Міністерство освіти і науки
України Національний
технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра
(бакалавра, магістра)

Здобувача вищої освіти Атрошенко Андрія Анатолійовича
(ПІБ)

академічної групи 126-21-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)

(офіційна назва)

на тему Розробка online-сервісу тестування знань здобувачів

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Доц. Гаркуша І.М.			
розділів:				
Розділ 1				
Розділ 2				

Рецензент				

Нормоконтролер	Проф. Коротенко Г.М.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних систем та технологій та комп'ютерної інженерії
(повна назва)

_____ В.В. Гнатушенко
(підпис) (ініціали та прізвище)

«05» травня 2025 року

ЗАВДАННЯ

на кваліфікаційну роботу ступеня бакалавра

(бакалавра, магістра)

здобувача вищої освіти Атрошенко А.А. **академічної групи** 126-21-1
(прізвище та ініціали) (шифр)

спеціальності 126 «Інформаційні системи та технології»

спеціалізації за освітньою-професійною програмою _____
(за наявності)

на тему Розробка online-сервісу тестування знань здобувачів

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

Розділ	Зміст	Термін виконання
Розділ 1. Дослідження предметної області	Дослідження предметної області, огляд існуючих рішень, огляд технологій, формування задачі.	21.04.2025-10.05.2025
Розділ 2. Проєктні рішення	1. Проєктування системи, опис її роботи. 2. Опис бази даних. 3. Опис розробки backend-частини системи. 4. Опис розробки frontend-частини системи. 5. Опис результатів тестування розробленої системи.	11.05.2025-08.06.2025

Завдання видано _____ Гаркуша І.М
(підпис керівника) (ініціали та прізвище)

Дата видачі 20.03.2025

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Атрошенко А.А.
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 110 с., 33 рис., 1 табл., 3 додатки, 16 джерел.

АВТОРИЗАЦІЯ, ТЕСТУВАННЯ ЗНАНЬ, JAVASCRIPT, NODE.JS, ONLINE-СЕРВІС, POSTGRESQL, REACT, REST API.

Об'єкт кваліфікаційної роботи: online-сервіс для тестування знань здобувачів.

Предмет кваліфікаційної роботи: процес розробки Web-сервісу із використанням REST API та реляційної бази даних.

Мета роботи: створення простого та зручного сервісу для проведення тестування.

У вступі розглянуто стан проблеми та описано цінність рішення такої проблеми.

У першому розділі наведені приклади вже існуючих проєктних рішень та розглянуті можливості, які інші проєкти надають. Розглянуто переваги та недоліки цих систем, особливості технологій, які використовувались для їх розробки.

У другому розділі описано проєктні рішення, які були прийняті під час розробки проєкту з описом взаємодії елементів системи та особливості роботи із системою. Було проведено тестування створеної системи за допомогою платформи Postman.

ABSTRACT

Explanatory note: 110 pages, 33 figures, 1 table, 3 applications, 16 resources.

AUTHORIZATION, JAVASCRIPT, KNOWLEDGE TESTING, NODE.JS, ONLINE SERVICE, POSTGRESQL, REACT, REST API.

Object of the qualification work: an online service for knowledge testing of students.

Subject of the qualification work: the process of developing a web service using a REST API and a relational database.

Purpose of the work: to create a simple and convenient service for conducting testing.

The introduction examines the current state of the problem and describes the value of solving it.

The first chapter presents examples of existing project solutions and analyzes the features offered by other projects. The advantages and disadvantages of these systems are discussed, as well as the characteristics of the technologies used in their development.

The second chapter describes the design decisions made during the development of the project, including the interaction of system components and features of working with the system. Testing of the developed system was conducted using the Postman application.

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 9

1.1 Огляд різновидів сервісів для тестування 9

1.2 Аналіз існуючих систем управління навчанням 9

1.3 Аналіз існуючих сервісів, розроблених для створення тестів 12

1.4 Огляд можливостей створення тестів на основі інструментів Moodle 15

1.5 Огляд технологій, що використовувались у згаданих системах 18

1.6 Backend-технології розробки 20

1.7 Frontend-технології розробки 21

1.8 Підходи до створення API 22

1.9 СУБД для задач online-сервісів 23

1.10 Висновки до першого розділу. Постановка задачі на розробку сервісу 24

РОЗДІЛ 2 ПРОЄКТНІ РІШЕННЯ 27

2.1 Опис структури системи 27

2.2 Опис алгоритму взаємодії з системою 28

2.3 Діаграма розгортання системи 30

2.4 Діаграми послідовності системи 31

2.5 Опис бази даних 34

2.6 Опис технологій backend-фрагменту системи 36

2.7 Опис кінцевих точок backend-фрагменту 37

2.8 Опис контролерів backend-фрагменту 39

2.9 Опис технологій frontend-фрагменту системи 42

2.10 Огляд роботи з веб інтерфейсом 43

2.11 Тестування API та системи за допомогою Postman 51

ВИСНОВКИ 58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60

ДОДАТОК А. Програмний код 62

А.1 Текст програмного коду backend-частини 62

А.2 Текст програмного коду frontend-частини 80

ДОДАТОК Б. Відгук керівника кваліфікаційної роботи 109

ДОДАТОК В. Рецензія 110

ВСТУП

У сучасному інформаційному суспільстві, де цифрові технології все більше проникають у всі сфери людської діяльності, зростає кількість можливостей, які ці технології надають. Завдяки цьому розвитку способів для контакту між людьми стає все більше. Так само це стосується освітнього процесу, який також зазнає суттєвих трансформацій. У цій сфері діяльності обов'язковим є постійний зв'язок між учасниками процесу, тому, при умовах, які потребують дистанційної взаємодії, гнучких графіків роботи через обмеження, викликаних зовнішніми факторами є необхідним мати можливість переходу на альтернативні методи роботи та взаємодії з людьми.

Рішенням багатьох проблем стала дистанційна освіта та ресурси, які дозволяють перейти на форму дистанційного навчання. Вже створено багато сервісів для взаємодії між студентами та викладачами, які задовольняють потреби такого формату навчального процесу. Такі сервіси надають інструментарій для створення та проведення online зустрічей, призначення завдань, їхньої перевірки та загальної взаємодії між учасниками освітнього процесу.

Одним із найважливіших етапів освітнього процесу є тестування знань здобувачів, тому при дистанційному навчанні необхідно мати можливість проводити online тестування. Це зумовлює необхідність у нових, більш гнучких та ефективних формах освітньої взаємодії. Уже існуючі ресурси для online тестування забезпечують автоматизацію процесу перевірки знань, дозволяють швидко отримувати результати, зменшують навантаження на викладачів і дають змогу студентам проходити тестування у зручний для них час. Завдяки таким рішенням освіта стає доступнішою, динамічнішою та орієнтованою на індивідуальні потреби здобувача.

Велика кількість навчальних закладів уже використовують або планують впровадити подібні рішення. Проте не всі наявні системи відповідають вимогам зручності, масштабованості, безпеки та адаптивності.

Багато з них є складними у використанні, не мають можливості налаштування під конкретні освітні програми або не підтримують аналітику результатів.

У зв’язку з цим виникає потреба у створенні нового online-сервісу, який буде відповідати сучасним вимогам. Такий сервіс має забезпечувати інтуїтивно зрозумілий інтерфейс, підтримку різних типів завдань, а також зручну систему оцінювання результатів.

Розробка подібного сервісу потребує поєднання знань з різних галузей: програмування, проєктування баз даних, користувацького інтерфейсу. Під час розробки подібного програмного забезпечення необхідно звертати увагу як на технічні, так і на освітні вимоги.

Метою даної роботи є розробка online-сервісу, який надаватиме можливість проводити тестування знань здобувачів. Завдання передбачає створення системи, яка надає можливість створювати та керувати тестами, проходити їх та розмежовує користувачів на здобувачів та викладачів. Таким чином потрібно розробити логіку роботи системи у backend-частині проєкту, спроєктувати базу даних для збереження даних про користувачів та тести, надати зручний інтерфейс на frontend-частині системи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

У розділі проведено огляд сучасних систем тестування знань. Розглянуті системи проаналізовано з точки зору зручності використання і наданих можливостей для проведення тестування. Також розглянуто використані технології при розробці описаних систем. Представлений опис технологій, які можна використати для розробки проєкту. У висновках до розділу поставлено задачу на розробку online-сервісу та описано обраний стек технологій.

1.1 Огляд різновидів сервісів для тестування

При пошуку сервісів для тестування можна знайти багато ресурсів, які використовують з різною метою. В першу чергу потрібно розрізняти системи оцінювання знань. Для навчальних закладів поширеним є система управління навчанням, або LMS (Learning management system) [1], яка представляє програмне забезпечення, призначене для адміністрування, документування, відстеження, звітування, автоматизації та надання навчальних курсів, програм підготовки, навчальних матеріалів або програм з розвитку та навчання . Такі системи є складними проєктами, які можуть включати в себе інструменти для створення та проведення тестування у межах цієї системи. Такими системами є Moodle, Canvas LMS, Forma LMS тощо. Іншим варіантом для тестування є невеликі online-сервіси, які використовують для створення вікторин, тестів, опитувальників. Такі сервіси використовують у випадках, коли є недоцільним впроваджувати систему управління навчанням у заклади осівти або у якості розваги. Такими сервісами є Hot Potatoes, iSpring QuizMaker, Google Forms тощо.

Системи управління навчанням та online-сервіси відрізняються за метою створення, оскільки LMS створені з метою створення умов для переходу на дистанційне навчання, тому є доцільним розглядати ці сервіси незалежно один від одного.

1.2 Аналіз існуючих систем управління навчанням

Moodle (рис. 1.1) [2] — це платформа з відкритим програмним кодом, що призначена для організації навчального процесу в цифровому середовищі. Вона забезпечує можливість створення онлайн-курсів, тестових завдань і дозволяє реалізовувати як комбіноване навчання, так і повністю дистанційні курси. Платформу можна інсталювати на локальному сервері, що знижує залежність від інтернет-з'єднання та розширює можливості її використання.

За допомогою вбудованого конструктора контенту користувач може формувати тести, анкети, лекційні матеріали. За потреби до навчального контенту можна додавати відео-, аудіофайли та зображення, які завантажуються адміністратором у систему.

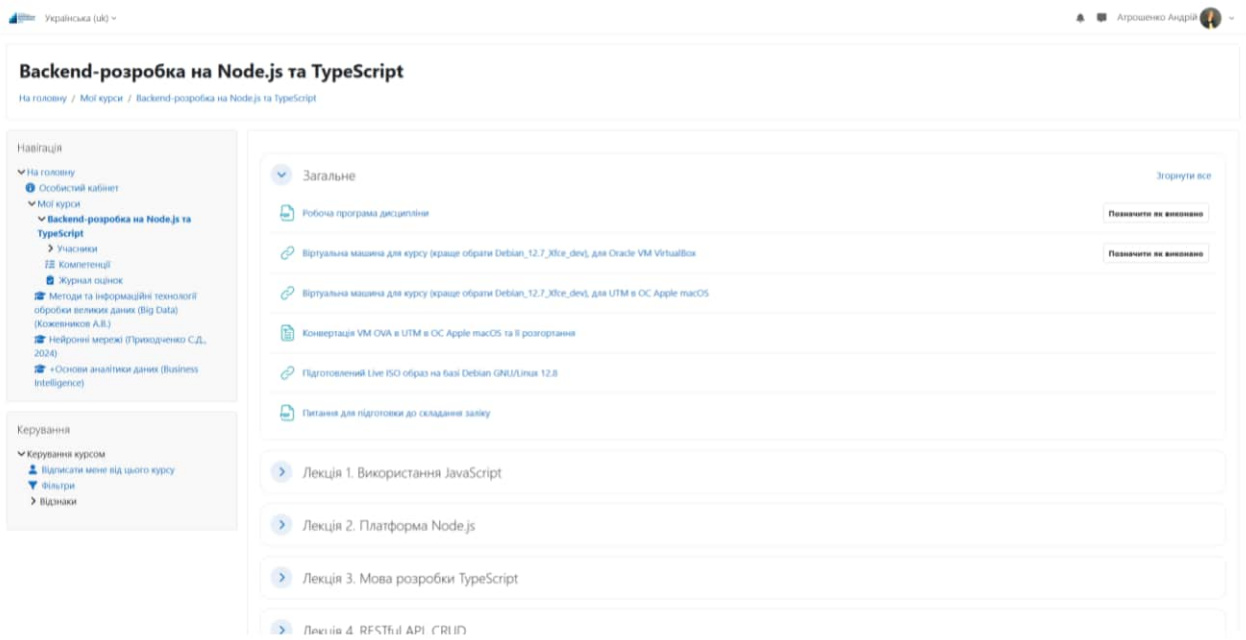


Рисунок 1.1 – Інтерфейс курсу на Moodle

Canvas LMS (рис. 1.2) [3] – це сучасна хмарна система управління навчанням, що орієнтована переважно на вищу освіту, але також активно використовується в школах, онлайн-курсах та корпоративному навчанні.

Інструментарій Canvas дозволяє створювати та адмініструвати курси, створювати тести та проводити оцінювання знань, також має вбудовані чати та інтеграцію з Zoom, що спрощує комунікацію. Значним недоліком є розділення на безкоштовну та комерційну версію, де безкоштовна версія має меншу потужність серверів та обмежений функціонал. Оскільки цей сервіс є хмарною системою, то робота з ним повністю залежить від підключення до глобальної мережі.

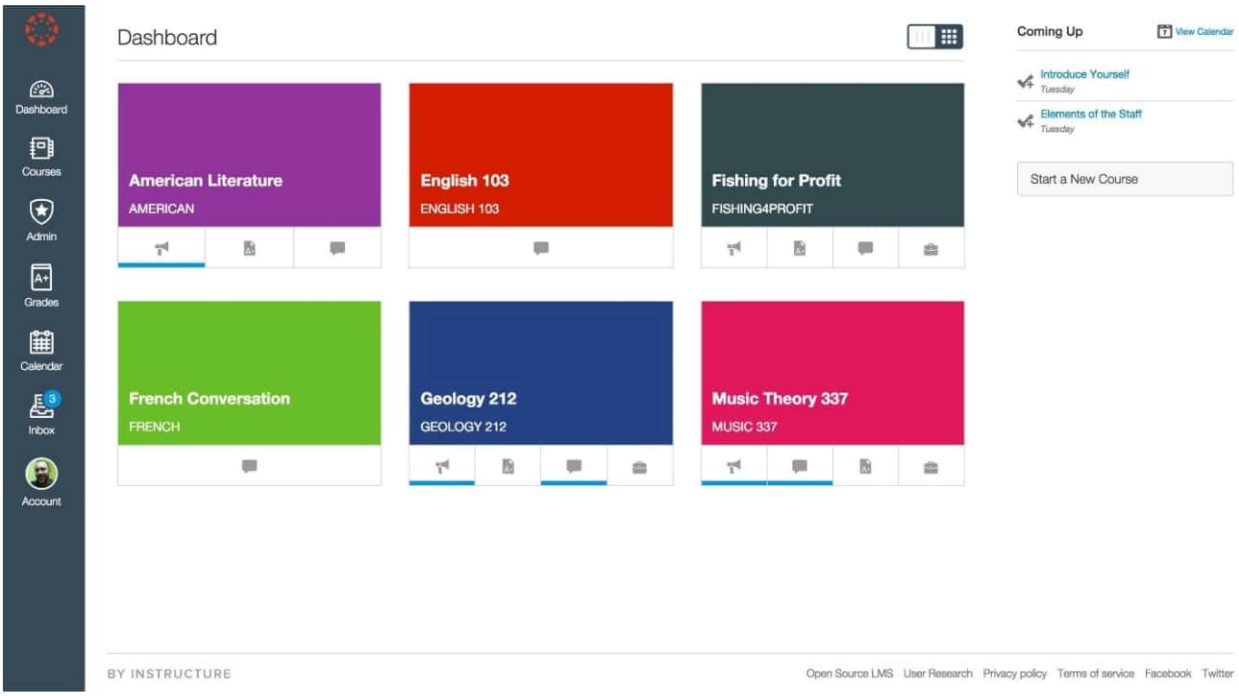


Рисунок 1.2 – Інтерфейс Canvas LMS

Forma LMS (рис. 1.3) [4] – це безкоштовна система управління навчанням з відкритим вихідним кодом, яка спочатку була орієнтована на корпоративний сегмент, але також використовується в освітніх установах.

Платформа надає можливість створювати курси, має вбудований редактор тестів, серед особливостей має систему ролей, що дозволяє чітко розділяти повноваження користувачів. Окрім цього Forma LMS має можливість автоматично видавати сертифікати після проходження курсів або успішного складання тестів.

Платформа може розгортатися як локально, так і можна знайти пропозиції оренди виділених серверів, що надає гнучкості у використанні системи.

Рисунок 1.3 – Інтерфейс Forma LMS

1.3 Аналіз існуючих сервісів, розроблених для створення тестів

Hot Potatoes (рис. 1.4) [5] – це набір інструментів для створення тестів та вправ. Програми дозволяють створювати декілька типів завдань: питання на вибір правильної відповіді, встановлення відповідностей, кросворди, вставку пропущених слів, сортування тощо.

Особливістю є можливість зберігати тести у вигляді HTML-сторінок, що робить їх придатними для інтеграції у вебсайти чи локальні навчальні ресурси, але це означає, що такий інструмент не є самостійним та для ефективного використання його потрібно реалізовувати у комбінації з іншими сервісами. Окрім цього, наданий інструментарій вже не відповідає сучасним потребам та має застарілий інтерфейс.

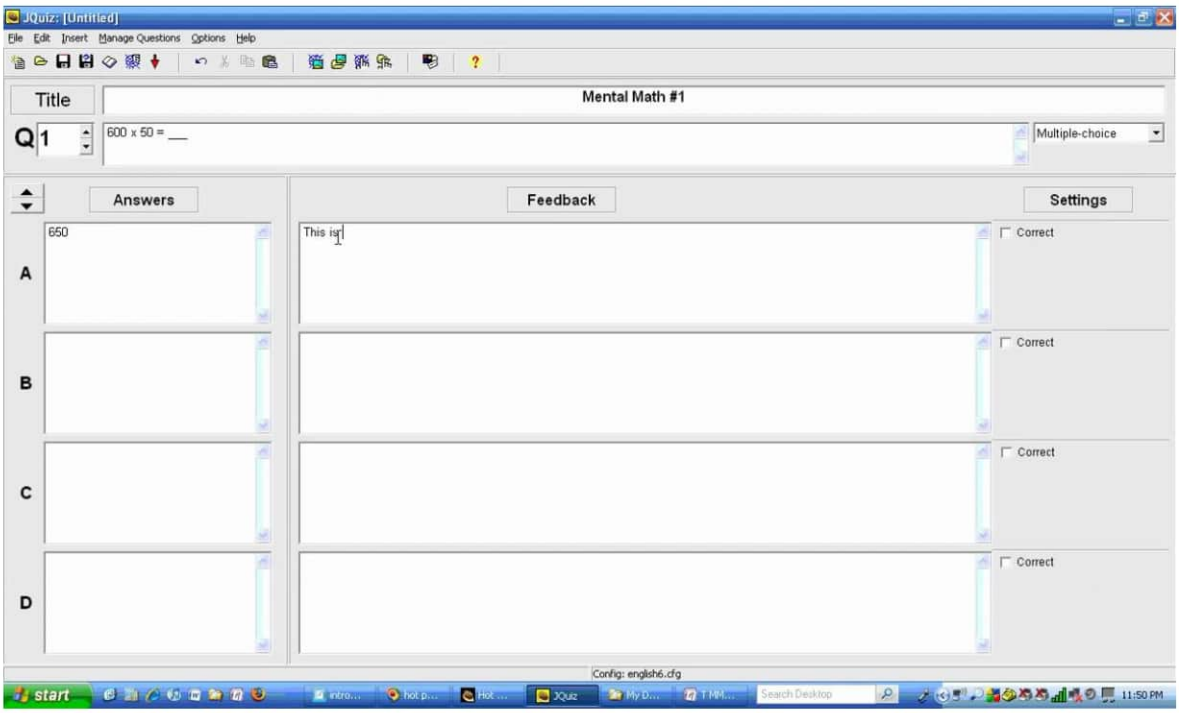


Рисунок 1.4 – Інтерфейс одного з інструментів Hot Potatoes

iSpring QuizMaker (рис. 1.5) [6] – це сучасний інструмент для створення професійних тестів, який входить до складу пакету iSpring Suite. Можливості додатку дозволяють створювати тести з більш ніж 14 типами запитань, у тому числі з відео, аудіо та графікою. Розроблена програма надає великі можливості для розробки комплексних тестів, має сучасний та зрозумілий інтерфейс, а створені тести можна інтегрувати у інші системи, особливо у велику кількість різних LMS.

Найбільшим недоліком для вибору цього забезпечення є комерційна основа додатку. iSpring QuizMaker є гарним вибором для великих компаній, що в змозі оплачувати задіяні ресурси та потребують сучасний інструмент для регулярного проведення тестування.

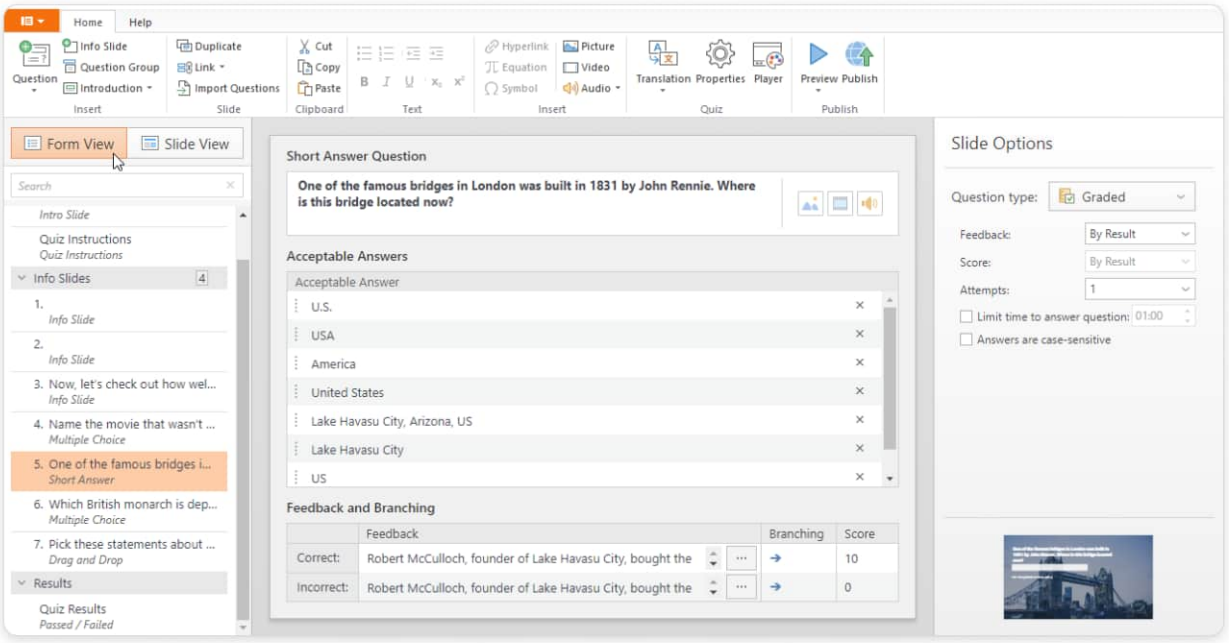


Рисунок 1.5 – Інтерфейс iSpring QuizMaker

Google Forms (рис. 1.6) [7] – це зручний online-сервіс для створення простих тестів і опитувань, що не потребує спеціальних технічних навичок. За допомогою Google Forms можна швидко підготувати завдання з варіантами відповідей.

Інтерфейс зрозумілий, що робить інструмент доступним як для викладачів, так і для студентів. Незважаючи на свою доступність

, функціональність сервісу обмежена у порівнянні з професійними рішеннями для тестування.

На відміну від минулих сервісів, Google Forms є повноцінним online-сервісом та не залежить від операційної системи чи типу пристрою, на якому відкривається. Це значна перевага порівняно з програмами, які потребують встановлення на конкретний комп’ютер та не мають повноцінної хмарної версії.

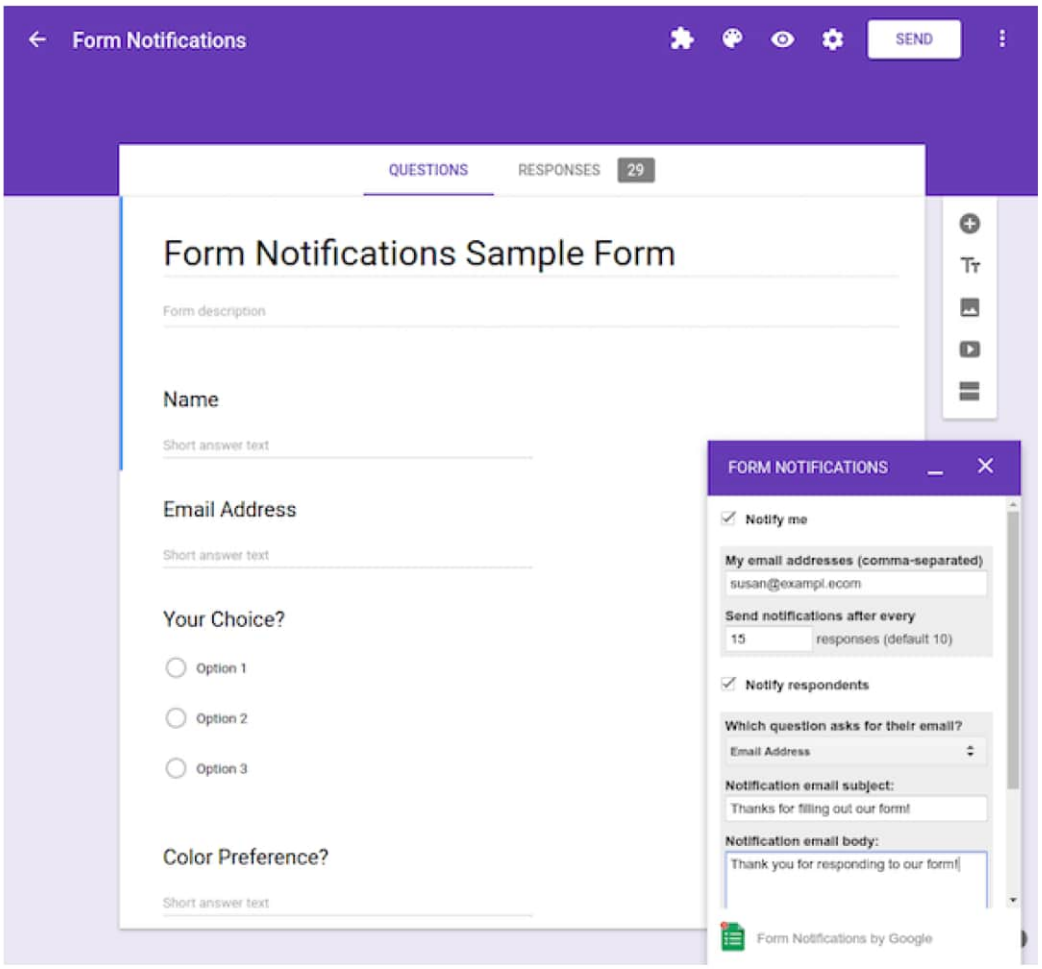


Рисунок 1.6 – Інтерфейс Google Forms

1.4 Огляд можливостей створення тестів на основі інструментів Moodle

Для більш глибокого розуміння роботи з такими сервісами буде доцільним розглянути можливості створення тестів та технічні аспекти роботи з середовищем.

