МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Криворізький національний університет
Кафедра моделювання програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка веб-орієнтованої системи онлайн-навчання з використанням технологій React та Node.js

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-22-2
_____/К. А. Смаглюк/

Керівник
кваліфікаційної роботи _____ /А. М. Стрюк

Завідувач кафедри _____ /А. М. Стрюк

Кривий Ріг
2026

Криворізький національний університет

Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри _____ А. М. Стрюк
«_» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-22-2 Смаглюк Катерині Андріївні

1. Тема: Розробка веб-орієнтованої системи онлайн-навчання з використанням технологій React та Node.js.
2. Термін подання студентом закінченої роботи: «10» червня 2026р.
3. Вихідні дані по роботі: розроблювана система повинна мати веб-орієнтовану архітектуру, підтримувати адаптивність інтерфейсу та три ролі користувачів (гість, учень, викладач) із захистом даних через JWT-токени
4. Зміст пояснювальної записки: Зміст, Вступ, Сучасний стан і актуальність кваліфікаційної роботи, Розробка функціональних схем і алгоритмів, Розробка компонентів, Реалізація програмного забезпечення, Висновки
5. Перелік ілюстративного матеріалу: схематичні діаграми, блок-схеми розроблених алгоритмів, схема взаємодії модулів системи, знімки екранних форм.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	24.02.2026 – 28.02.2026
2	Проведення порівняльного аналізу існуючих платформ онлайн-навчання	01.03.2026 – 09.03.2026
3	Формалізація вимог та моделювання бізнес-процесів (BPMN, Use Case)	10.03.2026 – 18.03.2026
4	Підготовка матеріалів першого розділу роботи	19.03.2026 – 23.03.2026
5	Проєктування архітектури системи та структури бази даних PostgreSQL	24.03.2026 – 31.03.2026
6	Підготовка матеріалів другого розділу роботи	01.04.2026 – 05.04.2026
7	Програмна реалізація серверної та клієнтської частин додатка (React, Node.js)	06.04.2026 – 12.05.2026
8	Підготовка матеріалів третього розділу роботи	13.05.2026 – 20.05.2026
9	Функціональне тестування системи та підготовка матеріалів четвертого розділу	21.05.2026 – 26.05.2026
10	Оформлення пояснювальної записки та фінальних висновків	27.05.2026 – 05.06.2026

Дата видачі завдання: « » _____ 2026 р.

Студент: _____ / К. А Смаглюк /

Керівник роботи: _____ / А. М. Стрюк /

РЕФЕРАТ

ВЕБ-ОРІЄНТОВАНА СИСТЕМА, ОНЛАЙН-НАВЧАННЯ, ПЛАТФОРМА NODE.JS, БІБЛІОТЕКА REACT.JS, СУБД POSTGRESQL, АРХІТЕКТУРА REST API, АВТЕНТИФІКАЦІЯ JWT, РОЗМЕЖУВАННЯ РОЛЕЙ, МОДЕЛЮВАННЯ БІЗНЕС-ПРОЦЕСІВ.

Пояснювальна записка: 49 с., 18 рис., 3 ліст., 3 дод., 9 джерел.

Метою кваліфікаційної роботи є проєктування та реалізація безпечної та веб-орієнтованої системи онлайн-навчання для дистанційного навчання, керування контентом та інтерактивного контролю знань.

У роботі проведено аналіз існуючих аналогів систем керування навчанням (Moodle, Google Classroom, HUMAN), виявлено їхні переваги та недоліки, а також обґрунтовано доцільність розробки індивідуального рішення. Сформульовано функціональні та нефункціональні вимоги до системи, побудовано різноманітні діаграми.

Обґрунтовано вибір технологічного стеку для реалізації клієнт-серверної архітектури програми. Розроблено структуру реляційної бази даних PostgreSQL. На базі платформи Node.js та фреймворку Express.js реалізовано серверне REST API із впровадженням пулу з'єднань (Pool), механізмів Middleware та автентифікації на основі токенів JSON Web Token (JWT). Клієнтську частину додатка спроєктовано за допомогою бібліотеки React.js.

Виконано комплексне функціональне тестування розробленої системи, яке підтвердило її стабільність, швидкодію, надійність розмежування доступу користувачів (гість, учень, викладач) та повну відповідність поставленим технічним вимогам.

ABSTRACT

WEB-ORIENTED SYSTEM, ONLINE LEARNING, NODE.JS PLATFORM, REACT.JS LIBRARY, POSTGRESQL DBMS, REST API ARCHITECTURE, JWT AUTHENTICATION, ROLE-BASED ACCESS CONTROL, BUSINESS PROCESS MODELING.

Explanatory note: 49 pages, 18 figures, 3 app, 3 listing, 9 sources.

The purpose of the qualification work is the design and implementation of a secure and web-oriented online learning system for distance education, content management, and interactive knowledge assessment.

The work provides an analysis of existing analogues of learning management systems (Moodle, Google Classroom, HUMAN), identifies their advantages and disadvantages, and substantiates the feasibility of developing an individual solution. Functional and non-functional requirements for the system have been formulated, and various diagrams have been constructed.

The choice of the technology stack for the implementation of the client-server architecture of the application is substantiated. The structure of the PostgreSQL relational database has been developed. Based on the Node.js platform and the Express.js framework, a server REST API has been implemented with the integration of a connection pool (Pool), Middleware mechanisms, and authentication based on JSON Web Tokens (JWT). The client side of the application is designed using the React.js library.

Comprehensive functional testing of the developed system has been performed, which confirmed its stability, performance, reliability of user access rozpodil / access control (guest, student, teacher), and full compliance with the specified technical requirements.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології є необхідними як для освіти та професійної діяльності, так і для повсякденного життя. Потреба у дистанційному навчанні, а саме використання онлайн-платформ для цього, стала особливо відчутною у 2020 році під час пандемії COVID-19, коли заклади освіти були змушені терміново перейти на дистанційний формат роботи. Після 2022 року, під час воєнного часу в Україні, дистанційна форма навчання набула ще більшої важливості. Завдяки таким системам можна забезпечити безперервність навчального процесу незалежно від географічного розташування тих, хто бере участь у навчанні.

Отже, метою даної роботи є розробка веб-орієнтованої системи онлайн-навчання, яка поєднає сучасні технологічні рішення, зручний інтерфейс та необхідний функціонал для викладачів та здобувачів освіти.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Критичний аналіз сучасних систем онлайн-навчання

Онлайн-навчання вже є важливою частиною сучасної освіти. Завдяки йому можна отримувати знання у будь-якому місці та в будь-який час. Такі системи активно впроваджують у школах, університетах та для самостійного навчання.

Такі платформи підтримують різні формати навчальних завдань, зокрема текстові матеріали, презентації, відео, а також інтерактивні завдання. Використання інтерактивних елементів стимулює цікавість учнів та підвищує ефективність навчання.

Сучасні системи онлайн-навчання зазвичай мають веб-орієнтовану архітектуру. Доступ до них здійснюється через браузер. Користувачам не потрібно встановлювати додаткове програмне забезпечення. Це полегшує не тільки використання, а й оновлення системи.

Основними функціями таких платформ є реєстрація та авторизація користувачів, ознайомлення з навчальними матеріалами, проходження тестів, перегляд відеоуроків та контроль успішності. Зазвичай, є поділ ролей, наприклад: викладач та студент.

Важливою вимогою до сучасних додатків є адаптивність. Інтерфейс повинен правильно відображатися на різних гаджетах, таких як стаціонарний комп’ютер, планшет та мобільний пристрій. Також значну роль відіграє швидкодія та надійність захисту даних користувачів.

Таким чином, сучасні платформи для дистанційного навчання є комплексними веб-додатками, які поєднують навчальні матеріали, інтерактивність та засоби контролю знань. Це робить їх актуальним об’єктом для аналізу та подальшої розробки.

1.2 Аналіз предметної області та ідентифікація стейкхолдерів

Основною цільовою аудиторією є студенти, викладачі та різні навчальні центри, які потребують зручного інструменту для дистанційного навчання, проходження курсів, перегляду відеоматеріалів та перевірки знань. Також система навчання підійде користувачам, які хочуть проходити різноманітні навчальні курси.

Вимоги до системи:

– доступ до навчальних матеріалів через веб-браузер;
– можливість перегляду відеоуроків;
– проходження онлайн-тестування;
– зберігання результатів навчання;
– взаємодія між викладачем та студентом;
– доступ до навчання з різних пристроїв (ПК, планшет, смартфон).

Головні переваги розроблювального додатку:

– використання сучасного стеку технологій React та Node.js;
– зручний та інтуїтивний інтерфейс;
– адаптивний дизайн для мобільних пристроїв;
– можливість масштабування системи та додавання нових модулів.

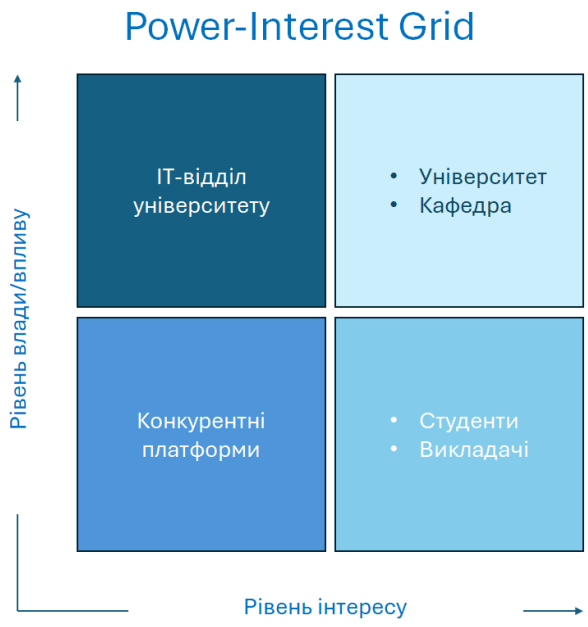

Рисунок 1.1 – Матриця стейкхолдерів

На рисунку 1.1 зображена матриця стейкхолдерів, яка відображає вплив людей або організацій на систему та їх зацікавленість в ній.

Отже, найбільший вплив та зацікавленість мають університет та кафедра, оскільки вони будуть вирішувати чи буде впроваджуватися ця системи чи ні.

Крім того, великий вплив має ІТ-відділ університету, бо саме вони будуть розповсюджувати систему в університеті.

Також, у цій системі зацікавлені студенти та викладачі, оскільки вони будуть найбільше користуватися системою, але мають малий вплив, адже головні вимоги буде висувати університет.

