі уроки. Критерії фільтру та результати фільтрації можна бачити на рисунках 5.21 – 5.22.

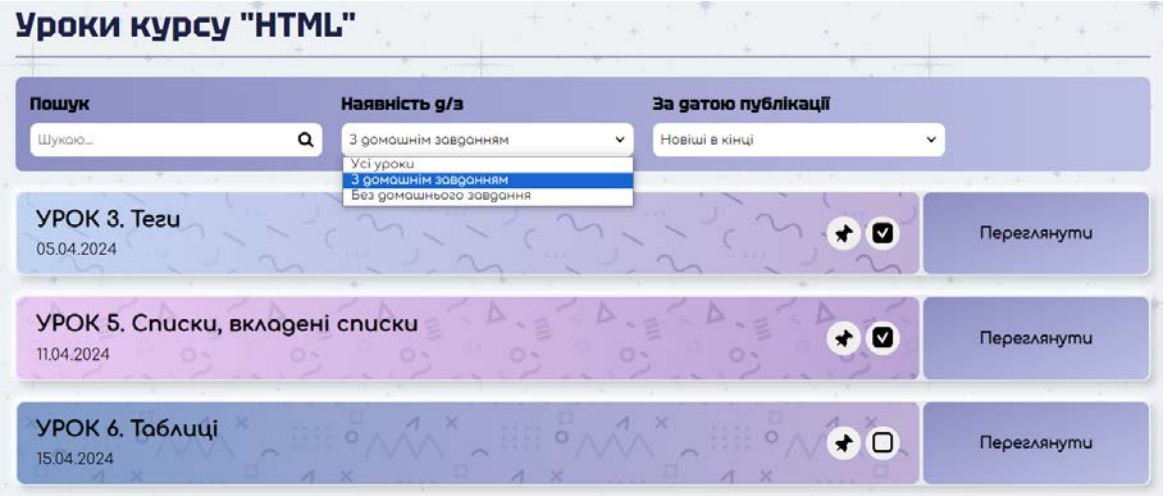


Рисунок 5.21 – Критерії фільтру «Наявність д/з»

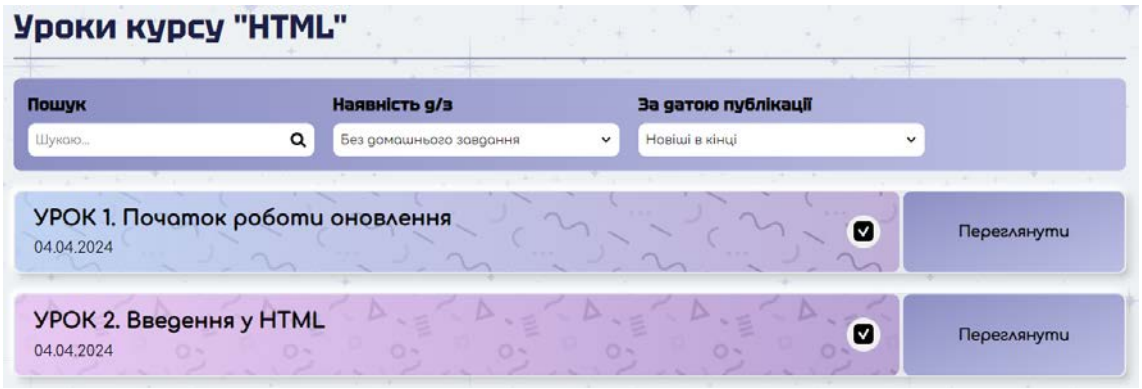


Рисунок 5.22 – Результати фільтрації за критерієм «Без домашнього завдання»

Користувачам також доступний фільтр який відсортовує уроки за новизною додавання на платформу. Його роботу можна бачити на рисунку 5.23.

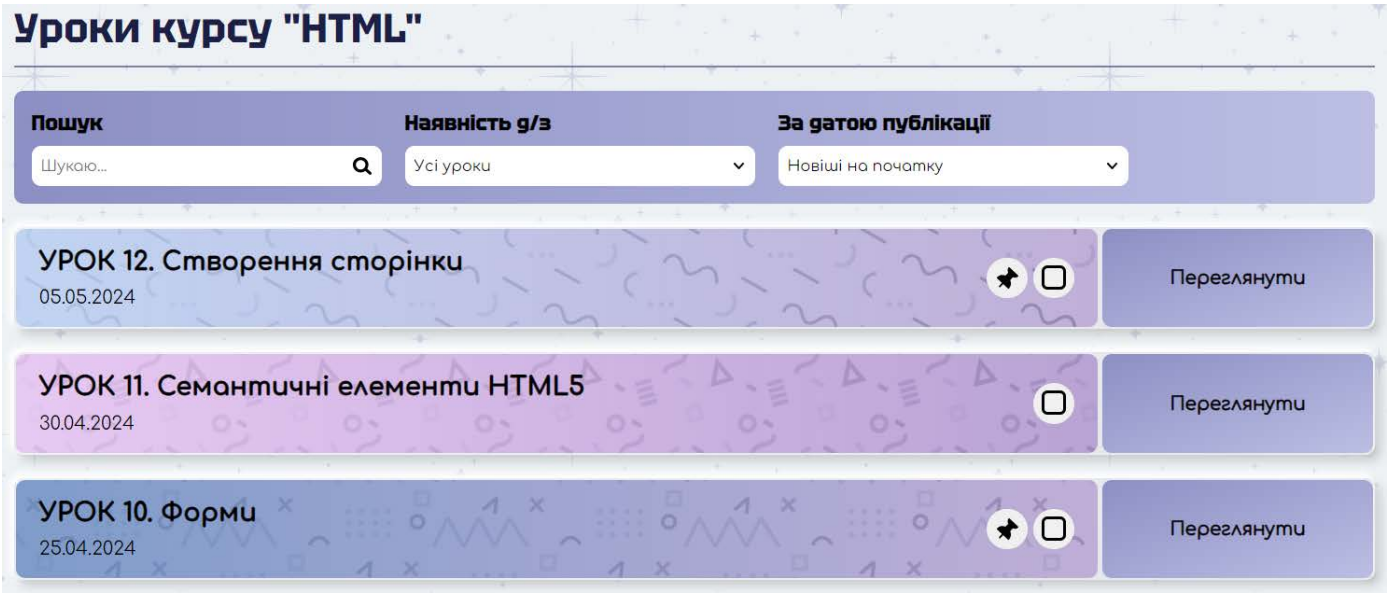


Рисунок 5.23 – Результати фільтрації за критерієм «Новіші на початку»

Користувачам доступна можливість використання декількох фільтрів одночасно. Демонстрація спільного використання фільтрів на рисунку 5.24.

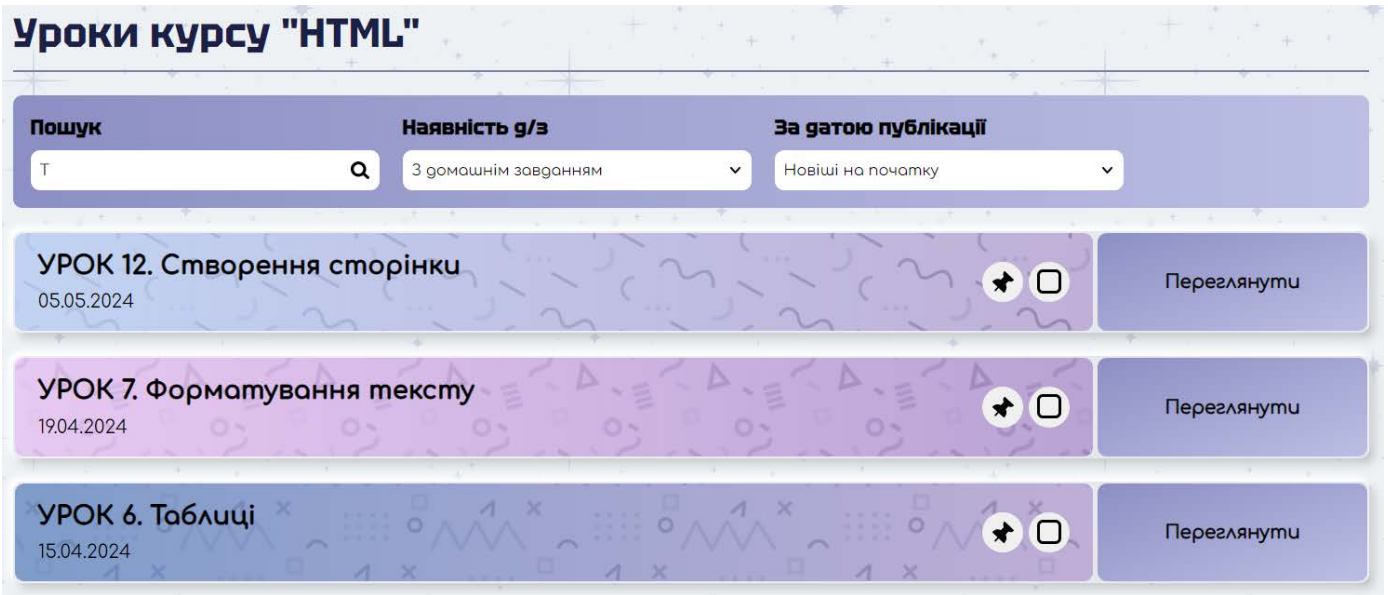


Рисунок 5.24 – Результати роботи мультифільтрації

Користувач може переглядати вміст кожного уроку. Кожен укрок може містити текстові та відео матеріали. Сторінка одного з уроків продемонстрована на рисунку 5.25.

