МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет Навчально-науковий інститут економіки та бізнес-освіти

Кафедра Економіки та цифрового бізнесу

Спеціальність 122 «Комп'ютерні науки»

Форма навчання Денна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Каменського Іллі Миколайовича
(прізвище, ім'я, по батькові здобувача)

на тему Розробка Web-ресурсу для створення тестів та перевірки знань
(повна назва теми)

за матеріалами

(повна назва бази дослідження)

науковий керівник к.т.н. *(наук. ступінь, вчене звання)* _____ *(підпис)* Селезньов М.Є. *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 09 червня 2025р. № 12
Завідувач кафедри _____
(підпис)
к.е.н., доцент *наук. ступень, вчене звання* Радько В.М. прізвище, ініціали

Кривий Ріг – 2025

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384
Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"07" квітня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА ЗДОБУВАЧУ

Каменському Іллі Миколайовичу

1. Тема роботи «Розробка Web-ресурсу для створення тестів та перевірки знань»

науковий керівник роботи Селезньов Максим Євгенович,
затверджені наказом вищого навчального закладу від «04» квітня 2025 р. № 224-ст (д/ф)
№ 151-ст (з/ф)

2. Строк подання здобувачем роботи 31.05.2025р.

3. Зміст кваліфікаційної роботи бакалавра, об'єкт, предмет та мета дослідження:

Розділ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ
Розділ 2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ WEB-РЕСУРСУ
Розділ 3 РОЗРОБКА ОКРЕМИХ МОДУЛІВ WEB-РЕСУРСУ ДЛЯ СТВОРЕННЯ ТЕСТІВ

Об'єкт дослідження - процес створення інформаційної системи для тестування та перевірки знань у вигляді Web-ресурсу

Предмет дослідження - функціональні та технічні аспекти розробки Web-ресурсу для створення тестів та перевірки знань

Мета кваліфікаційної роботи бакалавра – розробка Web-ресурсу, що дозволяє створювати тести, призначати їх учням та забезпечувати проходження з урахуванням ролей користувачів, а також автоматичним фіксуванням результатів

4. Дата видачі завдання 04.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 28.04.2025р.	25.04.2025
2	Підготовка розділу 2	до 16.05.2025р.	15.05.2025
3	Підготовка розділу 3	до 30.05.2025р.	29.05.2025
4	Реєстрація завершеної дипломної роботи	до 31.05.2025р.	30.05.2025
5	Отримання відгуку від наукового керівника	03-04.06.2025р.	04.06.2025
6	Отримання зовнішньої рецензії	05-06.06.2025р.	06.06.2025
7	Перевірка кваліфікаційної роботи на плагіат	02-09.06.2025р.	04.06.2025
8	Попередній захист кваліфікаційної роботи на кафедрі	03.06.2025р.	03.06.2025
9	Допуск кафедрою кваліфікаційної роботи до захисту	09.06.2025р.	09.06.2025
10	Підготовка студента до захисту в ЕК	до 17.06.2025р.	17.06.2025

Завдання підготував науковий керівник _____ **Селезньов М. Є**
(підпис) (прізвище та ініціали)

Завдання одержав здобувач _____ **Каменський. І. М.**
(підпис) (прізвище та ініціали)

Примітки:

1. Форму призначено для видачі завдання здобувачу на виконання кваліфікаційної роботи бакалавра і контролю за ходом роботи з боку кафедри.
2. Розробляється керівником кваліфікаційної роботи. Видається кафедрою.
3. Формат бланка А4 (210×297 мм), 2 сторінки.

РЕФЕРАТ

Робота містить 48 сторінок, 16 рисунків, 40 джерел, 1 таблицю і 2 додатки.

Об'єкт дослідження: процес створення інформаційної системи для тестування та перевірки знань у вигляді Web-ресурсу.

Предмет дослідження: функціональні та технічні аспекти розробки Web-ресурсу для створення тестів та перевірки знань.

Мета дослідження: розробка Web-ресурсу, що дозволяє створювати тести, призначати їх учням та забезпечувати проходження з урахуванням ролей користувачів, а також автоматичним фіксуванням результатів.

Розроблений Web-застосунок підтримує три ролі користувачів: учень, вчитель, адміністратор; дозволяє створювати тести з різними типами запитань, проходити їх у таймерному режимі та зберігати результати у базі даних. Реалізацію здійснено з використанням сучасного технологічного стеку.

Область застосування: програмний комплекс орієнтований на використання у закладах середньої та вищої освіти, забезпечує авторизацію, безпечну обробку даних і можливість масштабування у майбутньому.

ТЕСТУВАННЯ, ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, WEB-РЕСУРС, WEB-РОЗРОБКА, WEB-ІНТЕРФЕЙС.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД - База Даних;

СКБД - Система Керування Базами Даних;

API - Application Programming Interface;

HTTP - Hypertext Transfer Protocol;

JSON - JavaScript Object Notation;

JSONB - Binary JSON;

JWT - JSON Web Token;

ORM - Object-Relational Mapping;

REST - Representational State Transfer;

SQL - Structured Query Language;

UML - Unified Modeling Language;

UUID - universally unique ID;

UX - user experience.