У підсумку, замовником буде виступати університет та кафедра. Головними користувачами системи будуть студенти та викладачі. А сповільнювати розробку додатку можуть технічні обмеження такі як, відсутність серверної підтримки.

Розглянемо для кращого розуміння завдання відповідні історії користувачів. Оскільки user stories зазвичай короткий опис вимог різних користувачів.

Користувач:

- Як користувач, я хочу зареєструватися, щоб отримати доступ до системи.

Студент:

- Як студент, я хочу переглядати навчальні матеріали, щоб проходити тести;
- Як студент, я хочу бачити свій результат, щоб оцінити свої знання.

Викладач:

- Як викладач, я хочу, щоб автоматично перевірялися тести;
- Як викладач, я хочу переглядати результати всіх студентів, щоб оцінити їх успішність;
- Як викладач, я хочу створювати завдання, щоб студенти могли їх робити.

1.3 Огляд популярних платформ онлайн-навчання

Аналіз існуючих аналогій є важливим етапом під час розробки будь-якого програмного забезпечення, оскільки ми передивляємося вже готові рішення, інтерфейси, функції та інше. Це допоможе нам тим, що ми вже будемо розуміти, чого не вистачає в цих застосунках, що потрібно буде додати до нашого проєкту, а що можна використати готовим.

Наразі у вільному доступі вже є кілька популярних платформ, зокрема:

а) Google Classroom;

б) Moodle;

в) HUMAN. [1]

Розглянемо кожну з цих платформ детальніше.

1.3.1 Переваги та недоліки Google Classroom

Google Classroom – це інтуїтивно зрозумілий інструмент, що найчастіше використовується в різних закладах освіти. Ця платформа тісно інтегрована з екосистемою сервісів Google, охоплюючи Google Drive, Google Docs, Google Meet, електронну пошту Gmail та інші додатки.

Основними перевагами є швидке налаштування, велика кількість інструментів, які необхідні в плануванні і створенні якісного віддаленого навчання, зручність та безкоштовне використання. Також великою перевагою є оффлайн доступ до інформації. Навіть якщо у студента тимчасово відсутній доступ до інтернету, то він зможе ознайомитися з матеріалом.

Недоліками Google Classroom є обмежений функціонал, відсутність розширених засобів аналітики, тестування та невисокий рівень кастомізації.

Рисунок 1.2 – Google Classroom

1.3.2 Переваги та недоліки Moodle

Moodle – одна з найпоширеніших платформ, яка широко застосовується у вищих та середніх навчальних закладах. Це відкрита та безкоштовна система з потужними функціями. Moodle підтримує створення курсів, публікацію навчальних матеріалів, проведення тестів, контроль прогресу навчання та організацію взаємодії між викладачем та учнем.

Її переваги включають гнучкість, можливість розширення за допомогою плагінів та підтримку кількох ролей користувачів.

Водночас, недоліками можна вважати дещо заплутаний інтерфейс, необхідність постійного адміністрування серверної частини, а також відносно високий рівень технічної підготовки, який вимагається від осіб, відповідальних за її налаштування.

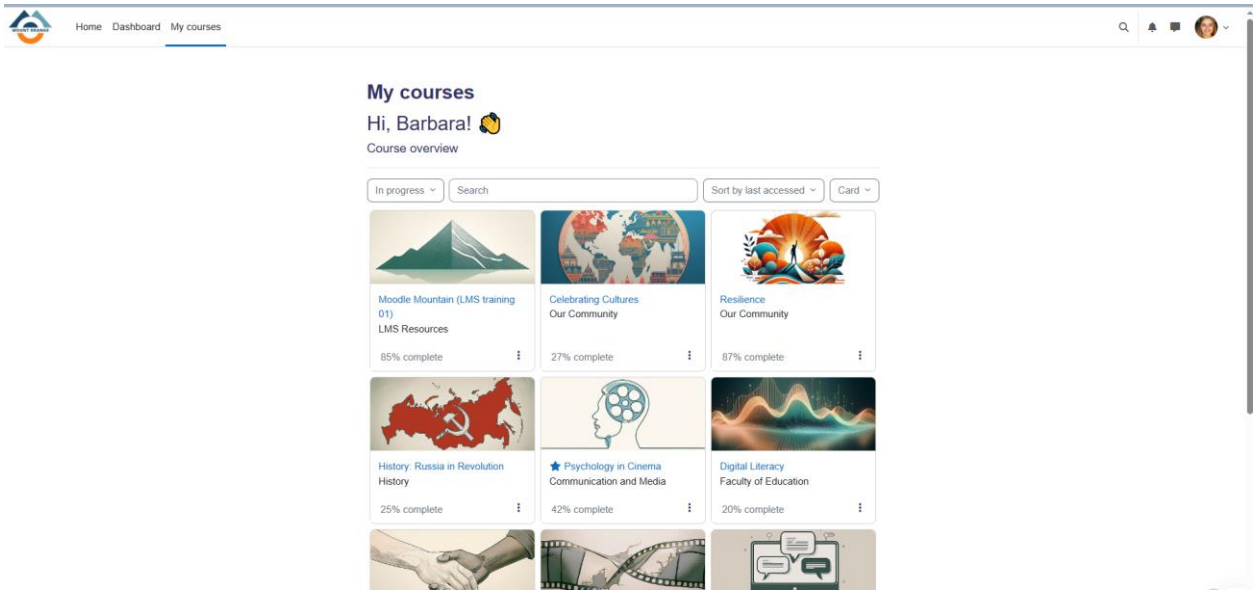

Рисунок 1.3 – Moodle

1.3.3 Переваги та недоліки HUMAN

Платформа HUMAN є сучасною українською системою для організації освітнього процесу, яка активно використовується в школах. Вона орієнтована на дистанційне та змішане навчання і надає можливості для створення навчальних курсів, завантаження навчального матеріалу, комунікації між учнями та викладачами, а також контролю виконання завдань.

До переваг платформи HUMAN належать зручний інтерфейс, адаптованість до потреб української освіти та відсутність складного технічного налаштування.

Однак платформа має обмеження щодо налаштування та інтеграції з іншими онлайн-додатками.

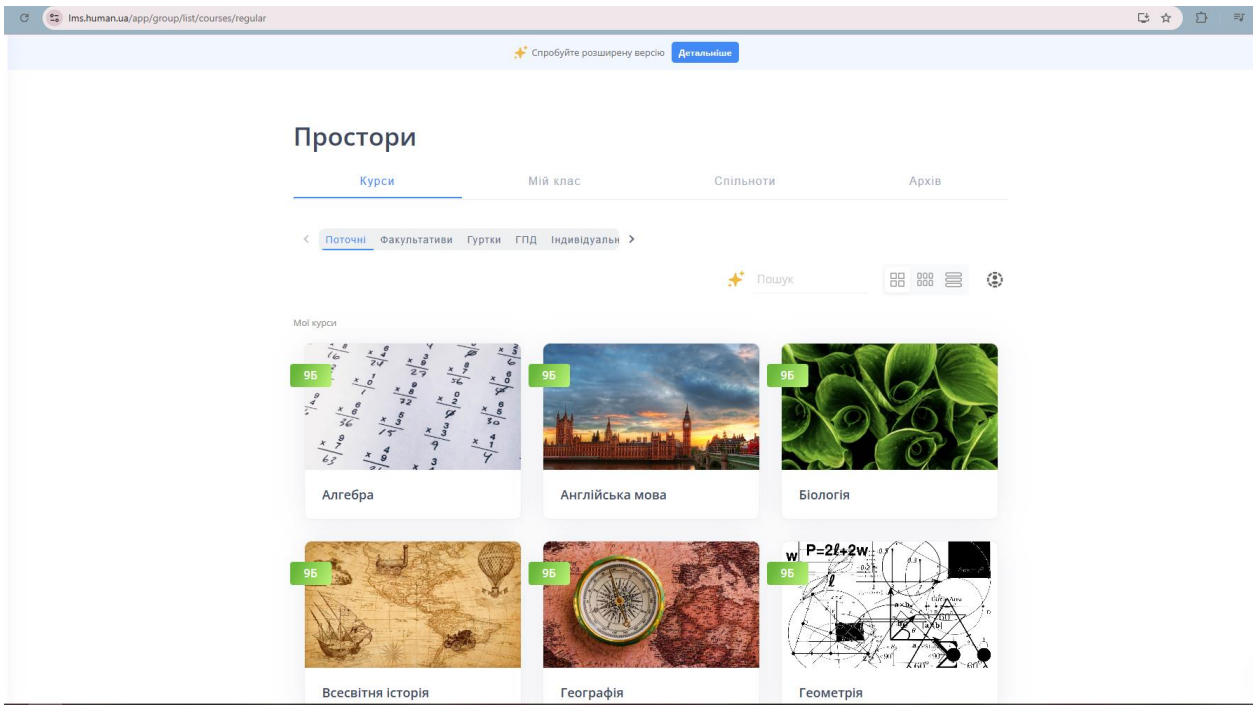

Рисунок 1.4 – HUMAN

На даний момент процес організації онлайн-навчання часто здійснюється з використанням різних інструментів, таких як Google Classroom або Moodle.

Викладач створює навчальні матеріали та надсилає їх студентам. Студенти отримують матеріали, виконують завдання та надсилають відповіді назад викладачу. Виконані завдання завантажуються студентами у систему. Це створює чергу «вхідних» для викладача, де немає автоматичного визначення пріоритетів. При цьому викладач має відкрити кожну роботу окремо, вивчити її, за потреби завантажити на комп'ютер. Це забирає багато часу. Також при великій кількості студентів (наприклад, потік у 100+ осіб) зростає ризик помилок при виставленні балів або неуважності під час перевірки.

В Classroom фінальні оцінки часто переносяться вручну в Excel-таблиці, Google Sheets або паперові відомості для звітності перед кафедрою/адміністрацією. Зворотний зв’язок також надається індивідуально кожному студенту. Часто викладач змушений писати одне й те саме різним студентам, оскільки немає бази типових помилок або авто-відповідей.

У дипломній роботі системи онлайн-навчання всі процеси автоматизуються та об’єднуються в єдину платформу.

Викладач створює курс, додає навчальні матеріали та формує тести безпосередньо в системі. Студенти отримують доступ до матеріалів, проходять тестування та миттєво отримують результати.

Система автоматично зберігає результати, формує статистику успішності та забезпечує швидкий зворотний зв’язок.

Це дозволяє значно зменшити обсяг ручної роботи та підвищити ефективність навчального процесу.