Для аналізу можливостей було обрано систему Moodle, оскільки вона є повноцінною системою, яка має повноцінний функціонал створення тестів та цю систему можна розгорнути на локальному комп’ютері.

В першу чергу після розгортання серверу потрібно створити акаунт та створити курс. Після цього можна створювати події для курсу, додавати оголошення, лекції та створювати тести.

Після створення курсу, потрібно створити подію типу тест, та налаштувати його. Для наповнення тесту потрібно додати нові питання або використати вже створені з банку питань. Moodle надає велику кількість різних видів питань для створення (рис. 1.7).

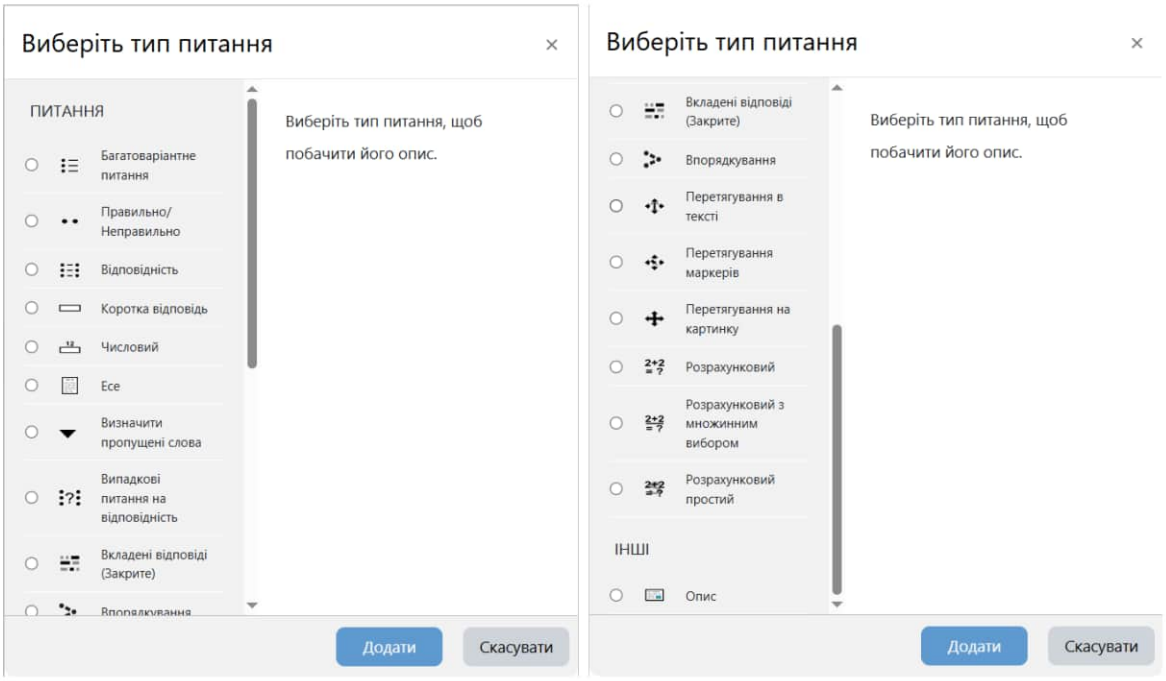


Рисунок 1.7 – Види питань, які можна створити

Під час створення питання Moodle надає широкі можливості налаштування для створення питань. У якості прикладу обрано питання типу коротка відповідь, яка передбачає коротку фразу у якості відповіді. У разі, якщо є декілька варіантів відповіді то можна додати декілька правильних варіантів відповіді, які, за потреби, можуть оцінюватися по різному. Для кожного питання і варіанту відповіді можна додати описання, налаштувати принцип оцінювання. До усіх коментарів можна додати різні елементи мультимедія, такі як аудіо-, відео-файли, фото, гіперпосилання тощо. Весь написаний текст можна форматувати.

З таким функціоналом можна побачити, що з такою кількістю різновидів питань та можливістю їхнього налаштування Moodle надає універсальні

інструменти для проведення тестування. Основними недоліками платформи є складний інтерфейс та відносно повільна робота сервісу.

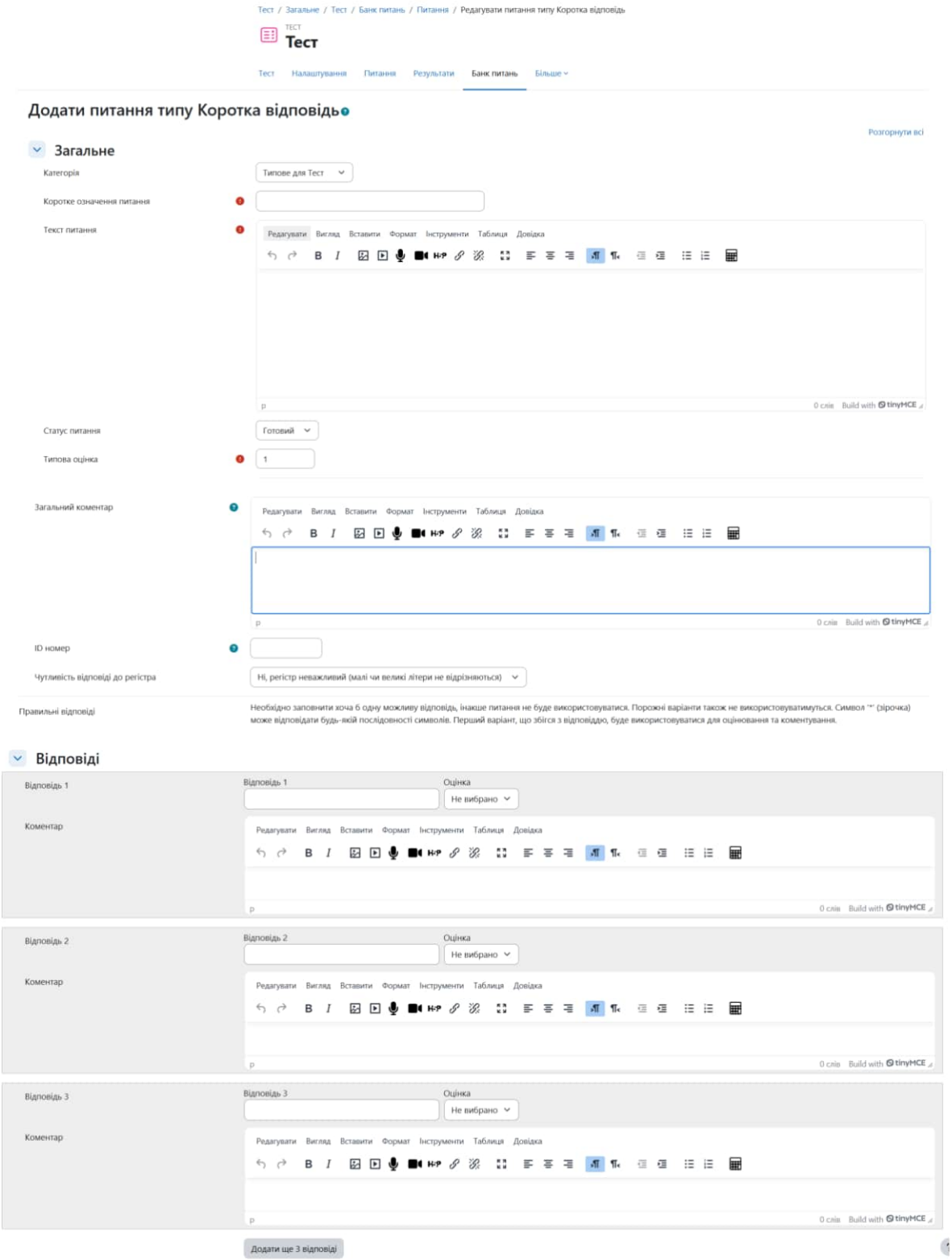


Рисунок 1.8 – Вигляд сторінки створення питання

1.5 Огляд технологій, що використовувались у згаданих системах

Платформа **Moodle** [8] – [9] розроблена мовою програмування PHP і може працювати на будь-якому вебсервері, що підтримує цей інтерпретатор. Для збереження даних використовується реляційна база даних — MySQL, PostgreSQL, MariaDB, Microsoft SQL Server або Oracle. Всі завантажені файли та додаткові дані користувачів зберігаються у спеціальному каталозі moodledata — позафайловій системі бази даних.

Архітектура Moodle є модульною: система побудована на великій кількості плагінів, які відповідають за різні функції — від створення курсів до звітності, тестування, інтеграції з іншими сервісами та зовнішніми стандартами. Цей підхід забезпечує гнучкість та масштабованість, дозволяючи адміністраторам додавати або видаляти функціональність без змін у ядрі платформи. Для розробників Moodle пропонує великий набір Core API, які дозволяють створювати нові функціональні модулі, керувати користувачами, доступами, курсами та контентом. Серед найпоширеніших:

- Access API — керування ролями та дозволами.
- Navigation API — формування навігаційного меню.
- Output API — керування HTML-виводом.
- Data manipulation API — робота з базами даних через.

Крім того, Moodle підтримує вебсервіси на основі REST, SOAP та XML-RPC, що дозволяє зовнішнім системам (наприклад, мобільним додаткам або порталам) здійснювати взаємодію з LMS, наприклад, для реєстрації студентів, отримання результатів тестів або створення нових курсів.

Canvas LMS [10] – [11] побудована на фреймворку Ruby on Rails та забезпечує чітке розділення логіки. Для зберігання даних використовується PostgreSQL, а для кешування та обробки фонових завдань — Redis. Фронтенд платформи поступово переходить на React, хоча частково ще використовуються компоненти на jQuery.

Canvas LMS надає REST API, який дозволяє зовнішнім застосункам взаємодіяти з платформою. За допомогою API можна керувати курсами, користувачами, завданнями, тестами, оцінками та іншими елементами.

Платформа також підтримує стандарт LTI (Learning Tools Interoperability), що забезпечує інтеграцію з зовнішніми освітніми інструментами та сервісами.

Forma LMS [11] – [12] розроблена мовою програмування PHP. Для зберігання даних застосовується реляційна база даних, така як MySQL або MariaDB. Фронтенд платформи реалізований з використанням HTML, CSS та JavaScript, що забезпечує адаптивний дизайн та зручний інтерфейс для користувачів.

Платформа підтримує модульну архітектуру, що дозволяє розширювати функціональність за допомогою плагінів та додаткових модулів. Це забезпечує можливість налаштування системи відповідно до специфічних вимог організації.

Forma LMS надає набір API, які дозволяють інтегрувати платформу з іншими системами, такими як HRM, CRM або ERP. Це забезпечує автоматизацію процесів, таких як управління користувачами, курсами та звітністю.

Платформа також підтримує SSO (Single Sign-On), що дозволяє користувачам входити в систему за допомогою облікових даних інших сервісів, що спрощує управління доступом та підвищує безпеку системи.

Hot Potatoes [5] є настільною програмою, що працює в середовищі Windows. Програмне забезпечення розробленою на основі Delphi. Кожен тип тестів створюється в окремому модулі: JQuiz (тести з вибором), JCloze (вправи на заповнення пропусків), JMatch (з'єднання), JCross (кросворди) тощо.

Основним результатом роботи є HTML-файли з вбудованим JavaScript-кодом, які можна зберігати локально або розміщувати на вебсервері. Ці HTML-сторінки не потребують серверної обробки, тому їх легко інтегрувати в прості вебсайти або LMS-системи, що підтримують статичні файли.

Google Forms [12] є частиною екосистеми Google Workspace і надає можливості для створення, управління та аналізу опитувань і тестів у хмарному середовищі. Для розробників, які хочуть інтегрувати Google Forms у власні системи або автоматизувати роботу з формами, Google пропонує спеціальний Forms API — сучасний інтерфейс програмування додатків за правилами REST, який дозволяє створювати, читати, оновлювати та керувати формами й відповідями програмно.

За допомогою Google Forms API можна автоматизувати такі процеси, як створення запитань різних типів, оновлення структури форми. API підтримує операції із керування формами, що дозволяє організовувати тестування або опитування без необхідності вручну працювати з інтерфейсом Google Forms.

API працює в межах інфраструктури Google, отже, всі дії відбуваються в хмарі, і для їх виконання потрібен доступ до облікового запису Google з відповідними дозволами.

1.6 Backend-технології розробки

Платформа **Node.js** з фреймворком **Express** є оптимальним вибором для швидкої розробки бекенду, орієнтованого на обробку API. Node.js забезпечує роботу за допомогою мови програмування JavaScript. Архітектура побудована на подіях та асинхронна природа дозволяють ефективно обробляти численні одночасні з'єднання, що важливо для систем тестування з активною взаємодією користувачів. Express надає мінімалістичний каркас, а для складніших проектів можна використовувати NestJS з його модульною структурою. відсутність вбудованих інструментів для роботи з базами даних, що змушує використовувати сторонні бібліотеки.

Django — фреймворк, який є рішенням у разі потреби комплексного інструмента з вбудованою ORM, адміністративним інтерфейсом та системою аутентифікації. Django забезпечує роботу за допомогою мови програмування Python. Його ORM дозволяє працювати з базою даних без необхідності

написання SQL-запитів, що значно прискорює процес розробки. Django REST Framework спрощує створення API. Серед недоліків можна виділити синхронну модель виконання, яка обмежує продуктивність системи порівняно з асинхронними рішеннями.

Laravel є фреймворком та пропонує зручний ORM (Eloquent), систему маршрутизації та шаблонізатор Blade, що робить його корисним при необхідності рендерингу HTML на стороні сервера. Laravel забезпечує роботу за допомогою мови програмування PHP. Вбудована система міграцій БД спрощує управління схемою даних. Недоліками є обмежена продуктивність у порівнянні з сучасними мовами програмування, базова відсутність асинхронності в PHP, хоча сучасні розширення частково компенсують це.

Ruby on Rails є фреймворком та реалізує принцип "convention over configuration, тобто принцип розробки програмного забезпечення, що передбачає використання стандартних, передбачуваних налаштувань, замість того, щоб повністю конфігурувати кожен аспект програмного забезпечення що значно пришвидшує розробку. Ruby on Rails забезпечує роботу за допомогою мови програмування Ruby. Його ORM ActiveRecord дозволяє легко взаємодіяти з базою даних без написання складного SQL-коду. Вбудовані засоби тестування та генератори коду допомагають підтримувати якість і стабільність проєкту. До недоліків можна віднести відносно високе споживання ресурсів, що робить фреймворк менш оптимальним вибором для високонавантажених систем без додаткової оптимізації.

1.7 Frontend-технології розробки

React.js — провідна бібліотека для побудови веб-інтерфейсів з компонентною архітектурою, що забезпечує повторне використання коду. Віртуальний DOM оптимізує оновлення при зміні стану, що є зручним для інтерактивних додатків. Часто використовується разом із додатковими бібліотеками. Для швидкого запуску проєктів можна використати Vite —

сучасний збирач з швидким гарячим перезавантаженням (HMR) і мінімальною конфігурацією.

Vue.js — фреймворк із реактивною системою і однофайловими компонентами. Підходить для поетапного впровадження, від невеликих елементів до SPA. Інтегрується з тестовими бібліотеками (Jest, Vue Test Utils). Менша кількість UI-бібліотек і обмежена масштабованість у порівнянні з React.

Angular — повноцінний фреймворк з модульною архітектурою, двонаправленим зв'язком даних і реактивним програмуванням на RxJS. Підходить для великих корпоративних проєктів, має розвинені інструменти для роботи з формами і валідацією, а також підтримку ізольованого тестування модулів. Вимагає значного часу на навчання, надмірний для малих проектів.

Svelte — компілятор, що переносить більшість роботи на етап збірки, виключаючи віртуальний DOM. Дає високу продуктивність і компактний код. Має просту систему управління станом і вбудовані анімації. Екосистема ще молода, з обмеженою кількістю бібліотек.

Solid.js — бібліотека, що поєднує React-подібний синтаксис із реактивністю без віртуального DOM. Відзначається високою продуктивністю. Підходить для додатків із частими оновленнями стану, таких як системи тестування в реальному часі.

1.8 Підходи до створення API

REST API — класичний підхід до побудови веб-сервісів на основі HTTP з використанням методів GET, POST, PUT, DELETE. Простий у реалізації й широко підтримується. Підходить для CRUD-операцій у системах тестування. Основна перевага полягає у широкій підтримці клієнтських бібліотек і простоті інтеграції. Недоліки: обмежена гнучкість для складних операцій, відсутність стандартизації для операцій, що не вписуються в CRUD.

GraphQL — мова запитів, що дозволяє клієнту самостійно формувати структуру відповідей. Особливо корисний для систем тестування з великою кількістю взаємозв'язаних даних. Переваги: гнучкість, типізація, мінімізація надлишкових даних. Недоліки: складна реалізація на сервері, складне кешування, перенавантаження сервера через складні запити.

gRPC — високопродуктивний RPC-фреймворк на базі HTTP/2 і Protocol Buffers. Актуальний для мікросервісної взаємодії всередині систем тестування. Переваги: строга типізація, авто-генерація клієнтів. Недоліки: складне налагодження, висока вимога до технічної кваліфікації.

WebSocket — двосторонній канал зв'язку між клієнтом і сервером. Підходить для реалізації функцій у реальному часі: таймери, оновлення результатів. Переваги: миттєві оновлення без опитування. Недоліки: складне масштабування, проблеми з безпекою, відсутність вбудованого кешування.

1.9 СУБД для задач online-сервісів

PostgreSQL є відкритою реляційною системою керування базами даних, яка підтримує складні зв'язки між таблицями, транзакції відповідно до моделі ACID та зберігання JSON-даних. У контексті систем тестування це дозволяє ефективно зберігати як структуровану інформацію про тести, так і різні формати питань. Основним обмеженням є складність горизонтального масштабування при великому навантаженні.

MySQL забезпечує прийнятний баланс між продуктивністю та простотою реалізації. Вона оптимізована для читання та добре підходить для роботи з відносно статичними даними. Недоліками є менш ефективна робота з JSON і обмежені можливості при виконанні складних аналітичних запитів.

MongoDB є СУБД, яка дозволяє зберігати дані без попередньо визначеної схеми. Це спрощує розробку систем з гнучкою структурою тестів. Підтримується горизонтальне масштабування. Однак обмеження включають

слабку підтримку складних зв’язків між документами та обмежену підтримку транзакцій у попередніх версіях.

Redis — це сховище даних у оперативній пам’яті, яке зазвичай використовується для кешування або зберігання тимчасових даних. Підтримує різноманітні типи структур і механізм публікації/підписки, що дозволяє реалізовувати обмін повідомленнями в режимі реального часу. Основні обмеження стосуються збереження даних після перезапуску та залежності від доступного обсягу оперативної пам’яті.

SQLite — вбудована СУБД, яка не потребує окремого сервера. Її доцільно використовувати в настільних або мобільних додатках для зберігання локальних даних. Рішення не підтримує одночасний доступ великої кількості користувачів і не призначене для роботи в мережевому середовищі.

Microsoft SQL Server — комерційна реляційна СУБД, що орієнтована на корпоративне використання. Має інтеграцію з інструментами аналітики та підтримку складних бізнес-процесів. Її використання передбачає витрати на ліцензування, а також вищі вимоги до обчислювальних ресурсів.

1.10 Висновки до першого розділу. Постановка задачі на розробку сервісу

Розглядаючи вже розроблені проєктні рішення в предметній області, було приведено приклади різних сервісів, які можна використовувати для проведення тестування знань здобувачів.

Існують системи контролю навчанням, такі як Moodle, Canvas LMS, Forma LMS, які розроблювались для використання у великих компаніях, освітніх закладах. Вони надають інструменти для переходу на дистанційне навчання та проведення курсів, можуть також мати інструменти для проведення тестування.

Інший варіант це сервіси, які реалізують тільки можливість створення тестів, які потрібно інтегрувати у інші системи, такі як Hot Potatoes, iSpring

QuizMaker, або online-сервіси, які мають свою платформу для проведення тестів, наприклад Google Forms.

Відповідно до теми роботи «Розробка online-сервісу тестування знань здобувачів» потрібно розробити online-сервіс для тестування знань. Такий сервіс не передбачає весь функціонал систем управління навчанням, але він має бути самостійним та незалежним, на відміну від Hot Potatoes або iSpring QuizMaker.

Для роботи такого сервісу потрібно спроєктувати та розробити наступні елементи системи:

- база даних, яка зберігатиме тести та інформацію про користувачів;
- модуль керування тестами, який відповідає за створення та обробку тестів;
- модуль аутентифікації та авторизації користувачів для розділення ролей;
- модуль проходження та оцінювання тестів;
- інтерфейс користувача.

Система орієнтована на обмежене коло користувачів і не передбачає високого навантаження або великого обсягу даних. Основні вимоги зводяться до забезпечення стабільної роботи базових функцій: автентифікації, створення та проходження тестів, збереження результатів. Особлива увага приділяється підтримуваності коду та можливості локального розгортання без складної інфраструктури. Висока масштабованість не є критичною вимогою для даного проєкту.

Для розробки серверної частини використано платформу Node.js та фреймворк Express. Node.js реалізує асинхронну подієво-орієнтовану модель, яка дозволяє обробляти велику кількість запитів без блокування. Express забезпечує базову маршрутизацію, роботу з middleware і дозволяє організувати структуру застосунку без зайвої складності. Використання мови програмування JavaScript як на клієнтській, так і на серверній стороні спрощує розробку та підтримку коду.

Для зберігання даних використовується PostgreSQL — реляційна система, яка підтримує складні SQL-запити та зберігання JSON-даних у

форматі JSONB. Це забезпечує гнучкість у роботі з різними структурами даних при збереженні переваг реляційної моделі. Враховуючи масштаб проєкту, PostgreSQL забезпечує необхідний функціонал без потреби у складній конфігурації.

Клієнтська частина реалізована за допомогою React у поєднанні з Vite. React дозволяє будувати інтерфейс на основі незалежних компонентів, що спрощує розробку і тестування. Vite використовується для швидкого запуску та збірки проєкту, що є ефективним для невеликої frontend-архітектури.

REST API є доцільним вибором для даного веб-сервісу, оскільки він забезпечує стандартизований механізм взаємодії між клієнтом і сервером на основі протоколу HTTP. Завдяки підтримці CRUD-операцій та простоті у впровадженні, REST дозволяє ефективно впровадити основну функціональність системи тестування без ускладнень, пов’язаних із налаштуванням додаткових протоколів чи форматів. Крім того, REST широко підтримується бібліотеками та фреймворками як на стороні клієнта, так і сервера, що спрощує розробку, тестування та обслуговування проєкту. У порівнянні з альтернативними підходами, REST не потребує додаткової складної інфраструктури або високого рівня технічної підготовки, що робить його оптимальним для даного проєкту.

РОЗДІЛ 2. ПРОЄКТНІ РІШЕННЯ

У розділі описано структуру розробленої системи, особливості її роботи. Надано опис бази даних, таблиць бази даних. Описано розроблені backend- і frontend-фрагменти системи, програмний код цих фрагментів та можливості, які створений сервіс надає. Представлено результати тестування системи виконаного на основі платформи Postman.

2.1 Опис структури системи

Розроблена система побудована за клієнт-серверною моделлю, що передбачає взаємодію між клієнтською частиною, або веб-клієнтом, серверною частиною та базою даних.

Веб-клієнт — це частина системи, яка працює у веб-браузері користувача. У даному випадку веб-клієнтом є фронтенд, реалізований за допомогою React.js. Він надає графічний інтерфейс для авторизації, проходження тестів, перегляду результатів тощо. Веб-клієнт взаємодіє з сервером через HTTP-запити.

Користувачі взаємодіють із системою через браузер, де на фронтенді відображається інтерфейс для введення даних. Усі дії користувача відправляються у вигляді HTTP-запитів до бекенд-сервера.

Бекенд виконує роль API-сервера. Він приймає запити від клієнта, обробляє їх, реалізує бізнес-логіку та взаємодіє з базою даних для зчитування чи збереження інформації.

API-сервер використовує для функціонування протокол HTTP та порт з номером 5000. У запитах з фронтенду зазначаються конкретні маршрути, які відповідають заданим діям. Сервер використовує контролери для обробки цих маршрутів і middleware для перевірки авторизації.

База даних PostgreSQL розміщується на тому ж сервері, що і бекенд. Вона використовується для зберігання інформації про користувачів, тести, відповіді та результати.

Після обробки запиту сервер повертає відповідь у форматі JSON, яка містить або результати операції, або повідомлення про помилки. Ця відповідь обробляється на фронтенді для оновлення інтерфейсу користувача у браузері.

Таким чином, відбувається двосторонній обмін даними між клієнтом та сервером: клієнт надсилає команди або дані, а сервер повертає відповідь у відповідь на ці запити.