ВСТУП

Сучасний освітній процес стрімко трансформується під впливом інформаційних технологій. Одним із ключових напрямів таких змін є автоматизація процесів контролю знань учнів, що особливо актуально в умовах дистанційного та змішаного навчання. Традиційні способи проведення тестування - паперові тести, усне опитування - мають низку обмежень, таких як складність перевірки, суб’єктивність оцінювання та значні витрати часу з боку викладача. У зв’язку з цим виникає потреба у створенні зручних, доступних та адаптованих до українських освітніх реалій програмних засобів для проведення онлайн-тестування.

Розробка Web-ресурсів для перевірки знань дозволяє автоматизувати значну частину освітнього процесу, зробити його прозорим, швидким та незалежним від людського фактору. Крім того, такі системи можуть забезпечити гнучке управління доступом, підтримку різних типів користувачів (учень, вчитель, адміністратор), збір статистичних даних про результати тестування та підвищення рівня цифрової грамотності всіх учасників освітнього процесу.

Об'єкт дослідження: процес створення інформаційної системи для тестування та перевірки знань у вигляді Web-ресурсу.

Предмет дослідження: функціональні та технічні аспекти розробки Web-ресурсу для створення тестів та перевірки знань.

Метою дослідження є розробка Web-ресурсу, що дозволяє створювати тести, призначати їх учням та забезпечувати проходження з урахуванням ролей користувачів, а також автоматичним фіксуванням результатів.

Для досягнення поставленої мети були сформульовані такі завдання:

- проаналізувати існуючі Web-рішення та виділити їх переваги й недоліки;
- визначити вимоги до майбутньої системи;
- спроєктувати архітектуру Web-застосунку та структуру бази даних;

- реалізувати клієнтську та серверну частини системи з розмежуванням ролей;
- забезпечити авторизацію, створення тестів, проходження і збереження результатів;
- перевірити працездатність системи та сформулювати можливі напрямки її розвитку.

Використовувалися такі методики:

- порівняльний аналіз;
- моделювання за допомогою UML;
- об’єктно-орієнтований підхід
- емпіричне тестування;
- системний аналіз.

Практична значущість результатів дослідження полягає в тому, що створений Web-застосунок може бути використаний як основа для впровадження в навчальних закладах, що прагнуть автоматизувати процес оцінювання знань учнів, знизити навантаження на викладачів та забезпечити об'єктивність перевірки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ

1.1 Аналіз існуючих рішень для створення тестів та перевірки знань

Для ефективної реалізації Web-ресурсу для створення тестів та перевірки знань необхідно детально дослідити наявні аналогічні рішення. Це дозволить визначити їхні переваги та недоліки, а також зрозуміти, яких функціональних можливостей бракує існуючим системам і як ці недоліки можна усунути у новому застосунку. На початковому етапі доцільно провести аналіз декількох популярних Web-ресурсів для тестування: Moodle, Google Forms, платформи «На Урок» та Google Classroom, що є поширеними та активно використовуються у сучасному освітньому середовищі. Окрім того, існуючі системи часто не враховують сучасні вимоги до адаптивного дизайну, інтеграції з мобільними пристроями та гнучкої масштабованості, що обмежує їх використання в нових освітніх моделях [1-4].

Moodle - це безкоштовна платформа з відкритим кодом, яка широко використовується для створення освітніх сайтів і дистанційних курсів. Система орієнтована переважно на вищу освіту, але успішно застосовується також у школах та інших освітніх закладах. Moodle пропонує багатий функціонал: створення різноманітних типів тестів (одиночний вибір, множинний вибір, відкриті питання), гнучке управління доступом і ролями користувачів, можливість налаштування термінів проведення тестування та багато інших опцій [2].

Переваги Moodle:

- Безкоштовність та відкритість коду, що дозволяє повністю адаптувати систему під конкретні потреби закладу.
- Велика кількість типів тестових завдань і можливість налаштування складних сценаріїв тестування.

- Гнучке управління ролями користувачів (адміністратор, вчитель, студент тощо).
- Наявність потужних аналітичних інструментів для оцінювання результатів тестувань.

Недоліки Moodle:

- Інтерфейс складний для початківців і може бути незручним для учнів молодших класів або вчителів, що мають недостатні навички роботи з IT-технологіями.
- Через великий функціонал система виглядає перевантаженою для невеликих шкіл або простих сценаріїв тестування.
- Вимагає порівняно високих технічних знань для встановлення, налаштування та підтримки платформи.

Google Forms - це безкоштовний Web-сервіс від Google, що дозволяє легко створювати та проводити онлайн-опитування й тестування. Він орієнтований на максимально простий та зрозумілий інтерфейс, завдяки чому широко використовується в освітній сфері для створення швидких тестів і опитувань без необхідності додаткових технічних знань. Вбудована інтеграція з Google Sheets дає змогу автоматично збирати та аналізувати відповіді користувачів [1].

Переваги Google Forms:

- Повністю безкоштовний сервіс з простим та зрозумілим інтерфейсом.
- Інтеграція з іншими сервісами Google (наприклад, автоматичний експорт результатів у Google Sheets).
- Швидке створення базових тестів з вибором одного або декількох варіантів відповідей, а також відкритими питаннями.
- Можливість миттєвого аналізу результатів, побудова графіків і діаграм безпосередньо після проходження тестів.