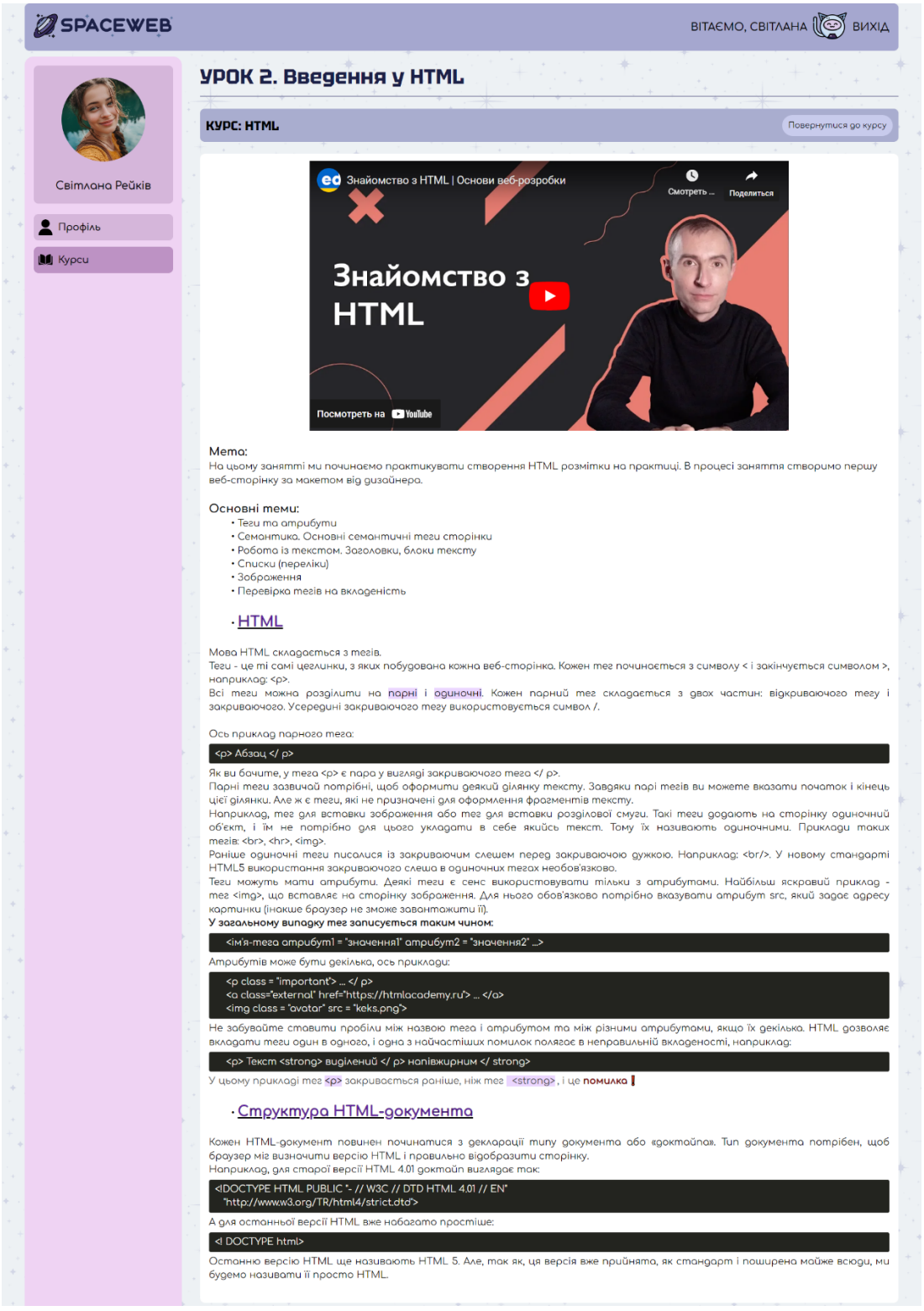


Рисунок 5.25 – Сторінка уроку курсу

Якщо до уроку немає домашнього завдання, то картка з домашнім завданням не буде відображена, це можна бачити на рисунку 5.25, якщо ж домашнє завдання є – картка з домашнім завданням буде міститися в кінці уроку, продемонстровано на рисунку 5.26.

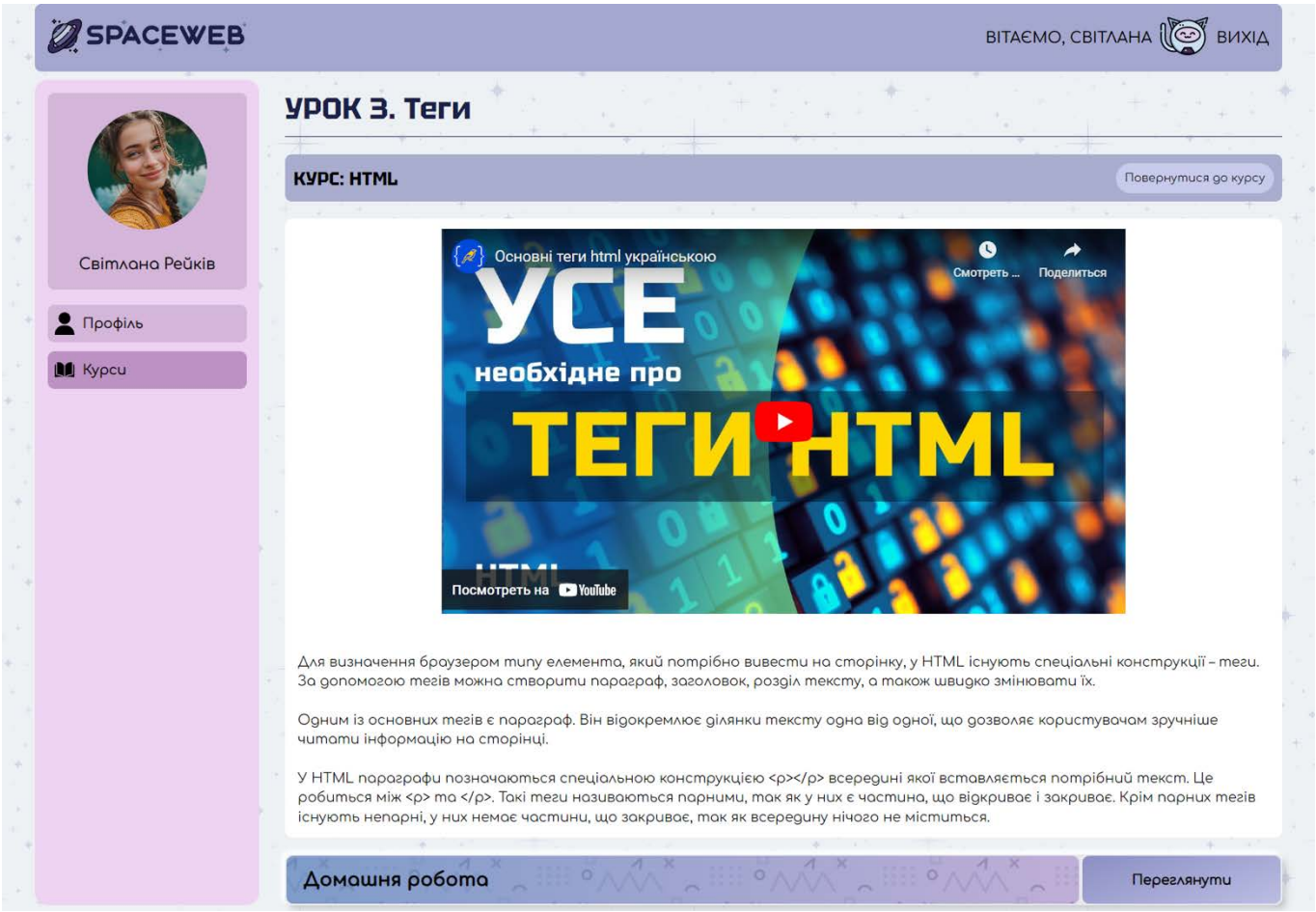


Рисунок 5.25 – Сторінка уроку курсу з присутньою домашньою роботою

Натиснувши на кнопку на картці домашньої роботи «Переглянути», користувач буде перенаправлений на сторінку домашнього завдання. На цій сторінці містить тест домашнього завдання, можливість надіслати або завантажити надіслані файли, переглянути оцінки, написаим коментарі. На рисунку 5.26 продемонстровано сторінку виконаного домашнього завдання до уроку «Теги».

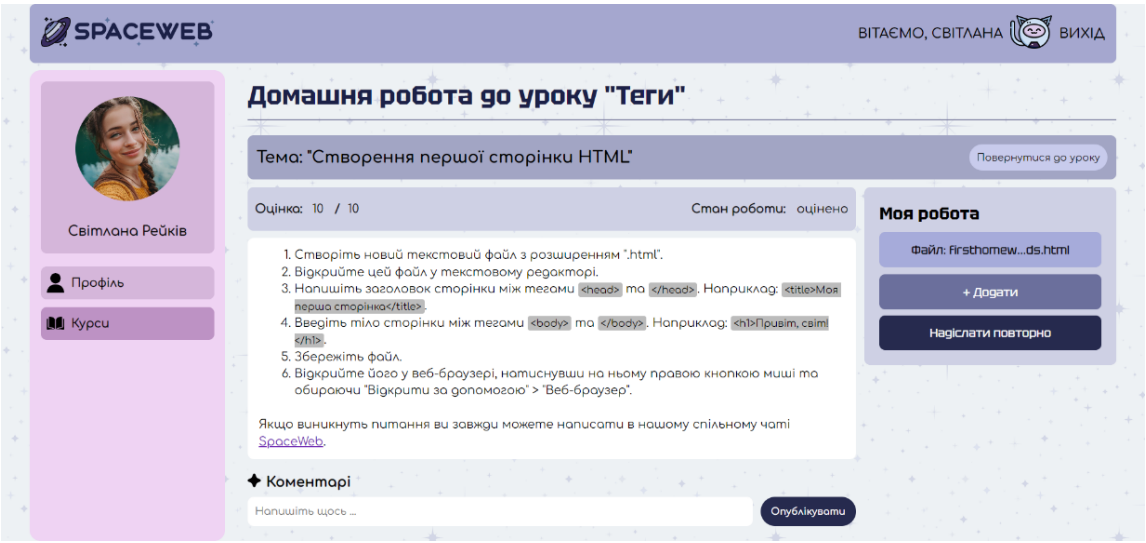


Рисунок 5.26 – Виконане та оцінене домашнє завдання студента

Домашня робота може набувати декількох станів:

- Назначена: робота призначена студенту, але ще не надіслана та не оцінена.
- Здана: робота надіслана на перевірку, але ще не оцінена.
- Оцінена: робота перевірена викладачем, оцінка за неї виставлена.

На рисунку 5.27 ми можемо бачити назначену домашню роботу студенту.