Для забезпечення взаємодії між цими двома частинами на етапі розробки використовується механізм проксі або ручна вказівка адреси API у запитах фронтенду. Це дозволяє фронтенду надсилати HTTP-запити до сервера

Схему клієнт-серверної взаємодії наведено на рисунку 2.1.

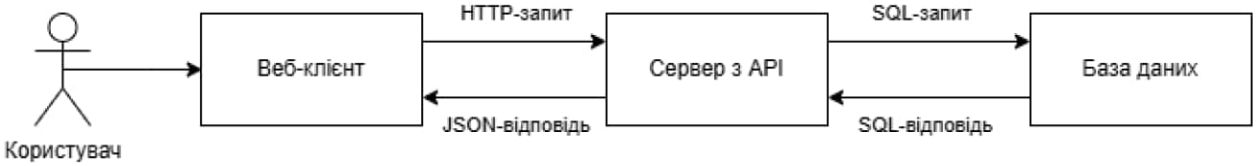


Рисунок 2.1 – Схема взаємодії системи

2.2 Опис алгоритму взаємодії з системою

Для опису поведінки системи створено діаграму станів на рисунку 2.2. Така діаграма описує можливі послідовності станів і переходів, які характеризують поведінку системи.

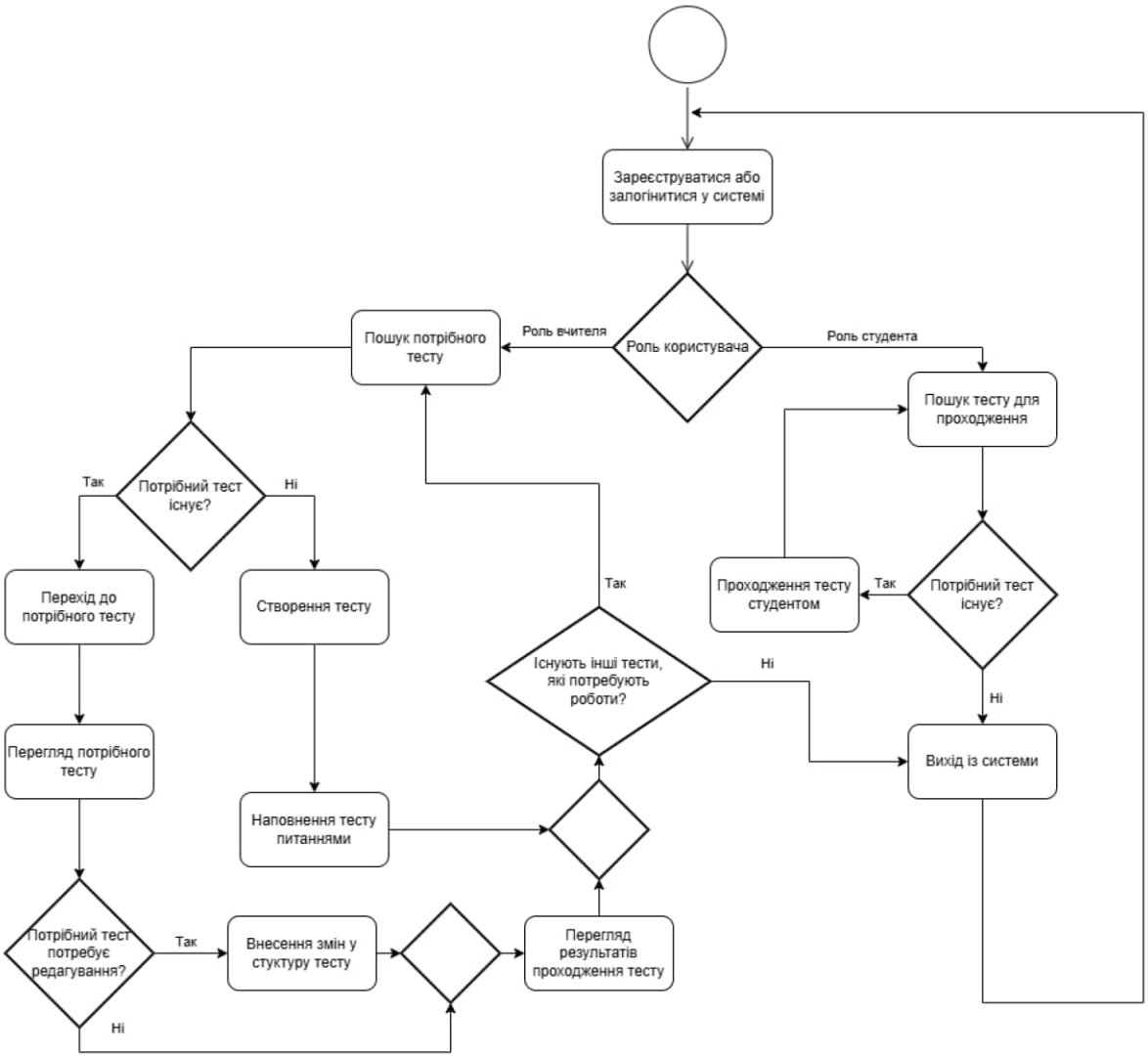


Рисунок 2.2 – Діаграма станів

Як видно з діаграми, сесія використання системи розпочинається з логіну або реєстрації у системі. Є дві ролі користувача: вчитель та студент. У залежності від ролі, користувачу надаються різні можливості.

Вчитель виконує пошук по наявним тестам, якщо потрібного тесту не існує, він створює новий. Якщо тест існує, він переглядає результати проходження тесту, у разі потреби виконання редагування тесту, він виконує зміни. Якщо більше тестів, які потребують роботи, немає, то вчитель виходить із системи.

Для студентів реалізована можливість вибору тестів з переліку усіх створених у системі та їх проходження.

Таким чином, розроблений сервіс надає основні можливості для тестування знань, а саме створення та редагування тестів, їх проходження та перевірка результатів.

2.3 Діаграма розгортання системи

Розгортання створеної системи описано на рисунку 2.3. Дана діаграма розгортання ілюструє фізичну структуру системи — як програмні компоненти розміщені на апаратних або віртуальних вузлах і як між ними відбувається обмін даними через мережеві протоколи. Вона показує, які модулі відповідають за інтерфейс, серверну логіку та зберігання даних, а також канали зв’язку між клієнтом, сервером і базою даних.

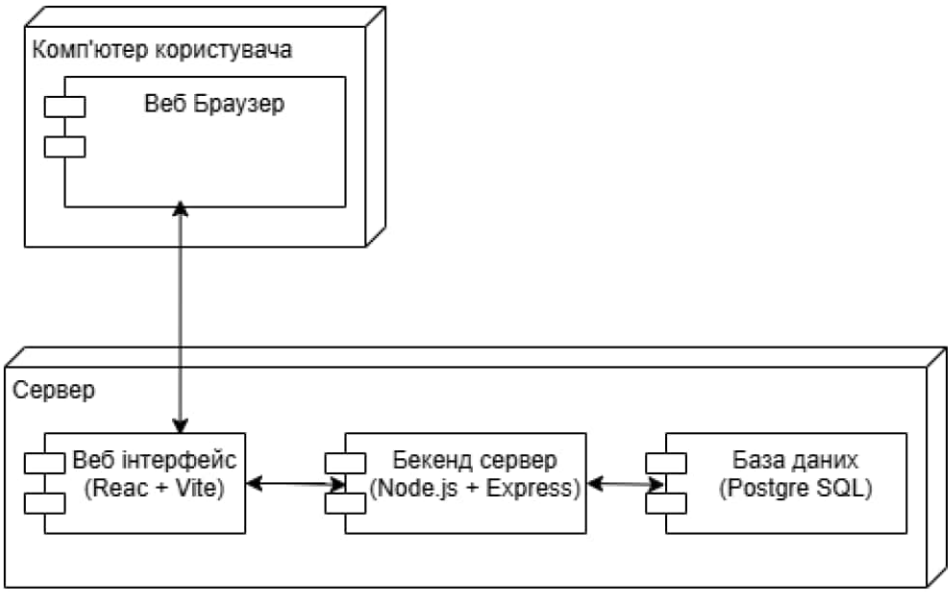


Рисунок 2.3 – Діаграма розгортання

Описана система реалізована за принципом клієнт-серверної архітектури з використанням REST API для взаємодії між фронтендом і бекендом. Користувач працює через веб-браузер, який надсилає HTTP/HTTPS-запити до веб-сервера. Веб-сервер включає два основні модулі: модуль веб-інтерфейсу, реалізований за допомогою Vite та React, що відповідає за відображення користувацького інтерфейсу та взаємодію з користувачем, і модуль серверної логіки реалізований, на Node.js з Express, що обробляє

запити, перевіряє авторизацію, виконує бізнес-логіку та взаємодіє з базою даних.

Веб-інтерфейс надсилає REST API-запити до бекенд-сервера через HTTP, додаючи токен авторизації у заголовок запиту. Серверна частина, у свою чергу, взаємодіє з базою даних PostgreSQL через TCP-протокол, зчитуючи або змінюючи дані згідно з запитами користувача. Таким чином, система побудована у вигляді чітко структурованих компонентів, що взаємодіють через стандартизовані протоколи, забезпечуючи масштабованість та безпеку у використанні.

2.4 Діаграми послідовності системи

Розроблені діаграми послідовності використовуються для візуалізації послідовності взаємодій між користувачами, клієнтським інтерфейсом та серверною частиною системи. Оскільки ролі відрізняються за конкретними діями, ці діаграми відповідно до ролей надані на рисунках 2.4 – 2.5. Вони демонструють, у якій послідовності відбуваються запити та відповіді, які компоненти взаємодіють, і як протікає обробка даних у часі. Це дозволяє краще зрозуміти функціональність системи, процеси авторизації, передачі даних та виконання бізнес-логіки, що особливо важливо при розробці систем із ролями та розподіленою архітектурою.

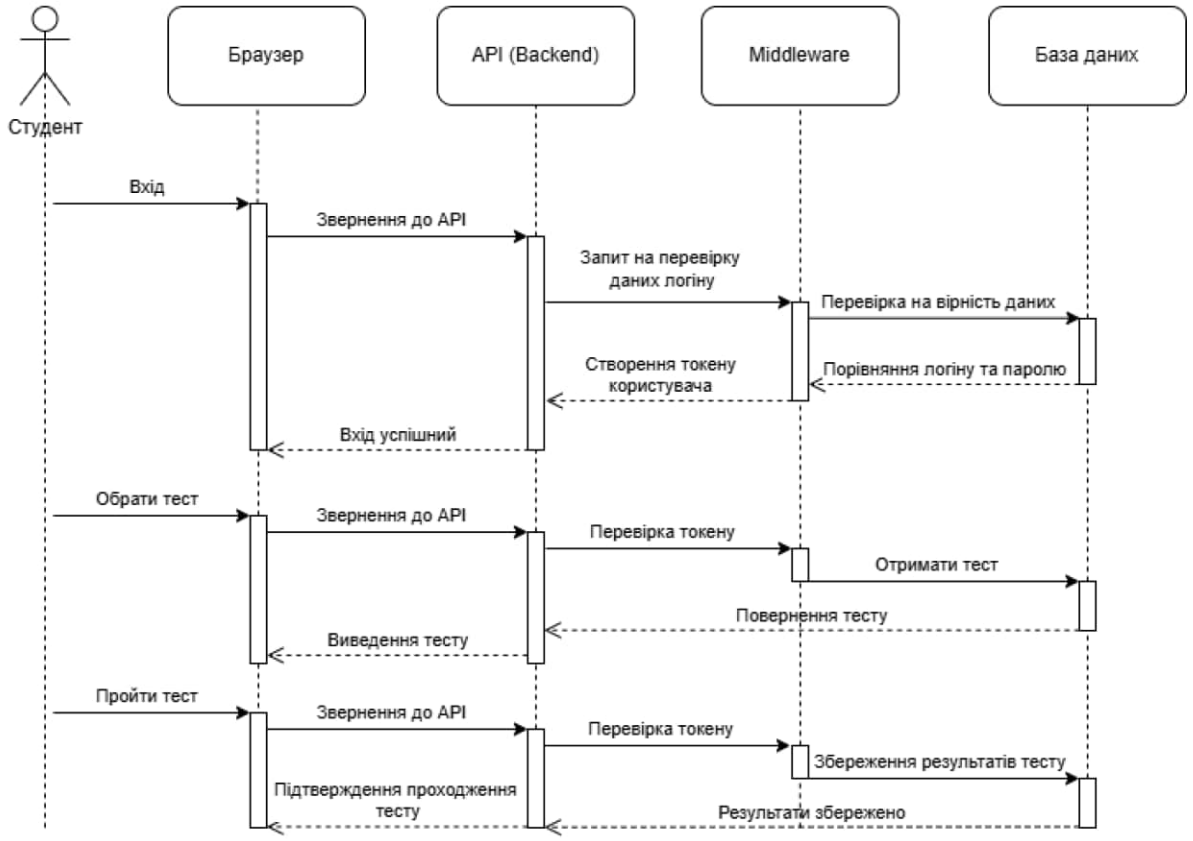


Рисунок 2.4 – Діаграма послідовності для ролі студента

Діаграма послідовності для студента ілюструє покроковий процес взаємодії користувача із системою під час проходження тестування. Спершу студент вводить логін та пароль у клієнтському додатку, після чого відбувається перевірка цих даних на сервері через API. Якщо аутентифікація успішна, сервер генерує токен доступу, який зберігається у браузері та використовується для авторизації подальших запитів. Студент може отримати список доступних тестів, обрати потрібний тест і завантажити питання через запити до API.

Наступним кроком є заповнення тесту: відповіді користувача відправляються на сервер, де вони зберігаються у базі даних разом із результатом. Протягом усієї взаємодії реалізовані механізми перевірки прав доступу через middleware, що гарантує безпеку системи.

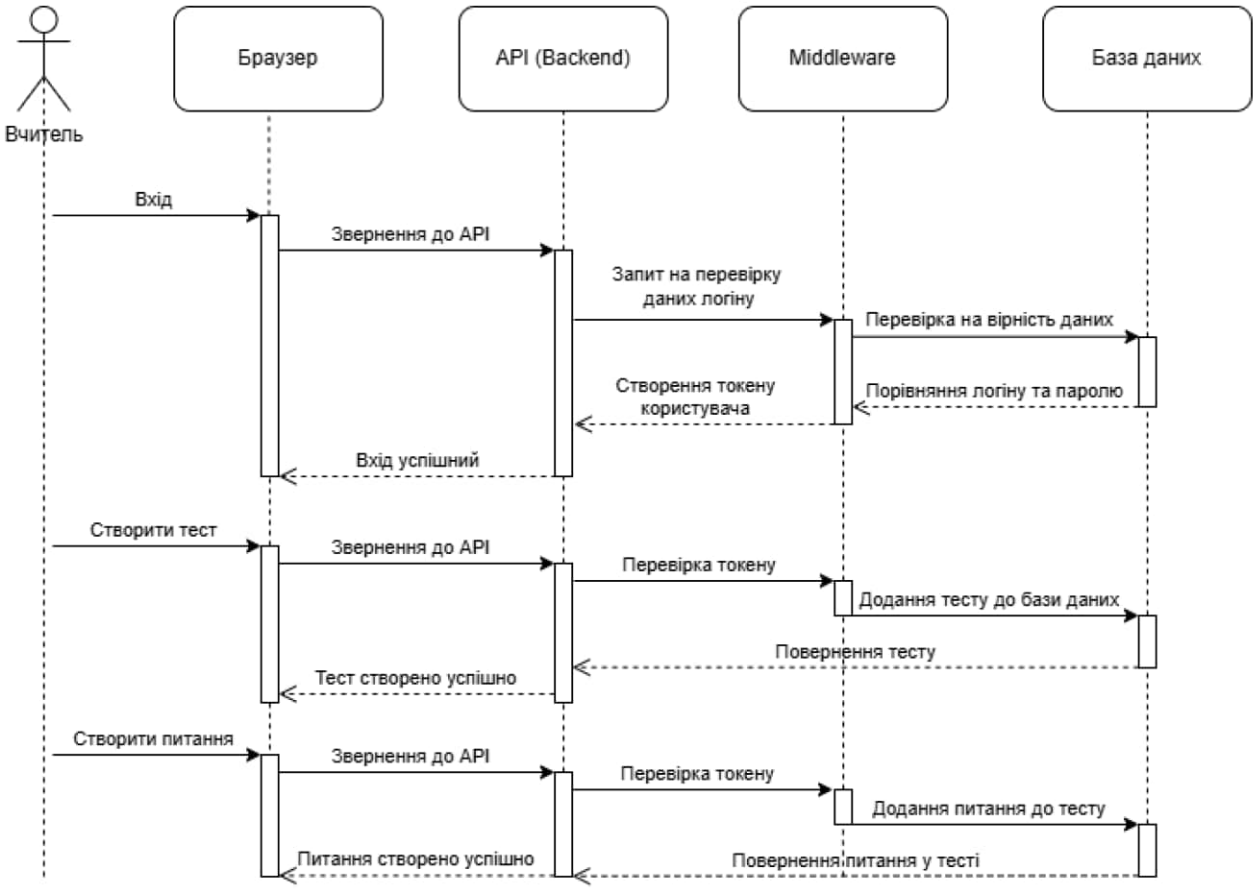


Рисунок 2.5 – Діаграма послідовності для ролі вчителя

Діаграма послідовності для вчителя відображає процес створення, редагування та управління тестами в системі. Вчитель також проходить аутентифікацію через API, отримуючи токен доступу, який дозволяє здійснювати захищені операції. Після успішного входу вчитель надсилає запит на створення нового тесту, сервер приймає ці дані, зберігає тест у базі та повертає підтвердження.

Далі вчитель може додавати або редагувати питання у тесті, надсилаючи відповідні HTTP-запити до API. Сервер обробляє ці запити, перевіряє права доступу, зберігає зміни у базі даних і повертає оновлену інформацію.

Обидві діаграми послідовності демонструють ключові процеси системи з урахуванням ролей користувачів і допомагають описати як взаємодіють клієнтська частина та сервер через API.

2.5 Опис бази даних

Для системи онлайн-тестування необхідне збереження та управління різноманітною інформацією, що стосується користувачів, тестів, питань, відповідей і результатів проходження тестів. Схема бази даних надана на рисунку 2.6.

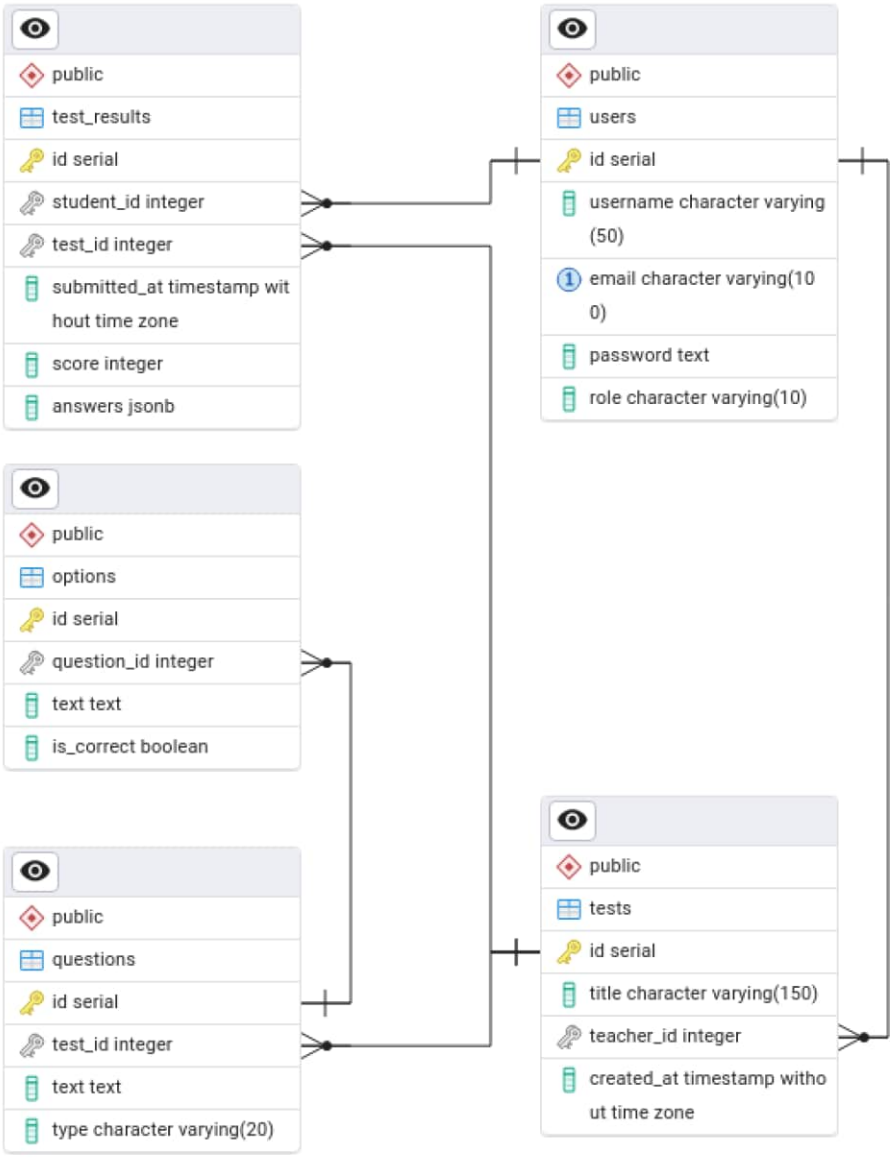


Рисунок 2.6 – Схема таблиць бази даних

Таблиця users зберігає інформацію про користувачів системи, які можуть мати роль вчителя або студента. Поле id є унікальним ідентифікатором користувача. Поля username та email зберігають ім’я користувача та його

електронну пошту, причому пошта є унікальною для кожного запису. Пароль зберігається у зашифрованому вигляді в полі password. Поле role обмежене двома значеннями — teacher або student, що визначає роль користувача в системі.

- id (SERIAL) – унікальний ідентифікатор користувача;
- username (VARCHAR(50)) – ім'я користувача;
- email (VARCHAR(100)) – унікальна електронна пошта користувача;
- password (TEXT) – зашифрований пароль;
- role (VARCHAR(10)) – роль користувача (teacher або student).

Таблиця tests містить інформацію про тести, які створюють вчителі. Кожен тест має унікальний ідентифікатор id, назву title та зв'язок із вчителем через поле teacher_id. Поле created_at фіксує дату й час створення тесту.

- id (SERIAL) – унікальний ідентифікатор тесту (первинний ключ);
- title (VARCHAR(150)) – назва тесту;
- teacher_id (INTEGER) – ідентифікатор вчителя, який створив тест;
- created_at (TIMESTAMP) – дата і час створення тесту.

У таблиці questions зберігаються питання, що належать до певного тесту. Поле test_id пов'язує питання з тестом, при видаленні тесту питання видаляються автоматично. Поле text містить текст питання, а type визначає тип відповіді — одиночний вибір (single), множинний вибір (multiple) або текстова відповідь (text).

- id (SERIAL) – унікальний ідентифікатор питання;
- test_id (INTEGER) – ідентифікатор тесту, якому належить питання;
- text (TEXT) – текст питання;
- type (VARCHAR(20)) – тип відповіді (single, multiple, text);

Таблиця options містить варіанти відповідей для питань з типом вибору. Кожен варіант пов'язаний з питанням через question_id. Поле text зберігає текст варіанту, а is_correct позначає, чи є цей варіант правильною відповіддю.

- id (SERIAL) – унікальний ідентифікатор варіанту;
- question_id (INTEGER) – ідентифікатор питання, якому належить варіант;
- text (TEXT) – текст варіанту відповіді;
- is_correct (BOOLEAN) – позначка правильності варіанту (за замовчуванням FALSE).

У таблиці test_results зберігаються результати проходження тестів студентами. Поля student_id та test_id вказують відповідно на студента і тест. Поле submitted_at фіксує дату і час здачі тесту. В score зберігається кількість набраних балів, а в answers — детальна відповідь студента у форматі JSONB.

- id (SERIAL) — унікальний ідентифікатор результату;
- student_id (INTEGER) — ідентифікатор студента;
- test_id (INTEGER) — ідентифікатор тесту;
- submitted_at (TIMESTAMP) — дата і час здачі тесту;
- score (INTEGER) — набрані бали;
- answers (JSONB) — відповіді студента у форматі JSON.

2.6 Опис технологій backend-фрагменту системи

Проєкт було реалізовано на основі клієнт-серверної моделі взаємодії, що передбачає розділення логіки між фронтендом і бекендом. Основна мета серверної частини — забезпечення обробки вхідних запитів від користувача, реалізація бізнес-логіки, взаємодія з базою даних та формування відповідей. Для досягнення такої мети було обрано перелік технологій, що надають відповідні можливості для розробки такої системи

В якості середовища виконання обрано платформу Node.js, що дає змогу запускати JavaScript на стороні сервера. Однією з причин вибору Node.js стала його висока швидкодія та велика кількість готових бібліотек, доступних через менеджер пакетів npm. Основним фреймворком для побудови HTTP-сервера виступає Express, який значно спрощує роботу з маршрутизацією, запитами та

відповідями, а також має зручну систему middleware-функцій для обробки логіки доступу, авторизації тощо.

Для роботи з конфіденційними змінними середовища, такими як порт сервера чи секретний ключ, використовується бібліотека dotenv. Це дозволяє зберігати конфігураційні параметри у окремому .env файлі та легко змінювати їх без внесення змін до коду. Забезпечення безпечного зберігання паролів реалізовано за допомогою бібліотеки bcrypt, що виконує хешування введених користувачем паролів перед збереженням у базу даних. А для реалізації механізму аутентифікації на основі токенів використовується JSON Web Token (JWT), який дозволяє створювати маркери доступу з обмеженим терміном дії.

Уся інформація про користувачів, тести, питання, варіанти відповідей та результати зберігається у реляційній базі даних PostgreSQL, яка підключається до Node.js за допомогою клієнтської бібліотеки pg. З'єднання з базою даних та виконання SQL-запитів здійснюється без використання ORM, що дозволяє краще контролювати структуру запитів та взаємодію з таблицями.