Таким чином, аналіз існуючих платформ онлайн-навчання показує, що кожна з них має певні переваги та недоліки. Це підтверджує доцільність розробки веб-орієнтованої системи онлайн-навчання, яка поєднуватиме зручність використання, необхідний функціонал та сучасні технологічні рішення. [2]

1.4 Цілі та завдання кваліфікаційної роботи

Сучасний етап розвитку освіти характеризується активним упровадженням цифрових технологій та зростанням обсягів навчальної інформації. У зв’язку з цим ефективна організація дистанційного освітнього процесу та зручний доступ до навчальних матеріалів набувають особливої важливості. Основна мета кваліфікаційної роботи полягає у створенні функціональної та зручної веб-орієнтованої системи онлайн-навчання, яка забезпечить ефективну взаємодію між викладачами та здобувачами освіти.

Для досягнення цієї мети в межах роботи необхідно виконати такі завдання:

- розробити функціонал управління навчальними матеріалами (забезпечити можливість створення, перегляду, редагування та видалення навчальних курсів і матеріалів);
- спроєктувати та реалізувати базу даних системи (для зберігання інформації про користувачів, курси, завдання та результати навчання);

- розробити інтерфейс користувача (інтуїтивно зрозумілий та зручний інтерфейс для різних категорій користувачів);
- забезпечити адаптивність інтерфейсу для коректної роботи на комп’ютерах, планшетах та мобільних пристроях;
- реалізувати функціонал автентифікації та авторизації користувачів з різними рівнями доступу (адміністратор, викладач, здобувач освіти);
- додати інтерактивні елементи, які підвищують зацікавленість користувачів та сприяють ефективному навчанню;
- провести тестування розробленого додатку та оцінити його працездатність.

1.5 Вибір технологій реалізації

Для успішної реалізації системи для навчання необхідно ретельно підійти до вибору технологій. З огляду на вимоги до функціональності, масштабованості та зручності користувацького інтерфейсу, будуть використовуватися наступні технології.

Технології фронтенд-частини (Client-side):

- React чудово підходить для створення динамічних та інтерактивних інтерфейсів, що є важливим для системи онлайн-навчання. Архітектура React дозволить легко організувати та підтримувати код; [7]
- Bootstrap значно спростить створення адаптивного дизайну, що є критично важливим для мобільних пристроїв. Готові компоненти Bootstrap дозволять швидко створити приємний та професійний інтерфейс.

Технології бекенд-частини (Server-side):

- Node.js забезпечує високу продуктивність та масштабованість, що важливо для обробки пошукових запитів та роботи з базою даних; [3]
- Express.js мінімалістичний та гнучкий фреймворк для Node.js, призначений для проєктування REST API;
- Javascript – єдина мова програмування, що використовується як на стороні клієнта (скрипти в React), так і на стороні сервера (Node.js).

Дані/Алгоритми:

- PostgreSQL – потужна та надійна реляційна базою даних, яка підходить для зберігання структурованих даних системи онлайн-навчання. Вона забезпечує ефективний пошук та підтримку складних запитів; [8]
- Хеш-таблиця, вона забезпечує амортизовану швидкість виконання операцій пошуку, додавання та видалення елементів за ключем на рівні O(1) у середньому випадку. [9]

2 РОЗРОБКА ФУНКЦІОНАЛЬНИХ СХЕМ І АЛГОРИТМІВ

Для розробки будь-якого додатку потрібно мати чітке уявлення як працює система та які задачі виконує, тому для візуалізації проєкту використовують різни типи діаграм. Найпоширенішими діаграмами є:

– Структурна схема;
– Діаграма варіантів використання (Use case diagram);
– Функціональна діаграма (Functional flow block diagram);
– Діаграми «сутність-зв'язок» (ER-diagram);
– Діаграма потоків даних (Data Flow Diagram).

2.1 Структурна схема

Структурна схема відображає основні об’єкти системи, їх взаємодію між собою та головні функції цієї системи.

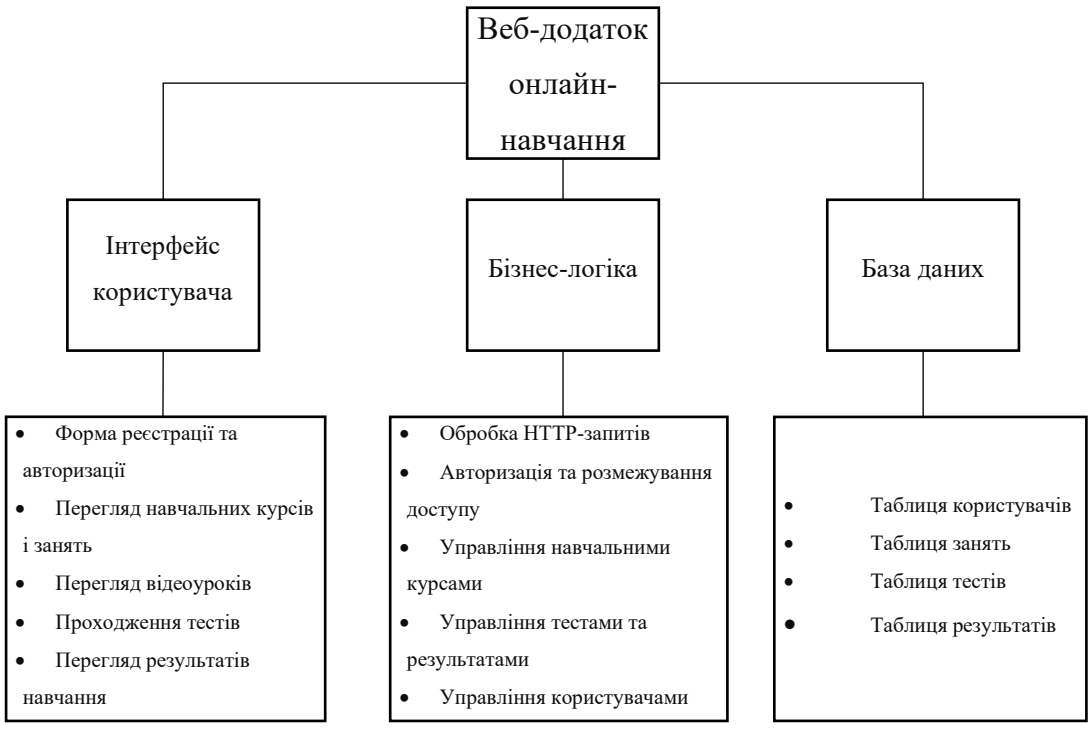

Рисунок 2.1 – Структурна схема програми

На рисунку 2.1 зображена структурна схема програми, яка відображає головні компоненти додатку: інтерфейс користувача, бізнес-логіку та базу даних. Кожен з цих компонентів відповідає за певні дії. Інтерфейс користувача відповідає за взаємодію з людиною, тобто там мають бути різноманітні форми, та сторінки. Бізнес-логіка відповідає за головні функції додатку, тобто обробка запитів, розмежування прав користувачів та інше. База даних відповідає за збереження потрібних даних, таких як інформація про користувачів, заняття, тести та результати.

2.2 Діаграма варіантів використання системи

Діаграма варіантів використання (Use case diagram) – це діаграма, яка відображає як різні користувачі можуть взаємодіяти з системою, які вони можуть виконувати дії та інше. Вони використовуються для відображення головного функціоналу системи, щоб розуміти, які саме функції мають бути реалізовані в додатку. [6]

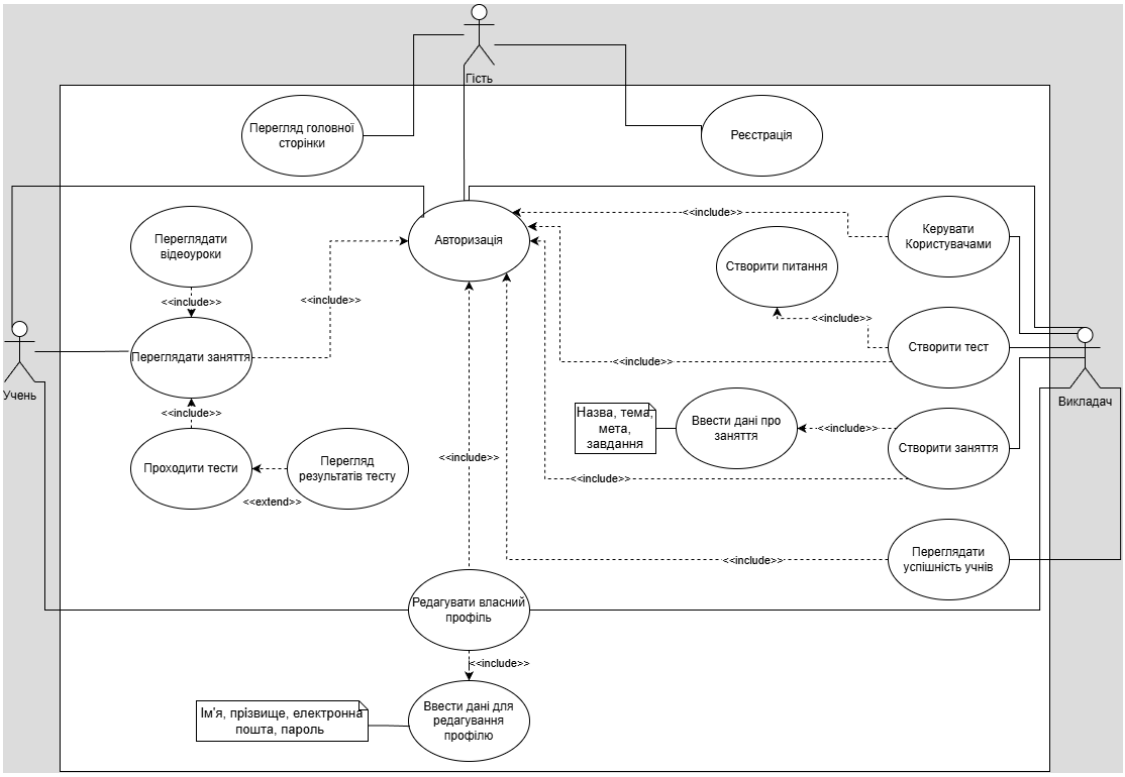

Рисунок 2.2 – Діаграма варіантів використання системи

На рисунку 2.2 зображена діаграма варіантів використання. Розглянемо детальніше схему.

Маємо 3 головних акторів, а саме:

- Гість (незареєстрований користувач);
- Студент (зареєстрований користувач);
- Викладач (зареєстрований користувач з особливими правами).