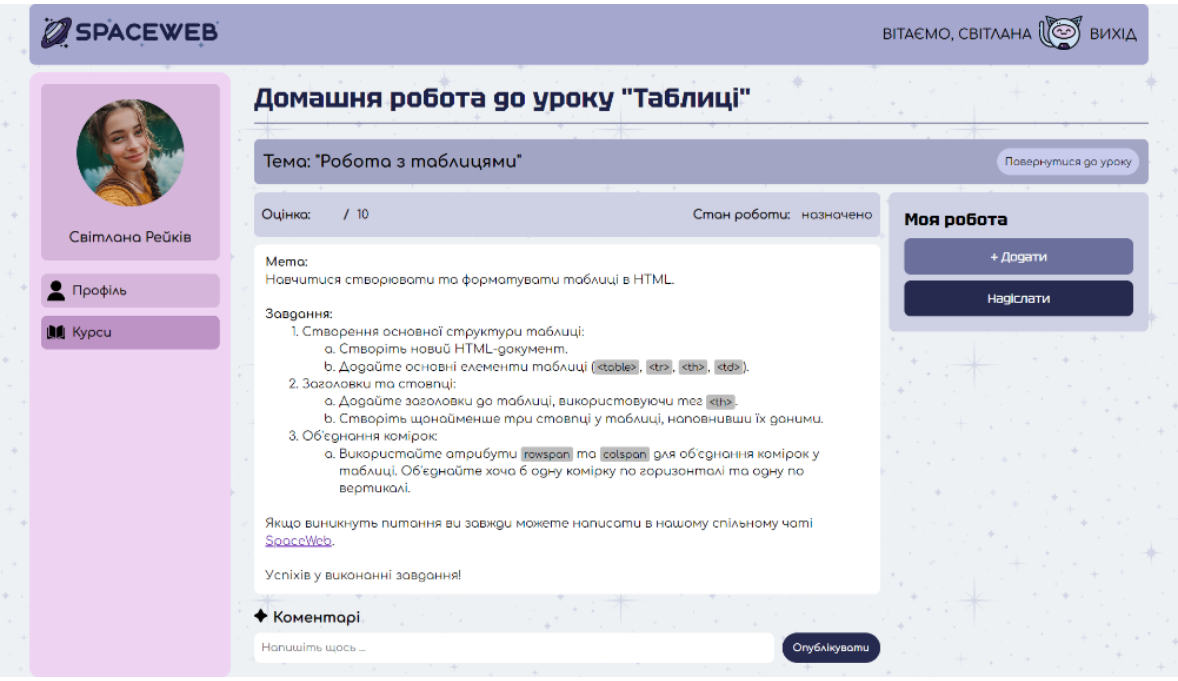


Рисунок 5.27 – Назначена для виконання домашня робота

Студент має можливість прикріпити файл для здачі домашньої роботи. Після натискання кнопки «Додати», системою буде запропоновано обрати файл, який міститься на пристрої користувача. Після того, як користувач обере відповідний файл, назва файлу буде відображена та з’явиться кнопка «Відкріпити», яка дозволяє відкріпити поточний файл і прикріпити новий. Результат прикріплення файлу продемонстровано на рисунку 5.28.

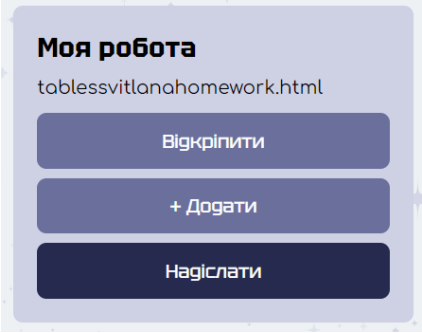


Рисунок 5.28 – Результат прикріплення файлу

Після натискання кнопки «Надіслати», робота буде відправлена на перевірку викладачу і її стан зміниться на здано (рис. 5.29).

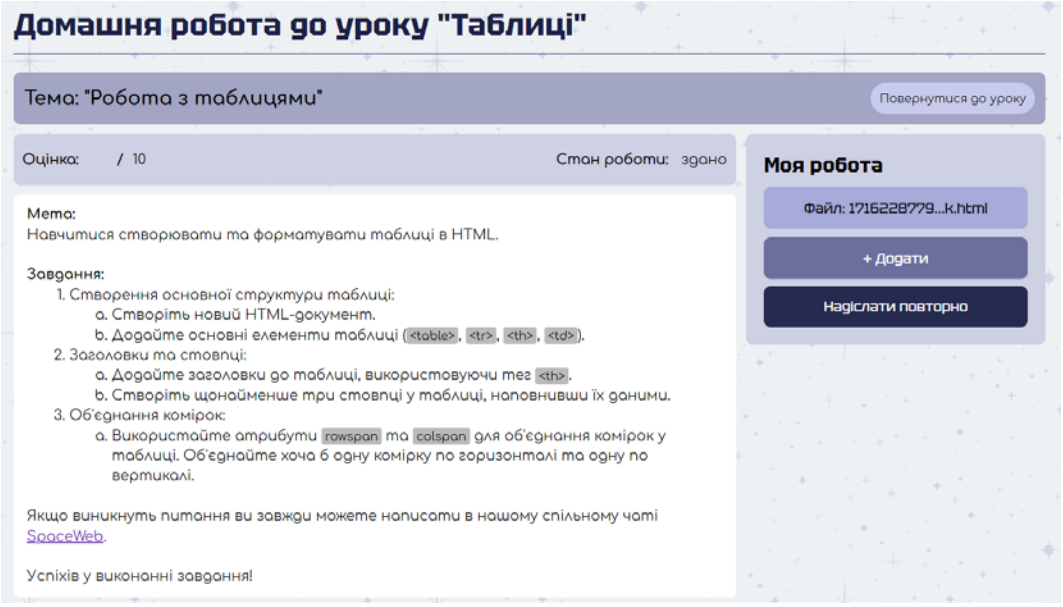


Рисунок 5.29 – Надіслана на перевірку домашня робота

Розглянемо доступні функціональні можливості для викладача. В особистому кабінеті викладача міститься перелік усіх курсів, які опубліковані на платформі (рис. 5.30). Викладач має можливість редагувати та видаляти ці курси.

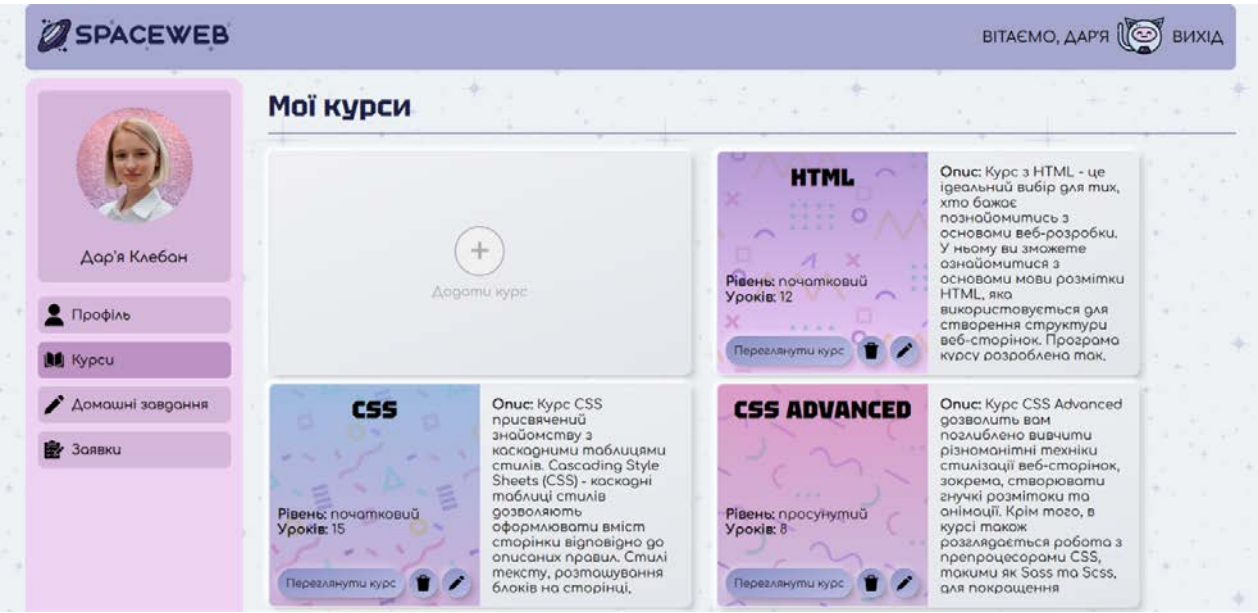


Рисунок 5.30 – Особистий кабінет викладача

При натисканні на кнопку з іконкою олівця викладачу відображається сторінка редагування курсу (рис. 5.30). На даній сторінці міститься поточні назва, рівень складності та назва курсу.

Рисунок 5.31 – Сторінка редагування курсу

Після внесенні змін у назву та їх збереження можна спостерігати результат зображений на рисунку 5.31.

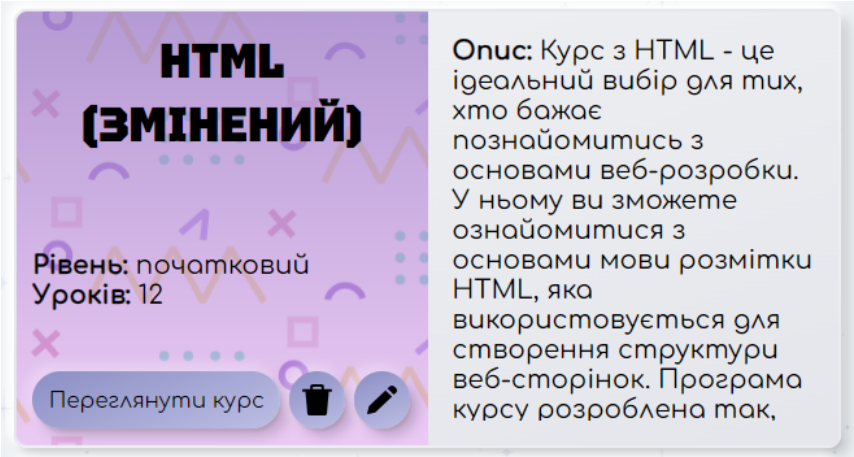


Рисунок 5.31 – Вигляд картки курсу після змін

Викладач має можливість додавати нові курси натиснувши на картку «Додати курс». Після натискання викладача перенесе до сторінки, де він може внести інформацію про курс, а саме: назва, рівень складності та опис (рис. 5.32).

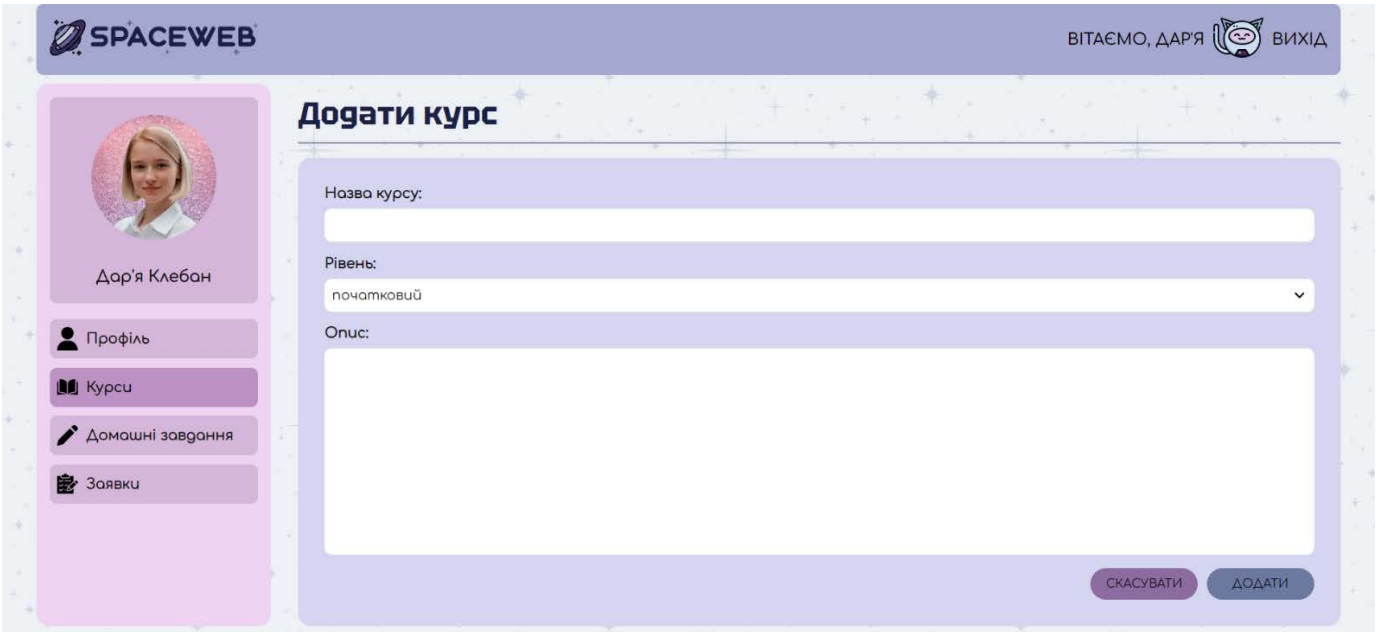


Рисунок 5.32 – Сторінка додавання курсу

Викладач, як і студент, може переглядати вміст курсу та користуватися фільтрацією. Викладачу, на відміну від студентів, доступний додатковий функціонал, який дозволяє створювати уроки та домашні завдання, видаляти та редагувати уроки. Для забезпечення даних функціональних можливостей на карточці уроку містяться відповідні кнопки (рис. 5.33).