На етапі розробки використовувався Node.js-модуль nodemon, який автомстуатично перезапускає сервер при зміні коду. Це значно прискорює розробку, дозволяючи одразу бачити результати внесених змін. Для забезпечення взаємодії з клієнтом з іншого домену чи порту додано middleware cors, яке дозволяє обробляти крос-доменні запити.

2.7 Опис кінцевих точок backend-фрагменту

У серверній частині застосунку розроблено кінцеві точки у архітектурному стилі REST API, це забезпечує взаємодію клієнта з базою даних через набір розділених маршрутів. Кожен маршрут відповідає за певну функціональну область системи — автентифікацію, керування тестами, обробку питань, збереження результатів тощо. Така структура дозволяє

розмежувати відповідальність кожного сегменту системи. Список кінцевих точок описано у таблиці 2.1.

Таблиця 2.1 – Кінцеві точки з’єднання

Метод HTTP-запиту	Кінцева точка	Опис
POST	/api/auth/register	Реєстрація нового користувача
POST	/api/auth/login	Вхід у систему, отримання токена
GET	/api/tests/public	Отримання списку тестів для студента з використанням фільтрів
GET	/api/tests/:testId/questions	Отримання питань для проходження тесту студентом
GET	/api/tests	Отримання тестів, створених викладачем
POST	/api/tests	Створення нового тесту викладачем
PUT	/api/tests/:id	Оновлення тесту викладачем
DELETE	/api/tests/:id	Видалення тесту викладачем
GET	/api/tests/:testId/questions	Отримання всіх питань до конкретного тесту для редагування
POST	/api/tests/:testId/questions	Додавання нового питання до тесту викладачем
PUT	/api/questions/:id	Редагування питання викладачем
DELETE	/api/questions/:id	Видалення питання викладачем
GET	/api/student/tests/:testId	Отримання тесту з питаннями для проходження студентом

POST	/api/student/tests/:testId/submit	Розрахунок балу за результатами тесту
POST	/api/results	Збереження результатів проходження тесту
GET	/api/results	Отримання результатів проходження тестів студента
GET	/api/results/test/:testId	Отримання оброблених та розрахованих результатів по тесту
GET	/api/teachers	Отримання списку всіх викладачів

2.8 Опис контролерів backend-фрагменту

У проєкті розроблена структура контролерів, які відповідають за певні набори функцій. Контролери приймають HTTP-запити, обробляють їх, взаємодіють із базою даних і повертають результати клієнту. Далі наведено опис контролерів та їхніх методів.

AuthController відповідає за управління реєстрацією та входом користувачів у систему. Він працює з базою даних, забезпечує перевірку коректності введених даних, хешування паролів і генерацію токенів для авторизації.

Цей контролер має дві основні функції: реєстрація нового користувача та авторизація існуючого.

- register — обробляє створення нового користувача, перевіряє всі необхідні поля, унікальність email, хешує пароль і повертає дані користувача разом з токеном (додаток А.1, стор. 65).
- login — приймає email і пароль, перевіряє їх коректність, порівнює з інформацією в базі, і у випадку успіху повертає інформацію

користувача і токен для подальшої авторизації (додаток А.1, стор. 65).

QuestionsController відповідає за керування питаннями в тестах. Він дозволяє викладачам отримувати, створювати, оновлювати та видаляти питання разом із варіантами відповідей. Контролер забезпечує перевірку прав доступу, щоб викладач міг працювати лише зі своїми тестами.

Цей контролер реалізує основні функції роботи з питаннями в тестах.

- getQuestionsForTest — отримує список усіх питань для певного тесту, разом з варіантами відповідей (додаток А.1, стор. 67).
- createQuestion — створює нове питання у вказаному тесті, а також додає до нього варіанти відповідей (додаток А.1, стор. 67).
- updateQuestion — оновлює текст і тип питання, замінює старі варіанти відповідей на нові (додаток А.1, стор. 68).
- deleteQuestion — видаляє питання разом з усіма варіантами відповідей (додаток А.1, стор. 69).

StudentTestController відповідає за отримання інформації про тест із питаннями та за обробку проходження тесту студентом. Він надає можливість завантажити тест із повним набором питань і варіантів відповідей, а також приймає результати тестування, оцінює їх і зберігає в базі.

Основні функції контролера пов’язані з роботою з тестами та їх проходженням.

- getTestWithQuestions — отримує інформацію про конкретний тест, включаючи його назву, ім’я викладача та список питань із варіантами відповідей. Повертає повний опис тесту для відображення студенту (додаток А.1, стор. 69).
- submitTest — приймає відповіді студента на питання тесту, перевіряє їх правильність, обчислює підсумковий бал і зберігає його в базі даних (додаток А.1, стор. 70).

TeacherController відповідає за отримання списку викладачів із бази даних. Він використовується, щоб надати фронтенду перелік усіх користувачів із роллю "teacher", впорядкований за іменами.

Основна функція контролера:

- getAllTeachers — отримує з бази даних усіх користувачів із роллю викладача, повертає масив із їхніх ідентифікаторів і імен користувачів. Використовується для заповнення списку викладачів у інтерфейсі (додаток А.1, стор. 71).

TeacherResultController працює з результатами тестів, зокрема надає викладачеві можливість отримати результати студентів для певного тесту, якщо викладач є власником цього тесту.

Основні функції контролера:

- isTestOwnedByTeacher — перевіряє, чи належить тест з вказаним testId викладачу з teacherId (додаток А.1, стор. 72).
- getResultsByTest — отримує результати тестування студентів для конкретного тесту (додаток А.1, стор. 72).

TestController відповідає за управління тестами з боку викладача: створення, отримання, оновлення, видалення та пошук доступних тестів.

Основні функції контролера:

- createTest — створює новий тест з заданою назвою. В якості викладача прив’язує тест до користувача, який виконав запит. Повертає створений тест (додаток А.1, стор. 73).
- getTests — отримує список усіх тестів, створених викладачем, який виконує запит (додаток А.1, стор. 73).
- updateTest — оновлює назву тесту за його id, якщо тест належить викладачу, який виконує запит (додаток А.1, стор. 73).
- deleteTest — видаляє тест за id, якщо він належить викладачу, який виконує запит (додаток А.1, стор. 74).
- getAvailableTests — повертає перелік тестів, доступних для студентів. Підтримує фільтри: teacherId — показує тести

конкретного викладача, search — пошук по назві тесту. Повертає інформацію про тести з ім’ям викладача (додаток А.1, стор. 74).

TestResultController відповідає за прийом та збереження результатів проходження тестів студентами, а також за отримання історії результатів для конкретного студента.

Основні функції контролера:

submitTest — приймає результати тесту від студента та зберігає ці дані у базу даних (додаток А.1, стор. 75).

getResults — повертає список результатів тестів, які проходив поточний студент. Для кожного результату надається інформація про тест та викладача, який створив тест, а також дата та оцінка (додаток А.1, стор. 76).

2.9 Опис технологій frontend-фрагменту системи

Клієнтська частина проєкту реалізована з використанням бібліотеки React. Для організації маршрутизації у проєкті використовується бібліотека react-router-dom. Вона надає інструменти для створення SPA (Single Page Application), де навігація між сторінками не вимагає повного перезавантаження. Через визначення маршруту можна пов’язати URL з конкретним компонентом. Наприклад, маршрутизація дозволяє відображати різні сторінки — вхід, список тестів, проходження тесту, перегляд результатів — залежно від поточного шляху в адресному рядку. Крім базової маршрутизації, react-router-dom підтримує вкладені маршрути, обробку параметрів URL, перенаправлення та навігаційні хуки, що дозволяє керувати навігаційною логікою в межах клієнтського застосунку.

Для збірки застосунку використовується Vite — сучасний інструмент розробки, що забезпечує швидке завантаження і компіляцію JavaScript та JSX-коду. Основною перевагою Vite є попередня обробка залежностей з використанням ES-модулів, що дозволяє значно зменшити час запуску у режимі розробки. Крім того, Vite підтримує HMR (Hot Module Replacement),

що дає змогу бачити зміни в коді без повного перезавантаження сторінки, зберігаючи при цьому поточний стан компонентів.

Для перевірки якості коду та дотримання єдиного стилю програмування проєкт налаштований на використання ESLint — інструмента статичного аналізу JavaScript-коду. ESLint перевіряє код на предмет потенційних помилок, неправильного використання синтаксису або недотримання прийнятих стандартів. Це допомогло знизити кількість багів і забезпечити узгодженість коду в командній розробці. У проєкті використано кілька плагінів ESLint.

2.10 Огляд роботи з веб інтерфейсом

Основним компонентом, з якого починається рендеринг додатку, є App.jsx. Саме в ньому організована логіка маршрутизації між сторінками, залежно від того, чи авторизований користувач, а також яку роль він має.

Компонент App обгорнутий у контекст авторизації, який дозволяє зберігати інформацію про поточного користувача та забезпечує функції авторизації й виходу з системи. Якщо користувач не авторизований, на головній сторінці одночасно відображаються форми входу та реєстрації. Це дозволяє новим користувачам одразу зареєструватися, а існуючим — увійти в систему, що зображено на рисунку 2.7.

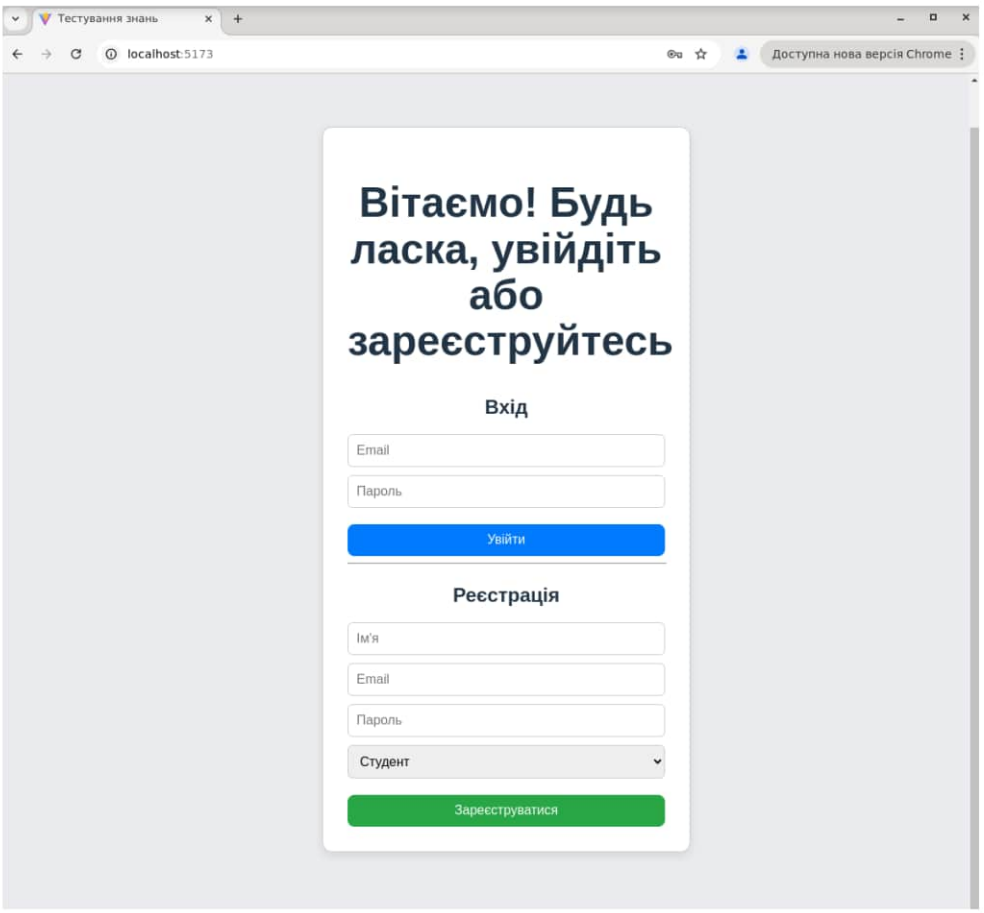


Рисунок 2.7 – Сторінка авторизації та реєстрації

Компоненти Login.jsx та Register.jsx реалізують відповідні форми авторизації та реєстрації. У формі входу користувач вводить свою електронну пошту та пароль, після чого запит надсилається на сервер. Якщо автентифікація успішна, користувач автоматично входить у систему. У разі помилки виводиться повідомлення на екран. Аналогічно працює форма реєстрації, яка додатково дозволяє обрати роль: студент або викладач. Успішна реєстрація також завершується автоматичним входом у систему.

Сторінка вчителя (рис 2.8) призначена для управління тестами, які він створює. Для опису сторінок створено користувача «Тетяна Мартинюк». Сторінка складається з кількох основних частин. У верхній частині виводиться заголовок з іменем викладача. Під ним розташована форма створення нового тесту — текстове поле для введення назви та кнопка «Створити». Після успішного створення тест додається до загального списку.

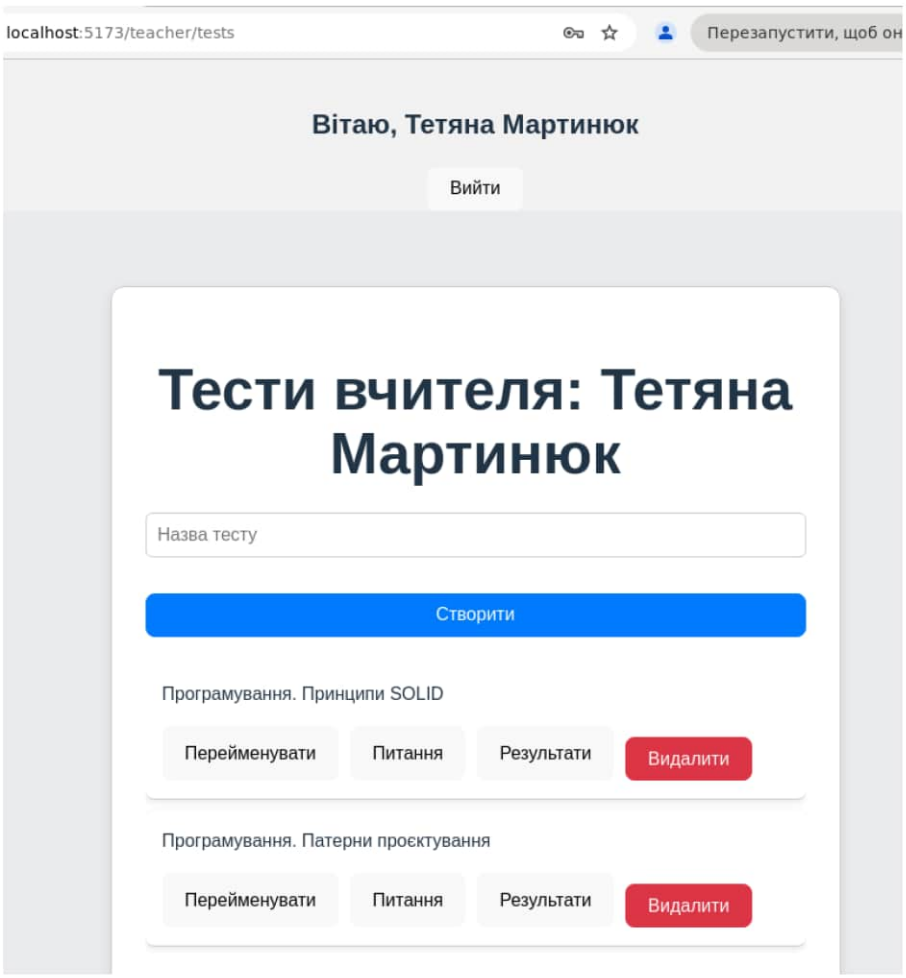


Рисунок 2.8 – Сторінка вчителя

Основна частина сторінки — список тестів, які були створені викладачем. Кожен тест можна перейменовувати, переглядати питання або результати, а також видаляти. Якщо викладач натискає кнопку «Перейменувати», відповідний елемент списку переходить у режим редагування — назва тесту стає доступною для зміни, а кнопки дозволяють зберегти нову назву або скасувати редагування. Кнопки «Питання» та «Результати» ведуть до відповідних сторінок керування питаннями тесту та перегляду результатів проходження.

Сторінка редагування тестів (рис. 2.9) відповідає за відображення та керування питаннями до конкретного тесту.

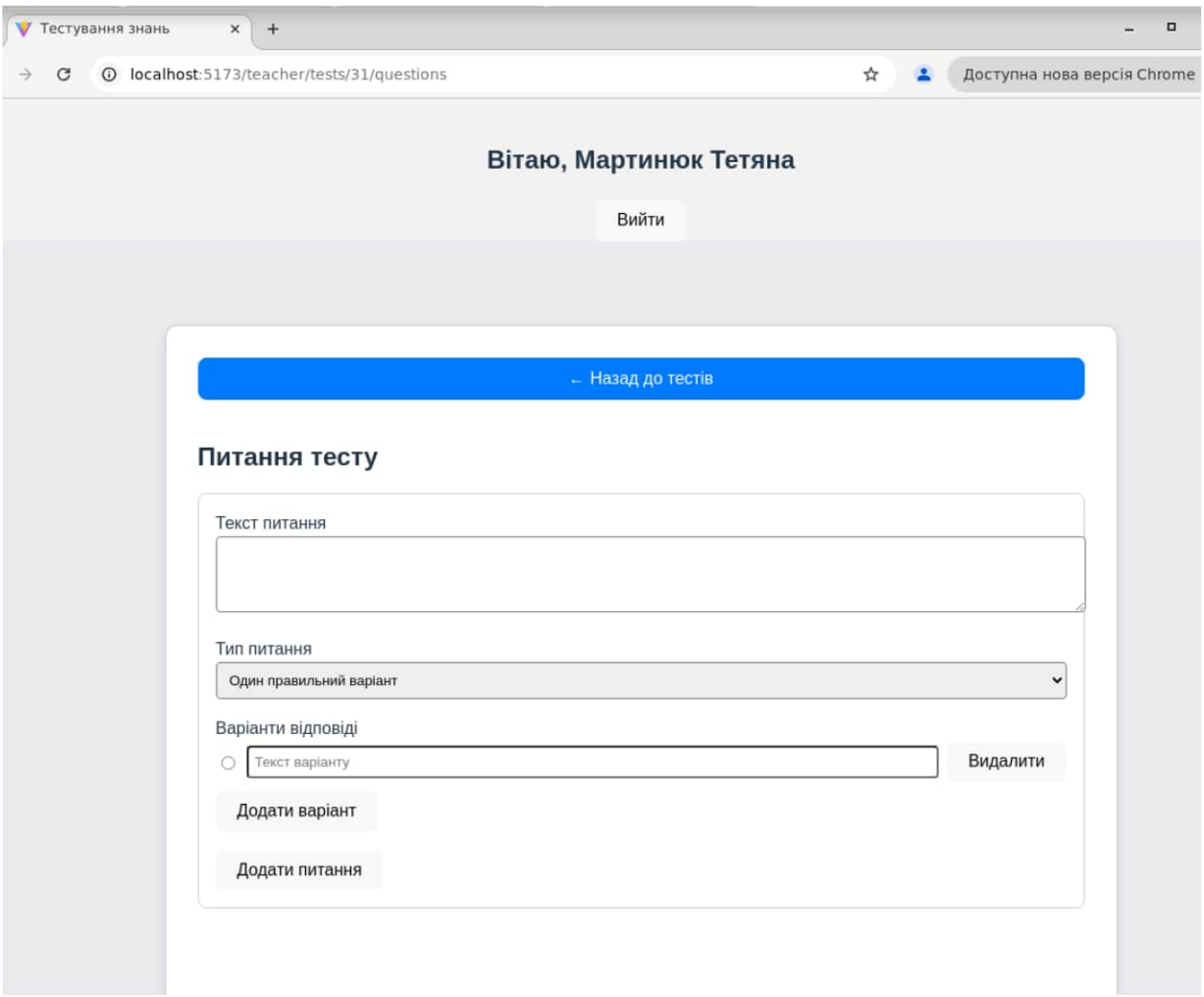


Рисунок 2.9 – Сторінка редагування питань

Основна частина інтерфейсу складається з форми для створення або редагування питання й переліку вже існуючих питань. У формі можна ввести текст питання, вибрати тип питання, це питання з одним правильним варіантом, декількома правильними варіантами та коротка текстова відповідь, і додати варіанти відповіді. Для типів з правильними варіантами ці варіанти позначаються відповідно радіокнопками або чекбоксами для вибору правильних відповідей, а для текстового типу просто вводиться список правильних текстових варіантів. Користувач може редагувати текст варіантів, додавати нові або видаляти існуючі. Нижче форми виводиться список усіх питань тесту з позначенням правильних відповідей, а також кнопками "Редагувати" й "Видалити" біля кожного елементу.

Рисунок 2.10 – Частина питань створеного тесту

Використовуючи створені можливості було створено тест для перевірки знань щодо патернів програмування на рисунку 2.10.

Таким чином, компонент забезпечує повний цикл створення, редагування та видалення питань до тесту вчителем в інтерфейсі.

Сторінка тестів студента відповідає за відображення для студента списку доступних тестів, які він може пройти. Інтерфейс складається з центрального контейнера з білою карткою, в якому відображено заголовок "Доступні тести", поле пошуку за назвою та селектор викладачів. Список тестів виводиться нижче: якщо він порожній — показується повідомлення "Немає доступних тестів", інакше кожен тест виводиться у вигляді рядка з назвою тесту, іменем викладача та кнопкою "Почати тест".

Приклад сторінки тестів студента наведено на рисунку 2.11.

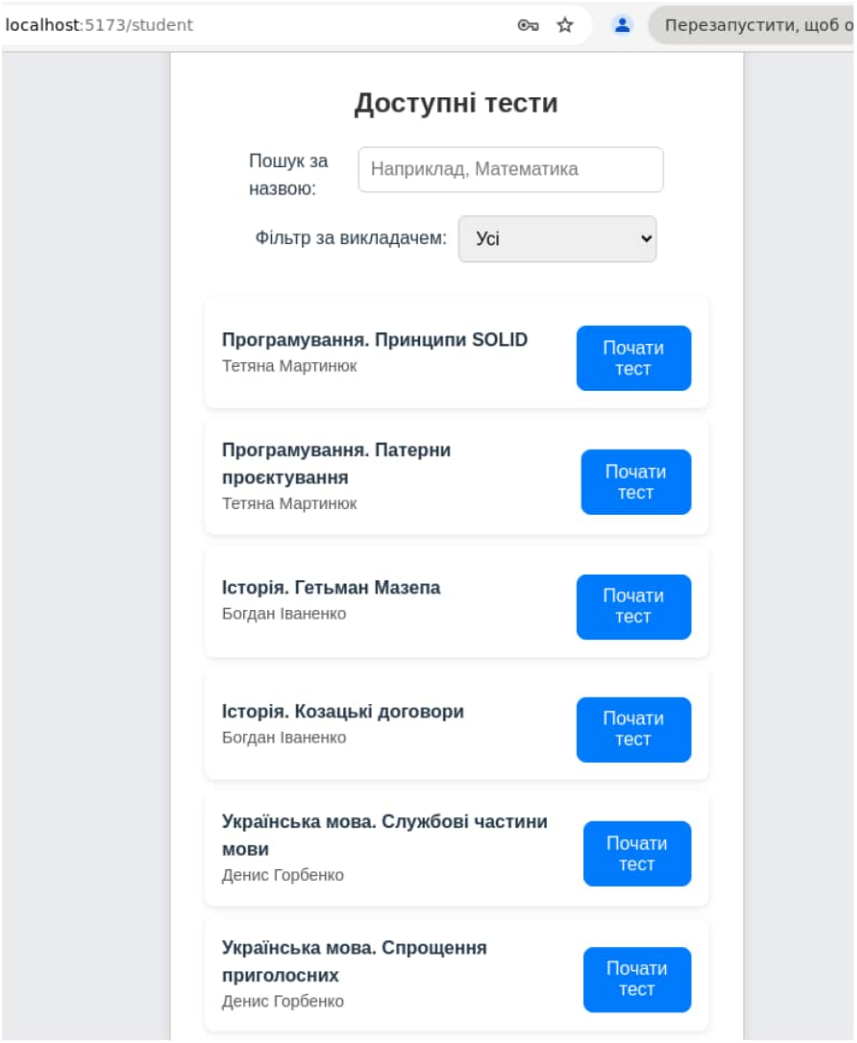
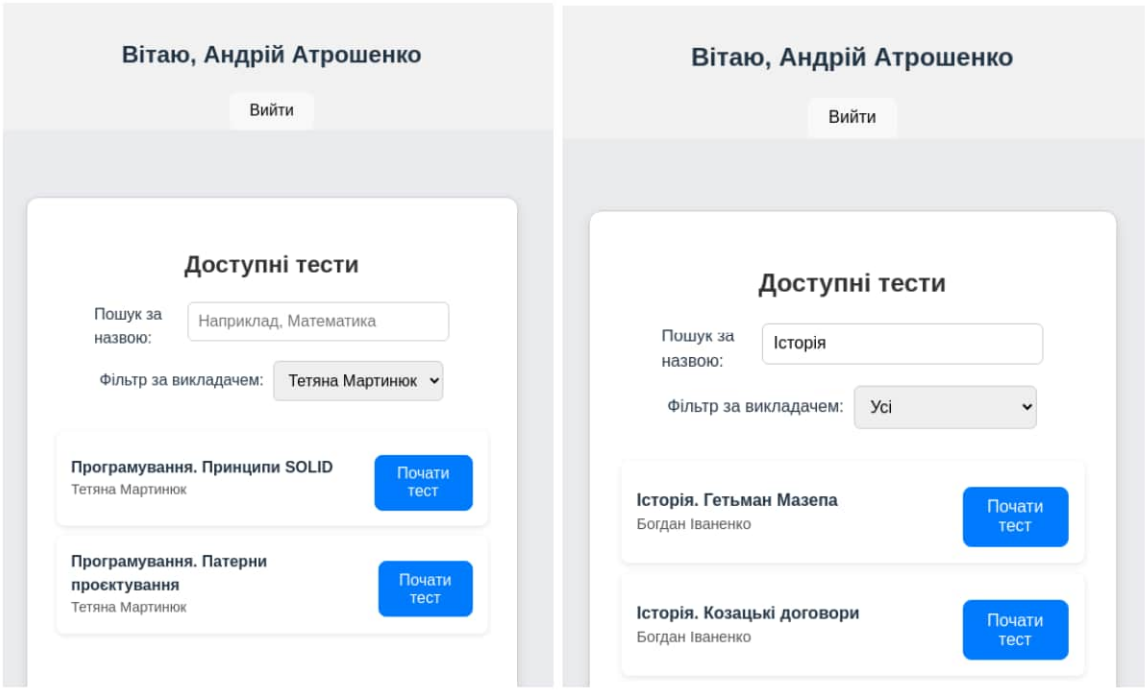


Рисунок 2.11 – Список доступних тестів усіх викладачів

Приклад фільтрації тестів наведено на рисунку 2.12.