Недоліки Google Forms:

- Відсутність розвиненої системи управління ролями та доступами - всі користувачі мають однакові права.

- Обмежений функціонал для більш складних сценаріїв (немає таймерів, обмежень на кількість проходжень тесту тощо).
- Не підходить для створення складних навчальних курсів чи масштабних тестувань через відсутність розширеного адміністрування та інструментів аналізу.
- Відсутність механізмів ідентифікації учасника у випадку відкритого доступу до форми створює загрозу недостовірності результатів.

Платформа «На Урок» - це український онлайн-ресурс, орієнтований на школи, вчителів та учнів. Він дозволяє створювати та проходити навчальні тести, а також містить значну базу вже готових матеріалів, що полегшує процес підготовки до уроків. Сервіс користується популярністю серед вчителів завдяки зручному україномовному інтерфейсу та адаптованості під шкільну програму України [3].

Переваги платформи «На Урок»:

- Україномовний інтерфейс і повна адаптація до українських освітніх стандартів.
- Велика бібліотека готових тестів, якими можна вільно користуватись або адаптувати під власні потреби.
- Простота створення тестів та інтуїтивно зрозумілий інтерфейс як для учнів, так і для вчителів.
- Можливість швидкого перегляду результатів учнів, формування статистики успішності та проведення аналізу класу в цілому.

Недоліки платформи «На Урок»:

- Обмежені можливості налаштування тестів: недостатньо гнучке управління питаннями та типами відповідей.
- Відсутність розвиненої системи управління ролями користувачів: немає окремої ролі «адміністратор» з розширеними можливостями управління системою.
- Відсутність деяких важливих функцій, наприклад, таймерів тестування або чітких обмежень кількості спроб проходження одного тесту.

- Платформа передбачає обмежену гнучкість у візуальному налаштуванні тестів, а інтерфейс не завжди адаптується під мобільні пристрої, що знижує зручність для учнів.

Google Classroom - це освітній Web-сервіс від компанії Google, призначений для спрощення процесів викладання та навчання у школах та університетах. Система дозволяє організовувати класи, роздавати та перевіряти завдання, комунікувати зі студентами й централізовано керувати навчальним процесом. Однією зі складових цього сервісу є можливість створення та проведення тестувань, які інтегровані з іншими сервісами Google (зокрема Google Forms) [4].

Переваги Google Classroom:

- Простий, інтуїтивний інтерфейс і зручність інтеграції з Google Workspace (Google Drive, Gmail, Calendar тощо).
- Швидке створення класів, автоматичне додавання студентів за допомогою посилань або кодів доступу.
- Легка організація навчального процесу: роздача та перевірка завдань, виставлення оцінок і залишення коментарів.
- Автоматична інтеграція з Google Forms для створення простих тестів з миттєвою аналітикою результатів.

Недоліки Google Classroom:

- Тестування в Classroom є додатковою опцією, яка реалізована через Google Forms, що обмежує функціональні можливості складних тестових сценаріїв.
- Відсутність детальних налаштувань безпеки, таких як обмеження за кількістю спроб або часом проходження тестів.
- Обмежена система управління ролями: наявні лише базові ролі викладача та студента, немає ролі адміністратора з повним доступом до керування системою.

– Оскільки створення тестів здійснюється через сторонній сервіс (Google Forms), відсутня єдина точка керування тестуванням та результатами, що створює незручності для викладача.

На основі аналізу наведених вище аналогів було сформовано порівняльну таблицю, в якій відображено ключові особливості, переваги та недоліки існуючих Web-ресурсів для створення тестів та перевірки знань. Для більшої наочності та систематизації отриманої інформації результати порівняння представлені у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих Web-ресурсів

Критерій порівняння	**Moodle**	**Google Forms**	**На Урок (Naurok)**	**Google Classroom**
Простота використання	Низька	Дуже висока	Висока	Висока
Керування ролями	Розширене	Відсутнє	Обмежене	Базове
Створення різних типів тестів	Дуже широкі можливості	Обмежено	Обмежено	Обмежено (через Google Forms)
Україномовний інтерфейс	Є (частково)	Є	Є	Є
Таймер тестування	Є	Немає	Немає	Немає
Безпека та адміністрування	Висока	Низька	Середня	Середня
Інтеграція з іншими сервісами	Обмежена	Дуже висока	Низька	Дуже висока
Адаптація до шкільної освіти України	Часткова	Відсутня	Повна	Часткова
Підтримка мобільних пристроїв	Обмежена	Повна	Часткова	Повна
Наявність аналітики результатів	Розширена	Базова	Середня	Базова

У результаті проведеного аналізу встановлено, що кожен із розглянутих аналогів має як вагомі переваги, так і суттєві недоліки. Сервіс Moodle пропонує широкі можливості для тестування, але складний для звичайних користувачів. Google Forms вирізняється простотою використання, однак не має достатньо інструментів для адміністрування та глибокого аналізу. Платформи «На Урок» та Google Classroom добре адаптовані під освітні потреби, але мають обмежений функціонал управління тестами та ролями. Таким чином, виникає необхідність у створенні нового Web-ресурсу, який поєднає простоту використання, гнучке управління ролями користувачів, адаптацію до українських навчальних закладів та зручні інструменти для створення, проведення й аналізу результатів тестувань. Окрім того, нова система має бути побудована із застосуванням сучасних фреймворків, які забезпечують зручність розробки, підтримку адаптивного інтерфейсу та легкість у розширенні функціоналу. Це дозволить створити більш ефективний та гнучкий інструмент для потреб освітнього процесу.