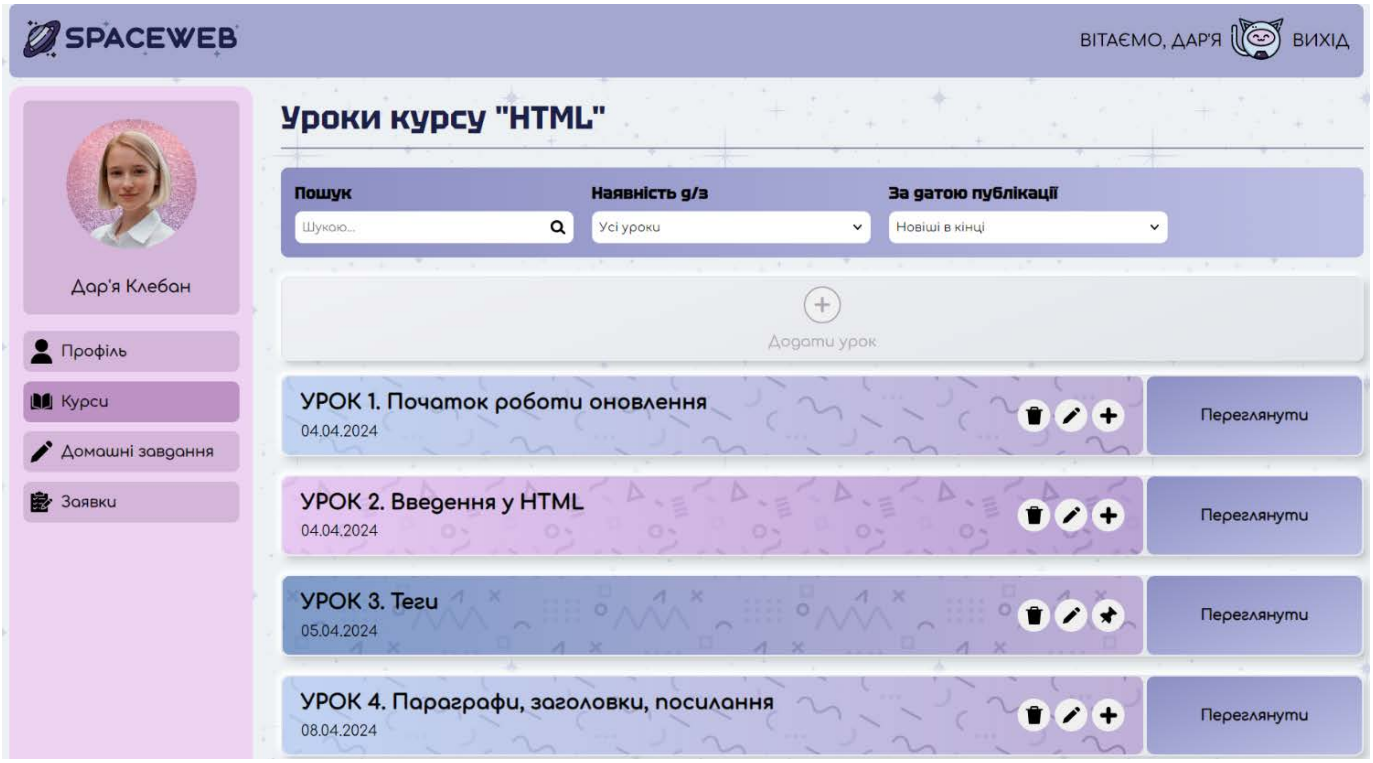


Рисунок 5.33 – Вигляд вмісту курсу для викладача

При натисканні кнопки «Додати урок» викладача перенаправить до сторінки, де він може внести інформацію про урок. Викладачу доступний текстовий редактор, який підтримує багато накреслень стилей тексту. За допомогою редактора викладач може використовувати різноманітні елементи для редагування тексту: виділяти текст курсивом, робити його напівжирним, встановлювати список, додавати гіперпосилання тощо. Приклад роботи редактора продемонстровано на рисунку 5.34.

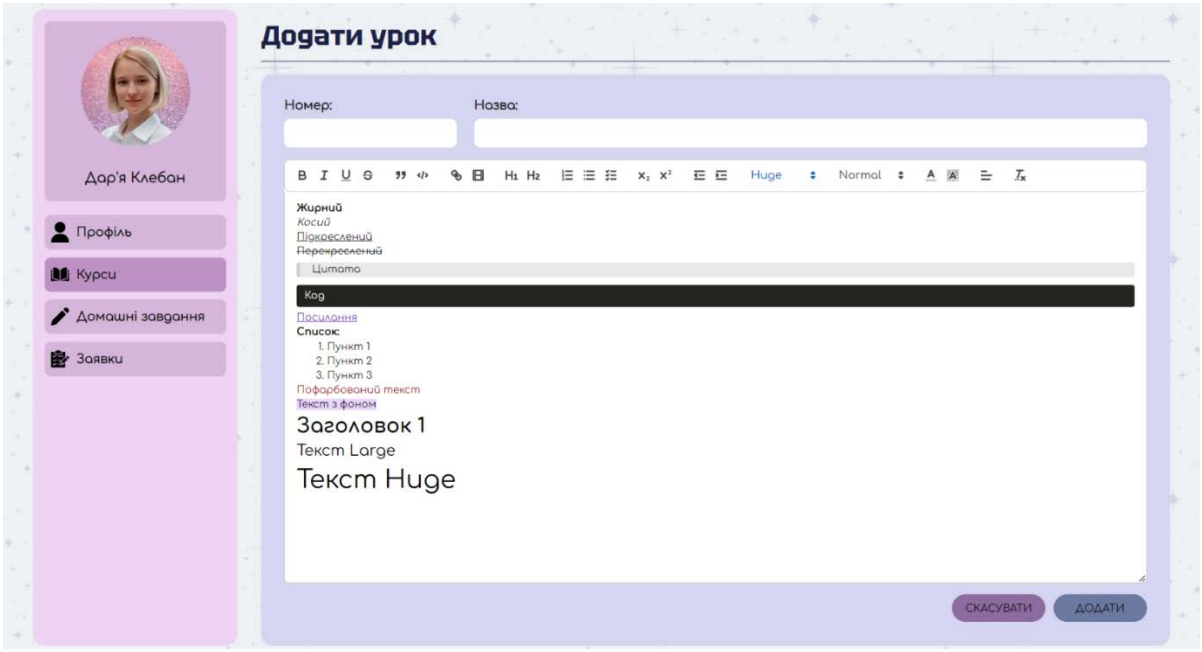


Рисунок 5.34 – Сторінка створення уроку

Так само як і редагування курсу, викладачу доступне редагування вмісту уроку (рис. 5.35).

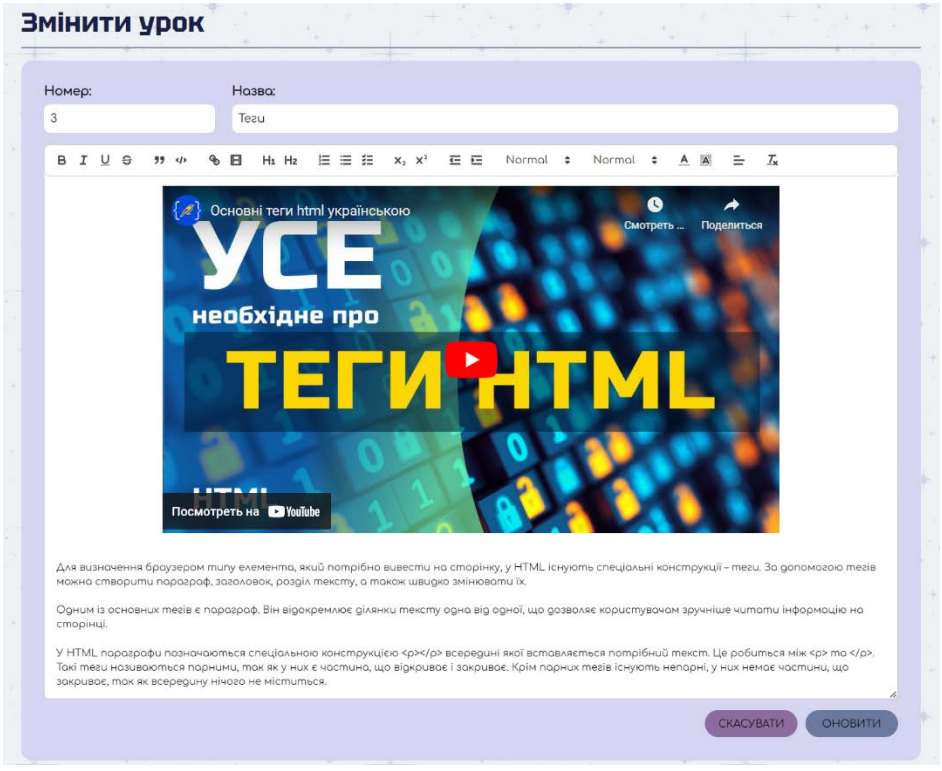


Рисунок 5.35 – Сторінка редагування уроку

До кожного уроку можна додати домашню роботи. Якщо урок наразі не містить домашньої робити, то в кінці сторінки уроку буде розміщуватись кнопка для додавання домашньої роботи (рис. 5.36). Якщо урок вже містить домашню роботу, то знизу уроку буде міститися картка з домашньою роботою і кнопки редагування та видалення (рис. 5.37).