Кожен з них може робити різні дії

Гість, оскільки він незареєстрований користувач, має найменше можливостей. Він може лише:

- Переглядати головну сторінку, на якій він зможе дізнатися про додаток;
- Авторизуватися, якщо вже є акаунт;
- Зареєструватися.

Студент, на відміну від гостя, має більший функціонал, а саме:

- Переглядати заняття;
- Дивитися відеоуроки;
- Проходити тести, також він може переглянути свої результати, тобто перевірити в яких питаннях допущена помилка;
- Редагувати власний профіль.

Всі ці дії студент може робити лише після авторизації в системі.

Найголовнішим в цій системі є Викладач, в нього найбільше прав та функцій.

- Керувати користувачами;
- Створювати заняття;
- Створення тестів також включає в себе розробку питань та варіантів відповідей;
- Редагувати власний профіль;
- Переглядати успішність учнів.

Викладач також має бути авторизований в системі, щоб мати можливість виконувати всі ці дії.

2.3 Функціональна схема програми

Функціональна діаграма (IDEF0) відображає взаємозв'язки між функціями у системі.

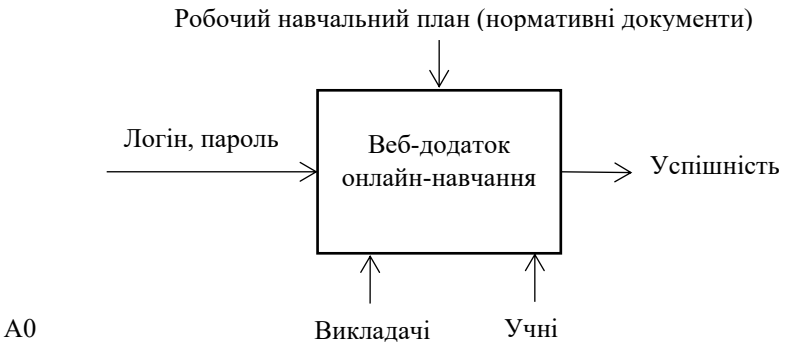

Рисунок 2.3 – Контекстна діаграма

На рисунку 2.3 зображена контекстна діаграма, яка містить лише один загальний блок – це головна функція системи, яка проєктується. Цей блок визначає «основну місію» системи.

Рисунок 2.4 – Функціональна діаграма

На рисунку 2.4 зображена функціональна діаграма першого рівня декомпозиції. Тобто головна функція розбита на декілька простих, для кращого розуміння системи. Розглянемо кожну функцію детальніше. Вхід – авторизація в системі. Користувач повинен ввести логін та пароль. Після цього в системі перевіряється роль користувача, від цього залежить його функціонал.

Розглянемо спочатку Студент. Після входу у систему він може переглянути завдання, після ознайомлення він має пройти тест для закріплення матеріалу. Одразу після проходження він може побачити свій результат. Також студент може переглядати свою успішність, а саме, скільки завдань пройдено на курсі, які отримано оцінки та інше.

Розглянемо тепер роль Викладач. Після входження у систему він має створити завдання для курсу та додати до нього тестове завдання. Після того як учні пройдуть тест, він може переглядати їх результати. Він також може переглядати успішність своїх учнів, тобто хто на якому етапі проходження, в кого які оцінки та інше. На всіх цих етапах Вчитель керується робочим навчальним планом та іншими нормативними документами.

2.4 Моделювання бізнес-процесів (BPMN 2.0)

Нотація BPMN (Business Process Model and Notation) – це стандартизована діаграма, яка відображає бізнес процеси та дії учасників в ній. Їх використовують для аналізу завдання, вона допомагає оптимізувати та автоматизувати відповідні процеси. [4]

На рисунку 2.5 зображена BPMN діаграма для системи онлайн-навчання. На початку роботи користувач вводить логін та пароль. Ці дані надаються системі, яка перевіряє чи є зареєстрований користувач у системі чи ні. Якщо користувача не знайдено, система виведе відповідне повідомлення. В іншому випадку система почне перевіряти вашу роль. Якщо користувач є Викладачем, то він має створити завдання та відповідний тест для закріплення пройденого

матеріалу. Після цього Учні зможуть опрацювати матеріал та пройти відповідний тест. Після тестування система перевірить відповіді та збереже результати. У кінці Учні зможуть переглянути свої результати, а Викладач зможе побачити загальну успішність.

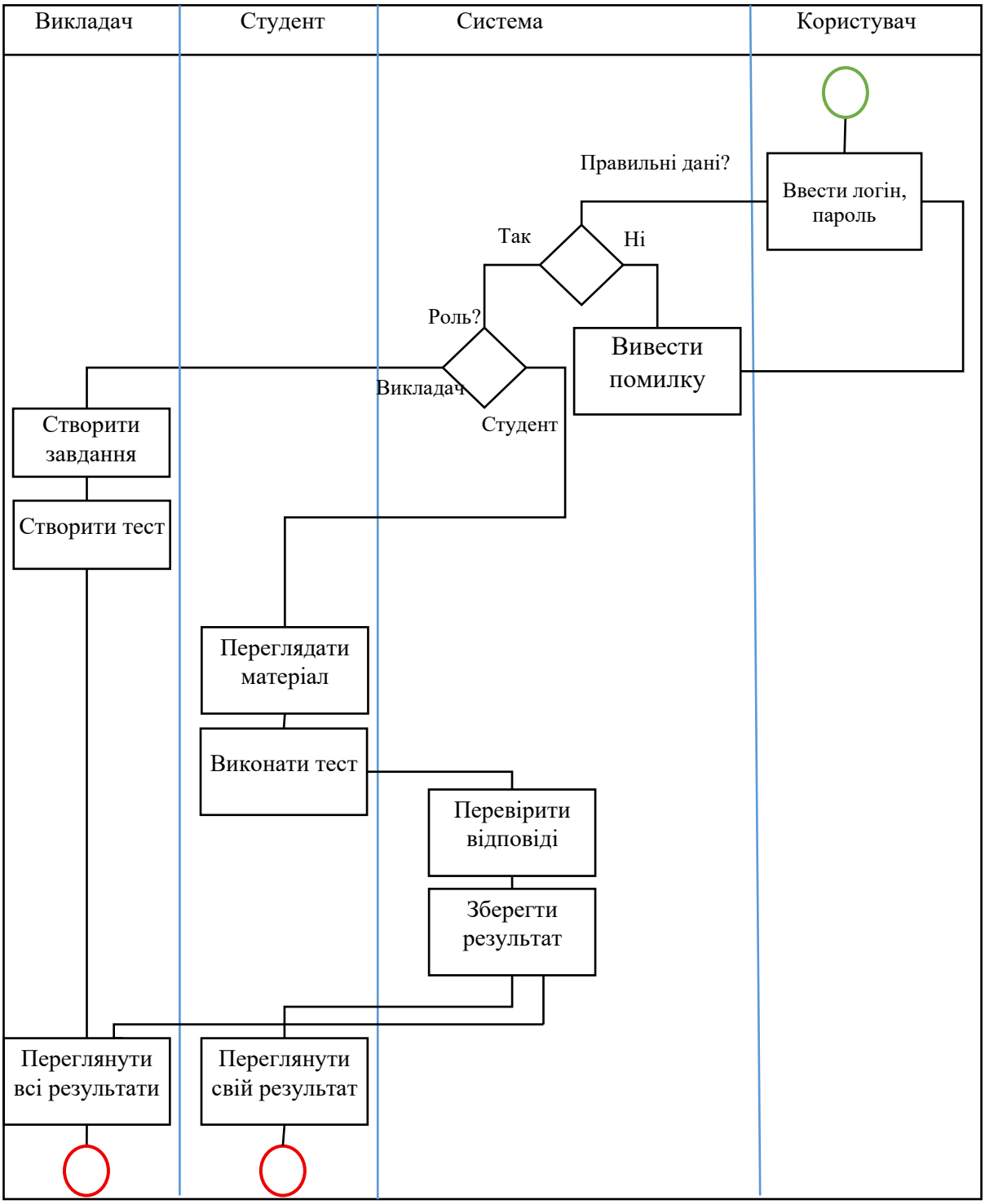

Рисунок 2.5 – BPMN діаграма

2.5 Діаграма сутність-зв’язок

Діаграма сутність-зв’язок (ERD або ER-діаграма) – діаграма, яка відображає як головні об’єкти (сутності) проєктованої системи взаємодіють між собою. Ці діаграми використовують для кращого розуміння архітектури систем та баз даних.[5]

На рисунку 2.6 зображена ER-діаграма з такими сутностями: Користувач, Заняття, Тест, Курс, Питання та Відповідь. Далі розглянемо зв’язки між сутностями із зазначенням їхньої модальності:

- Один користувач може проходити декілька курсів, але кожен курс проходить конкретний користувач. Тому зв’язок – «один до багатьох»;
- Один користувач може проходити багато тестів, але кожен тест проходиться конкретним користувачем. Тому зв’язок – «один до багатьох»;
- Один курс може містити багато занять, але кожне заняття належить лише до одного курсу. Тому зв’язок – «один до багатьох»;
- Одне заняття може містити багато тестів, але кожен тест належить лише одному заняттю. Тому зв’язок – «один до багатьох»;
- Один тест може містити багато запитань, але кожне запитання належить лише одному тесту. Тому зв’язок – «один до багатьох»;
- Одне запитання може мати багато варіантів відповідей, але кожна відповідь належить лише одному запитанню. Тому зв’язок – «один до багатьох».