1.2 Аналіз вимог до функціоналу Web-ресурсу для створення тестів

Для успішної реалізації інформаційної системи необхідно чітко сформулювати її мету, визначити основні завдання та описати вимоги до функціоналу. Це дозволяє сформувати загальне бачення структури системи, її можливостей, а також критеріїв, за якими можна оцінити досягнення мети. Тому попередньо необхідно чітко сформульювати цілі роботи, перелік завдань, які необхідно реалізувати, та ключові вимоги до майбутнього Web-ресурсу.

Метою роботи є розробка Web-ресурсу для створення тестів та перевірки знань, що дозволяє автоматизувати процес проведення тестування, аналізу та оцінювання результатів учнів, а також спростити роботу викладачів і адміністраторів системи. Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Обґрунтувати вибір інструментів і технологій для реалізації Web-ресурсу.

2. Розробити архітектуру застосунку, включаючи структуру бази даних і логіку взаємодії між компонентами системи.

3. Реалізувати клієнтську та серверну частини системи з урахуванням розмежування ролей користувачів (учень, вчитель, адміністратор).

4. Забезпечити безпечну авторизацію та збереження даних, включаючи захист паролів.

5. Протестувати систему, перевірити її функціональність, зручність та стабільність роботи. Зокрема, може бути доцільно провести тестування системи з урахуванням сценаріїв несанкціонованого доступу та перевірити надійність обробки помилкових запитів.

Розроблювана система має підтримувати три основні ролі користувачів: учень, вчитель та адміністратор. Для кожної з ролей передбачено окремий набір функціональних можливостей. З урахуванням цільової аудиторії та функціонального призначення системи, було визначено три основні ролі користувачів, кожна з яких має власний набір можливостей:

- Учень (за замовчуванням надається усім новим користувачам):
    - Реєстрація та авторизація в системі.
    - Перегляд доступних тестів за класами та предметами.
    - Проходження тестів (одноразово або з можливістю повтору за дозволом вчителя).
    - Перегляд власних результатів тестування.
- Вчитель:
    - Можливість створення тестів (назва, клас, предмет, тривалість, питання та варіанти відповідей).
    - Редагування тестів до моменту їх публікації.
    - Перегляд результатів учнів, які проходили власні тести.
    - Надання можливості повторного проходження тесту обраним учням.
- Адміністратор:

- Перегляд і керування усіма зареєстрованими користувачами.
- Редагування даних користувачів (ім’я, прізвище, email, пароль, роль).
- Призначення ролі «вчитель» за запитом учня.
- Отримання повного доступу до системи та виконання будь-яких адміністративних дій.

Окрім функціональних можливостей, система повинна відповідати ряду нефункціональних вимог, які забезпечать її стабільну та безпечну роботу, зручність для користувачів і можливість масштабування в майбутньому.

- Безпека:
  - Паролі користувачів зберігаються у зашифрованому вигляді.
  - Авторизація та перевірка прав доступу реалізовані з урахуванням розмежування ролей.
  - Неможливість проходження тесту без авторизації.
- Зручність інтерфейсу:
  - Інтерфейс має бути інтуїтивно зрозумілим як для учнів, так і для вчителів.
  - Необхідно передбачити адаптивну верстку для коректного відображення на різних пристроях (комп’ютери, планшети, смартфони).
- Масштабованість:
  - Архітектура системи повинна передбачати можливість розширення функціоналу без повного переписування коду.
  - Можливість додавання нових ролей або модулів у майбутньому.
- Продуктивність:
  - Швидке завантаження сторінок та обробка запитів користувачів.
  - Ефективна робота з базою даних навіть при великій кількості тестів та користувачів.

Таким чином, на цьому етапі визначено перелік завдань, які необхідно реалізувати, а також детально описано функціональні та нефункціональні

вимоги до системи. Чітке розуміння поставлених задач та очікуваного функціоналу є необхідною передумовою для ефективного проєктування інформаційної системи, яка відповідатиме сучасним вимогам до якості, зручності та безпеки Web-застосунків в освітній сфері. Усі визначені вимоги будуть враховані при проєктуванні архітектури, реалізації структури бази даних та побудові користувацького інтерфейсу.

1.3 Огляд технологій для Web-розробки та вибір технологічного стеку

Вибір технологій для розробки Web-застосунку має важливе значення для ефективності реалізації проєкту, подальшої підтримки, масштабування та зручності використання кінцевого продукту. Правильно підібраний технологічний стек дозволяє забезпечити оптимальну швидкодію, зручність розробки та простоту в обслуговуванні Web-ресурсу. Тому доцільно виконати огляд популярних технологій frontend та backend-розробки, а також систем управління базами даних і засобів для роботи з ними, після чого можна буде зробити обґрунтований вибір конкретних технологій для реалізації Web-ресурсу створення тестів та перевірки знань. При цьому, особливу увагу необхідно приділити адаптивності інтерфейсу, швидкості взаємодії з сервером та ефективному управлінню даними [5-14].