а) б)

Рисунок 2.12 – Фільтрація тестів: а) за викладачем; б) за назвою

Наступна сторінка призначена для проходження студентом конкретного тесту (рис. 2.13). При завантаженні вона автоматично отримує testId з адресного рядка та, використовуючи токен авторизації, надсилає запит до сервера для завантаження структури тесту. Коли тест успішно завантажено, студент бачить заголовок тесту, список запитань та варіанти відповідей. Відповіді зберігаються в локальному стані. Після заповнення студент натискає кнопку "Завершити тест", яка надсилає відповіді на сервер. Якщо все успішно — відображається повідомлення про надсилання та виконується перехід на сторінку студента. У разі помилки надсилання — виводиться відповідне повідомлення. Ця сторінка реалізує повноцінну логіку проходження тесту з підтримкою різних типів запитань, відправкою результатів і базовою обробкою помилок.

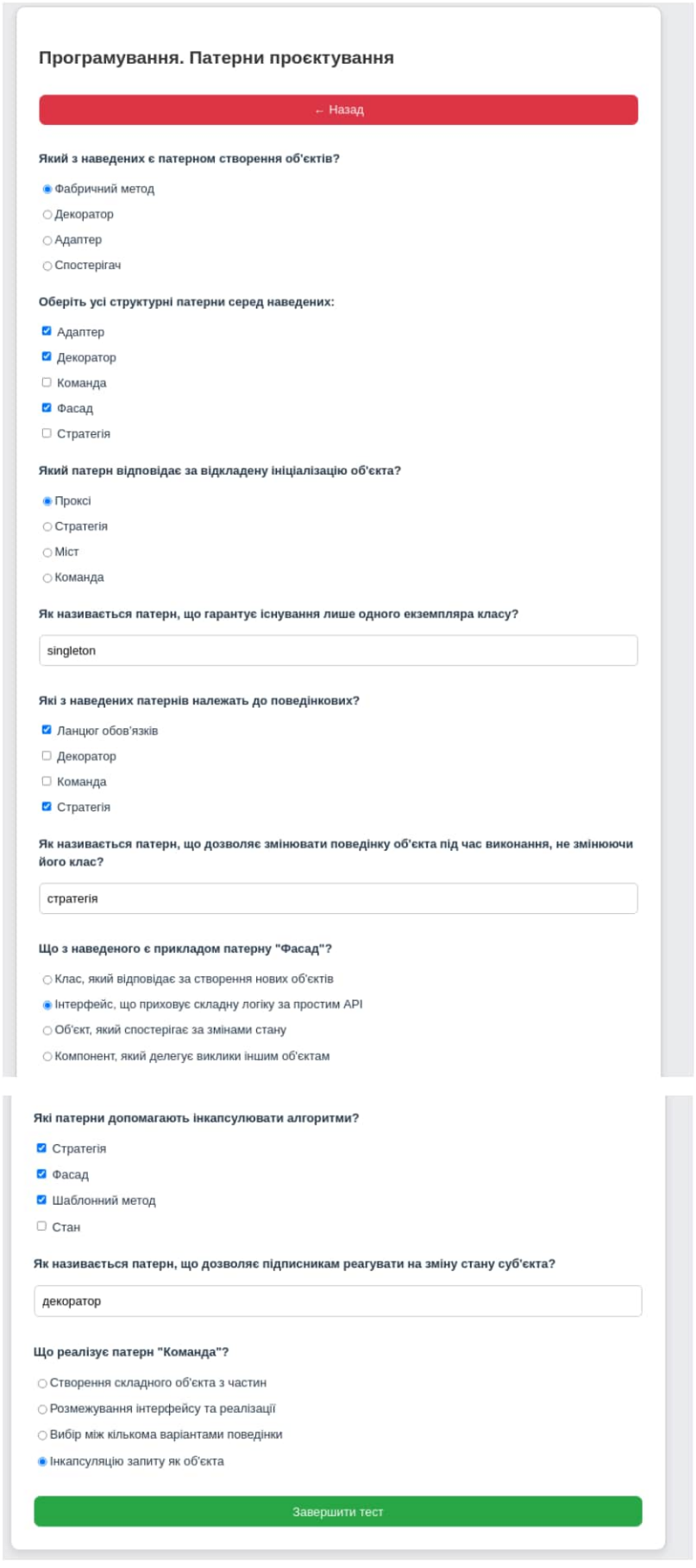


Рисунок 2.13 – Сторінка проходження тесту

Створений тест пройдено і тепер запис про його проходження можна перевірити в акаунті викладача через список тестів (рис 2.14). Під час проходження тесту було навмисно зроблено три помилки, що тепер видно на сторінці результатів.

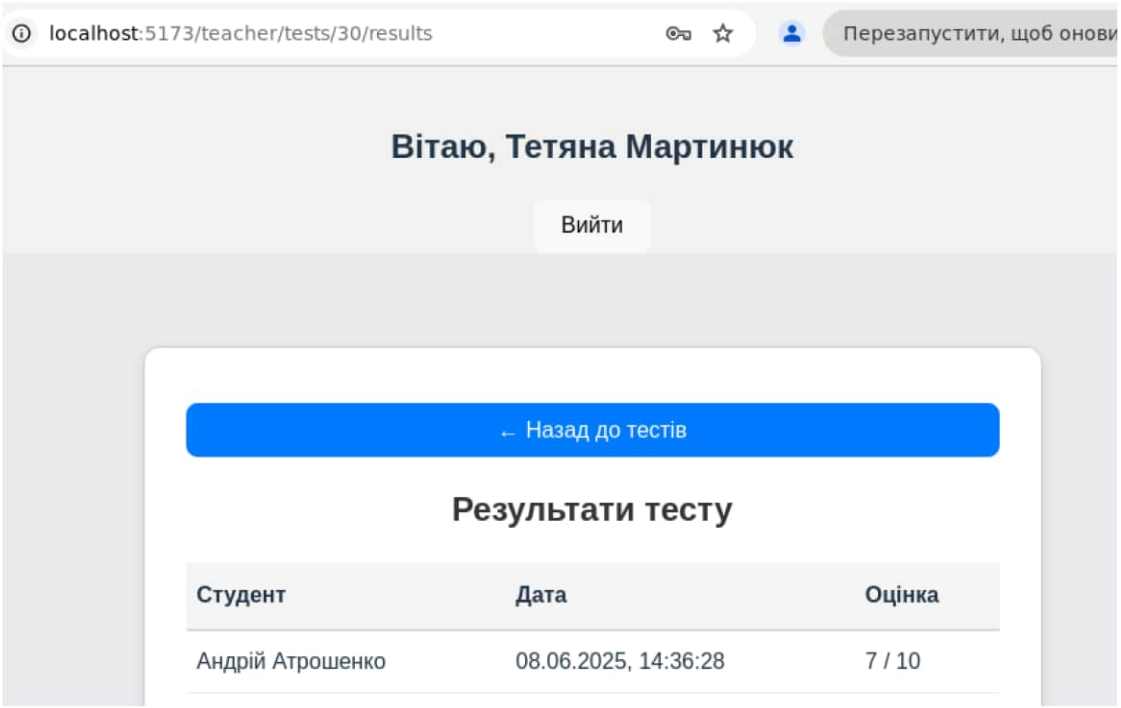


Рисунок 2.14 – Сторінка результатів тесту

Ця сторінка призначена для викладача, щоб переглядати результати студентів за певним тестом. Після завантаження сторінки з URL-адреси, що ідентифікує конкретний тест. Якщо сервер повернув порожній список, то з’явиться повідомлення “Немає результатів”. Якщо ж дані отримано, вони відображаються у вигляді таблиці.

У таблиці для кожного результату вказано:

- логін студента;
- дату та час здачі тесту;
- набрану оцінку.

Також на сторінці є кнопка повернення назад до списку тестів викладача.

2.11 Тестування API та системи за допомогою Postman

Для тестування розробленої системи використано платформу Postman, який використовується для тестування REST API, оскільки дозволяє швидко надсилати HTTP-запити до сервера й переглядати відповіді. Це використано для перевірки чи правильно працюють маршрути, передається чи тіло запиту, як реагує сервер на помилки або неправильні дані.

За допомогою методу POST за посиланням http://localhost:5000/api/auth/register створено запит на створення нового користувача, запит виконано успішно (рис. 2.15).

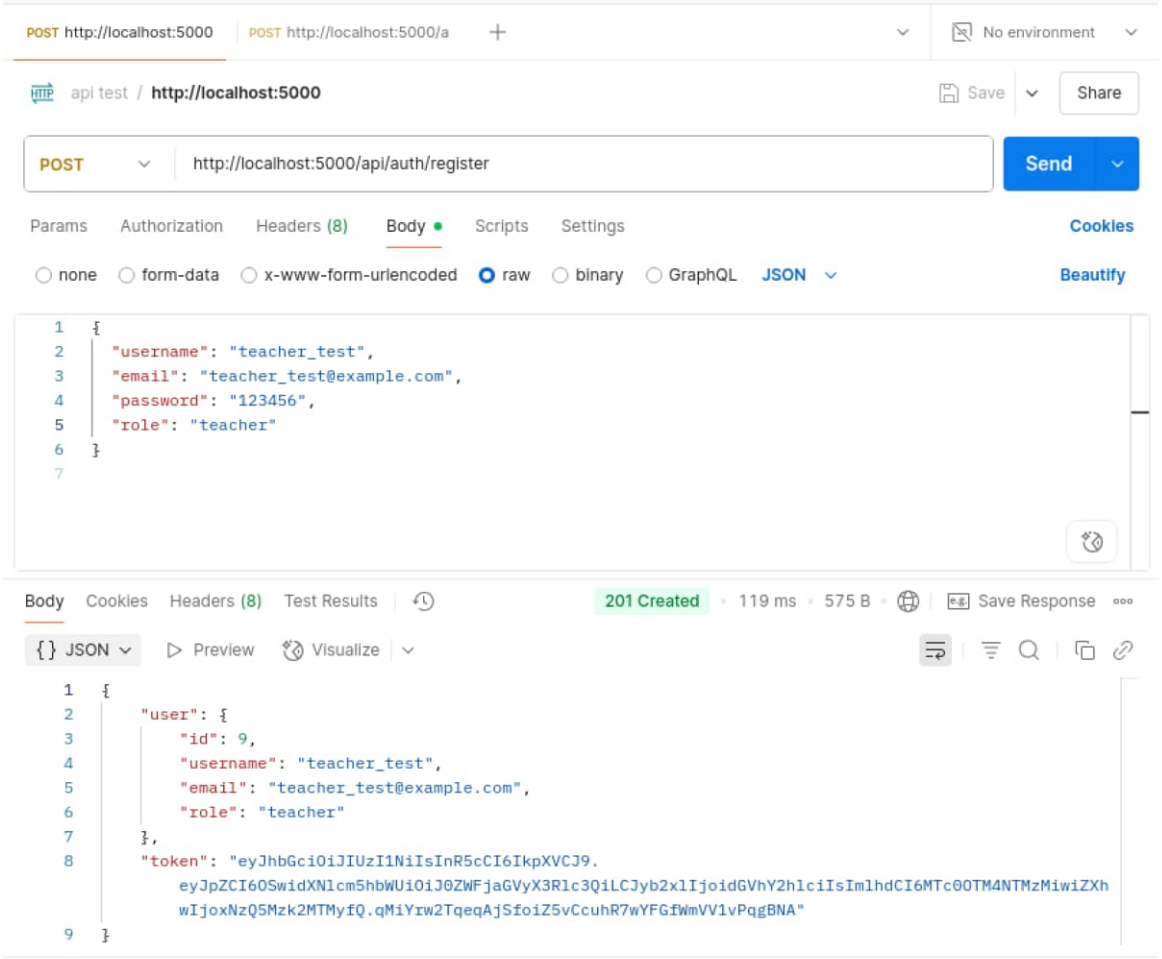


Рисунок 2.15 – Запит на створення користувача

Наступним для перевірки є логін у систему. Для цього використано метод POST за посиланням http://localhost:5000/api/auth/login. Вхід успішний (рис 2.16).

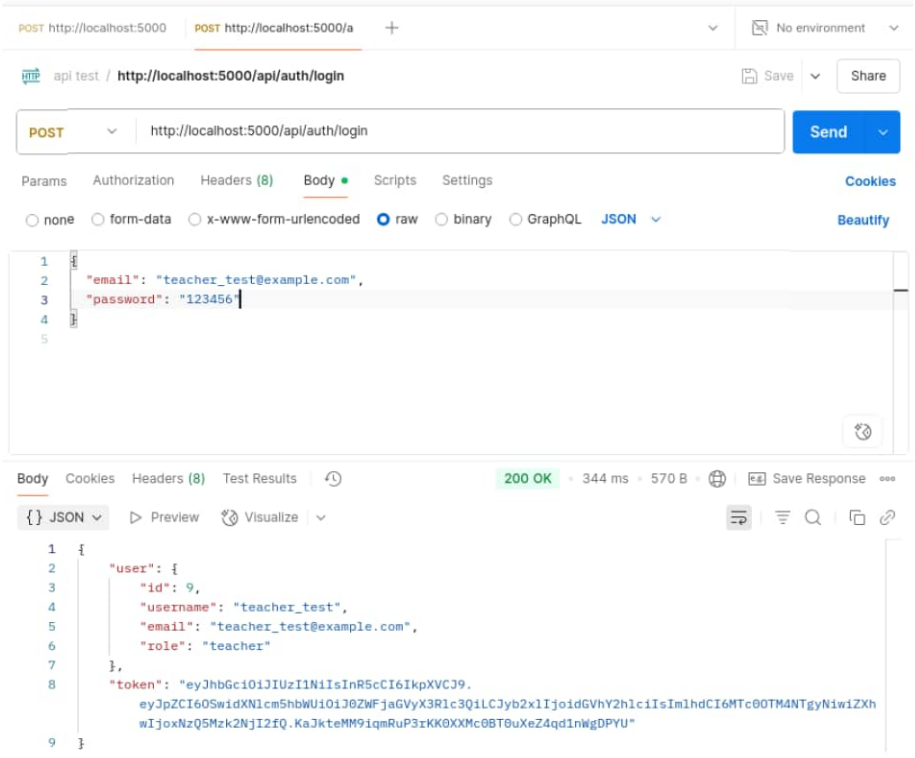


Рисунок 2.16 – Запит на логін користувача

Далі тестування створення тесту. Для авторизації отриманий токен введено у вкладку Headers. За посиланням http://localhost:5000/api/tests методом POST успішно створено новий тест (рис 2.17).

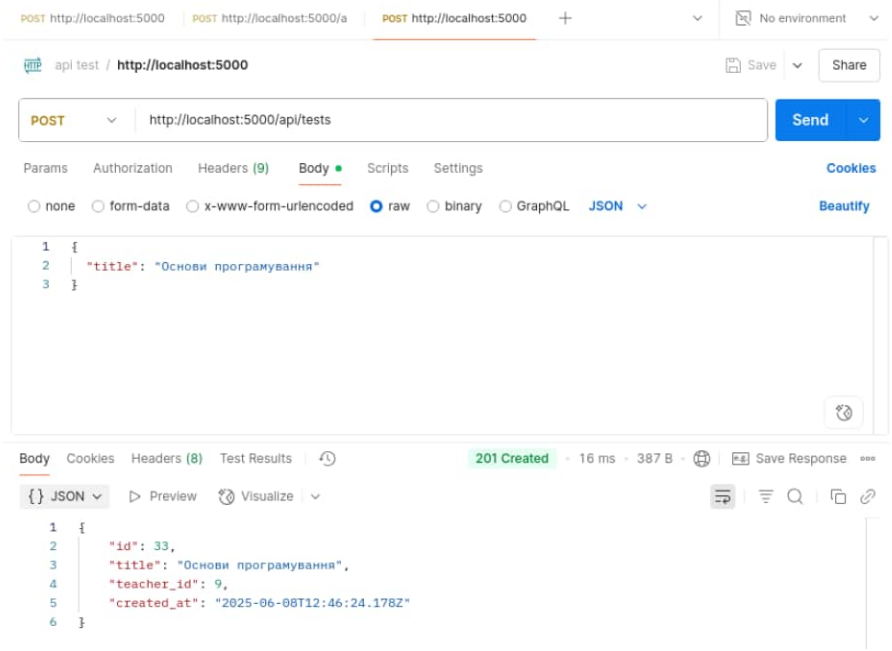


Рисунок 2.17 – Запит на створення тесту

Методом POST за посиланням http://localhost:5000/api/tests/33/questions створено три питання до тесту (рис. 2.18). У вкладці Headers теж вказаний токен.

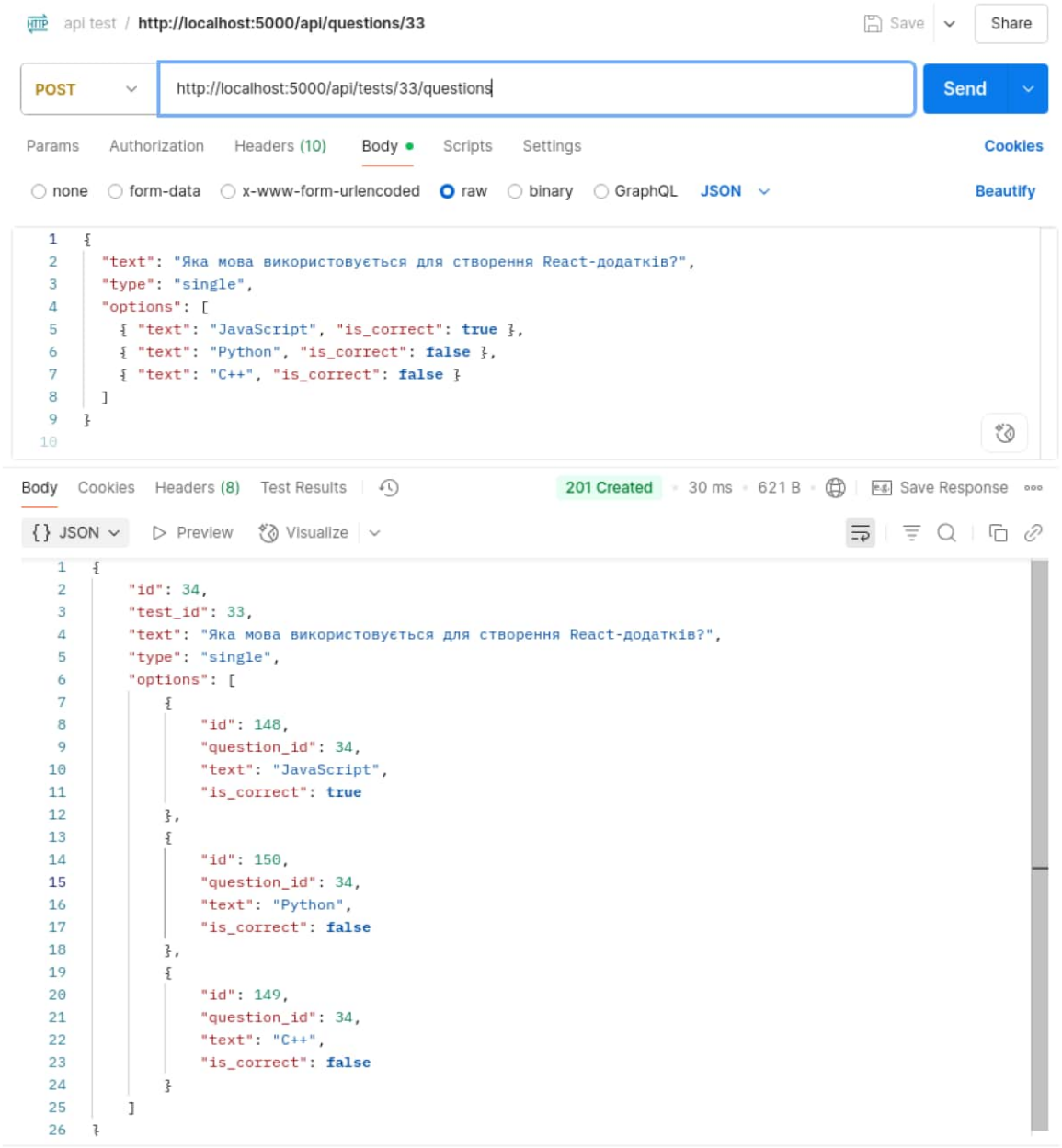


Рисунок 2.18 – Запит на створення одно з трьох питань

Для перевірки створених питань використано метод GET за посиланням http://localhost:5000/api/tests/33/questions. Отримані питання на рисунку 2.19. Запит відправлено з токеном.

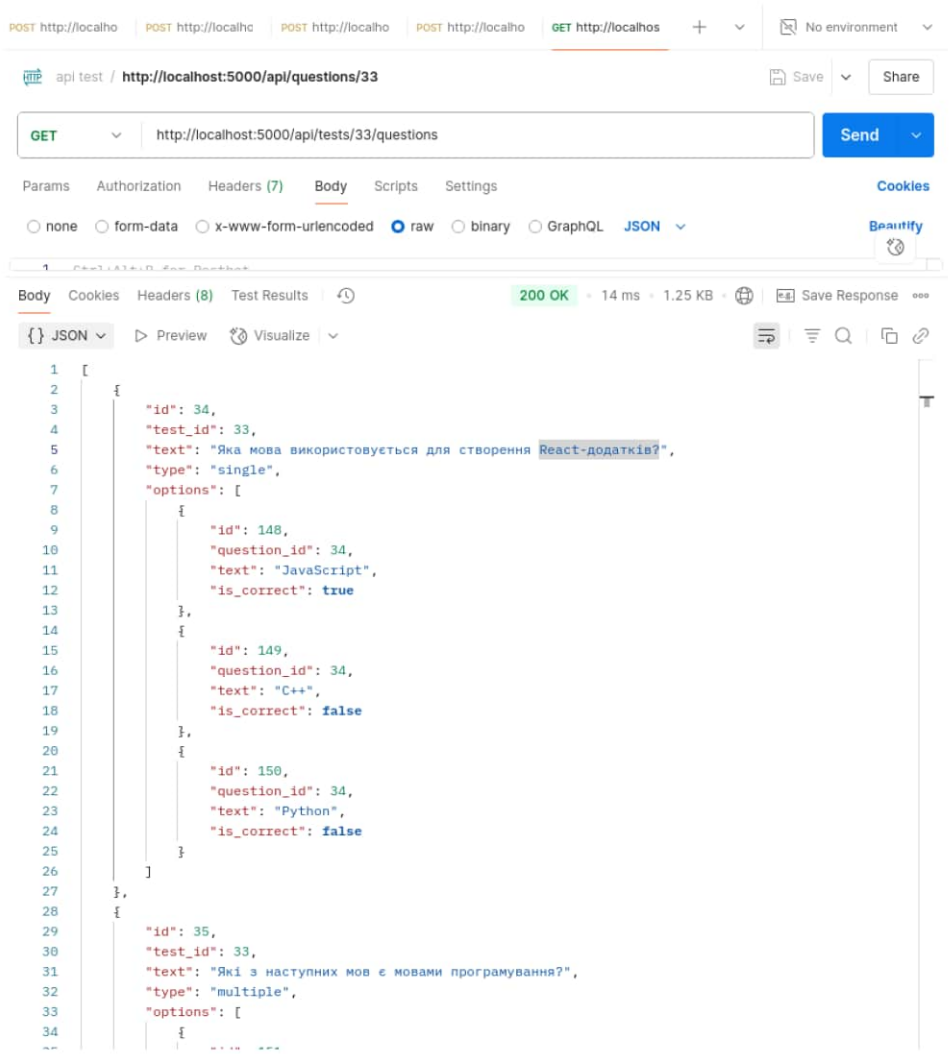


Рисунок 2.19 – Запит на отримання тестів

У разі, якщо до запиту не додано токен, то система повертає помилку про необхідність авторизації (рис. 2.20).

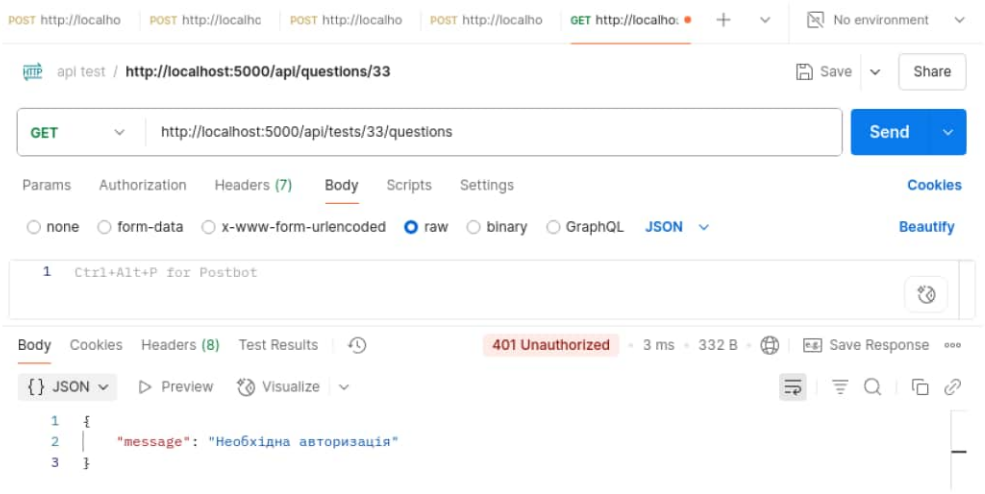


Рисунок 2.20 – Невдалий запит через відсутність токену

Тепер було створено користувача студента для проходження тесту. За допомогою метода POST за посиланням http://localhost:5000/api/results надіслано відповіді до серверу на створений тест (рис 2.21).