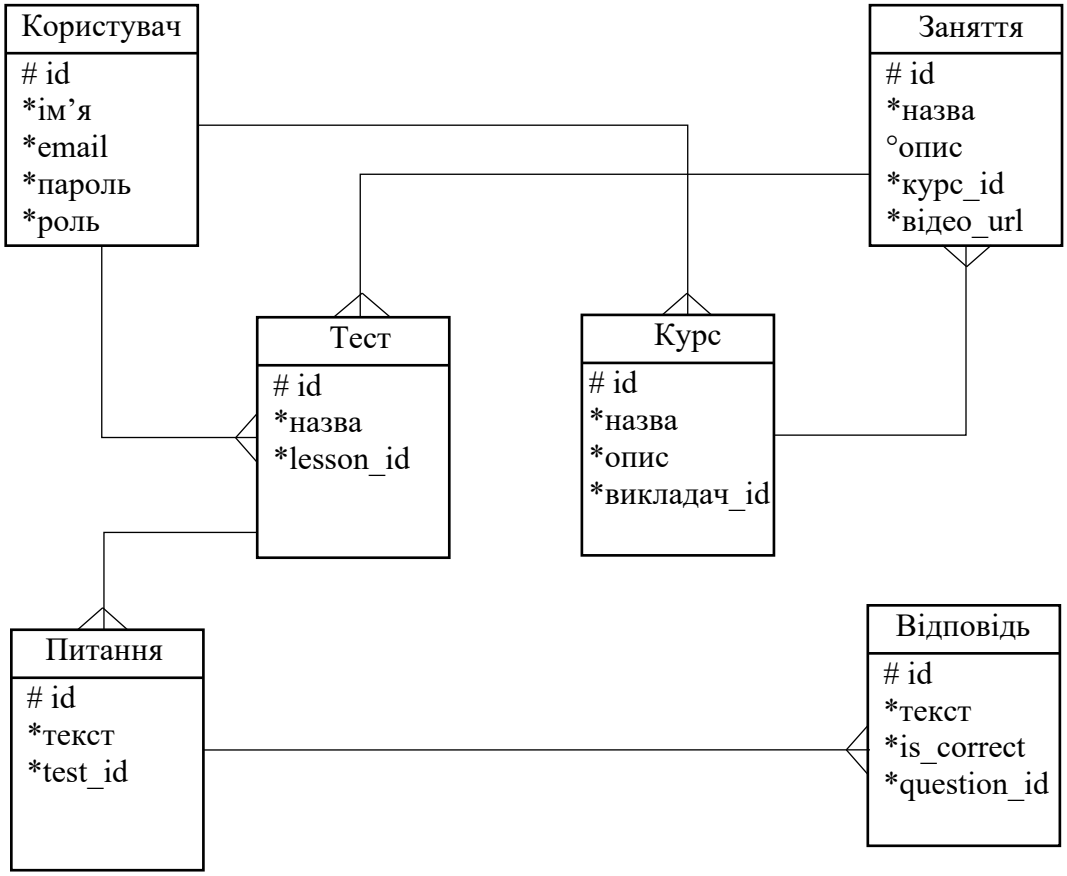

Рисунок 2.6 – Діаграма сутність-зв’язок

3 РОЗРОБКА КОМПОНЕНТІВ

3.1 Основна структура програми

Проєкт базується на архітектурі клієнт-сервер, що забезпечує чітку межу між користувацьким інтерфейсом (фронтенду) та обробленням даних, доступу до них (бекенду).

Клієнтська частина. Вона написана на React.js – сучасній бібліотеці JavaScript для створення інтерфейсів користувача. Усі сторінки реалізовані у вигляді компонентів, що полегшує розширення і підтримку коду. Стилізація виконана з використанням Bootstrap та власних CSS стилів. Також налаштовано маршрутизацію за допомогою react-router-dom, що дозволяє здійснити навігацію між сторінками без обов'язкового перезавантаження.

Серверна частина. Вона написана на Node.js та фреймворку Express.js. Також створено REST API для роботи з базою даних та обробки запитів. Для зберігання інформації про користувачів, курсів та тестів використовується база даних PostgreSQL, перевірена часом реляційна система управління базами даних (СУБД). Запроваджено JWT-автентифікацію з метою обмеження доступу до сторінок згідно з ролями користувачів ().

Основна структура проєкту виглядає наступним чином:

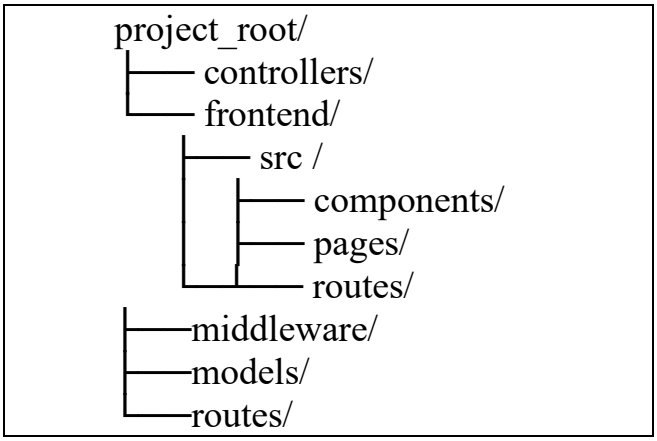

Рисунок 3.1 – Схема структури програмного проєту

Кожна з цих папок відповідає за окрему логіку додатку

- controllers/ Тут зосереджена серверна частина, яка реагує на HTTP-запити. Вона визначає, як опрацьовуються запити на авторизацію, створення курсів, завдань та тестів. Кожен контролер відповідає за окрему функцію та взаємодіє з моделями, що зберігають дані в базі.
- frontend/ Це основна директорія фронтенду, розробленого з використанням React. У ній зберігаються всі файли, необхідні для створення інтерфейсу користувача
- src/ Тут розташовані всі вихідні файли React. У цьому місці зосереджується основний процес розробки візуальної частини сайту:
- components/ – містить компоненти, які можна використовувати повторно, наприклад кнопки, форми та інші. Ці компоненти застосовуються на різних сторінках;
- pages/ – розміщує повноцінні сторінки застосунку, котрі відображаються відповідно до маршруту (тобто шляху, зазначеного в адресному рядку). Наприклад: LoginPage.js, AdminPage.js та інші.
- routes/ Це частина серверного коду, що визначає маршрути та керує взаємодією з базою даних, використовуючи моделі та контролери;
- middleware/ – містить проміжне програмне забезпечення, яке здійснює перевірку авторизації, прав доступу користувачів, токенів та інших аспектів безпеки;
- models/ – тут знаходиться модель бази даних, що описує структуру таблиць у PostgreSQL;
- routes/ – містить маршрути API, які зв’язують URL-адреси з відповідними функціями з контролерів.

3.2 Класи для роботи з базами даних

У цьому проєкті реалізовано централізований підхід підключення до бази даних PostgreSQL. У каталозі models/ міститься файл, відповідальний за конфігурацію пул з'єднань з базою даних. З цією метою використовується модуль pg, який є офіційною бібліотекою для роботи з PostgreSQL у середовищі Node.js.

Лістинг 3.1

```
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  connectionString: process.env.DATABASE_URL
});

module.exports = pool;
```

Цей файл створює пул з’єднань – набір активних підключень до бази даних, які використовуються повторно для обробки запитів. Це збільшує швидкість роботи та зменшує час відгуку за великого навантаження на сервер.

Використання dotenv гарантує зберігання налаштувань у середовищі (.env), що підвищує захищеність та дає змогу швидко переналаштовувати систему без потреби змінювати код.

Підключення виокремлено в додатковий модуль для зручного імпортування та використання в кожному маршруті чи контролері, який потребує взаємодії з базою даних.

3.3 Класи для роботи з налаштуваннями

У розробці застосовується конфігураційний файл .env, призначений для зберігання параметрів, чутливих до середовища розгортання. Ці параметри включають:

– Адреса бази даних (рядок з'єднання);
– Секретний ключ для токенів (JWT).

Для імпорту змінних середовища використовується популярна бібліотека dotenv. Під час ініціалізації серверної частини коду викликається метод `require('dotenv').config()`. Цей метод зчитує вміст файлу .env, парсить його та динамічно додає визначені змінні до глобального об'єкта `process.env`.

Такий підхід дає змогу уникнути прямого зберігання секретних даних у програмному коді. . Це суттєво підвищує рівень безпеки застосунку, спрощує процес безперервної інтеграції та розгортання (CI/CD), а також робить конфігурацію системи гнучкою та адаптивною до зміни інфраструктури.

3.4 Автентифікація

У проєкті реалізовано комплексну систему автентифікації та авторизації користувачів на основі використання JWT-токенів (JSON Web Token). Оскільки архітектура застосунку побудована за принципом REST API, використання JWT дозволяє реалізувати концепцію Stateless (збереження стану на стороні клієнта), що знижує навантаження на сервер та покращує масштабованість системи. Токен забезпечує безпечне розпізнавання користувача під час кожного запиту та чітке визначення його ролі й відповідних обмежень на доступ до ресурсів.

Для автентифікації передбачено два ключові маршрути:

– POST /api/auth/register – для створення нового облікового запису;
– POST /api/auth/login – для перевірки облікових даних та видачі токена.

У процесі входу користувача, сервер здійснює верифікацію електронної пошти та пароля. За умови успішної перевірки, генерується JWT-токен, що містить зашифровану інформацію про користувача (ID та роль). Токен підписується секретним ключем і надсилається клієнту.

Для розмежування прав доступу (авторизації) до захищених маршрутів API було розроблено спеціальні проміжні функції (middleware). Вони перехоплюють запит, вилучають токен, декодують його та верифікують за допомогою секретного ключа. У разі, якщо токен є дійсним, актуальним (не вичерпано термін дії exp) та містить необхідний рівень доступу, middleware передає керування наступному обробнику маршруту. Якщо токен відсутній або невалідний, сервер повертає помилку з HTTP-статусом 401 Unauthorized або 403 Forbidden.

В системі визначено такі ролі користувачів:

– Гість (незареєстрований користувач);
– Студент (зареєстрований користувач);
– Викладач (зареєстрований користувач з розширеним правами).

Кожен з них має відповідний функціонал. Викладач створює курси, заняття та тести, тоді як учні можуть їх лише проходити. А гості можуть переглядати головну сторінку, зареєструватися та потім авторизуватися.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Серверна частина

Серверна частина веб-орієнтованої системи онлайн-навчання реалізована на платформі Node.js із використанням фреймворку Express.js. Архітектура бекенду побудована за принципами REST API.