Backend-частина Web-застосунку відповідає за обробку даних, бізнес-логіку, взаємодію з базою даних та обслуговування запитів від frontend-інтерфейсу. Найбільш поширеними технологіями для реалізації серверної частини є Node.js (Express.js), Python (Django, Flask) та Java (Spring). Розглянемо їх докладніше.

Node.js (Express.js) - платформа для виконання JavaScript на сервері, що дозволяє створювати швидкі та масштабовані серверні застосунки. Express.js - це мінімалістичний фреймворк для Node.js, що забезпечує простоту в створенні REST API та обробці HTTP-запитів. Node.js має низький поріг входження,

високу швидкодію, велику кількість додаткових модулів та активно розвинену спільноту [11].

Python (Django, Flask) - популярна мова програмування, яка широко використовується для Web-розробки завдяки простому синтаксису та високій читабельності. Django - повноцінний фреймворк, що забезпечує готові рішення для роботи з БД, авторизацією, формами та маршрутизацією. Flask є більш легким фреймворком, що дозволяє створювати прості REST API без зайвих компонентів, однак вимагає більше ручного налаштування [12, 13].

Java (Spring) - потужний та стабільний фреймворк, який широко застосовується в ентерпрайз-розробці. Spring надає готові механізми для роботи з базами даних, транзакціями, безпекою, а також глибокий контроль над конфігурацією застосунку. Однак Spring є складним для початківців, вимагає ґрунтовних знань Java та значних витрат часу на навчання та налаштування проєкту.

На основі аналізу, для реалізації даного проєкту було обрано Express.js як головний серверний фреймворк, який працює поверх Node.js. Цей вибір обґрунтовано його мінімалістичним підходом, високою продуктивністю, широкою підтримкою REST API та сумісністю з сучасними ORM-інструментами. Express.js забезпечує ефективну обробку HTTP-запитів, чітке розмежування маршрутів і логіки, а також легко інтегрується з іншими бібліотеками, що є важливо для побудови масштабованого бекенду.

Збереження, обробка та доступ до структурованих даних у Web-застосунках здійснюється за допомогою систем керування базами даних (СКБД). У сучасній розробці найпоширенішими реляційними СКБД є MySQL та PostgreSQL. Для зручної взаємодії із базою даних зазвичай використовуються інструменти типу ORM (Object-Relational Mapping), серед яких популярними є Prisma та Sequelize [15, 16].

MySQL - одна з найпопулярніших СКБД у світі з відкритим кодом. Вона забезпечує високу продуктивність, підтримує транзакції, реплікацію, зручна в

налаштуванні та використанні. MySQL активно підтримується спільнотою, має велику кількість документації та інтегрується з більшістю мов програмування.

PostgreSQL - потужна об’єктно-реляційна СКБД, що відзначається високим рівнем відповідності стандартам SQL та широкими можливостями у роботі з даними. Вона має розширений функціонал (наприклад, підтримку JSONB, тригерів, збережених процедур), але може бути складнішою в адмініструванні для недосвідчених користувачів.

У контексті даного проєкту PostgreSQL було обрано також через його розширені можливості обробки складних типів даних, зокрема підтримку JSONB, повнотекстовий пошук і гнучке управління транзакціями. На відміну від MySQL, PostgreSQL краще масштабується для великих застосунків і надає більше можливостей для адаптації структури БД під специфіку освітніх процесів.

Prisma - це сучасний ORM-інструмент для Node.js, який дозволяє працювати з базами даних на високому рівні абстракції. Prisma має власний синтаксис для опису схеми даних, автоматично генерує типи та запити, підтримує міграції та має чудову інтеграцію з TypeScript. Основна перевага Prisma - простота, надійність та висока продуктивність [17].

Sequelize - ORM-бібліотека для Node.js, яка також підтримує роботу з MySQL, PostgreSQL, SQLite та MSSQL. Sequelize є потужним інструментом, проте має складніший синтаксис, менш гнучку типізацію та менш структуровану систему міграцій порівняно з Prisma [18].

У порівнянні з Sequelize, Prisma надає більш сучасний підхід до роботи з даними. Зокрема, Prisma забезпечує типобезпечність, автоматичну генерацію запитів, зручну систему міграцій та глибоку інтеграцію з TypeScript, що значно знижує ймовірність помилок на етапі розробки. Крім того, Prisma має простіший і логічніший синтаксис, що робить її зручною для використання навіть початківцями.

Зважаючи на мету роботи, для реалізації було обрано PostgreSQL у поєднанні з ORM Prisma, оскільки таке поєднання забезпечує високу

відповідність стандартам SQL, потужну підтримку складних запитів (у тому числі JSON), а також масштабованість у випадку зростання кількості користувачів. Prisma дозволяє ефективно моделювати структуру бази даних, автоматизувати створення таблиць і спростити взаємодію між сервером та СКБД.