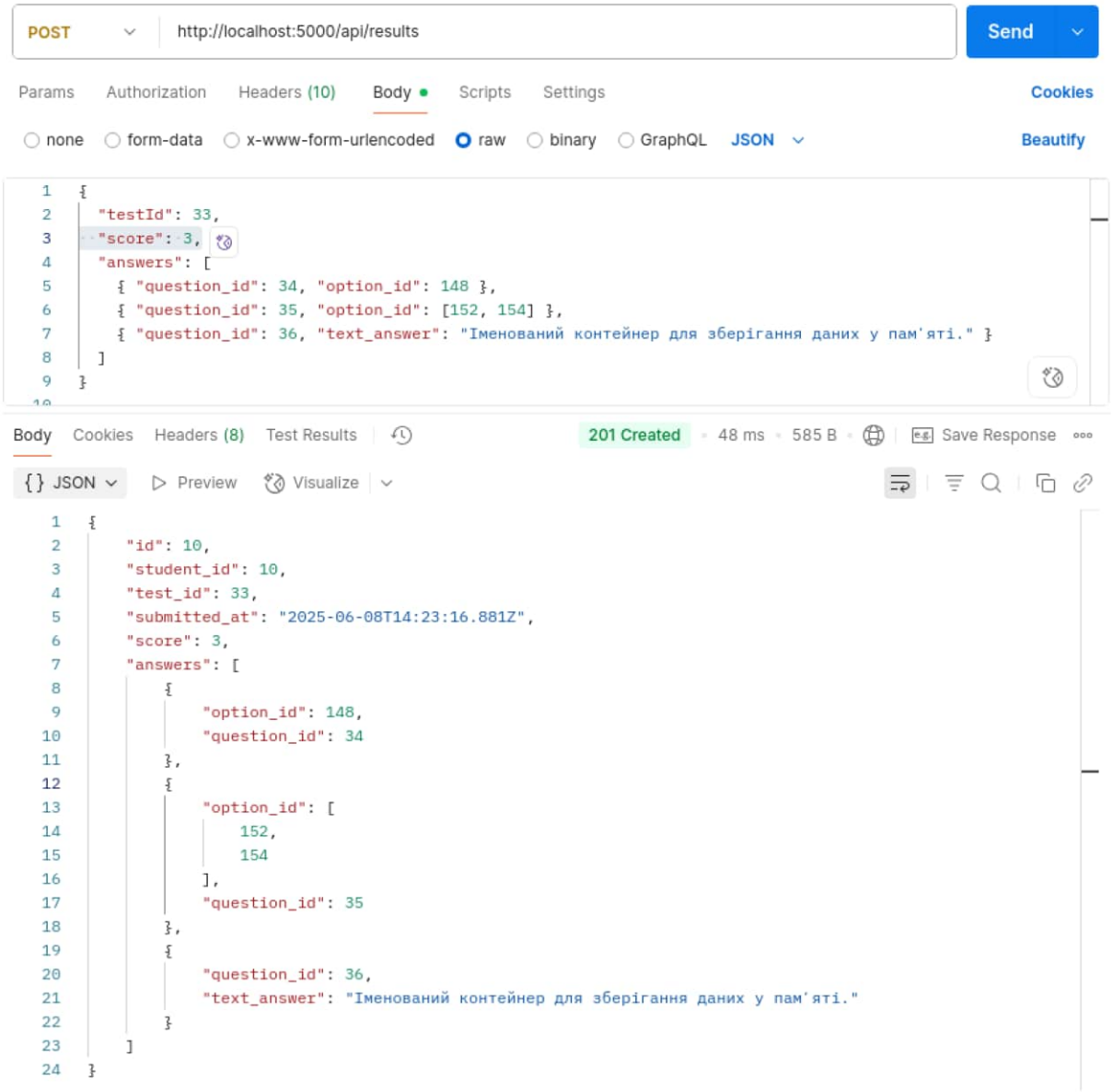


Рисунок 2.21 – Проходження створеного тесту

Для перевірки результатів тесту, виконано вхід у профіль вчителя та надіслано GET запит за посиланням http://localhost:5000/api/results/test/33. Система повернула результат тесту та оцінку (рис 2.22).

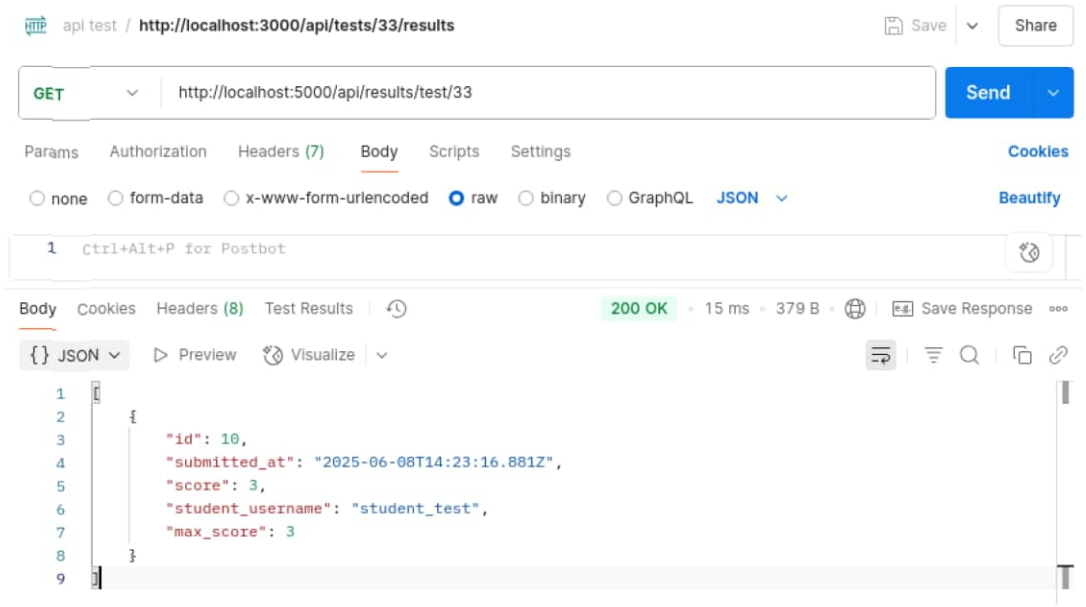


Рисунок 2.22 – Результати проходження тесту

Тепер можна перевірити створені дані у веб-браузері. Сторінка із тестом у створеному профілі викладача на рисунку 2.23.

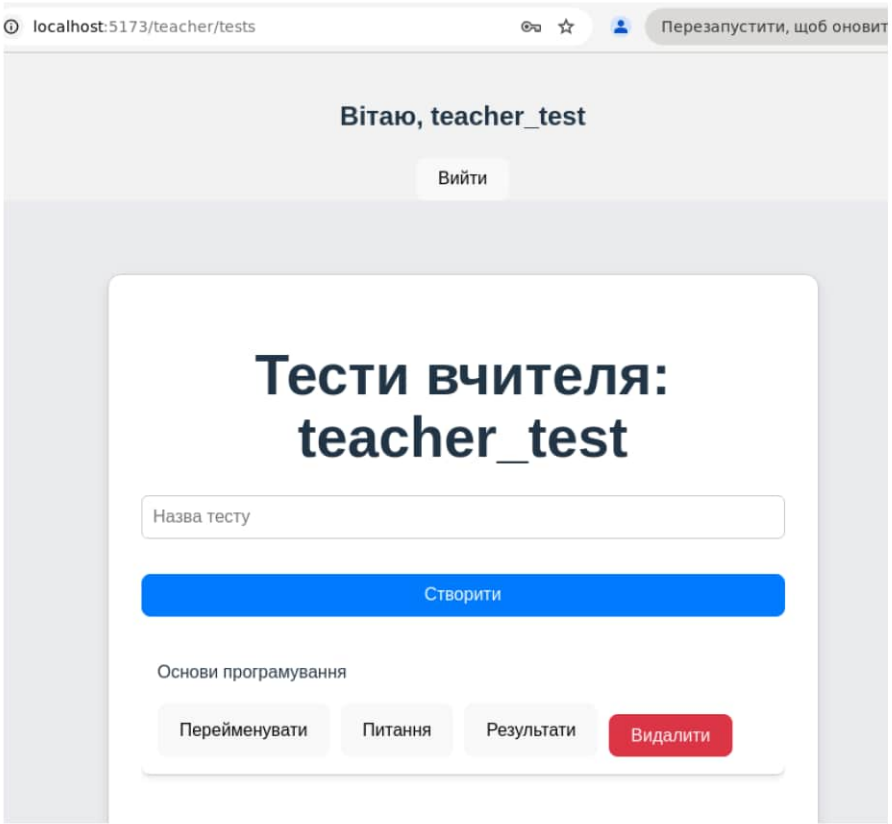


Рисунок 2.23 – Створений тест через API-запити

Створені питання для тесту надані на рисунку 2.24 Видно три питання кожного з типів.

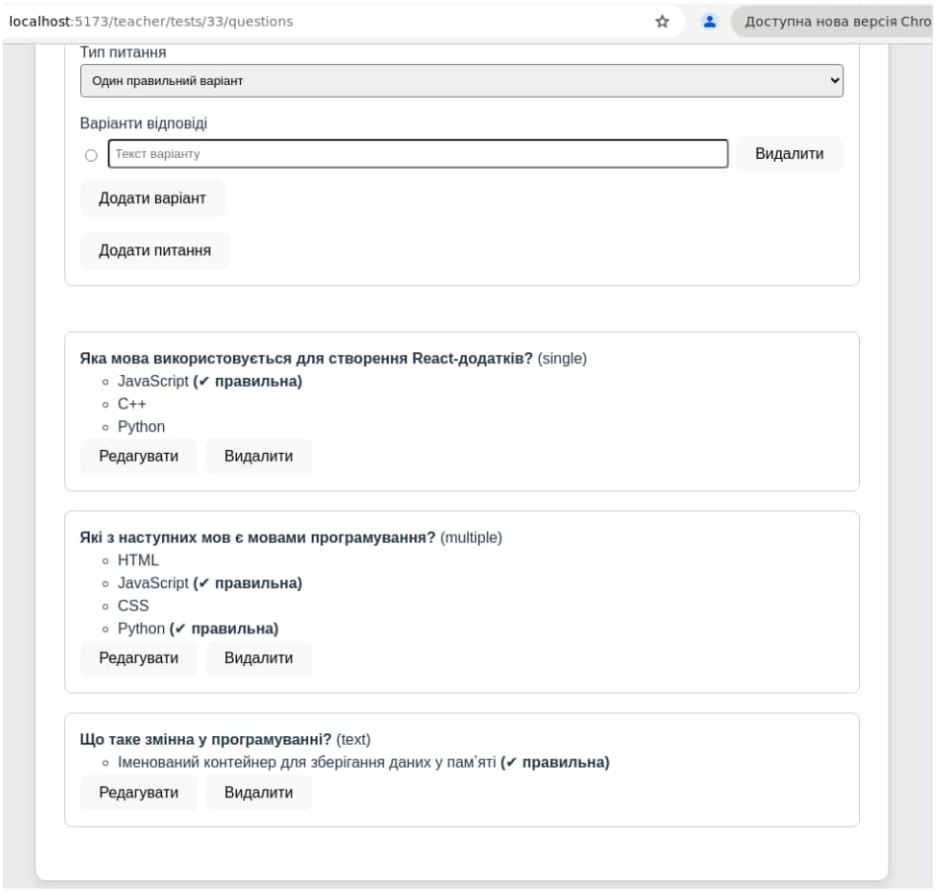


Рисунок 2.24 – Створені питання до тесту

Вивід результатів показано на рисунку 2.25. Існує єдиний запис проходження тесту від створеного користувача student_test.

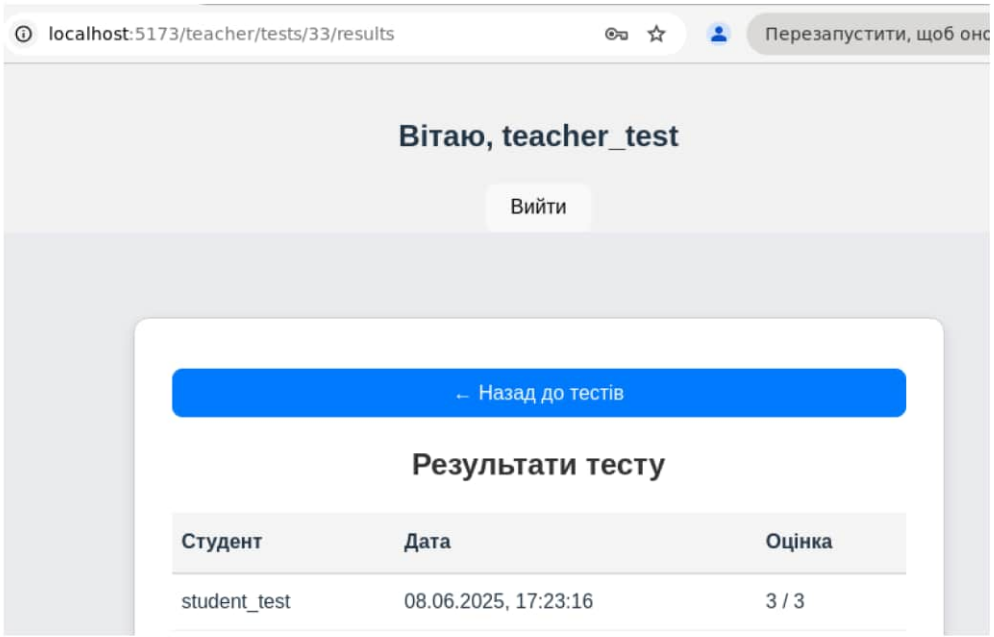


Рисунок 2.25 – Результат проходження створеного тесту

ВИСНОВКИ

У ході роботи над проєктом онлайн-сервісу тестування знань здобувачів було створено комплексну систему, яка дозволяє викладачам створювати тести, додавати декілька типів питань, а здобувачам – проходити ці тести з подальшим оцінюванням.

Перш за все, було спроєктовано структуру бази даних, що включає таблиці користувачів, тестів, питань, варіантів відповідей та результатів тестування.

З використанням бази даних стало можливим працювати з інформацією, підтримувати різні типи питань, а саме одинарний вибір, множинний вибір, текстові відповіді, та зберігати результати у вигляді JSON-структури, що забезпечує подальший аналіз та звітність.

В бекенд частині було реалізовано REST API на базі Node.js та Express.js, що обробляє запити користувачів і взаємодіє з базою даних PostgreSQL. Впроваджено аутентифікацію та авторизацію користувачів за допомогою токенів, що дозволяє розмежувати права доступу між студентами та викладачами. Завдяки цьому викладачі мають змогу створювати та редагувати тести, а студенти — проходити створені тести.

Фронтенд частина, розроблена з використанням JavaScript-бібліотек React та Vite, забезпечує веб інтерфейс для користувачів. Реалізовано сторінки для реєстрації, входу, перегляду доступних тестів, їх проходження та перегляду результатів.

Для тестування API і перевірки коректності роботи серверних ендпоінтів використовувався Postman. Цей інструмент дозволив детально перевірити всі ключові функції бекенду, включаючи аутентифікацію, авторизацію, створення тестів, додавання питань, проходження тестів і збереження результатів. Процес тестування включав послідовне відпрацювання різних сценаріїв використання системи — як з боку викладача, так і зі сторони студента.

У результаті реалізовано систему, яка відповідає основним вимогам проєкту: забезпечує створення і проходження тестів, коректно обробляє різні типи відповідей, гарантує безпеку і розмежування прав користувачів, а також зберігає та відображає результати тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Learning management system [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Learning_management_system (дата звернення: 03.05.2025).

2. Що таке платформа Moodle [Електронний ресурс]. URL: https://hostpro.ua/blog/ua/what-is-the-moodle-platform/ (дата звернення: 03.05.2025).

3. What is Canvas? [Електронний ресурс]. URL: https://community.canvaslms.com/t5/Canvas-Basics-Guide/What-is-Canvas/ta-p/45 (дата звернення: 03.05.2025).

4. Forma Lms Features [Електронний ресурс]. URL: https://www.formalms.org/features.html (дата звернення: 03.05.2025).

5. Hot Potatoes version 7 [Електронний ресурс]. URL: https://hotpot.uvic.ca/hotpot7_help.pdf (дата звернення: 03.05.2025).

6. iSpring QuizMaker Overview [Електронний ресурс]. URL: https://www.ispringsolutions.com/ispring-quizmaker (дата звернення: 02.05.2025).

7. How to Use Google Forms: The Ultimate Guide [Електронний ресурс]. URL: https://paperform.co/google-forms/how-to-use/ (дата звернення: 03.05.2025).

8. Moodle architecture [Електронний ресурс]. URL: https://docs.moodle.org/dev/Moodle_architecture#How_Moodle_code_is_organised (дата звернення: 08.05.2025).

9. Moodle API Guides [Електронний ресурс]. URL: https://moodledev.io/docs/5.1/apis (дата звернення: 08.05.2025).

10. Canvas LMS Overview [Електронний ресурс]. URL: https://deepwiki.com/instructure/canvas-lms/1-canvas-lms-overview (дата звернення: 08.05.2025).

11. Canvas LMS - REST API and Extensions Documentation [Електронний ресурс]. URL: https://www.canvas.instructure.com/doc/api/ (дата звернення: 08.05.2025).

12. Google Forms API Overview [Електронний ресурс]. URL: https://developers.google.com/workspace/forms/api/guides?hl=en (дата звернення: 08.05.2025).

13. The 8 Best Online Test Makers for Teachers [Електронний ресурс]. URL: https://www.ispringsolutions.com/blog/5-test-makers-for-teachers-and-e-learning-developers (дата звернення: 08.05.2025).

14. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language, 7th edition. O'Reilly Media, 2020. 704 p.

15. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs. O'Reilly Media, 2013. 280 p.

16. David Herron. Node.js Web Development, 4th edition. Packt Publishing, 2018. 450 p.

ДОДАТОК А

КОД ПРОГРАМИ

А.1 Текст програмного коду backend-частини

Код файлу server.js:

```
/**
 * Основний серверний файл проєкту. Ініціалізує Express-додаток,
 * підключає middleware, маршрути API та запускає сервер на вказаному порту.
 * Реалізує точку входу для клієнт-серверної взаємодії.
 */

import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';

import authRoutes from './routes/authRoutes.js';
import testRoutes from './routes/testRoutes.js';
import studentTestRoutes from './routes/studentTestRoutes.js';
import resultRoutes from './routes/resultRoutes.js';
import teacherRoutes from './routes/teacherRoutes.js';
import questionRoutes from './routes/questionRoutes.js';

dotenv.config();

const app = express();

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

app.use('/api/auth', authRoutes);
app.use('/api/tests', testRoutes);
app.use('/api/student', studentTestRoutes);
app.use('/api/results', resultRoutes);
app.use('/api/teachers', teacherRoutes);
app.use('/api/questions', questionRoutes);

const PORT = process.env.PORT;

app.listen(PORT, () => {
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

Код файлу authMiddleware.js:

```
/**
 * Middleware-функції для перевірки JWT-токенів авторизації.
 * Забезпечує доступ до ресурсів лише для авторизованих користувачів
 * та обмежує дії для ролі "teacher".
 */
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

dotenv.config();
```

```
const JWT_SECRET = process.env.JWT_SECRET;

function verifyToken(req, res, next) {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Необхідна авторизація' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ message: 'Токен недійсний' });
  }
}

function verifyTeacher(req, res, next) {
  if (req.user.role !== 'teacher') {
    return res.status(403).json({ message: 'Доступ дозволений лише для
вчителів' });
  }
  next();
}

export { verifyToken, verifyTeacher };
```

Код файлу authRoutes.js:

```
/**
 * Маршрути для реєстрації та входу користувачів у систему.
 * Підключає контролери для обробки відповідних запитів.
 */
import { Router } from 'express';
import { register, login } from '../controllers/authController.js';

const router = Router();

router.post('/register', register);
router.post('/login', login);

export default router;
```

Код файлу questionRoutes.js:

```
/**
 * Маршрути для оновлення та видалення питань тестів.
 * Доступні лише авторизованим вчителям.
 */
import express from 'express';
import { verifyToken, verifyTeacher } from '../middlewares/authMiddleware.js';
import * as questionController from '../controllers/questionController.js';
```

```
const router = express.Router();

router.use(verifyToken, verifyTeacher);

router.put('/:id', questionController.updateQuestion);
router.delete('/:id', questionController.deleteQuestion);

export default router;
```

Код файлу resultRoutes.js:

```
/**
* Маршрути для надсилання та перегляду результатів тестування.
 * Доступ учнів та вчителів розмежовується за роллю.
 */
import express from 'express';
import { verifyToken, verifyTeacher } from '../middlewares/authMiddleware.js';
import * as teacherResultController from
'../controllers/teacherResultController.js';
import * as testResultController from
'../controllers/testResultController.js';

const router = express.Router();

router.post('/', verifyToken, testResultController.submitTest);
router.get('/', verifyToken, testResultController.getResults);

router.get('/test/:testId', verifyToken, verifyTeacher,
teacherResultController.getResultsByTest);

export default router;
```

Код файлу studentTestRoutes.js:

```
/**
 * Маршрути для отримання тестів на проходження та розрахунку баллів з тесту.
 */
import express from 'express';
import { verifyToken } from '../middlewares/authMiddleware.js';
import * as studentTestController from
'../controllers/studentTestController.js';

const router = express.Router();

router.use(verifyToken)

router.get('/tests/:testId', studentTestController.getTestWithQuestions);
router.post('/tests/:testId/submit', studentTestController.submitTest);

export default router;
```

Код файлу teacherRoutes.js:

```
/**
 * Маршрути для отримання списку вчителів.
*/
import express from 'express';
import * as teacherController from '../controllers/teacherController.js';

const router = express.Router();

router.get('/', teacherController.getAllTeachers);

export default router;
```

Код файлу testRoutes.js:

```
/**
 * Маршрути для роботи з тестами та питаннями для вчителів.
 * Вивід тестів з можливістю фільтрування для студентів.
 */
import express from 'express';
import { verifyToken, verifyTeacher } from '../middlewares/authMiddleware.js';
import * as testController from '../controllers/testController.js';
import * as questionController from '../controllers/questionController.js';

const router = express.Router();

router.get('/public', verifyToken, testController.getAvailableTests);
router.get('/tests/:testId/questions', verifyToken,
questionController.getQuestionsForTest);

router.use(verifyToken, verifyTeacher);

router.get('/', testController.getTests);
router.post('/', testController.createTest);
router.put('/:id', testController.updateTest);
router.delete('/:id', testController.deleteTest);

router.get('/:testId/questions', questionController.getQuestionsForTest);
router.post('/:testId/questions', questionController.createQuestion);

export default router;
```

Код файлу authController.js:

```
/**
 * Модуль контролера аутентифікації.
 * Відповідає за реєстрацію та вхід користувачів.
 */
import pool from '../config/db.js';
import { hashPassword, comparePasswords } from '../utils/hash.js';
import { generateToken } from '../utils/jwt.js';

// Створює нового користувача в базі даних
async function createUser({ username, email, password, role }) {
  const query = `
```

```
    INSERT INTO users (username, email, password, role)
    VALUES ($1, $2, $3, $4)
    RETURNING id, username, email, role
  `;
  const values = [username, email, password, role];
  const result = await pool.query(query, values);
  return result.rows[0];
}

// Отримує користувача за email з бази даних
async function getUserByEmail(email) {
  const query = 'SELECT * FROM users WHERE email = $1';
  const result = await pool.query(query, [email]);
  return result.rows[0];
}

// Обробник реєстрації нового користувача
async function register(req, res) {
  try {
    const { username, email, password, role } = req.body;

    if (!username || !email || !password || !role) {
      return res.status(400).json({ message: 'Всі поля обов\'язкові' });
    }
    if (!['teacher', 'student'].includes(role)) {
      return res.status(400).json({ message: 'Невірна роль' });
    }

    const existingUser = await getUserByEmail(email);
    if (existingUser) {
      return res.status(400).json({ message: 'Користувач з таким email вже
існує' });
    }

    const hashedPassword = await hashPassword(password);
    const newUser = await createUser({ username, email, password:
hashedPassword, role });

    const token = generateToken(newUser);

    res.status(201).json({ user: newUser, token });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

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

    if (!email || !password) {
      return res.status(400).json({ message: 'Email і пароль обов\'язкові' });
    }

    const user = await getUserByEmail(email);
```

```
    if (!user) {
      return res.status(400).json({ message: 'Невірний email або пароль' });
    }

    const isValid = await comparePasswords(password, user.password);
    if (!isValid) {
      return res.status(400).json({ message: 'Невірний email або пароль' });
    }

    const token = generateToken(user);

    const { password: _, ...userWithoutPassword } = user;

    res.json({ user: userWithoutPassword, token });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

export { register, login };
```

Код файлу questionController.js:

```
/**
 * Модуль контролера питань.
 * Відповідає за CRUD операції з питаннями тестів.
 */
import pool from '../config/db.js';

// Отримує всі питання для конкретного тесту
async function getQuestionsForTest(req, res) {
  const { testId } = req.params;
  const teacherId = req.user.id;

  try {
    const testCheck = await pool.query(
      'SELECT * FROM tests WHERE id = $1 AND teacher_id = $2',
      [testId, teacherId]
    );

    if (testCheck.rowCount === 0) {
      return res.status(404).json({ message: 'Тест не знайдено або доступ
заборонено' });
    }

    const questionsRes = await pool.query(
      'SELECT * FROM questions WHERE test_id = $1 ORDER BY id ASC',
      [testId]
    );

    const questionIds = questionsRes.rows.map(q => q.id);
    const optionsRes = await pool.query(
      'SELECT * FROM options WHERE question_id = ANY($1::int[])',
      [questionIds]
    );
```

```
    const questionsWithOptions = questionsRes.rows.map(question => ({
      ...question,
      options: optionsRes.rows.filter(opt => opt.question_id === question.id)
    }));

    res.json(questionsWithOptions);
  } catch (err) {
    console.error('Помилка при отриманні питань:', err);
    res.status(500).json({ message: 'Помилка сервера при отриманні питань' });
  }
}