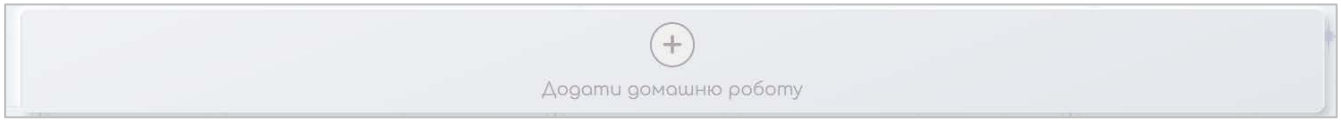


Рисунок 5.36 – Кнопка додавання нової домашньої роботи

Рисунок 5.37 – Картка домашньої роботи

На рисунку 5.38 зображено вікно додавання нового домашнього завдання, яке працює аналогічно як додавання уроку.

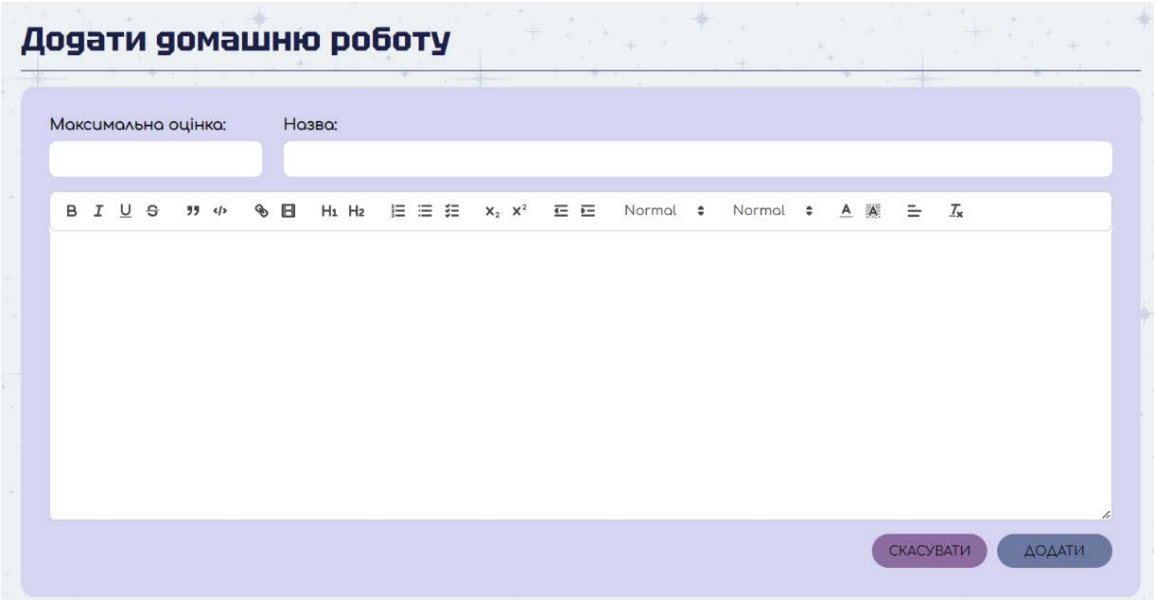


Рисунок 5.38 – Вікно додавання домашнього завдання

На рисунку 5.39 зображено вікно перегляду домашнього завдання.

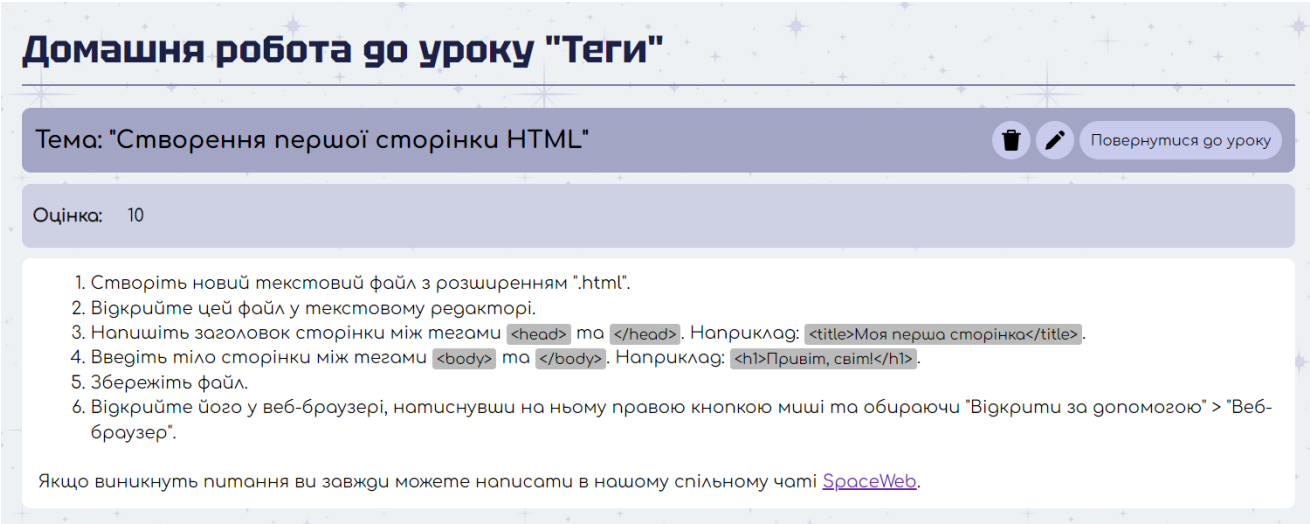


Рисунок 5.39 – Вікно додавання домашнього завдання

На рисунку 5.40 зображено вікно зміни домашнього завдання.

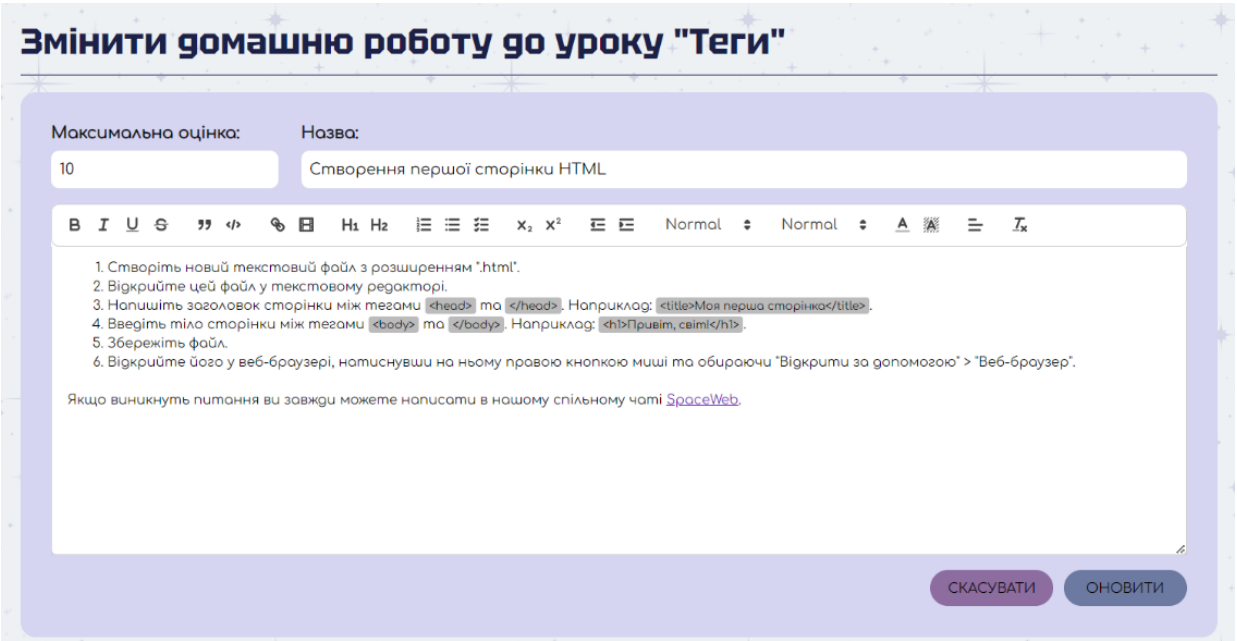


Рисунок 5.40 – Вікно редагування домашнього завдання

Викладач може переглядати заявки на курси від студентів та приймати або відхиляти їх (рис. 5.41).

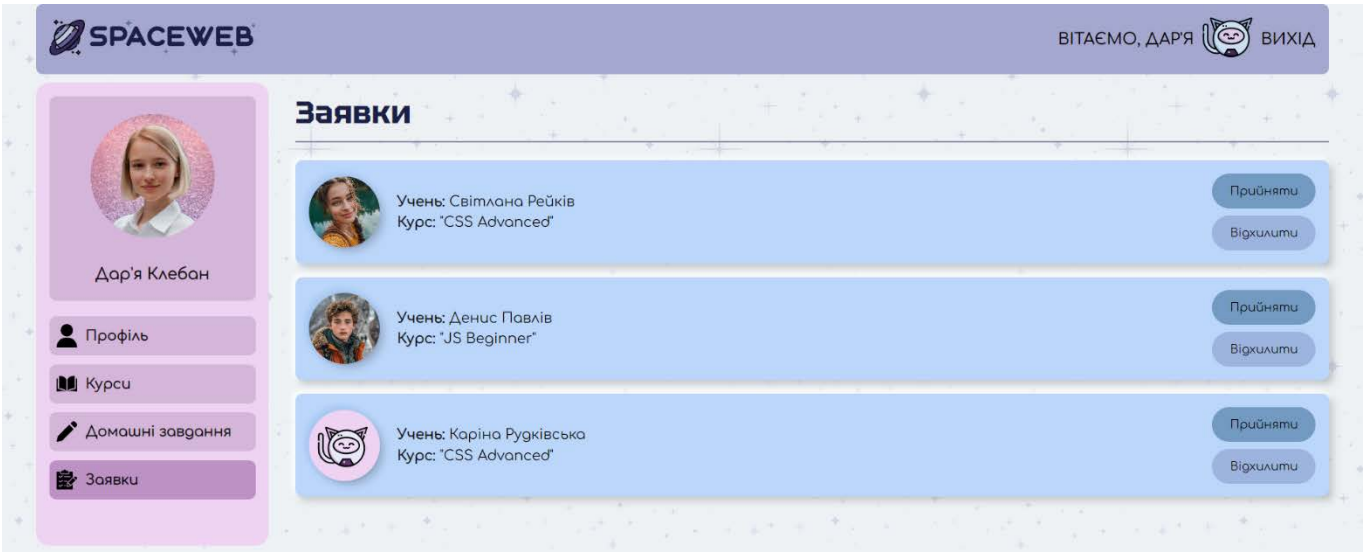


Рисунок 5.41 – Заявки реєстрації на курси від студентів

У вкладці «Домашні завдання» викладач може переглядати усі надіслані роботи студентами (рис. 5.42). Також викладачу доступна зручна фільтрація робіт: пошук по імені або назві уроку, за наявністю перевірки роботи, за датою надсилання роботи та за назвою курсу. Перевірені роботи виділені окремим кольором, щоб навіть без використання фільтрів було зрозуміло, які роботи є перевіреними.