Головною точкою входу в додаток є файл server.js, який ініціалізує Express-сервер, налаштовує глобальні проміжні шари (Middleware), підключає маршрутизатори та запускає прослуховування HTTP-портів. Для забезпечення безпеки конфігураційні дані такі як секрети JWT, параметри підключення до СКБД, завантажуються динамічно зі змінних середовища .env за допомогою бібліотеки dotenv

Лістинг 4.1

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");

const authRoutes    = require("./routes/auth.routes");
const coursesRoutes = require("./routes/courses.routes");
const lessonsRoutes = require("./routes/lessons.routes");

const app = express();

app.use(cors());
app.use(express.json());

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

app.use("/api/courses", coursesRoutes);
app.use("/api/lessons", lessonsRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port
${PORT}`));
```

У лістингу 4.1 підключено механізм CORS для розв'язання проблеми крос-доменних запитів між фронтендом та бекендом, а також активовано парсер JSON-тіла запитів (express.json()).

Логіка обробки запитів, пов'язаних із життєвим циклом облікових записів користувачів, винесена в окремий шар контролерів. У системі реалізовано 2 основні функції: register та login.

Реєстрація (register): Приймає вхідні дані нового користувача. Задля криптографічного захисту персональних даних пароль користувача ніколи не зберігається у відкритому вигляді – перед записом у базу даних він піддається односторонньому асиметричному хешуванню за допомогою алгоритму bcrypt. Також впроваджено правило безпеки, а саме при реєстрації через загальну форму всі нові профілі за замовчуванням створюються із роллю student. Після успішного збереження генерується стартовий JWT-токен.

Авторизація (login): Перевіряє наявність користувача за його унікальним email. За допомогою методу bcrypt.compare здійснюється порівняння введеного пароля із захешованим значенням з бази даних. У разі збігу клієнту повертається підписаний JWT-токен та дані користувача, при цьому об'єкт пароля безпечно видаляється з відповіді сервера.

Лістинг 4.2

```
const bcrypt = require("bcrypt");
const { getUserByEmail, createUser, getUserById, updateUser } =
require("../models/user.model");
const { generateToken } = require("../utils/jwt");

async function register(req, res) {
  try {
    const { name, email, password, role } = req.body;

    if (!name || !email || !password) {
      return res.status(400).json({ message: "Заповніть всі
поля" });
    }

    const existing = await getUserByEmail(email);
    if (existing) {
      return res.status(409).json({ message: "Email вже
використовується" });
    }
```

```
    const safeRole = role === "teacher" ? "student" :
"student";

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await createUser({ name, email, password:
hashedPassword, role: safeRole });

    const token = generateToken({ id: user.id, email:
user.email, role: user.role });
    res.status(201).json({ user, token });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function login(req, res) {
  try {
    const { email, password } = req.body;

    const user = await getUserByEmail(email);
    if (!user) {
      return res.status(401).json({ message: "Невірний email
або пароль" });
    }

    const valid = await bcrypt.compare(password,
user.password);
    if (!valid) {
      return res.status(401).json({ message: "Невірний email
або пароль" });
    }

    const token = generateToken({ id: user.id, email:
user.email, role: user.role });
    const { password: _, ...safeUser } = user;