// Створює нове питання для тесту
async function createQuestion(req, res) {
  const { testId } = req.params;
  const { text, type, options } = req.body;
  const teacherId = req.user.id;

  try {
    const testCheck = await pool.query(
      'SELECT * FROM tests WHERE id = $1 AND teacher_id = $2',
      [testId, teacherId]
    );

    if (testCheck.rowCount === 0) {
      return res.status(403).json({ message: 'Недостатньо прав доступу до
тесту' });
    }

    const questionRes = await pool.query(
      'INSERT INTO questions (test_id, text, type) VALUES ($1, $2, $3)
RETURNING *',
      [testId, text, type]
    );

    const question = questionRes.rows[0];

    const optionsToInsert = options?.length
      ? await Promise.all(
          options.map(opt =>
            pool.query(
              'INSERT INTO options (question_id, text, is_correct) VALUES ($1,
$2, $3) RETURNING *',
              [question.id, opt.text, opt.is_correct]
            )
          )
        )
      : [];

    const optionsInserted = optionsToInsert.map(res => res.rows[0]);

    res.status(201).json({ ...question, options: optionsInserted });
  } catch (err) {
    console.error('Помилка при створенні питання:', err);
    res.status(500).json({ message: 'Помилка сервера при створенні питання'
});
  }
```

```
}

// Оновлює існуюче питання
async function updateQuestion(req, res) {
  const { id } = req.params;
  const { text, type, options } = req.body;
  const teacherId = req.user.id;

  try {
    const questionCheck = await pool.query(
      `SELECT q.*, t.teacher_id FROM questions q
       JOIN tests t ON q.test_id = t.id
       WHERE q.id = $1 AND t.teacher_id = $2`,
      [id, teacherId]
    );

    if (questionCheck.rowCount === 0) {
      return res.status(404).json({ message: 'Питання не знайдено або доступ
заборонено' });
    }

    const updatedQuestion = await pool.query(
      'UPDATE questions SET text = $1, type = $2 WHERE id = $3 RETURNING *',
      [text, type, id]
    );

    await pool.query('DELETE FROM options WHERE question_id = $1', [id]);

    const insertedOptions = await Promise.all(
      options.map(opt =>
        pool.query(
          'INSERT INTO options (question_id, text, is_correct) VALUES ($1, $2,
$3) RETURNING *',
          [id, opt.text, opt.is_correct]
        )
      )
    );

    res.json({ ...updatedQuestion.rows[0], options: insertedOptions.map(r =>
r.rows[0]) });
  } catch (err) {
    console.error('Помилка при оновленні питання:', err);
    res.status(500).json({ message: 'Помилка сервера при оновленні питання'
});
  }
}

// Видаляє питання
async function deleteQuestion(req, res) {
  const { id } = req.params;
  const teacherId = req.user.id;

  try {
    const questionCheck = await pool.query(
      `SELECT q.*, t.teacher_id FROM questions q
       JOIN tests t ON q.test_id = t.id
       WHERE q.id = $1 AND t.teacher_id = $2`,
```

```
      [id, teacherId]
    );

    if (questionCheck.rowCount === 0) {
      return res.status(404).json({ message: 'Питання не знайдено або доступ
заборонено' });
    }

    await pool.query('DELETE FROM options WHERE question_id = $1', [id]);
    await pool.query('DELETE FROM questions WHERE id = $1', [id]);

    res.json({ message: 'Питання та варіанти видалено' });
  } catch (err) {
    console.error('Помилка при видаленні питання:', err);
    res.status(500).json({ message: 'Помилка сервера при видаленні питання'
});
  }
}

export { getQuestionsForTest, createQuestion, updateQuestion, deleteQuestion
};
```

Код файлу studentTestController.js:

```
/**
 * Модуль для проходження тестів
 */
import pool from '../config/db.js';

// Отримує тест разом з питаннями та варіантами відповідей для студента
async function getTestWithQuestions(req, res) {
  const { testId } = req.params;

  try {
    const testRes = await pool.query(
      `SELECT t.id, t.title, u.username AS teacher_name
       FROM tests t
       JOIN users u ON u.id = t.teacher_id
       WHERE t.id = $1`,
      [testId]
    );

    if (testRes.rows.length === 0) {
      return res.status(404).json({ message: 'Тест не знайдено' });
    }

    const questionsRes = await pool.query(
      `SELECT id, text, type FROM questions WHERE test_id = $1 ORDER BY id`,
      [testId]
    );

    const questions = questionsRes.rows;

    for (const question of questions) {
      if (question.type !== 'text') {
        const optionsRes = await pool.query(
          `SELECT id, text FROM options WHERE question_id = $1`,
```

```
          [question.id]
        );
        question.options = optionsRes.rows;
      }
    }

    res.json({
      ...testRes.rows[0],
      questions,
    });
  } catch (err) {
    console.error('Error fetching test:', err);
    res.status(500).json({ message: 'Server error' });
  }
}

// Обробляє відправку тесту студентом та розраховує результат
async function submitTest(req, res) {
  const { testId } = req.params;
  const { answers } = req.body;
  const studentId = req.user.id;
  let score = 0;

  try {
    const questionsRes = await pool.query(
      `SELECT id, type FROM questions WHERE test_id = $1`,
      [testId]
    );
    const questions = questionsRes.rows;

    for (const question of questions) {
      const userAnswer = answers[question.id];
      if (!userAnswer) continue;

      if (question.type === 'single') {
        const correctOption = await pool.query(
          `SELECT id FROM options WHERE question_id = $1 AND is_correct =
true`,
          [question.id]
        );
        if (
          correctOption.rows.length === 1 &&
          String(correctOption.rows[0].id) === String(userAnswer)
        ) {
          score++;
        }
      }

      if (question.type === 'multiple') {
        const correctOptions = await pool.query(
          `SELECT id FROM options WHERE question_id = $1 AND is_correct =
true`,
          [question.id]
        );
        const correctIds = correctOptions.rows.map((o) =>
String(o.id)).sort();
```

```
        const userIds = Array.isArray(userAnswer) ?
userAnswer.map(String).sort() : [];

        if (JSON.stringify(correctIds) === JSON.stringify(userIds)) {
          score++;
        }
      }

      if (question.type === 'text') {
        const correctTexts = await pool.query(
          `SELECT text FROM options WHERE question_id = $1 AND is_correct =
true`,
          [question.id]
        );
        const normalized = (str) => str.trim().toLowerCase();
        const correctValues = correctTexts.rows.map((r) =>
normalized(r.text));
        const userText = normalized(userAnswer);
        if (correctValues.includes(userText)) {
          score++;
        }
      }
    }

    await pool.query(
      `INSERT INTO test_results (student_id, test_id, submitted_at, score,
answers)
       VALUES ($1, $2, NOW(), $3, $4)`,
      [studentId, testId, score, answers]
    );

    res.json({ message: 'Тест пройдено успішно', score });
  } catch (err) {
    console.error('Error submitting test:', err);
    res.status(500).json({ message: 'Server error' });
  }
}

export { getTestWithQuestions, submitTest };
```

Код файлу teacherController.js:

```
/**
 * Модуль для отримання інформації з бази даних про викладачів
 */
import pool from '../config/db.js';

// Отримує список усіх викладачів у системі
async function getAllTeachers(req, res) {
  try {
    const { rows } = await pool.query(`
      SELECT id, username
      FROM users
      WHERE role = 'teacher'
      ORDER BY username ASC
    `);
    res.json(rows);
```

```
  } catch (error) {
    console.error('Error getting teachers:', error);
    res.status(500).json({ message: 'Помилка сервера при отриманні викладачів'
});
  }
}

export { getAllTeachers };
```

Код файлу teacherResultController.js:

```
/**
 * Модуль для виводу оцінок за тести
 */

import pool from '../config/db.js';

// Перевіряє, чи належить тест вказаному викладачу
async function isTestOwnedByTeacher(testId, teacherId) {
  const result = await pool.query(
    'SELECT id FROM tests WHERE id = $1 AND teacher_id = $2',
    [testId, teacherId]
  );
  return result.rows.length > 0;
}

// Отримує результати проходження тесту для викладача
async function getResultsByTest(req, res) {
  const { testId } = req.params;
  const teacherId = req.user.id;

  try {
    const ownsTest = await isTestOwnedByTeacher(testId, teacherId);
    if (!ownsTest) return res.status(403).json({ message: 'Доступ заборонено'
});

    const result = await pool.query(`
      SELECT
        tr.id,
        tr.submitted_at,
        tr.score,
        u.username AS student_username
      FROM test_results tr
      JOIN users u ON tr.student_id = u.id
      WHERE tr.test_id = $1
      ORDER BY tr.submitted_at DESC
    `, [testId]);

    const maxScoreResult = await pool.query(`
      SELECT COUNT(*) AS max_score
      FROM questions
      WHERE test_id = $1
    `, [testId]);

    const max_score = Number(maxScoreResult.rows[0].max_score) || 0;

    const resultsWithMaxScore = result.rows.map(row => ({
```

```
      ...row,
      max_score,
    }));

    res.json(resultsWithMaxScore);
  } catch (err) {
    console.error('Помилка при отриманні результатів:', err);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

export { getResultsByTest };
```

Код файлу testController.js:

```
/**
 * Модуль для роботи з тестами (CRUD операції)
 * Дозволяє викладачам створювати, редагувати та переглядати тести
 */
import pool from '../config/db.js';

// Створює новий тест
async function createTest(req, res) {
  try {
    const { title } = req.body;
    const teacher_id = req.user.id;

    if (!title) return res.status(400).json({ message: 'Назва тесту потрібна'
});

    const result = await pool.query(
      'INSERT INTO tests (title, teacher_id) VALUES ($1, $2) RETURNING *',
      [title, teacher_id]
    );

    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

// Отримує список тестів для поточного викладача
async function getTests(req, res) {
  try {
    const teacher_id = req.user.id;

    const result = await pool.query(
      'SELECT * FROM tests WHERE teacher_id = $1 ORDER BY id DESC',
      [teacher_id]
    );

    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Помилка сервера' });
  }
```

```
}

// Оновлює існуючий тест
async function updateTest(req, res) {
  try {
    const testId = req.params.id;
    const { title } = req.body;
    const teacher_id = req.user.id;

    const testCheck = await pool.query(
      'SELECT * FROM tests WHERE id = $1 AND teacher_id = $2',
      [testId, teacher_id]
    );

    if (testCheck.rowCount === 0)
      return res.status(404).json({ message: 'Тест не знайдено або доступ
заборонено' });

    const result = await pool.query(
      'UPDATE tests SET title = $1 WHERE id = $2 RETURNING *',
      [title, testId]
    );

    res.json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

// Видаляє тест
async function deleteTest(req, res) {
  try {
    const testId = req.params.id;
    const teacher_id = req.user.id;

    const testCheck = await pool.query(
      'SELECT * FROM tests WHERE id = $1 AND teacher_id = $2',
      [testId, teacher_id]
    );

    if (testCheck.rowCount === 0)
      return res.status(404).json({ message: 'Тест не знайдено або доступ
заборонено' });

    await pool.query('DELETE FROM tests WHERE id = $1', [testId]);

    res.json({ message: 'Тест видалено' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

// Отримує доступні тести для студентів з фільтрацією
async function getAvailableTests(req, res) {
  try {
```

```
    const { teacherId, search } = req.query;
    const params = [];
    let query = `
      SELECT tests.*, users.username AS teacher_name
      FROM tests
      JOIN users ON tests.teacher_id = users.id
    `;

    const conditions = [];
    if (teacherId) {
      conditions.push(`tests.teacher_id = $${params.length + 1}`);
      params.push(teacherId);
    }
    if (search) {
      conditions.push(`LOWER(tests.title) LIKE LOWER($${params.length + 1})`);
      params.push(`%${search}%`);
    }

    if (conditions.length > 0) {
      query += ' WHERE ' + conditions.join(' AND ');
    }

    query += ' ORDER BY tests.id DESC';

    const result = await pool.query(query, params);
    res.json(result.rows);
  } catch (error) {
    console.error('Помилка при отриманні тестів для студентів:', error);
    res.status(500).json({ error: 'Помилка при отриманні тестів' });
  }
}

export {
  createTest,
  getTests,
  updateTest,
  deleteTest,
  getAvailableTests,
};
```

Код файлу testResultController.js:

```
/**
 * Модуль для роботи з результатами тестування
 * Дозволяє зберігати та переглядати результати тестів
 */
import pool from '../config/db.js';

// Зберігає результати проходження тесту студентом
async function submitTest(req, res) {
  const { testId, answers, score } = req.body;
  const studentId = req.user.id;

  try {
    const result = await pool.query(
      `INSERT INTO test_results (student_id, test_id, submitted_at, score,
answers)
```

```
       VALUES ($1, $2, NOW(), $3, $4)
       RETURNING *`,
      [studentId, testId, score, JSON.stringify(answers)]
    );

    res.status(201).json(result.rows[0]);
  } catch (error) {
    console.error('Помилка при збереженні результату тесту:', error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

// Отримує історію результатів тестування
async function getResults(req, res) {
  const studentId = req.user.id;

  try {
    const result = await pool.query(
      `SELECT tr.id, tr.test_id, tr.score, tr.submitted_at, t.title AS
test_title, u.username AS teacher_name
       FROM test_results tr
       JOIN tests t ON tr.test_id = t.id
       JOIN users u ON t.teacher_id = u.id
       WHERE tr.student_id = $1
       ORDER BY tr.submitted_at DESC`,
      [studentId]
    );

    res.json(result.rows);
  } catch (error) {
    console.error('Помилка при отриманні результатів тестів:', error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
}

export {
  submitTest,
  getResults,
};
```

Код файлу db.js:

```
/**
 * Модуль для роботи з базою даних PostgreSQL.
 * Створює пул з'єднань з БД, використовуючи змінні оточення.
 * Експортує пул для використання в інших модулях.
 */
import { Pool } from 'pg';
import dotenv from 'dotenv';

// Завантаження змінних оточення з .env файлу
dotenv.config();

// Ініціалізація пулу з'єднань з параметрами з .env
const pool = new Pool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
```

```
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  port: process.env.DB_PORT,
});
export default pool;
```

Код файлу hash.js:

```
/**
 * Модуль для хешування та перевірки паролів.
 * Використовує bcrypt з сіллю (10 раундів).
 * Надає функції для створення хешу та порівняння паролів.
 */
import bcrypt from 'bcrypt';

const SALT_ROUNDS = 10;

// Генерує хеш пароля з використанням солі
async function hashPassword(password) {
  return await bcrypt.hash(password, SALT_ROUNDS);
}

// Порівнює введений пароль з хешем з БД
async function comparePasswords(password, hash) {
  return await bcrypt.compare(password, hash);
}
export { hashPassword, comparePasswords };
```

Код файлу jwt.js:

```
/**
 * Модуль для роботи з JWT токенами.
 * Відповідає за генерацію та верифікацію токенів доступу.
 * Використовує секретний ключ з змінних оточення.
 */
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

// Завантаження змінних оточення
dotenv.config();

// Секретний ключ для підпису токенів
const JWT_SECRET = process.env.JWT_SECRET;

// Генерує JWT токен для користувача
function generateToken(user) {
  return jwt.sign(
    { id: user.id, username: user.username, role: user.role },
    JWT_SECRET,
    { expiresIn: '3h' }
  );
}

// Перевіряє валідність JWT токена
function verifyToken(token) {
  return jwt.verify(token, JWT_SECRET);
}
export { generateToken, verifyToken };
```

А.2 Текст програмного коду frontend-частини

Код файлу App.jsx:

```
/**
 * Головний компонент додатку. Відповідає за маршрутизацію та автентифікацію.
 * Рендерить різні сторінки в залежності від ролі користувача
 * (teacher/student).
 */
import React, { useContext } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-
router-dom';
import { AuthContext, AuthProvider } from './context/AuthContext';
import Register from './pages/Register';
import Login from './pages/Login';
import TeacherTestsPage from './pages/TeacherTestsPage.jsx';
import TestQuestionsPage from './pages/TestQuestionsPage.jsx';
import StudentTestsPage from './pages/StudentTestsPage.jsx';
import StudentPassPage from './pages/StudentPassPage.jsx';
import TeacherTestResultsPage from './pages/TeacherTestResultsPage.jsx';
import { styles } from './styles';

function AppContent() {
  const { user, logoutUser } = useContext(AuthContext);

  return (
    <Router>
      {!user ? (
        <Routes>
          <Route
            path="/"
            element={
              <div style={styles.fullPageCenter}>
                <div style={styles.card}>
                  <h1 style={{ textAlign: 'center' }}>
                    Вітаємо! Будь ласка, увійдіть або зареєструйтесь
                  </h1>
                  <Login />
                  <hr style={{ width: '100%' }} />
                  <Register />
                </div>
              </div>
            }
          />
          <Route path="*" element={<Navigate to="/" replace />} />
        </Routes>
      ) : (
        <div style={styles.baseContainer}>
          <h2>
            Вітаю, {user.username}
          </h2>
          <button style={styles.button} onClick={logoutUser}>
            Вийти
          </button>

          <Routes>
            {user.role === 'teacher' && (
              <>
```

```
                <Route path="/teacher/tests" element={<TeacherTestsPage />} />
                <Route path="/teacher/tests/:testId/questions"
element={<TestQuestionsPage />} />
                <Route path="/teacher/tests/:testId/results"
element={<TeacherTestResultsPage />} />
              </>
            )}
            {user.role === 'student' && (
              <>
                <Route path="/student" element={<StudentTestsPage />} />
                <Route path="/student/tests/:testId" element={<StudentPassPage
/>} />
              </>
            )}
            <Route
              path="*"
              element={
                <Navigate
                  to={user.role === 'teacher' ? '/teacher/tests' : '/student'}
                  replace
                />
              }
            />
          </Routes>
        </div>
      )}
    </Router>
  );
}

export default function App() {
  return (
    <AuthProvider>
      <AppContent />
    </AuthProvider>
  );
}
```

Код файлу Login.jsx:

```
/**
 * Компонент форми входу. Обробляє автентифікацію користувача.
 * Валідує дані та відображає помилки при невдалому вході.
 */
import React, { useState, useContext } from 'react';
import { login } from '../services/authService';
import { AuthContext } from '../context/AuthContext';
import { styles } from '../styles';

export default function Login() {
  const { loginUser } = useContext(AuthContext);
  const [form, setForm] = useState({ email: '', password: '' });
  const [error, setError] = useState(null);

  const handleChange = e => setForm({ ...form, [e.target.name]: e.target.value
});

  const handleSubmit = async e => {
```

```
    e.preventDefault();
    const response = await login(form);
    if (response.token) {
      loginUser(response.user, response.token);
    } else {
      setError(response.message || 'Помилка входу');
    }
  };

  return (
    <form onSubmit={handleSubmit} style={{ width: '100%', boxSizing: 'border-
box', margin: 0, padding: 0 }}>
      <h2 style={{ textAlign: 'center', marginBottom: '15px' }}>Вхід</h2>
      <input
        type="email"
        name="email"
        value={form.email}
        onChange={handleChange}
        placeholder="Email"
        required
        style={styles.input}
      />
      <input
        type="password"
        name="password"
        value={form.password}
        onChange={handleChange}
        placeholder="Пароль"
        required
        style={styles.input}
      />
      <button type="submit" style={styles.blueButton}>Увійти</button>
      {error && <p style={{ color: 'red', textAlign: 'center', marginTop:
'10px' }}>{error}</p>}
    </form>
  );
}
```

Код файлу Register.jsx:

```
/**
 * Компонент форми реєстрації. Дозволяє створити новий обліковий запис.
 * Підтримує вибір ролі (teacher/student) та валідацію даних.
 */
import React, { useState, useContext } from 'react';
import { register } from '../services/authService';
import { AuthContext } from '../context/AuthContext';
import { styles } from '../styles';

export default function Register() {
  const { loginUser } = useContext(AuthContext);
  const [form, setForm] = useState({
    username: '',
    email: '',
    password: '',
    role: 'student',
  });
  const [error, setError] = useState(null);
```

```
  const handleChange = e => setForm({ ...form, [e.target.name]: e.target.value
});

  const handleSubmit = async e => {
    e.preventDefault();
    const response = await register(form);
    if (response.token) {
      loginUser(response.user, response.token);
    } else {
      setError(response.message || 'Помилка реєстрації');
    }
  };

  return (
    <form onSubmit={handleSubmit} style={{ width: '100%', boxSizing: 'border-
box', margin: 0, padding: 0 }}>
      <h2 style={{ textAlign: 'center', marginBottom: '15px'
}}>Реєстрація</h2>
      <input
        name="username"
        value={form.username}
        onChange={handleChange}
        placeholder="Ім'я"
        required
        style={styles.input}
      />
      <input
        type="email"
        name="email"
        value={form.email}
        onChange={handleChange}
        placeholder="Email"
        required
        style={styles.input}
      />
      <input
        type="password"
        name="password"
        value={form.password}
        onChange={handleChange}
        placeholder="Пароль"
        required
        style={styles.input}
      />
      <select
        name="role"
        value={form.role}
        onChange={handleChange}
        style={styles.input}
      >
        <option value="student">Студент</option>
        <option value="teacher">Вчитель</option>
      </select>
      <button type="submit"
style={styles.greenButton}>Зареєструватися</button>
```

```
      {error && <p style={{ color: 'red', textAlign: 'center', marginTop:
'10px' }}>{error}</p>}
    </form>
  );
}
```

Код файлу StudentPassPage.jsx:

```
/**
 * Сторінка проходження тесту для студента.
 * Відображає питання тесту з варіантами відповідей, обробляє вибір відповідей
 * та відправляє результати на сервер.
 */
import React, { useEffect, useState, useContext } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';
import { styles } from '../styles';

function StudentPassPage() {
  const { token } = useContext(AuthContext);
  const { testId } = useParams();
  const navigate = useNavigate();

  const [test, setTest] = useState(null);
  const [answers, setAnswers] = useState({});
  const [submitting, setSubmitting] = useState(false);
  const [error, setError] = useState('');

  useEffect(() => {
    fetchTest();
  }, []);

  const fetchTest = async () => {
    try {
      const res = await fetch(`/api/student/tests/${testId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      if (!res.ok) throw new Error('Не вдалося завантажити тест');
      const data = await res.json();
      setTest(data);
    } catch (err) {
      console.error(err);
      setError('Помилка при завантаженні тесту');
    }
  };

  const handleChange = (questionId, value) => {
    setAnswers((prev) => ({ ...prev, [questionId]: value }));
  };

  const handleMultipleChange = (questionId, optionId) => {
    setAnswers((prev) => {
      const prevAnswers = prev[questionId] || [];
      const updated = prevAnswers.includes(optionId)
        ? prevAnswers.filter((id) => id !== optionId)
        : [...prevAnswers, optionId];
      return { ...prev, [questionId]: updated };
    });
```

```
  };

  const handleSubmit = async () => {
    setSubmitting(true);
    setError('');
    try {
      const res = await fetch(`/api/student/tests/${testId}/submit`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({ answers }),
      });

      if (!res.ok) throw new Error('Помилка при надсиланні результату');
      alert('Тест надіслано!');
      navigate('/student');
    } catch (err) {
      console.error(err);
      setError('Не вдалося надіслати результат');
    } finally {
      setSubmitting(false);
    }
  };

  if (error) {
    return (
      <div style={styles.fullPageCenter}>
        <div style={styles.card}>
          <p>{error}</p>
          <button onClick={() => navigate(-1)} style={styles.redButton}>
            ← Назад
          </button>
        </div>
      </div>
    );
  }

  if (!test) {
    return (
      <div style={styles.fullPageCenter}>
        <div style={styles.card}>
          <p>Завантаження...</p>
        </div>
      </div>
    );
  }

  return (
    <div style={styles.fullPageCenter}>
      <div style={{ ...styles.card, width: '60%', maxWidth: 900 }}>
        <h2 style={styles.headerText}>{test.title}</h2>
        <button onClick={() => navigate(-1)} style={{ ...styles.redButton,
marginBottom: '1rem' }}>
          ← Назад
        </button>
```

```
        <form
          onSubmit={(e) => {
            e.preventDefault();
            handleSubmit();
          }}
        >
          {test.questions.map((question) => (
            <div key={question.id} style={{ marginBottom: '1.5rem' }}>
              <p style={{ fontWeight: 'bold' }}>{question.text}</p>

              {question.type === 'single' &&
                question.options.map((opt) => (
                  <label key={opt.id} style={styles.label}>
                    <input
                      type="radio"
                      name={`question-${question.id}`}
                      value={opt.id}
                      checked={answers[question.id] === opt.id}
                      onChange={() => handleChange(question.id, opt.id)}
                    />
                    {' ' + opt.text}
                  </label>
                ))}

              {question.type === 'multiple' &&
                question.options.map((opt) => (
                  <label key={opt.id} style={{ ...styles.label, display:
'block' }}>
                    <input
                      type="checkbox"
                      value={opt.id}
                      checked={answers[question.id]?.includes(opt.id) ||
false}
                      onChange={() => handleMultipleChange(question.id,
opt.id)}
                    />
                    {' ' + opt.text}
                  </label>
                ))}

              {question.type === 'text' && (
                <input
                  type="text"
                  value={answers[question.id] || ''}
                  onChange={(e) => handleChange(question.id, e.target.value)}
                  placeholder="Ваша відповідь"
                  style={styles.input}
                />
              )}
            </div>
          ))}
          <button type="submit" disabled={submitting}
style={styles.greenButton}>
            {submitting ? 'Надсилання...' : 'Завершити тест'}
          </button>
        </form>
```

```
      </div>
    </div>
  );
}

export default StudentPassPage;
```

Код файлу StudentTestsPage.jsx:

```
/**
 * Сторінка зі списком доступних тестів для студента.
 * Надає можливість фільтрувати тести за викладачем та пошуку за назвою.
 * Дозволяє перейти до сторінки проходження обраного тесту.
 */
import React, { useEffect, useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';
import { styles } from '../styles';

function StudentTestsPage() {
  const navigate = useNavigate();
  const { token } = useContext(AuthContext);
  const [tests, setTests] = useState([]);
  const [teachers, setTeachers] = useState([]);
  const [selectedTeacherId, setSelectedTeacherId] = useState('');
  const [searchTitle, setSearchTitle] = useState('');

  useEffect(() => {
    fetchTeachers();
  }, []);

  useEffect(() => {
    fetchTests();
  }, [selectedTeacherId, searchTitle]);

  const fetchTeachers = async () => {
    try {
      const response = await fetch(`/api/teachers`, {
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
      });
      if (!response.ok) throw new Error('Помилка при завантаженні
викладачів');
      const data = await response.json();
      setTeachers(data);
    } catch (error) {
      console.error('Помилка при завантаженні викладачів:', error);
    }
  };

  const fetchTests = async () => {
    try {
      const params = new URLSearchParams();
      if (selectedTeacherId) params.append('teacherId', selectedTeacherId);
```