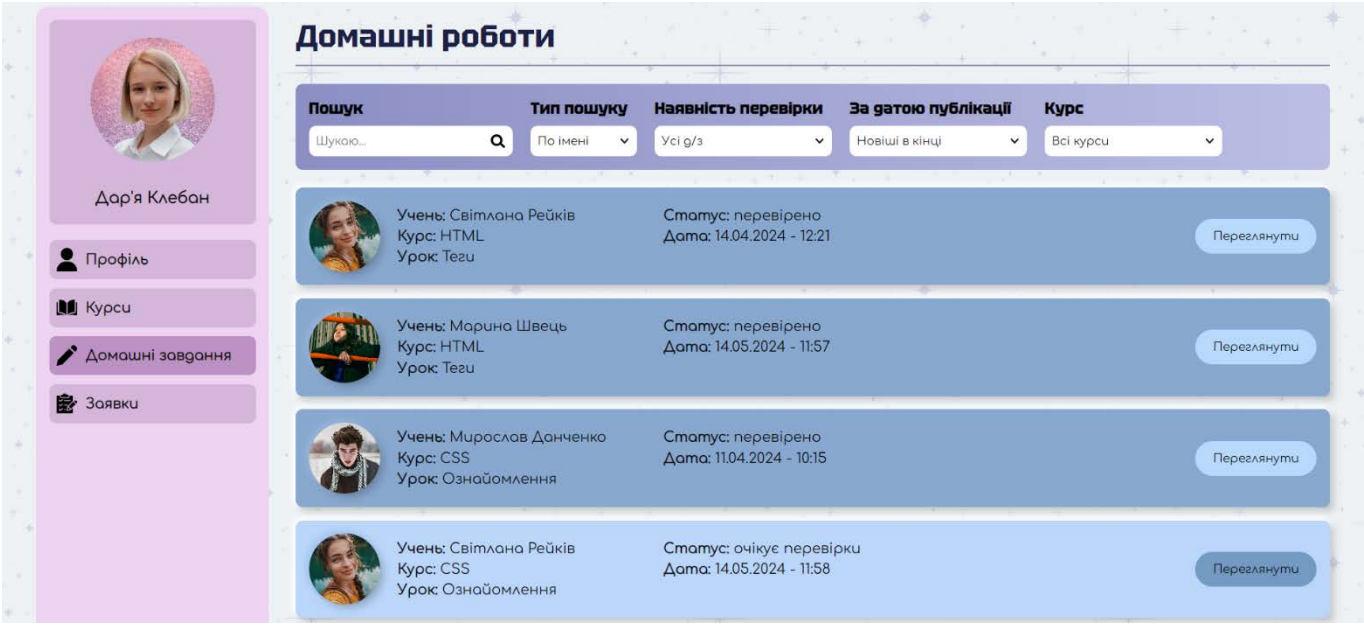


Рисунок 5.42 – Перегляд домашніх робіт від студентів

Викладач може перевіряти конкретну роботу перейшовши до її перегляду натиснувши на кнопку «Переглянути». На сторінці конкретної роботи викладач може бачити наступну інформацію: ім’я учня, назва курсу, назва та номер уроку, дата здачі роботи, завдання, файл з роботою учня. Викладач може завантажити файл собі на робочий пристрій та після перегляду і перевірки виставити оцінку у відповідне поле та оцінити роботу. На рисунку 5.43 зображено перегляд конкретної зданої роботи.

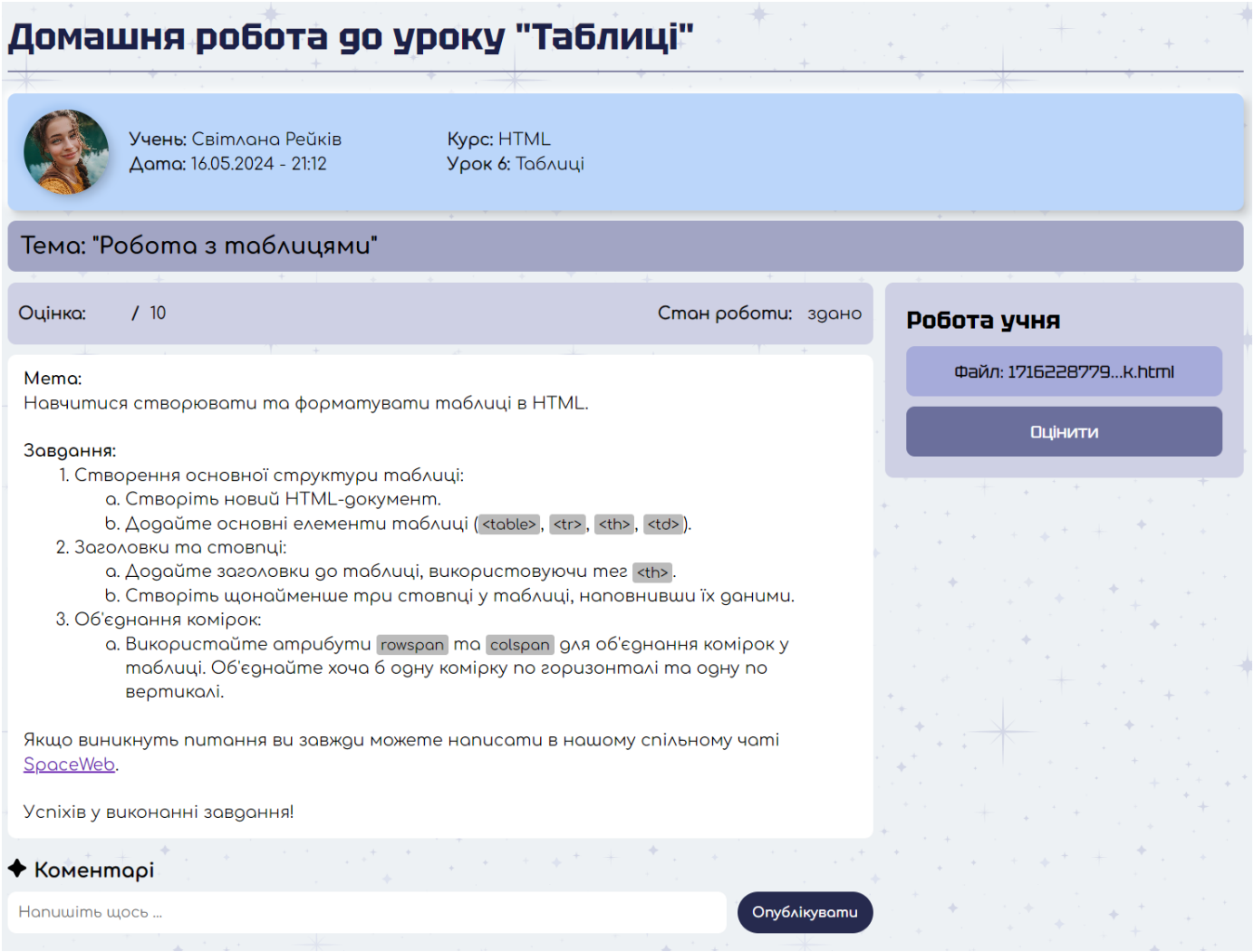


Рисунок 5.43 – Отримана домашня робота від студента

На рисунку 5.44 зображено вже оцінену роботу.

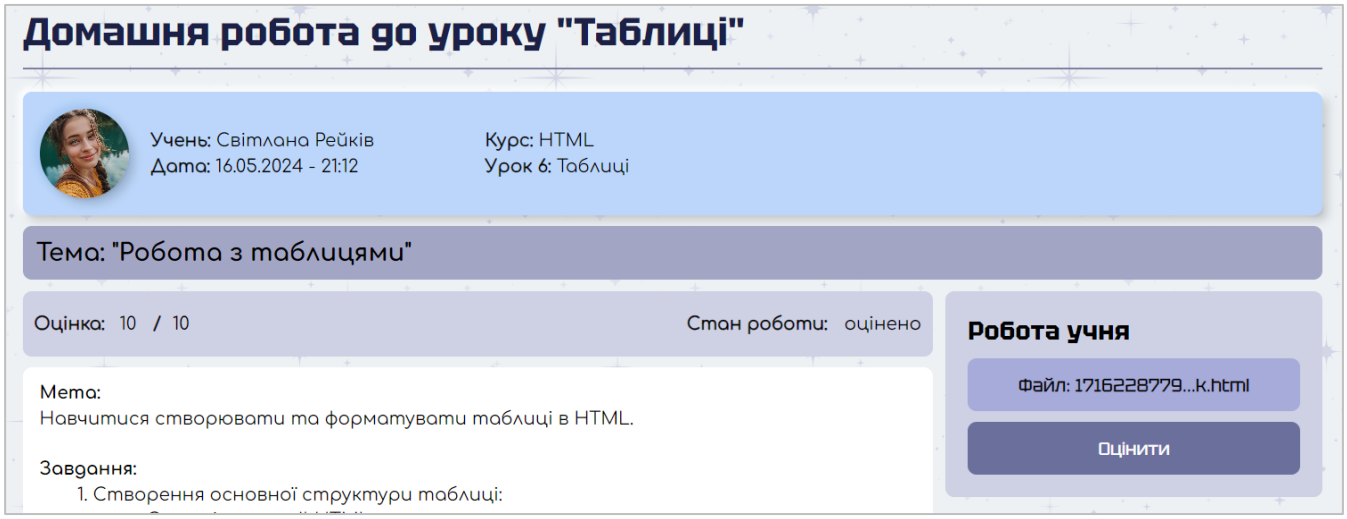


Рисунок 5.44 – Оцінена робота студента

Таким чином відбувається взаємодія користувачів з платформою. Користувачі можуть входити у свої облікові записи, реєструватися на платформі, реєструватися на обрані курси, переглядати вміст курсів, на які зареєстровані, виконувати на надсилати домашні завдання, отримувати і переглядати оцінки, за потреби використовувати фільтри. В залежності від ролі користувачам доступний додатковий функціонал, такий як додавання та редагування курсів, уроків, домашніх завдань, перегляд, схвалення чи відхилення заявок реєстрації на курси, оцінювання домашніх робіт. Розроблений інтерфейс та наявні функціональні можливості дозволяють ефективно взаємодіяти з платформою та отримувати максимальну користь від навчального процесу.

ВИСНОВКИ

Під час виконання роботи розглянуто актуальність онлайн навчання, зокрема актуальність вивчення веб-технологій та веб-дизайну.