Після аналізу сучасних технологій frontend та backend-розробки, систем керування базами даних і засобів взаємодії з ними можна обґрунтовано зробити вибір технологічного стеку для реалізації Web-застосунку. Зокрема, для клієнтської частини обрано React - популярну JavaScript-бібліотеку, яка дозволяє створювати інтерфейси на основі компонентів. У поєднанні з React Router для реалізації маршрутизації, Axios для виконання HTTP-запитів та Tailwind CSS для швидкої стилізації, React забезпечує гнучкість, адаптивність та високу продуктивність користувацького інтерфейсу. Серверна частина буде реалізована з використанням Express.js, а як СКБД застосовано PostgreSQL у поєднанні з Prisma ORM. Обраний технологічний стек дозволяє реалізувати Web-застосунок, який є масштабованим, зручним для підтримки, а також адаптованим до сучасних потреб освітніх установ.

Висновки до розділу 1

У першому розділі було здійснено детальний аналіз предметної області, що дозволив чітко окреслити контекст задачі, потреби цільової аудиторії та специфіку функціонування системи онлайн-тестування. На основі цього аналізу були визначені основні вимоги до функціональності, інтерфейсу, продуктивності та безпеки розроблюваного Web-додатку.

Проведений огляд існуючих Web-рішень для створення та проведення тестувань дав змогу зіставити різні підходи до реалізації подібних систем, виявити їхні сильні сторони (наприклад, зручність інтерфейсу або широку функціональність) і слабкі місця (такі як обмежена адаптивність, складність масштабування або відсутність модулів аналітики). Ці спостереження стали

підґрунтям для формування цілей дослідження та чітко сформульованих завдань дипломної роботи.

Обґрунтований вибір технологічного стеку, який найкраще відповідає поставленим вимогам до системи є запорукою створення якісного Web-додатку. Зокрема, для реалізації фронтенд-частини обрано бібліотеку React, що забезпечує швидку розробку інтерфейсу користувача з можливістю повторного використання компонентів. При цьому, серверна частина буде реалізована за допомогою Express.js - легкого та гнучкого фреймворку для Node.js, що дозволяє швидко організувати REST API. У якості СУБД обрано PostgreSQL, яка забезпечує високу надійність, розширюваність та відповідність вимогам до зберігання структурованих даних. Для зручної роботи з базою даних використовується Prisma ORM, яка забезпечує взаємодію з моделями даних, спрощує міграції та підвищує ефективність розробки. Загалом, такий технологічний стек дозволить створити якісний Web-додаток відповідно до встановлених вимог.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ WEB-РЕСУРСУ

2.1 Загальна архітектура Web-ресурсу

Для розробки Web-ресурсу, що забезпечує створення та проходження тестів, обрано клієнт-серверну архітектуру, яка є загальноприйнятим підходом у сучасному Web-програмуванні. Такий підхід передбачає чітке розділення обов'язків між клієнтською та серверною частинами системи: клієнт відповідає за відображення інтерфейсу користувача та ініціювання запитів, тоді як сервер виконує бізнес-логіку, обробляє запити й взаємодіє з базою даних.

Система складається з трьох основних компонентів: клієнтської частини (Frontend), серверної частини (Backend) та системи керування базами даних (СКБД). Клієнтська частина буде реалізована з використанням бібліотеки React, що дозволить створювати компонентну структуру інтерфейсу та ефективно керувати станом програми. При цьому Tailwind CSS буде використаний для швидкого створення адаптивного дизайну, а Axios - для виконання HTTP-запитів до серверної частини. Навігація між сторінками буде реалізована за допомогою React Router [19-21].

Серверна частина побудована з використанням Express.js - легкого та швидкого фреймворка для створення REST API, який працює поверх Node.js. Він обробляє запити від клієнта, виконує авторизацію, обробляє дані, що надходять, і відповідає за взаємодію з базою даних через ORM Prisma. У якості СКБД буде використана PostgreSQL - потужна реляційна система, що зберігає інформацію про користувачів, тести, результати тощо.

Для взаємодії між клієнтом та сервером буде використаний REST API (Representational State Transfer), який базується на стандартному HTTP-протоколі та передбачає використання основних методів (GET, POST, PUT, DELETE) для виконання операцій з даними. Такий підхід дозволяє чітко розмежувати відповідальність, забезпечує простоту у розширенні функціоналу та дає

можливість легко інтегрувати інші клієнтські додатки (наприклад, мобільні додатки) у майбутньому.

Технологічна взаємодія компонентів виглядає наступним чином: користувач через Web-інтерфейс (React) ініціює HTTP-запити, які формуються за допомогою бібліотеки Axios. Ці запити відправляються до серверної частини (Express.js), де відбувається їх обробка та перевірка прав доступу (авторизація за допомогою JWT-токенів). При цьому, для реалізації безпечного механізму авторизації застосовується технологія JSON Web Token (JWT), яка дозволяє зберігати інформацію про сесію користувача на стороні клієнта. Після успішного входу в систему сервер генерує токен, який містить у зашифрованому вигляді унікальний ідентифікатор користувача, його роль та термін дії. Цей токен передається клієнту й зберігається у локальному сховищі браузера, після чого автоматично прикріплюється до кожного подальшого запиту. На стороні сервера реалізовано middleware, яке перевіряє наявність та дійсність токена й надає або забороняє доступ до захищених ресурсів залежно від ролі користувача [23-27].