```
    if (searchTitle) params.append('search', searchTitle);

    const response = await fetch(`/api/tests/public?${params.toString()}`, {
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Помилка при завантаженні тестів');
    const data = await response.json();
    setTests(data);
  } catch (error) {
    console.error('Помилка при завантаженні тестів:', error);
  }
};

const handleStartTest = (testId) => {
  navigate(`/student/tests/${testId}`);
};

return (
  <div style={styles.fullPageCenter}>
            <div
        style={{...styles.card,
          width: '60%',
          maxWidth: 500,
          minHeight: '90vh',
        }}
      >
      <h2 style={styles.headerText}>Доступні тести</h2>

      <div style={{display: 'flex',
                  flexDirection: 'column',
                  alignItems: 'center',
                  width: '100%',
                  maxWidth: '450px',
                  marginBottom: '20px',}}>

        <label style={styles.label}>
          Пошук за назвою:
          <input
            type="text"
            value={searchTitle}
            onChange={(e) => setSearchTitle(e.target.value)}
            placeholder="Наприклад, Математика"
            style={styles.input}
          />
        </label>

        <label style={styles.label}>
          Фільтр за викладачем:
          <select
            value={selectedTeacherId}
            onChange={(e) => setSelectedTeacherId(e.target.value)}
            style={{  padding: '10px',
              fontSize: '1em',
```

```
                borderRadius: '6px',
                border: '1px solid #ccc',
                marginLeft: '10px',
                outline: 'none',}}
              >
                <option value="">Усі</option>
                {teachers.map((teacher) => (
                  <option key={teacher.id} value={teacher.id}>
                    {teacher.username}
                  </option>
                ))}
              </select>
            </label>
          </div>

          <ul style={{listStyleType: 'none',
                      padding: 0,
                      width: '100%',
                      maxWidth: 650,}}>
            {tests.length === 0 ? (
              <li>Немає доступних тестів</li>
            ) : (
              tests.map((test) => (
                <li key={test.id} style={styles.listItem}>
                  <div style={{ flexGrow: 1, marginRight: '20px' }}>
                    <strong>{test.title}</strong>
                    <div style={{ fontSize: '0.9em', color: '#555' }}>
                      {test.teacher_name}
                    </div>
                  </div>
                  <button
                    style={{
                            ...styles.blueButton,
                            width: 100,
                          }}
                    onClick={() => handleStartTest(test.id)}
                  >
                    Почати тест
                  </button>
                </li>
              ))
            )}
          </ul>
        </div>
      </div>
    );
}

export default StudentTestsPage;
```

Код файлу TeacherTestResultsPage.jsx:

```
/**
 * Сторінка перегляду результатів тесту для викладача.
 * Відображає таблицю з результатами студентів (ім'я, дата проходження,
 * оцінка).
```

```
 * Дозволяє викладачу аналізувати успішність студентів.
 */
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { styles } from '../styles';

const TeacherTestResultsPage = () => {
  const { testId } = useParams();
  const navigate = useNavigate();
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchResults = async () => {
      try {
        const response = await fetch(`/api/results/test/${testId}`, {
          headers: {
            Authorization: `Bearer ${localStorage.getItem('token')}`,
          },
        });
        if (!response.ok) throw new Error('Помилка при завантаженні
результатів');
        const data = await response.json();
        setResults(data);
      } catch (err) {
        console.error('Failed to fetch results:', err);
      } finally {
        setLoading(false);
      }
    };

    fetchResults();
  }, [testId]);

  if (loading) return <div style={styles.baseContainer}>Завантаження
результатів...</div>;

  return (
    <div style={styles.fullPageCenter}>
      <div
        style={{
          ...styles.card,
          maxWidth: 650,
          minHeight: '90vh',
        }}
      >
        <button onClick={() => navigate('/teacher/tests')}
style={styles.blueButton}>
          ← Назад до тестів
        </button>

        <h2 style={styles.headerText}>Результати тесту</h2>

        {results.length === 0 ? (
          <p>Немає результатів</p>
        ) : (
          <table style={{
```

```
            width: '100%',
            borderCollapse: 'collapse',
            marginTop: '20px',}}>
            <thead>
              <tr>
                <th style={styles.tableHeaderCell}>Студент</th>
                <th style={styles.tableHeaderCell}>Дата</th>
                <th style={styles.tableHeaderCell}>Оцінка</th>
              </tr>
            </thead>
            <tbody>
              {results.map(({ id, student_username, submitted_at, score,
max_score }) => (
                <tr key={id}>
                  <td style={styles.tableCell}>{student_username}</td>
                  <td style={styles.tableCell}>{new
Date(submitted_at).toLocaleString()}</td>
                  <td style={styles.tableCell}>
                    {score} / {max_score !== undefined ? max_score : '-'}
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        )}
      </div>
    </div>
  );
};

export default TeacherTestResultsPage;
```

Код файлу TeacherTestsPage.jsx:

```
/**
 * Головна сторінка викладача для керування тестами.
 * Надає можливість створювати, редагувати та видаляти тести.
 * Містить посилання на сторінки керування питаннями та перегляду результатів.
 * CRUD операції інкапсулювано у кастомний хук.
 */
import React, { useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';
import { styles } from '../styles';
import { useTeacherTestsPage } from '../hooks/useTeacherTestsPage';

export default function TeacherTestsPage() {
  const { user, token } = useContext(AuthContext);
  const navigate = useNavigate();

  const {
    title,
    setTitle,
    tests,
    error,
    loading,
    editId,
    editTitle,
```

```
    setEditTitle,
    handleCreateTest,
    startEdit,
    cancelEdit,
    handleUpdateTest,
    handleDeleteTest,
  } = useTeacherTestsPage(token);

  const goToManageQuestions = (testId) => {
    navigate(`/teacher/tests/${testId}/questions`);
  };

  const goToTestResults = (testId) => {
    navigate(`/teacher/tests/${testId}/results`);
  };

  return (
    <div style={styles.fullPageCenter}>
          <div
      style={{
        ...styles.card,
        maxWidth: 650,
        height: '90vh',

      }}
    >
        <h1 style={{ textAlign: 'center', marginBottom: 24 }}>
          Тести вчителя: {user?.username}
        </h1>

        <form
          onSubmit={handleCreateTest}
          style={{
            display: 'flex',
            gap: 12,
            marginBottom: 24,
            alignItems: 'center',
            flexWrap: 'wrap',
          }}
        >
          <input
            type="text"
            placeholder="Назва тесту"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            disabled={loading}
            style={styles.input}
          />
          <button type="submit" disabled={loading}
style={{...styles.blueButton, width: '100%'}}>
            Створити
          </button>
        </form>

        {error && (
          <p style={{ ...styles.error, marginBottom: 20, textAlign: 'center'
}}>
```

```
            {error}
          </p>
        )}

        <ul
          style={{
            paddingLeft: 0,
            listStyle: 'none',
            margin: 0,
            overflowX: 'auto',
          }}
        >
          {tests.map((test) => (
            <li
              key={test.id}
              style={{
                ...styles.listItem,
                display: 'flex',
                flexDirection: 'column', // <=== ключовий момент
                alignItems: 'flex-start',
                gap: 8,
                paddingBottom: 16,
                borderBottom: '1px solid #ccc',
              }}
            >
              {editId === test.id ? (
                <>
                  <input
                    type="text"
                    value={editTitle}
                    onChange={(e) => setEditTitle(e.target.value)}
                    disabled={loading}
                    style={styles.input}
                  />
                  <button
                    onClick={() => handleUpdateTest(test.id)}
                    disabled={loading}
                    style={styles.greenButton}
                  >
                    Зберегти
                  </button>
                  <button onClick={cancelEdit} disabled={loading}
style={styles.button}>
                    Відмінити
                  </button>
                </>
              ) : (
                <>
                  <span
                    style={{marginBottom: 8, display: 'block'}}
                  title={test.title}
                  >
                    {test.title}
                  </span>

                  <div style={{ display: 'flex', gap: '10px', flexWrap: 'wrap'
}}>
```

```
                    <button onClick={() => startEdit(test)} disabled={loading}
style={{...styles.button, width: 'auto'}}>
                      Перейменувати
                    </button>
                    <button onClick={() => goToManageQuestions(test.id)}
disabled={loading}
                    style={{...styles.button, width: 'auto'}}>
                      Питання
                    </button>
                    <button onClick={() => goToTestResults(test.id)}
disabled={loading} style={{...styles.button, width: 'auto'}}>
                      Результати
                    </button>
                    <button onClick={() => handleDeleteTest(test.id)}
disabled={loading} style={{...styles.redButton, width: 'auto'}}>
                      Видалити
                    </button>
                  </div>
                </>
              )}
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
}
styles
```

Код файлу useTeacherTestsPage.js:

```
/**
 * Кастомний хук для логіки сторінки керування тестами викладача, розбиття
 * коду виконано для легшого для легшого сприйняття коду.
 * Інкапсулює CRUD операції з тестами (створення, оновлення, видалення).
 * Повертає стан та методи для взаємодії з тестами.
 */
import { useState, useEffect } from 'react';
import { createTest, getTests, updateTest, deleteTest } from
'../services/testService';

export function useTeacherTestsPage(token) {
  const [title, setTitle] = useState('');
  const [tests, setTests] = useState([]);
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const [editId, setEditId] = useState(null);
  const [editTitle, setEditTitle] = useState('');

  useEffect(() => {
    async function fetchTests() {
      try {
        const data = await getTests(token);
        setTests(data);
      } catch (err) {
        setError(err.message);
      }
    }
```

```
    fetchTests();
  }, [token]);

  const handleCreateTest = async (e) => {
    e.preventDefault();
    if (!title.trim()) {
      setError('Введіть назву тесту');
      return;
    }
    setLoading(true);
    setError('');
    try {
      const newTest = await createTest(title.trim(), token);
      setTests((prev) => [newTest, ...prev]);
      setTitle('');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  const startEdit = (test) => {
    setEditId(test.id);
    setEditTitle(test.title);
    setError('');
  };

  const cancelEdit = () => {
    setEditId(null);
    setEditTitle('');
    setError('');
  };

  const handleUpdateTest = async (testId) => {
    if (!editTitle.trim()) {
      setError('Назва тесту не може бути порожньою');
      return;
    }
    setLoading(true);
    setError('');
    try {
      const updatedTest = await updateTest(testId, editTitle.trim(), token);
      setTests((prev) => prev.map((t) => (t.id === testId ? updatedTest :
t)));
      cancelEdit();
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  const handleDeleteTest = async (testId) => {
    if (!window.confirm('Ви впевнені, що хочете видалити цей тест?')) return;
    setLoading(true);
    setError('');
```

```
    try {
      await deleteTest(testId, token);
      setTests((prev) => prev.filter((t) => t.id !== testId));
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  return {
    title,
    setTitle,
    tests,
    error,
    loading,
    editId,
    editTitle,
    setEditTitle,
    handleCreateTest,
    startEdit,
    cancelEdit,
    handleUpdateTest,
    handleDeleteTest,
  };
}
```

Код файлу TestQuestionsPage.jsx:

```
/**
 * Сторінка керування питаннями тесту для викладача.
 * Надає інтерфейс для створення, редагування та видалення питань.
 * Підтримує різні типи питань.
 * CRUD операції інкапсульовано у кастомний хук для легшого сприйняття коду.
 */
import { useEffect, useState, useContext } from 'react';
import { useParams } from 'react-router-dom';
import { getQuestions, createQuestion, updateQuestion, deleteQuestion } from
'../services/questionService';
import { AuthContext } from '../context/AuthContext';

const emptyOption = () => ({ text: '', is_correct: false });

export function useTestQuestionsPage() {
  const { testId } = useParams();
  const { token } = useContext(AuthContext);

  const [questions, setQuestions] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [editingQuestionId, setEditingQuestionId] = useState(null);
  const [formQuestion, setFormQuestion] = useState({
    text: '',
    type: 'single',
    options: [emptyOption()],
  });
```

```
  useEffect(() => {
    loadQuestions();
  }, [testId, token]);

  async function loadQuestions() {
    setLoading(true);
    try {
      const data = await getQuestions(testId, token);
      setQuestions(data);
    } catch (e) {
      setError(e.message);
    } finally {
      setLoading(false);
    }
  }

  function resetForm() {
    setFormQuestion({
      text: '',
      type: 'single',
      options: [emptyOption()],
    });
    setEditingQuestionId(null);
  }

  function handleAddOption() {
    setFormQuestion(prev => ({
      ...prev,
      options: [...prev.options, emptyOption()],
    }));
  }

  function handleOptionChange(index, field, value) {
    setFormQuestion(prev => {
      const newOptions = [...prev.options];
      newOptions[index] = { ...newOptions[index], [field]: value };

      if (prev.type === 'single' && field === 'is_correct' && value === true)
{
        newOptions.forEach((opt, i) => {
          if (i !== index) opt.is_correct = false;
        });
      }

      if (prev.type === 'text') {
        newOptions[index].is_correct = true;
      }

      return { ...prev, options: newOptions };
    });
  }

  function handleQuestionChange(field, value) {
    setFormQuestion(prev => {
      let newOptions = prev.options;
      if (field === 'type') {
        if (value === 'text') {
```

```
          newOptions = [emptyOption()];
        } else if (prev.type === 'text') {
          newOptions = [emptyOption()];
        }
      } else if (field === 'options') {
        newOptions = value;
      }
      return { ...prev, [field]: value, options: newOptions };
    });
  }


  async function handleSubmit(e) {
    e.preventDefault();
    try {
      if (!formQuestion.text.trim()) {
        setError('Текст питання не може бути порожнім');
        return;
      }

      if (formQuestion.options.length === 0) {
        setError('Потрібно додати хоча б один варіант відповіді');
        return;
      }

      if (formQuestion.options.some(o => !o.text.trim())) {
        setError('Варіанти відповіді не можуть бути порожніми');
        return;
      }

      if ((formQuestion.type === 'single' || formQuestion.type === 'multiple')
&& !formQuestion.options.some(o => o.is_correct)) {
        setError('Потрібно позначити хоча б один правильний варіант');
        return;
      }

      if (editingQuestionId) {
        await updateQuestion(editingQuestionId, formQuestion, token);
      } else {
        await createQuestion(testId, formQuestion, token);
      }
      await loadQuestions();
      resetForm();
      setError('');
    } catch (e) {
      setError(e.message);
    }
  }

  async function handleEdit(question) {
    setEditingQuestionId(question.id);
    setFormQuestion({
      text: question.text,
      type: question.type,
      options: question.options.length > 0 ? question.options :
[emptyOption()],
    });
```

```
    setError('');
  }

  async function handleDelete(id) {
    if (window.confirm('Видалити це питання?')) {
      try {
        await deleteQuestion(id, token);
        await loadQuestions();
        if (editingQuestionId === id) resetForm();
      } catch (e) {
        setError(e.message);
      }
    }
  }

  return {
    questions,
    loading,
    error,
    formQuestion,
    editingQuestionId,
    handleAddOption,
    handleOptionChange,
    handleQuestionChange,
    handleSubmit,
    handleEdit,
    handleDelete,
    resetForm,
  };
}
```

Код файлу useTestQuestionsPage.js:

```
/**
 * Кастомний хук для логіки сторінки керування питаннями тесту.
 * Інкапсулює всю бізнес-логіку роботи з питаннями (CRUD операції).
 * Обробляє валідацію форми та синхронізацію з сервером.
 */
import { useEffect, useState, useContext } from 'react';
import { useParams } from 'react-router-dom';
import { getQuestions, createQuestion, updateQuestion, deleteQuestion } from
'../services/questionService';
import { AuthContext } from '../context/AuthContext';

const emptyOption = () => ({ text: '', is_correct: false });

export function useTestQuestionsPage() {
  const { testId } = useParams();
  const { token } = useContext(AuthContext);

  const [questions, setQuestions] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [editingQuestionId, setEditingQuestionId] = useState(null);
  const [formQuestion, setFormQuestion] = useState({
    text: '',
    type: 'single',
```

```
    options: [emptyOption()],
  });

  useEffect(() => {
    loadQuestions();
  }, [testId, token]);

  async function loadQuestions() {
    setLoading(true);
    try {
      const data = await getQuestions(testId, token);
      setQuestions(data);
    } catch (e) {
      setError(e.message);
    } finally {
      setLoading(false);
    }
  }

  function resetForm() {
    setFormQuestion({
      text: '',
      type: 'single',
      options: [emptyOption()],
    });
    setEditingQuestionId(null);
  }

  function handleAddOption() {
    setFormQuestion(prev => ({
      ...prev,
      options: [...prev.options, emptyOption()],
    }));
  }

  function handleOptionChange(index, field, value) {
    setFormQuestion(prev => {
      const newOptions = [...prev.options];
      newOptions[index] = { ...newOptions[index], [field]: value };

      if (prev.type === 'single' && field === 'is_correct' && value === true)
{
        newOptions.forEach((opt, i) => {
          if (i !== index) opt.is_correct = false;
        });
      }

      if (prev.type === 'text') {
        newOptions[index].is_correct = true;
      }

      return { ...prev, options: newOptions };
    });
  }

  function handleQuestionChange(field, value) {
    setFormQuestion(prev => {
```

```
      let newOptions = prev.options;
      if (field === 'type') {
        if (value === 'text') {
          newOptions = [emptyOption()];
        } else if (prev.type === 'text') {
          newOptions = [emptyOption()];
        }
      } else if (field === 'options') {
        newOptions = value;
      }
      return { ...prev, [field]: value, options: newOptions };
    });
  }


  async function handleSubmit(e) {
    e.preventDefault();
    try {
      if (!formQuestion.text.trim()) {
        setError('Текст питання не може бути порожнім');
        return;
      }

      if (formQuestion.options.length === 0) {
        setError('Потрібно додати хоча б один варіант відповіді');
        return;
      }

      if (formQuestion.options.some(o => !o.text.trim())) {
        setError('Варіанти відповіді не можуть бути порожніми');
        return;
      }

      if ((formQuestion.type === 'single' || formQuestion.type === 'multiple')
&& !formQuestion.options.some(o => o.is_correct)) {
        setError('Потрібно позначити хоча б один правильний варіант');
        return;
      }

      if (editingQuestionId) {
        await updateQuestion(editingQuestionId, formQuestion, token);
      } else {
        await createQuestion(testId, formQuestion, token);
      }
      await loadQuestions();
      resetForm();
      setError('');
    } catch (e) {
      setError(e.message);
    }
  }

  async function handleEdit(question) {
    setEditingQuestionId(question.id);
    setFormQuestion({
      text: question.text,
      type: question.type,
```

```
      options: question.options.length > 0 ? question.options :
[emptyOption()],
    });
    setError('');
  }

  async function handleDelete(id) {
    if (window.confirm('Видалити це питання?')) {
      try {
        await deleteQuestion(id, token);
        await loadQuestions();
        if (editingQuestionId === id) resetForm();
      } catch (e) {
        setError(e.message);
      }
    }
  }

  return {
    questions,
    loading,
    error,
    formQuestion,
    editingQuestionId,
    handleAddOption,
    handleOptionChange,
    handleQuestionChange,
    handleSubmit,
    handleEdit,
    handleDelete,
    resetForm,
  };
}
```

Код файлу styles.js:

```
/**
 * Об'єкт стилів для всього додатку.
 * Містить базові стилі компонентів, кнопок, форм та таблиць.
 * Допомогає підтримувати єдиний дизайн-систему в додатку.
 */
const baseContainer = {
  padding: '20px',
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
  backgroundColor: '#f2f2f2',
  minHeight: '100vh',
  boxSizing: 'border-box',
};

const buttonBase = {
  borderRadius: '8px',
  border: '1px solid transparent',
  padding: '0.6em 1.2em',
  fontSize: '1em',
  fontWeight: 500,
```

```
  fontFamily: 'inherit',
  cursor: 'pointer',
  transition: 'background-color 0.25s, border-color 0.25s',
  color: 'white',
  userSelect: 'none',
  width: '100%',
  boxSizing: 'border-box',
  marginTop: '10px',
};

const blueButton = {
  ...buttonBase,
  backgroundColor: '#007bff',
  borderColor: '#007bff',
};

const greenButton = {
  ...buttonBase,
  backgroundColor: '#28a745',
  borderColor: '#28a745',
};

const redButton = {
  ...buttonBase,
  backgroundColor: '#dc3545',
  borderColor: '#dc3545',
};

const input = {
  padding: '10px',
  fontSize: '1em',
  marginBottom: '10px',
  borderRadius: '6px',
  border: '1px solid #ccc',
  width: '100%',
  boxSizing: 'border-box',
  outline: 'none',
};

const card = {
  padding: '30px',
  border: '1px solid #ccc',
  borderRadius: '12px',
  backgroundColor: '#fff',
  width: '100%',
  maxWidth: '450px',
  boxShadow: '0 4px 12px rgba(0,0,0,0.1)',
  boxSizing: 'border-box',
};

const fullPageCenter = {
  display: 'flex',
  alignItems: 'center',
  justifyContent: 'center',
  minHeight: '100vh',
  width: '100vw',
  backgroundColor: '#e9ecef',
```

```
  padding: '20px',
};

const tableHeaderCell = {
  padding: '12px 8px',
  borderBottom: '2px solid #ddd',
  backgroundColor: '#f5f5f5',
  textAlign: 'left',
};

const tableCell = {
  padding: '10px 8px',
  borderBottom: '1px solid #eee',
};

const headerText = {
  fontWeight: 600,
  fontSize: '24px',
  color: '#333',
  marginBottom: '20px',
  textAlign: 'center',
};

const label = {
  marginBottom: '10px',
  display: 'flex',
  alignItems: 'center',
};

const listItem = {
  backgroundColor: '#fff',
  padding: '15px',
  marginBottom: '10px',
  borderRadius: '8px',
  boxShadow: '0 2px 6px rgba(0, 0, 0, 0.1)',
  display: 'flex',
  justifyContent: 'space-between',
  alignItems: 'center',
  boxSizing: 'border-box',
  width: '100%',
};

export const styles = {
  baseContainer,
  blueButton,
  greenButton,
  redButton,
  input,
  card,
  fullPageCenter,
  tableHeaderCell,
  tableCell,
  headerText,
  label,
  listItem,
};
```

Код файлу AuthContext.jsx:

```
/**
 * Контекст аутентифікації для керування станом користувача.
 * Зберігає дані користувача та токен, синхронізує з localStorage.
 * Надає методи для входу та виходу з системи.
 */
import React, { createContext, useState, useEffect } from 'react';

export const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(() => {
    const storedUser = localStorage.getItem('user');
    return storedUser ? JSON.parse(storedUser) : null;
  });

  const [token, setToken] = useState(() => localStorage.getItem('token'));

  useEffect(() => {
    if (user && token) {
      localStorage.setItem('user', JSON.stringify(user));
      localStorage.setItem('token', token);
    } else {
      localStorage.removeItem('user');
      localStorage.removeItem('token');
    }
  }, [user, token]);

  const loginUser = (userData, token) => {
    setUser(userData);
    setToken(token);
  };

  const logoutUser = () => {
    setUser(null);
    setToken(null);
  };

  return (
    <AuthContext.Provider value={{ user, token, loginUser, logoutUser }}>
      {children}
    </AuthContext.Provider>
  );
}
```

Код файлу authService.js:

```
/**
 * Сервіс для роботи з API аутентифікації.
 * Надає методи для реєстрації нового користувача та входу в систему.
 */
const API_URL = 'http://localhost:5000/api/auth';

export async function register(userData) {
  const response = await fetch(`${API_URL}/register`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify(userData),
  });
  return response.json();
}

export async function login(credentials) {
  const response = await fetch(`${API_URL}/login`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(credentials),
  });
  return response.json();
}
```

Код файлу questionService.js:

```
/**
 * Сервіс для роботи з питаннями тестів через API.
 * Надає CRUD операції для питань тестів (отримання, створення, оновлення,
видалення).
 */
const API_URL = 'http://localhost:5000/api';

export async function getQuestions(testId, token) {
  const response = await fetch(`${API_URL}/tests/${testId}/questions`, {
    headers: { Authorization: `Bearer ${token}` },
  });
  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка отримання питань');
  }
  return response.json();
}

export async function createQuestion(testId, question, token) {
  const response = await fetch(`${API_URL}/tests/${testId}/questions`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`
    },
    body: JSON.stringify(question),
  });
  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка створення питання');
  }
  return response.json();
}

export async function updateQuestion(questionId, updatedQuestion, token) {
  const response = await fetch(`${API_URL}/questions/${questionId}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify(updatedQuestion),
```

```
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка оновлення питання');
  }

  return response.json();
}

export async function deleteQuestion(questionId, token) {
  const response = await fetch(`${API_URL}/questions/${questionId}`, {
    method: 'DELETE',
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка видалення питання');
  }

  return response.json();
}
```

Код файлу testService.js:

```
/**
 * Сервіс для роботи з тестами через API.
 * Надає CRUD операції для тестів (створення, отримання, оновлення,
видалення).
 */
const API_URL = 'http://localhost:5000/api';

export async function createTest(title, token) {
  const response = await fetch(`${API_URL}/tests`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({ title }),
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка створення тесту');
  }

  return response.json();
}

export async function getTests(token) {
  const response = await fetch(`${API_URL}/tests`, {
    headers: {
      Authorization: `Bearer ${token}`,
    },
```

```
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка отримання тестів');
  }

  return response.json();
}

export async function updateTest(testId, title, token) {
  const response = await fetch(`${API_URL}/tests/${testId}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({ title }),
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка оновлення тесту');
  }

  return response.json();
}

export async function deleteTest(testId, token) {
  const response = await fetch(`${API_URL}/tests/${testId}`, {
    method: 'DELETE',
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    const error = await response.json();
    throw new Error(error.message || 'Помилка видалення тесту');
  }

  return response.json();
}
```