Порівняно різні платформи онлайн навчання. Оскільки немає аналогічних платформ спеціалізованих під вивчення веб-технологій та веб-дизайну, проаналізовано платформи загального застосування. В порівнянні з іншими платформами, перевагами створеної платформи є наявність особистого кабінету та змога його редагувати; можливість перегляду всіх курсів, які містяться на платформі; подача заявки на курс з подальшим відслідковуванням її стану; фільтрація по різним критеріям для зручного пошуку потрібного запису; наявність відслідковування процесу користувача; підтримка різних типів контенту; прикріплення файлів домашніх завдань; інтуїтивно зрозумілий інтерфейс. Також унікальністю платформи являється те, що вона спеціалізована саме під вивчення веб-технологій та веб-дизайну і немає аналогічних ресурсів для навчання в даній сфері.

Створено діаграму прецедентів, яка допомогла визначити функціональні можливості платформи та створити графічне відображення взаємодії користувача з системою.

Для реалізації платформи проаналізовано різні технології та відібрано саме ті, які найбільше підходять для її реалізації. Під час розробки платформи використані наступні технології: для клієнтської частини – React, React Router та Axios, для серверної частини – Node.js, Express.js, JSONWebToken та BcryptJS, для стилізації – SCSS, для управління базою даних – MySQL.

Розроблено різні інтерфейсні елементи для забезпечення потреб користувачів різних ролей. Такий підхід сприяв покращенню користувацького досвіду та забезпеченню більш ефективного використання платформи.

В результаті роботи розроблена інтерактивна платформа для вивчення веб-технологій та веб-дизайну, яка дозволяє організувати та керувати навчальними процесами онлайн. Створена платформа має ряд функціональних можливостей, які

сприяють зручному та продуктивному навчанню онлайн. Ці можливості включають вивід курсів у вигляді карток, реєстрація та авторизація користувачів, управління профілем, перегляд уроків курсів та виконання домашніх завдань до уроків, оцінювання, коментування, та розширене фільтрування даних. Запровадження хешування паролів гарантує безпеку особистих даних користувачів.

Реалізований застосунок містить необхідну інформацію, широкий спектр функціональних можливостей, простий і зручний інтерфейс, що виконує поставлені цілі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dos Santos L. M. Completing engineering degree programmes on-line during the Covid-19 pandemic: Australian international students' perspectives. Global Journal of Engineering Education. 2021. Vol. 23, no. 2. P. 143–149.

2. What are Online Learning Platforms? URL: https://www.mycomputercareer.edu/news/what-are-online-learning-platforms/#:~:text=An%20online%20learning%20platform%20is,teacher%20to%20monitor%20student%20progress.

3. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL. 3rd ed. 2012. 826 p.

4. The 7 Best Reasons to Choose MySQL for Organizing Your Web Database. URL: https://devm.io/databases/mysql-7-best-reasons-175591.

5. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020. 310 p.

6. React Router. URL: https://reactrouter.com/en/main/start/overview.

7. Importance of SCSS. URL: https://codeit.mk/home/followUs/blog/IMPORTANCE-OF-SCSS.html.

8. Powers S. Learning Node: Moving to the server side. 2016. 271 p.

9. Express js: встановлення, налаштування та особливості. URL: https://foxminded.ua/express-js/.

10. Axios. URL: https://axios-http.com/docs/intro.

11. JWT - JSON Web Tokens. URL: https://jwt.io/introduction#:~:text=What%20is%20JSON%20Web%20Token,because%20it%20is%20digitally%20signed.

12. JWT. URL: https://www.geeksforgeeks.org/jwt-authentication-with-node-js/.

13. BcryptJS. npm. URL: https://www.npmjs.com/package/bcrypt.

14. Single-page application vs multi-page application. Lvivity. URL: https://lvivity.com/single-page-app-vs-multi-page-app.

15. Архітектура програмного забезпечення. Wezom. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya.https://intellipaat.com/blog/what-is-client-server-architecture/

16. Client Server Architecture - Working, Types, and Examples. *Intellipaat*. URL: https://intellipaat.com/blog/what-is-client-server-architecture/.

17. What is Client-Server Architecture. URL: https://www.theknowledgeacademy.com/blog/client-server-architecture/.https://dou.ua/forums/topic/44636/.

18. Розуміння Клієнт-Серверної Архітектури на прикладах. URL: https://dou.ua/forums/topic/44636/.

19. Методологія моделювання прецедентів на основі мови UML. URL: http://ni.biz.ua/9/9_11/9_115925_diagramma-pretsedentov.html.

20. Data Modelling. URL: https://www.guru99.com/data-modelling-conceptual-logical.html#:~:text=A%20Conceptual%20Data%20Model%20is,on%20the%20actual%20database%20structure.

21. REST API. Robot dreams. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi.

22. Classroom Management Tools & Resources - Google for Education. Google for Education. URL: https://classroom.google.com.

23. Moodle. URL: https://moodle.org.

24. Платформа для створення курсів WeStudy. URL: https://westudy.ua.

25. Softbook. URL: https://softbook.app.

ДОДАТОК А

Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну

Текст програми

УКР.НТУУ«КПІ імені Ігоря Сікорського»

Аркушів 10

Київ 2024

Програмні засоби:

– мова програмування – JavaScript;

– запити до бази даних – sql;

– робота з sql в Node.js – mysql;

```
const app = express()
app.use(express.json())
app.use(cookieParser());

const storage = multer.diskStorage({
    destination: function (req, file, cb) {
      cb(null, "../client/public/upload");
    },
    filename: function (req, file, cb) {
      cb(null, Date.now() + file.originalname);
    },
  });

const upload = multer({ storage });

app.post("/api/upload", upload.single("file"), function (req, res) {
  const file = req.file;
  res.status(200).json(file.filename);
});

app.get('/api/download/:filename', (req, res) => {
  const filePath = `../client/public/upload/${req.params.filename}`;
  res.download(filePath);
});

app.use("/api/auth", authRoutes);
```

```
app.use("/api/users", userRoutes);
app.use("/api/courses", courseRoutes);
app.use("/api/lesson", lessonRoutes);
app.use("/api/applications", applicationRoutes);
app.use("/api/homework", homeworkRoutes);

app.listen(8800,()=>{
   console.log("Connected")
})

import { register, login, logout } from "../controllers/auth.js";
const router = express.Router()

router.post("/register", register);
router.post("/login", login);
router.post("/logout", logout);

export default router;

// РЕЄСТРАЦІЯ
export const register = (req, res) => {
   const q = "SELECT * FROM Users WHERE email = ?";

   db.query(q, [req.body.email], (err, data) => {
      if (err) return res.status(500).json(err);
      if (data.length)
         return res.status(409).json("Користувач вже зареєстрований!");

      const salt = bcrypt.genSaltSync(10);
      const hash = bcrypt.hashSync(req.body.password, salt);
```

```
        const q =
            "INSERT INTO Users(`email`, `name`, `surname`, `password`) VALUES (?)";
        const values = [req.body.email, req.body.name, req.body.surname, hash];

        db.query(q, [values], (err, data) => {
            if (err) return res.status(500).json(err);
            return res.status(200).json("Користувач успішно зареєстрований.");
        });
    });
};

// ВХІД
export const login = (req, res) => {
    const q = "SELECT * FROM users WHERE email = ?";

    db.query(q, [req.body.email], (err, data) => {
        if (err) return res.status(500).json(err);
        if (data.length === 0)
            return res.status(404).json("Користувача не знайдено!");

        const isPasswordCorrect = bcrypt.compareSync(
            req.body.password,
            data[0].password
        );

        if (!isPasswordCorrect)
            return res.status(400).json("Неправильний пароль або e-mail!");

        const token = jwt.sign({ id: data[0].id }, "jwtkey");
        const { password, ...other } = data[0];
```

```
        res.cookie("access_token", token, {
            httpOnly: true,
        })
            .status(200)
            .json(other);
    });
};

// ВИХІД
export const logout = (req, res) => {
    res.clearCookie("access_token", {
        sameSite: "none",
        secure: true,
    })
        .status(200)
        .json("Користувач вийшов із системи.");
};

import {
  getLesson,
  addLesson,
  updateLesson,
  deleteLesson,
} from "../controllers/lesson.js";

const router = express.Router();

router.get("/:id", getLesson);
router.post("/:id", addLesson);
router.put("/:id", updateLesson);
router.delete("/:id", deleteLesson);
```

```
export default router;

export const getLesson = (req, res) => {
   const q = "SELECT l.*, c.title AS courseTitle, h.id AS hwId  from Lessons l INNER JOIN Courses c ON l.course = c.id LEFT JOIN Homeworks h ON h.lesson_id = l.id WHERE l.id= ?";
   db.query(q, req.params.id, (err, data) => {
      if (err) return res.status(500).json(err);
      return res.status(200).json(data[0]);
   });
};

export const addLesson = (req, res) => {
   const courseId = req.params.id;
   const q = "INSERT INTO Lessons(`number`, `title`, `descript`, `course`) VALUES (?)";

   const values = [req.body.number, req.body.title, req.body.descript, courseId];

   db.query(q, [values], (err, data) => {
      if (err) return res.status(500).json(err);
      return res.json("Урок додано.");
   });
};

export const updateLesson = (req, res) => {
   const lessonId = req.params.id;
   const q = "UPDATE Lessons SET `number` = ?, `title` = ?, `descript` = ?  WHERE id = ?";
   const values = [req.body.number, req.body.title, req.body.descript, lessonId];
```

```
    db.query(q, values, (err, data) => {
        if (err) return res.status(500).json(err);
        return res.json("Урок оновлено.");
    });
};

export const deleteLesson = (req, res) => {
    const lessonId = req.params.id;
    const q = "DELETE FROM Lessons  WHERE `id` = ?";

    db.query(q, [lessonId], (err, data) => {
        if (err) return res.status(403).json(err);
        return res.json("Урок видалено!");
    });
};

const AddHomework = () => {
    const state = useLocation().state;
    const location = useLocation();
    const lessonId = location.pathname.split("/")[2];
    const [value, setValue] = useState(state?.descript || "");
    const [title, setTitle] = useState(state?.title || "");
    const [max_grade, setGrade] = useState(state?.max_grade || "");
    const navigate = useNavigate();

    const handleClick = async (e) => {
        e.preventDefault();

        try {
            if (state) {
```

```
            await axios.put(`/api/homework/${state.id}`, {
                title,
                descript: value,
                lesson_id: lessonId,
                max_grade,
            });
            navigate(`/homework/${state.id}`);
        } else {
            await axios.post(`/api/homework/${lessonId}`, {
                title,
                descript: value,
                lesson_id: lessonId,
                max_grade,
            });
            navigate(`/lesson/${lessonId}`);
        }
    } catch (err) {
        console.log(err);
    }
};
const toolbarOptions = [
    ["bold", "italic", "underline", "strike"],
    ["blockquote", "code-block"],
    ["link", "video"],

    [{ header: 1 }, { header: 2 }],
    [{ list: "ordered" }, { list: "bullet" }, { list: "check" }],
    [{ script: "sub" }, { script: "super" }],
    [{ indent: "-1" }, { indent: "+1" }],

    [{ size: ["small", false, "large", "huge"] }],
```

```
    [{ header: [1, 2, 3, 4, 5, 6, false] }],

    [{ color: [] }, { background: [] }],

    [{ align: [] }],

    ["clean"],
];
const modules = {
    toolbar: toolbarOptions,
};

return (
    <div className="addCourse">
        {state ? (
            <h2>Змінити домашню роботу до уроку "{state.lessonTitle}"</h2>
        ) : (
            <h2>Додати домашню роботу</h2>
        )}

        <div className="homework">
            <form className="formAddCourse">
                <div className="point">
                    <div className="inputBox">
                        <label>Максимальна оцінка:</label>
                        <input
                            type="text"
                            name="grader"
                            value={max_grade}
                            onChange={(e) => setGrade(e.target.value)}
                        />
```

```
    </div>
    <div className="inputBox">
      <label>Назва:</label>
      <input
        type="text"
        name="title"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
        required
      />
    </div>
  </div>
  <div className="editorContainer">
    <ReactQuill
      className="editor"
      modules={modules}
      theme="snow"
      value={value}
      onChange={setValue}
    />
  </div>
  <div className="formButtons">
    <Link className="link" to={`/lesson/${lessonId}`}>
      <button id="reset" type="reset">
        Скасувати
      </button>
    </Link>

    {state ? (
      <button onClick={handleClick}>Оновити</button>
    ) : (
```

```
                    <button onClick={handleClick}>Додати</button>
                )}
            </div>
        </form>
      </div>
    </div>
  );
};

export default AddHomework;
```