    res.json({ user: safeUser, token });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}
```

4.2 Клієнтська частина

Клієнтська частина веб-орієнтованої системи онлайн-навчання розроблена на базі бібліотеки React.js.

Для стилізації інтерфейсу використано методологію CSS-in-JS за допомогою бібліотеки styled-components, яка дозволяє інкапсулювати стилі всередині відповідних UI-компонентів, також підключити глобальні стилі (GlobalStyles) та забезпечити централізоване керування візуальною темою додатка (theme).

Для всіх користувачів є однакова головна сторінка, яка розказує про веб-додаток, його особливості та переваги

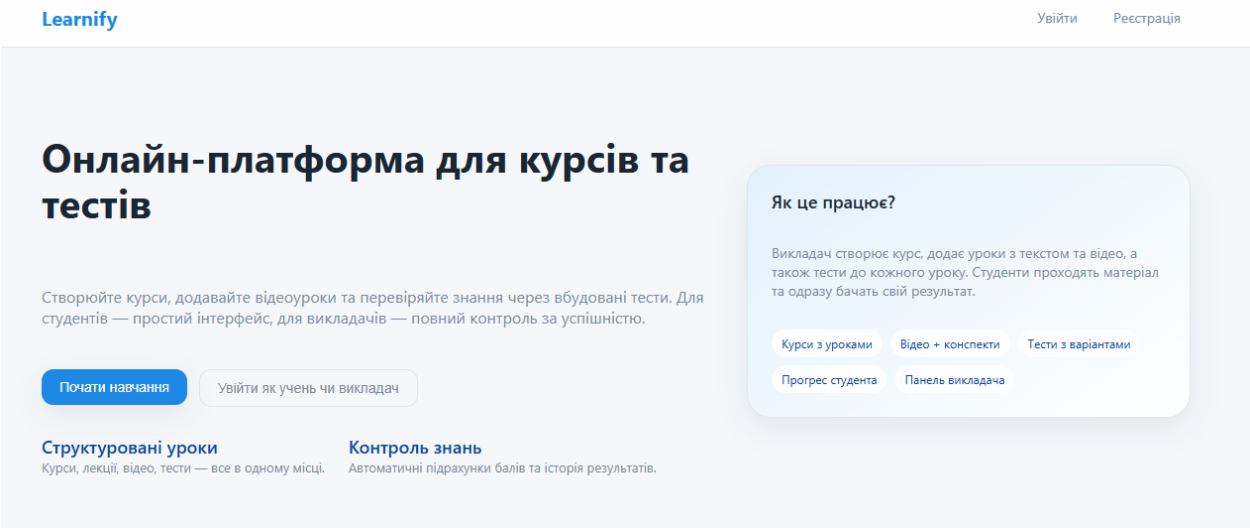

Рисунок 4.1 – Головна сторінка додатку

Для отримання доступу до персонального кабінету та навчальних матеріалів користувач повинен заповнити формою входу, що відображено на рисунку 4.2.

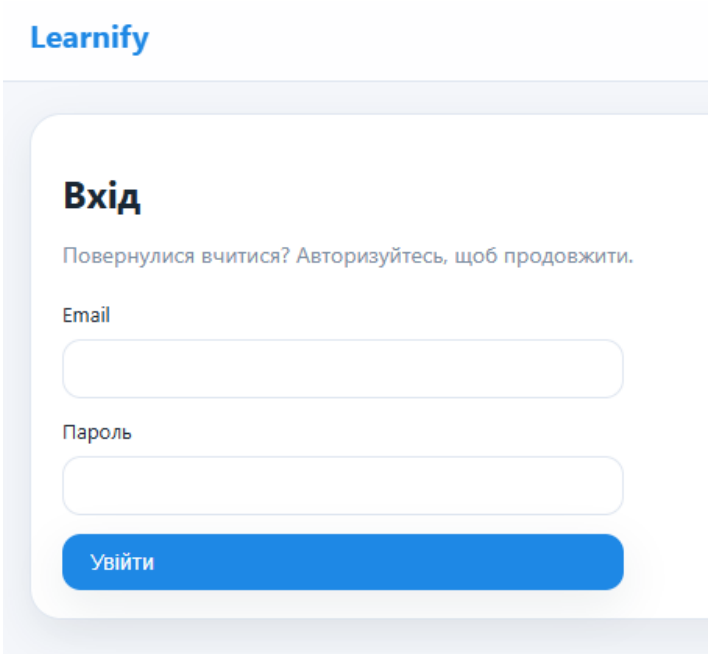

Рисунок 4.2 – Форма авторизації

Головним завданням викладача є створення курсів, занять та тестів. Для кожного цього завдання розроблені окремі форми з відповідними полями та інструментами. На рисунку 4.3 зображена форма для створення нового курсу.

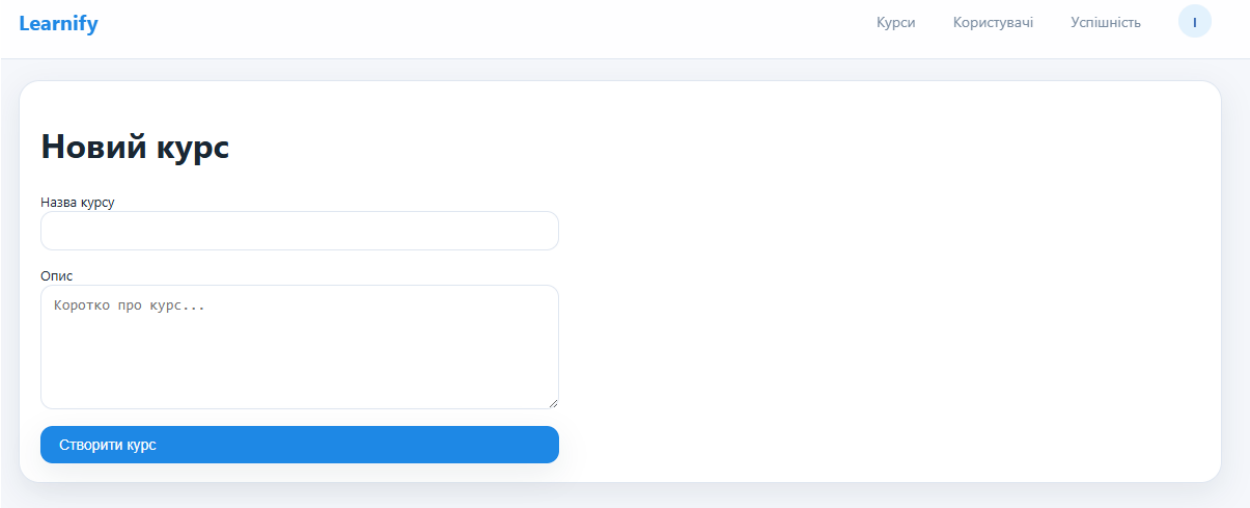

Рисунок 4.3 – Форма створення курсу

Кожен студент після опрацювання завдання має пройти відповідний тест для закріплення матеріалу по цій темі. На рисунку 4.4 зображена сторінка проходження тесту

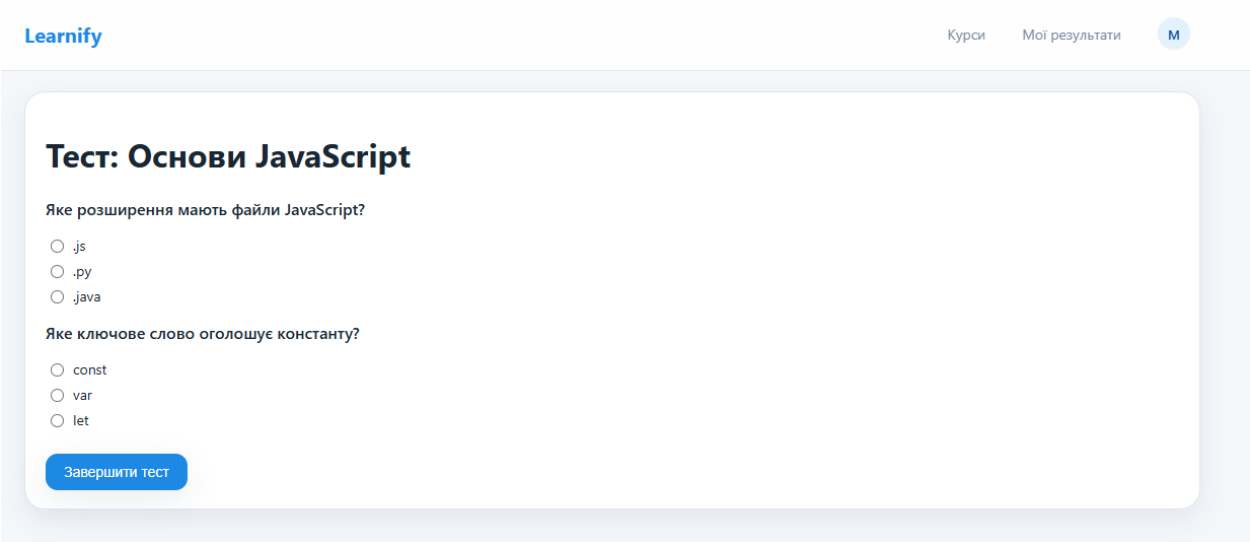

Рисунок 4.4 – Сторінка проходження тесту

4.3 Тестування

Після реалізації основного функціоналу було організовано низку тестувань аби перевірити роботу додатку, його стійкість та відповідність потребам користувачів.

Типи тестувань, які були проведені на різних етапах розробки:

– Функціональне тестування;
– Тестування безпеки;
– Тестування інтерфейсу (UI-тестування).

Розглянемо детальніше кожен вид тестування застосований у даному проєкті.

Функціональне тестування. Найперше тестування, яке було проведене через програми Postman. Було виконано перевірку коректності роботи всіх ключових можливостей:

– Реєстрація нового користувача;
– Авторизація користувача з різними правами доступу;
– Показ курсів, занять та тестів;
– Додавання нових курсів, занять, тесів;

– Обробка некоректних даних (порожні поля, невірний email та інше).

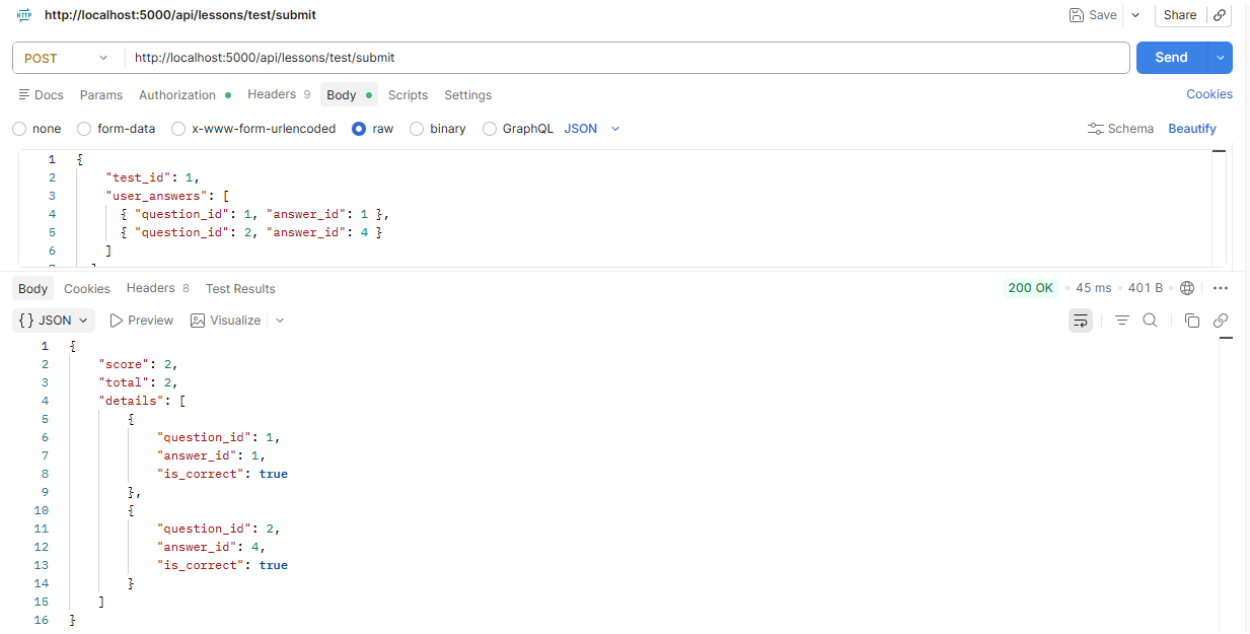

Рисунок 4.5 – Скріншот з програми Postman

Наступним етапом проводилося тестування безпеки, яке також проводилося також через програму Postman. Найголовнішими моментами перевірки були:

– Використання JWT-токенів забезпечує автентифікацію без передачі пароля;
– Захист маршрутів на основі ролі;
– Перевірка на обробку несанкціонованого доступу до захищених сторінок;
– Перевірка чи відхиляє сервер спроби додати або змінити дані без токена.

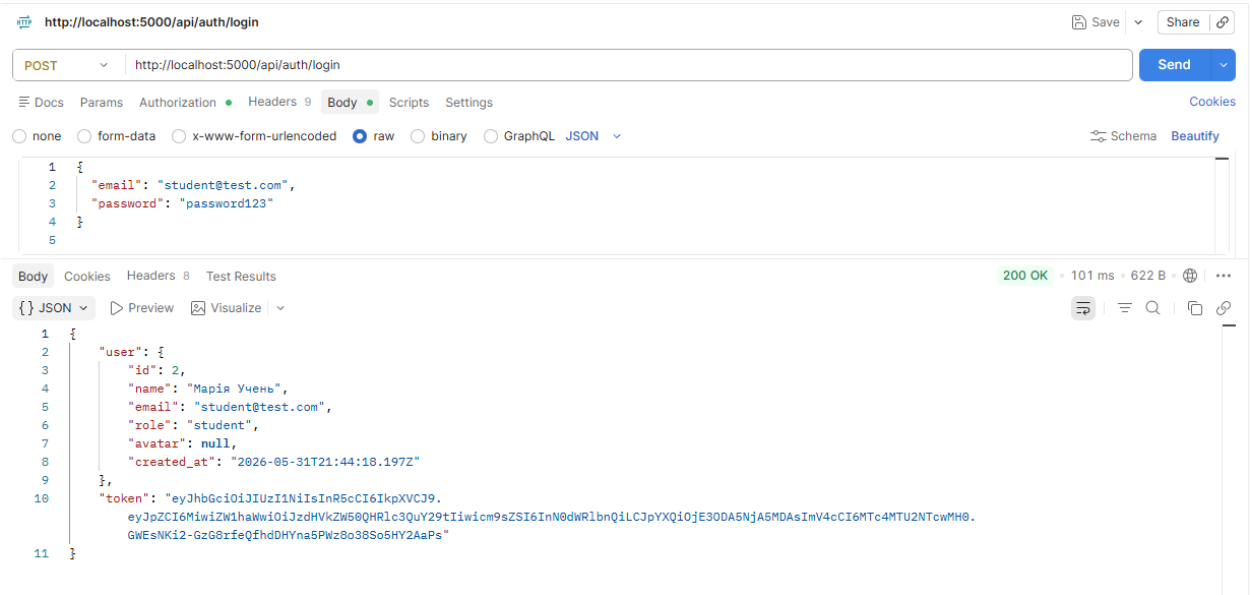

Рисунок 4.6 – Скріншот з програми Postman

Останнім проводилось *тестування інтерфейсу (UI-тестування)* після створення клієнтського інтерфейсу. Основними пунктами перевірки були:

– Зручність навігації;
– Адаптивність до різних розмірів екрана;
– Узгодженість стилів та елементів;
– Видимість і доступність кнопок та полів введення;
– Реакція системи на взаємодію з елементами (наведення, кліки тощо).

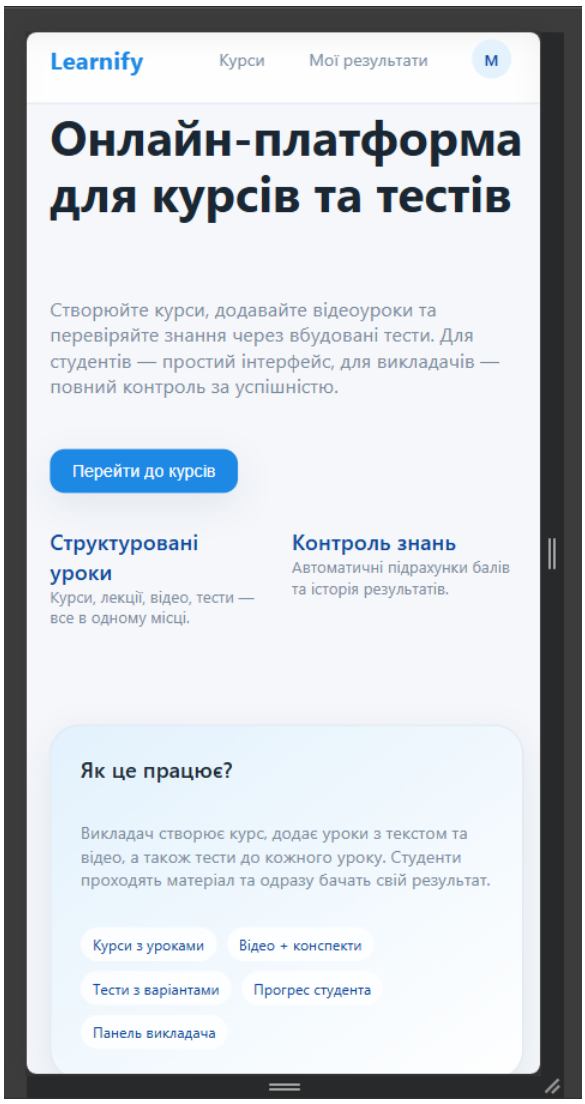

Рисунок 4.7 – Перевірка адаптивності сторінки

Результати тестувань:

- Усі ключові функції працюють без збоїв;
- Додаток не має помилок, що впливають на роботу;
- Інтерфейс зручний та інтуїтивно зрозумілий;
- Інтерфейс правильно відображається як на комп'ютерах, так і на мобільних платформах;
- Реакція на помилки, що виникають у користувачів, відбувається шляхом відображення відповідних повідомлень.

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи було успішно розроблено веб-орієнтовану систему онлайн-навчання. Додаток реалізує всі поставлені вимоги: підтримує повну логіку дистанційного освітнього процесу (перегляд лекцій, інтеграцію медіаконтенту, проходження тестів), дозволяє реєструвати нових користувачів із розмежуванням ролей (гість, студент, викладач), зберігати історію успішності й результатів у базі даних та забезпечує зручний інтуїтивно зрозумілий інтерфейс.

В процесі розробки було застосовано сучасні технології та підходи інтернет-програмування: бібліотека React.js для клієнтської частини, платформа Node.js із фреймворком Express для серверної частини, СКБД PostgreSQL для роботи з даними, методологія styled-components для безпечного стилювання елементів інтерфейсу, а також архітектура REST API для взаємодії компонентів. Особливу увагу було приділено безпеці даних (через валідацію JWT-токенів та хешування паролів за допомогою bcrypt) та стабільності системи (через використання пулу з'єднань Pool для фонових операцій з БД).

Реалізована система реєстрації та автентифікації дозволяє надійно ідентифікувати користувачів і розмежовувати їхні права доступу до приватних і адміністративних маршрутів, що гарантує конфіденційність інформації. Можливість автоматичного оцінювання тестів та перегляду журналів успішності дає викладачам змогу ефективно моніторити прогрес, а учням – аналізувати свої помилки та вдосконалювати знання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Найпопулярніші освітні платформи для організації дистанційного навчання URL: https://vseosvita.ua/blogs/naipopuliarnishi-osvitni-platformy-dlia-orhanizatsii-dystantsiinoho-navchannia-102669.html (дата звернення: 01.03.2026);
2. 5 платформ для організації дистанційного навчання URL: https://buki.com.ua/news/5-platform-dlya-orhanizatsiyi-dystantsiynoho-navchannya/(дата звернення: 01.03.2026);
3. Node.js vs PHP: Making the right choice for your project URL: https://www.lightflows.co.uk/blog/node-js-vs-php/ (дата звернення: 15.04.2025);
4. Що таке BPMN-діаграма і навіщо вона потрібна з прикладами URL: https://iampm.club/ua/blog/shho-take-bpmn-diagrama-i-navishho-vona-potribna-z-prikladami-2/(дата звернення: 15.04.2025);
5. Д. Б. Буй, Л. М. Сільвейструк ФОРМАЛІЗАЦІЯ МОДЕЛІ "СУТНІСТЬ-ЗВ'ЯЗОК": Київ, 2011. 174 с;
6. М. Дідковська Проектування програмного забезпечення засобами UML: Київ, 10 с;
7. React JavaScript-бібліотека для створення користувацьких інтерфейсів URL: https://uk.legacy.reactjs.org/ (дата звернення: 18.04.2025);
8. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту URL: https://dou.ua/forums/topic/51621/ (дата звернення: 01.03.2025);
9. Н. Ф. Хайрова, С. В. Петрасова СУЧАСНІ ТЕХНОЛОГІЇ WEB-ПРОГРАМУВАННЯ: Харків, 2020. 112с;

ДОДАТКИ

Додаток 1 – auth.controller.js