Для захисту облікових даних користувачів у системі використовується алгоритм хешування bcrypt. Перед збереженням у базу даних пароль кожного користувача перетворюється на хеш з унікальною "сіллю", що унеможливлює зворотне відновлення пароля навіть у разі компрометації бази даних. Під час авторизації введений користувачем пароль порівнюється з хешованим значенням, що зберігається, шляхом використання внутрішніх методів bcrypt. Такий підхід значно підвищує загальний рівень безпеки системи та відповідає сучасним рекомендаціям з кіберзахисту.

Якщо потрібна взаємодія з базою даних, сервер використовує ORM Prisma для створення відповідних запитів до PostgreSQL. База даних повертає серверу результати запиту, після чого сервер формує відповідь у вигляді JSON і повертає її клієнту. Клієнтська частина, отримавши відповідь, оновлює стан інтерфейсу відповідно до отриманих даних.

Загальну схему архітектури Web-застосунку можна представити наступною послідовністю взаємодії:

Користувач → Frontend (React, Axios) → HTTP-запит → Backend (Express.js) → Prisma ORM → PostgreSQL → Prisma ORM → Backend → JSON-відповідь → Frontend (Axios) → оновлення інтерфейсу → Користувач.

Така архітектура має низку суттєвих переваг, серед яких гнучкість, зручність розробки та підтримки, можливість незалежного масштабування окремих компонентів, а також чітке розмежування логіки клієнта та сервера, що спрощує подальшу розробку та обслуговування системи.

Таким чином отримана загальна архітектура Web-застосунку, що базується на клієнт-серверному підході з використанням сучасних фреймворків і бібліотек (React, Express.js), REST API та реляційної бази даних PostgreSQL. Вибір саме такої архітектури зумовлений її високою продуктивністю, гнучкістю та можливістю ефективної інтеграції додаткових компонентів у майбутньому, що є важливим фактором для успішної реалізації і підтримки розроблюваного проєкту. REST API дає змогу чітко структурувати маршрути HTTP-запитів, забезпечити уніфіковану передачу даних у форматі JSON та легко масштабувати функціонал відповідно до потреб системи.

2.2 Моделювання логіки роботи інформаційної системи

Для формалізації логіки роботи інформаційної системи використовуються засоби об'єктно-орієнтованого моделювання (UML-діаграми). Таке моделювання дозволяє чітко визначити функціональність, об'єкти системи та послідовність взаємодій між ними. Для цього проєкту було використано нотацію UML (Unified Modeling Language), яка є загальноприйнятим стандартом у сфері програмної інженерії. UML забезпечує універсальність, гнучкість та візуальну зрозумілість моделей, що дає змогу відобразити як статичні структури (класи, сутності), так і динамічні процеси (логіку виконання, сценарії взаємодії). На відміну від бізнес-орієнтованих нотацій (наприклад, BPMN), UML краще підходить для опису архітектури та програмної логіки інформаційних систем [29, 30].

В межах даного проєкту для опису основних сценаріїв використання та структури системи було побудовано діаграми варіантів використання (Use Case), класів (Class) і діяльності (Activity), які відображають ключові процеси, що реалізуються у Web-застосунку.

Використання UML-діаграм на етапі проєктування дозволяє:

- наочно представити структуру та поведінку системи до її реалізації;
- виявити потенційні помилки логіки або неузгодженості ще до написання коду;
- забезпечити єдине розуміння архітектури між усіма учасниками розробки;
- спростити підтримку й розвиток системи, оскільки діаграми слугують документацією;
- створити основу для генерації частин коду або автоматизації частини процесів (наприклад, при використанні UML-сумісних інструментів).

Взаємодію основних користувачів із системою наведено на діаграмі варіантів використання (рисунок 2.1).

Для опису взаємодії користувачів із системою було побудовано діаграму варіантів використання, яка відображає основні функціональні можливості, доступні для кожної з ролей: учня, вчителя та адміністратора. Така діаграма дозволяє наочно продемонструвати межі системи, доступні сценарії використання та призначення кожної ролі у рамках функціоналу Web-застосунку.

На рис. 2.1 представлено основні сценарії використання. Учень має можливість переглядати доступні тести, проходити їх, а також ознайомлюватися з результатами вже пройдених тестів. Вчитель, окрім базового функціоналу, може створювати тести, редагувати їх, переглядати результати учнів та надавати дозвіл на повторне проходження тесту. Адміністратор системи має найширші повноваження - він може переглядати інформацію про всіх зареєстрованих користувачів, редагувати їхні дані та змінювати ролі.