ДОДАТОК Б

Платформа онлайн-курсів для вивчення веб-технологій та веб-дизайну

Апробація

УКР.НТУУ«КПІ імені Ігоря Сікорського»

Аркушів 6

Київ 2024

МОЛОДІЖНА НАУКОВА ЛІГА

МАТЕРІАЛИ VII МІЖНАРОДНОЇ СТУДЕНТСЬКОЇ НАУКОВОЇ

КОНФЕРЕНЦІЇ

СУЧАСНІ АСПЕКТИ ТА ПЕРСПЕКТИВНІ НАПРЯМКИ РОЗВИТКУ НАУКИ

м. Дніпро, Україна
24 травня 2024 рік

Вінниця, Україна
«UKRLOGOS Group»
2024

Клебан Дар'я Андріївна, здобувач вищої освіти навчально-наукового інституту атомної та теплової енергетики
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

Науковий керівник: Полягушко Любов Григорівна, канд. техн. наук, доцент, доцент кафедри цифрових технологій в енергетиці
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна

ПОРІВНЯЛЬНИЙ АНАЛІЗ ОНЛАЙН ПЛАТФОРМ ДЛЯ НАВЧАННЯ

Зі зростанням швидкості технологічного прогресу і зміною потреб суспільства, світ постійно змінюється. Технології відіграють ключову роль у цьому процесі, впливаючи на всі сфери нашого життя, сфера освіти не є виключенням. Сучасне покоління виростає в умовах цифрової ери, де доступ до інформації є легким і миттєвим. Тому, для ефективного навчання і підготовки молоді до викликів сучасного світу, потрібні нові методи та підходи до освіти. Оскільки традиційні методи навчання часто виявляються недостатньо ефективними, використання інноваційних освітніх технологій та онлайн платформ стає необхідним кроком у вдосконаленні навчального процесу.

Після виникнення пандемії COVID-19 різко зросла потреба у використанні онлайн навчання, онлайн платформи стали відповіддю на цей виклик. Згідно дослідження [1] було виявлено, що під час пандемії COVID-19 кількість студентів онлайн зросла з 30% до 70%. Також враховуючи воєнний стан, в Україні продовжується дистанційне навчання з залученням онлайн платформ, тому для України залишається вкрай важливим розвиток онлайн-навчання.

Мета нашого дослідження полягає в проведенні аналізу різних платформ для онлайн-навчання з ціллю виявлення їх основних переваг та недоліків. Цей аналіз спрямований на детальне вивчення функціональних можливостей кожної платформи, а також на оцінку їхнього впливу на навчальний процес. Визначення сильних та слабких сторін платформ допоможе виявити, які аспекти вимагають уваги та можливих поліпшень. В подальшому це сприятиме вдосконаленню подібних програмних продуктів поліпшуючи якість онлайн-освіти та забезпечення кращого досвіду навчання для користувачів.

Платформа онлайн-навчання — це веб-простір або портал для освітнього контенту та ресурсів. Вона об'єднує в собі різноманітні компоненти, такі як лекції, навчальні матеріали, можливості для взаємодії між учасниками, та дозволяє вчителям та учням вести моніторинг та контроль за прогресом у навчанні.

Існують різні типи онлайн-платформ для навчання, основні з них:

1. Сайти навчальних ресурсів (англ. Learning Destination Sites, LDS) – це сайти, на яких багато різних розробників курсів пропонують свої навчальні матеріали. У студента є можливість вибрати теми та викладачів, та записатись на курс відповідно до уподобань. Прикладами таких платформ є Coursera, EdX та Udemy.

2. Системи управління навчанням (англ. Learning Management System, LMS) –

це веб-сайт або додаток для організації та проведення навчальних процесів. Вони зазвичай використовуються в межах конкретної організації чи установи. На відміну від LDS платформ, за створення курсів відповідають уповноважені особи установи. Прикладами таких платформ є Google Classroom, Moodle.

3. Екосистеми керування навчанням (англ. Learning Management Ecosystems, LME) – це розширений навчальний інструмент, який об'єднує кілька програмних рішень. Такі екосистеми зазвичай використовуються великими організаціями, які пропонують багато онлайн програм [2].

Детально розглянемо LMS платформи. Однією з таких платформ є Google Classroom – це безкоштовна платформа, розроблена Google для освітніх закладів, що дозволяє вчителям створювати, розповсюджувати та оцінювати завдання. Вона спрощує процес збору робіт від учнів, взаємодію між учасниками навчального процесу. У Google Classroom можна створювати віртуальні класи, куди вчителі запрошують учнів та співробітників, де можна додавати завдання, матеріали для читання, відео та інші ресурси. Вона також інтегрується з іншими продуктами Google, такими як Google Drive, Google Docs, Google Sheets та іншими, що дозволяє зручно працювати з документами.

Основні переваги даної платформи, які можна виділити:

- Наявність веб-застосунку та додатку, що забезпечує зручне навчання на будь-якому пристрої.
- Сповіщення про отримання оцінки, нагадування про необхідність здачі роботи до певного терміну.
- Інтеграція з іншими продуктами Google.
- Можливість надсилати запрошення учням.
- Можливість додавати декількох учителів або кураторів до класу з метою перевірки робіт.

Але існує і низка недоліків. Основним недоліками даної платформи являються:

- Відсутність фільтрування за датою публікації матеріалу. Задля того щоб переглянути найперший урок курсу доведеться перелискувати всі уроки.
- Розміщення оголошень та завдань усіх разом в одній стрічці ускладнює перегляд. При великій кількості уроків і оповіщень стає важко знайти потрібний запис та зорієнтуватися.
- Відсутність пошуку за назвою уроку/завдання.
- Відсутність відслідковування прогресу та отримання загальної оцінки по всьому курсу.
- Не можливо створювати інтерактивні тести або завдання.

Наступною платформою для аналізу було обрано Moodle. Moodle – це безкоштовна платформа для управління навчанням, яка надає широкі можливості. Завдяки своїй гнучкості, Moodle став популярним інструментом для організації онлайн-курсів у навчальних закладах та корпоративних середовищах.

Основні переваги використання, як можна виділити:

- Надає широкі можливості для взаємодії між учасниками навчального процесу, такі як форуми, чати, відеоконференції тощо.
- Гнучкий, що дозволяє налаштовувати робочий простір під індивідуальні вимоги.
- Можливість створювати тести з великою низкою налаштувань.

Недоліками даної платформи є:

•Складність використання для нових користувачів. Для тих, хто тільки починає знайомитися з Moodle, інтерфейс може здатися складним та заплутаним.

•Підтримка та налаштування Moodle може вимагати технічних знань або спеціаліста з IT, щоб забезпечити ефективну роботу платформи.

•Великі вимоги до хостингу. При великій кількості користувачів або обсязі навчального матеріалу, можуть виникнути проблеми з продуктивністю та швидкістю роботи Moodle.

Додатково було проаналізовано платформи Westudy та Softbook та внесено результати до таблиці 1.

У таблиці 1 наведено усі дані, які були зібрані в ході аналізу, та представлено результати. У цій таблиці "+" позначає присутність, а "–" відсутність певної функціональної можливості.

Таблиця 1

Порівняння платформ онлайн навчання

		Платформи онлайн-навчання			
		Google Classroom	Moodle	Westudy	Softbook
Функціональні можливості	Сортування по часу публікації	–	+	–	–
	Пошук по назві завдання/оголошення	–	+	–	–
	Підтримка різних типів контенту	+	+	+	+
	Інтуїтивно зрозумілий інтерфейс	+	–	–	+
	Інтерактивне тестування	–	+	+	+
	Додаток на мобільні пристрої	+	–	–	–
	Відстеження прогресу проходження курсу	–	+	+	+
	Безкоштовно (для викладача)	+	+	–	–

У результаті аналізу було досліджено, що платформи онлайн-навчання мають величезний потенціал для оптимізації навчального процесу. Однак, вони вимагають певного вдосконалення, щоб процес навчання був більш ефективним і зручним. Google Classroom спрощує взаємодію вчителів і учнів, але обмежений у функціональності. Moodle надає гнучкість у налаштуванні робочого простору, але має складний інтерфейс та вимагає технічної підтримки. Westudy та Softbook пропонують багато функціональних переваг, але їх недоліки та висока вартість переважають ці переваги. Навіть з усіма вищезазначеними обмеженнями та вадами, платформи онлайн-навчання є ключовим інструментом у сучасному освітньому середовищі, який має потенціал значно поліпшити якість та доступність навчання.

Список використаних джерел:

1. Dos Santos L. M. Completing engineering degree programmes on-line during the Covid-19 pandemic: Australian international students' perspectives. Global Journal of Engineering Education. 2021. Vol. 23, no. 2. P. 143–149.

2. What are Online Learning Platforms? URL: https://www.mycomputercareer.edu/news/what-are-online-learning-platforms/#:~:text=An%20online%20learning%20platform%20is,teacher%20to%20monitor%20student%20progress.
3. Classroom Management Tools & Resources - Google for Education. Google for Education. URL: https://classroom.google.com.
4. Moodle. URL: https://moodle.org.
5. Платформа для створення курсів WeStudy. URL: https://westudy.ua.
6. Softbook. URL: https://softbook.app.