```
const bcrypt = require("bcrypt");
const { getUserByEmail, createUser, getUserById, updateUser } =
require("../models/user.model");
const { generateToken } = require("../utils/jwt");

async function register(req, res) {
  try {
    const { name, email, password, role } = req.body;

    if (!name || !email || !password) {
      return res.status(400).json({ message: "Заповніть всі поля" });
    }

    const existing = await getUserByEmail(email);
    if (existing) {
      return res.status(409).json({ message: "Email вже
використовується" });
    }

    const safeRole = role === "teacher" ? "student" : "student";

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await createUser({ name, email, password:
hashedPassword, role: safeRole });

    const token = generateToken({ id: user.id, email: user.email, role:
user.role });
    res.status(201).json({ user, token });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function login(req, res) {
  try {
    const { email, password } = req.body;

    const user = await getUserByEmail(email);
    if (!user) {
      return res.status(401).json({ message: "Невірний email або пароль"
});
    }

    const valid = await bcrypt.compare(password, user.password);
    if (!valid) {
```

```
      return res.status(401).json({ message: "Невірний email або пароль"
});
    }

    const token = generateToken({ id: user.id, email: user.email, role:
user.role });
    const { password: _, ...safeUser } = user;

    res.json({ user: safeUser, token });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function getMe(req, res) {
  try {
    const user = await getUserById(req.user.id);
    if (!user) return res.status(404).json({ message: "Користувача не
знайдено" });
    res.json(user);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function updateProfile(req, res) {
  try {
    const { name, avatar } = req.body;
    const updated = await updateUser(req.user.id, { name, avatar });
    res.json(updated);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

module.exports = { register, login, getMe, updateProfile };
```

Додаток 2 – courses.controller.js

```
const {
  getAllCourses, getCourseById,
  createCourse, updateCourse, deleteCourse
} = require("../models/course.model");
```

```
const { getAllUsers, deleteUser } = require("../models/user.model");

async function getCourses(req, res) {
  try {
    const courses = await getAllCourses();
    res.json(courses);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function getCourse(req, res) {
  try {
    const course = await getCourseById(req.params.id);
    if (!course) return res.status(404).json({ message: "Курс не
знайдено" });
    res.json(course);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function addCourse(req, res) {
  try {
    const { title, description } = req.body;
    if (!title) return res.status(400).json({ message: "Назва курсу
обов'язкова" });
    const course = await createCourse({ title, description, teacher_id:
req.user.id });
    res.status(201).json(course);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function editCourse(req, res) {
  try {
    const updated = await updateCourse(req.params.id, req.body);
    if (!updated) return res.status(404).json({ message: "Курс не
знайдено" });
    res.json(updated);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function removeCourse(req, res) {
```

```
  try {
    await deleteCourse(req.params.id);
    res.json({ message: "Курс видалено" });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

// Управління користувачами (тільки teacher)
async function getUsers(req, res) {
  try {
    const users = await getAllUsers();
    res.json(users);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function removeUser(req, res) {
  try {
    await deleteUser(req.params.id);
    res.json({ message: "Користувача видалено" });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

module.exports = { getCourses, getCourse, addCourse, editCourse,
removeCourse, getUsers, removeUser };
```

Додаток 3 – lessons.controller.js

```
const {
  getLessonsByCourse, getLessonById,
  createLesson, updateLesson, deleteLesson
} = require("../models/lesson.model");

const pool = require("../db");

async function getLessons(req, res) {
  try {
    const lessons = await getLessonsByCourse(req.params.courseId);
    res.json(lessons);
  } catch (err) {
```

```
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function getLesson(req, res) {
  try {
    const lesson = await getLessonById(req.params.id);
    if (!lesson) return res.status(404).json({ message: "Урок не
знайдено" });
    res.json(lesson);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function addLesson(req, res) {
  try {
    const { course_id, title, video_url, content, order_index } =
req.body;
    if (!course_id || !title) {
      return res.status(400).json({ message: "course_id та title
обов'язкові" });
    }
    const lesson = await createLesson({ course_id, title, video_url,
content, order_index });
    res.status(201).json(lesson);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function editLesson(req, res) {
  try {
    const updated = await updateLesson(req.params.id, req.body);
    if (!updated) return res.status(404).json({ message: "Урок не
знайдено" });
    res.json(updated);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function removeLesson(req, res) {
  try {
    await deleteLesson(req.params.id);
    res.json({ message: "Урок видалено" });
  } catch (err) {
```

```
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

// Тести
async function getTestByLesson(req, res) {
  try {
    const result = await pool.query(
      `SELECT t.*,
        json_agg(
          json_build_object(
            'id', q.id, 'text', q.text,
            'answers', (
              SELECT json_agg(json_build_object('id', a.id, 'text',
a.text))
              FROM answers a WHERE a.question_id = q.id
            )
          )
        ) AS questions
       FROM tests t
       LEFT JOIN questions q ON q.test_id = t.id
       WHERE t.lesson_id = $1
       GROUP BY t.id`,
      [req.params.lessonId]
    );
    res.json(result.rows[0] || null);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function createTest(req, res) {
  try {
    const { lesson_id, title, questions } = req.body;
    // questions: [{ text, answers: [{ text, is_correct }] }]

    const testResult = await pool.query(
      "INSERT INTO tests (lesson_id, title) VALUES ($1, $2) RETURNING
*",
      [lesson_id, title]
    );
    const test = testResult.rows[0];

    for (const q of questions) {
      const qResult = await pool.query(
        "INSERT INTO questions (test_id, text) VALUES ($1, $2) RETURNING
id",
        [test.id, q.text]
      );
```

```
      const questionId = qResult.rows[0].id;

      for (const a of q.answers) {
        await pool.query(
          "INSERT INTO answers (question_id, text, is_correct) VALUES
($1, $2, $3)",
          [questionId, a.text, a.is_correct]
        );
      }
    }

    res.status(201).json({ message: "Тест створено", test });
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function submitTest(req, res) {
  try {
    const { test_id, user_answers } = req.body;
    // user_answers: [{ question_id, answer_id }]

    const correctAnswers = await pool.query(
      `SELECT a.id, a.question_id, a.is_correct
       FROM answers a
       JOIN questions q ON a.question_id = q.id
       WHERE q.test_id = $1 AND a.is_correct = TRUE`,
      [test_id]
    );

    const correctMap = {};
    for (const row of correctAnswers.rows) {
      correctMap[row.question_id] = row.id;
    }

    let score = 0;
    const snapshot = user_answers.map((ua) => {
      const isCorrect = correctMap[ua.question_id] === ua.answer_id;
      if (isCorrect) score++;
      return { ...ua, is_correct: isCorrect };
    });

    await pool.query(
      `INSERT INTO test_results (user_id, test_id, score,
answers_snapshot)
       VALUES ($1, $2, $3, $4)`,
      [req.user.id, test_id, score, JSON.stringify(snapshot)]
    );

    res.json({ score, total: user_answers.length, details: snapshot });
```

```
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function getMyResults(req, res) {
  try {
    const result = await pool.query(
      `SELECT tr.*, t.title AS test_title, l.title AS lesson_title
       FROM test_results tr
       JOIN tests t ON tr.test_id = t.id
       JOIN lessons l ON t.lesson_id = l.id
       WHERE tr.user_id = $1
       ORDER BY tr.created_at DESC`,
      [req.user.id]
    );
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

async function getAllResults(req, res) {
  try {
    const result = await pool.query(
      `SELECT tr.*, u.name AS student_name, u.email,
              t.title AS test_title, l.title AS lesson_title
       FROM test_results tr
       JOIN users u ON tr.user_id = u.id
       JOIN tests t ON tr.test_id = t.id
       JOIN lessons l ON t.lesson_id = l.id
       ORDER BY tr.created_at DESC`
    );
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ message: "Помилка сервера", error:
err.message });
  }
}

module.exports = {
  getLessons, getLesson, addLesson, editLesson, removeLesson,
  getTestByLesson, createTest, submitTest, getMyResults, getAllResults
};
```