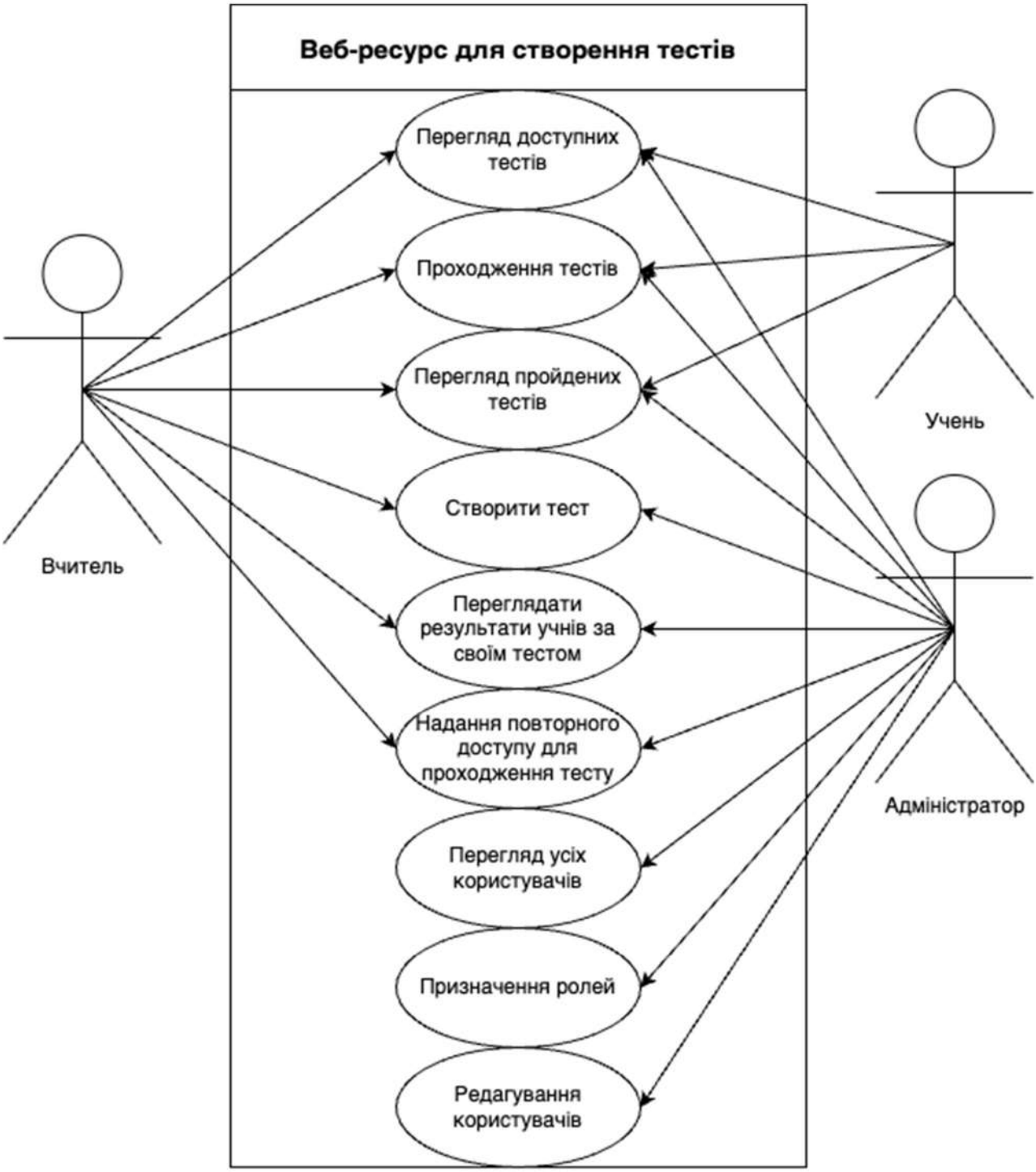

Рис. 2.1. Діаграма варіантів використання (Use Case Diagram)

Діаграма класів (Class Diagram) демонструє внутрішню структуру системи з точки зору об'єктно-орієнтованого підходу. Вона описує класи, їх атрибути, методи, а також зв’язки між класами (асоціації, агрегації, композиції, наслідування). Така діаграма є основою для реалізації програмної моделі

системи та дозволяє формалізувати сутності, визначені на етапі проєктування бази даних, у вигляді об’єктів.

У межах цього проєкту діаграма класів відображає основні сутності: User, Role, Test, Question, Answer, Result та TestAccess. Вона наочно показує їхні атрибути та типи зв’язків між ними (один-до-одного, один-до-багатьох, багато-до-багатьох), що відображає логіку зберігання, доступу та обробки даних у системі. Ця діаграма слугує підґрунтям як для побудови структури бази даних, так і для реалізації бізнес-логіки Web-застосунку.

На рис. 2.2 представлено діаграму класів Web-ресурсу для створення тестів і перевірки знань.

Для наочного представлення логіки проходження тесту користувачем було створено діаграму діяльності, яка демонструє послідовність ключових етапів процесу тестування. Діаграма охоплює всі основні дії, які виконує користувач під час взаємодії з системою: починаючи з авторизації, продовжуючи вибором відповідного тесту, проходженням окремих запитань, обробкою наданих відповідей, і завершуючи тестуванням - як у разі ручного завершення користувачем, так і у випадку автоматичного завершення за таймером.

Діаграма діяльності (Activity Diagram), як елемент мови моделювання UML, відображає динамічний аспект функціонування системи - послідовність і взаємозв'язок дій, які виконує користувач або система автоматично. Вона дає змогу візуалізувати керування системою, забезпечуючи наочне представлення логіки виконання операцій у межах різних процесів. Ця діаграма є вдосконаленим аналогом традиційної блок-схеми, але надає ширші можливості моделювання. Вона дає змогу описувати не лише послідовні дії, а й розгалуження з різноманітними умовами, паралельне виконання процесів, цикли (петлі), події, точки синхронізації та варіанти завершення сценаріїв. Важливою особливістю діаграми діяльності є її здатність описувати дії, що виконуються як користувачем, так і зовнішніми системами, а також внутрішні операції, що відбуваються автоматично.

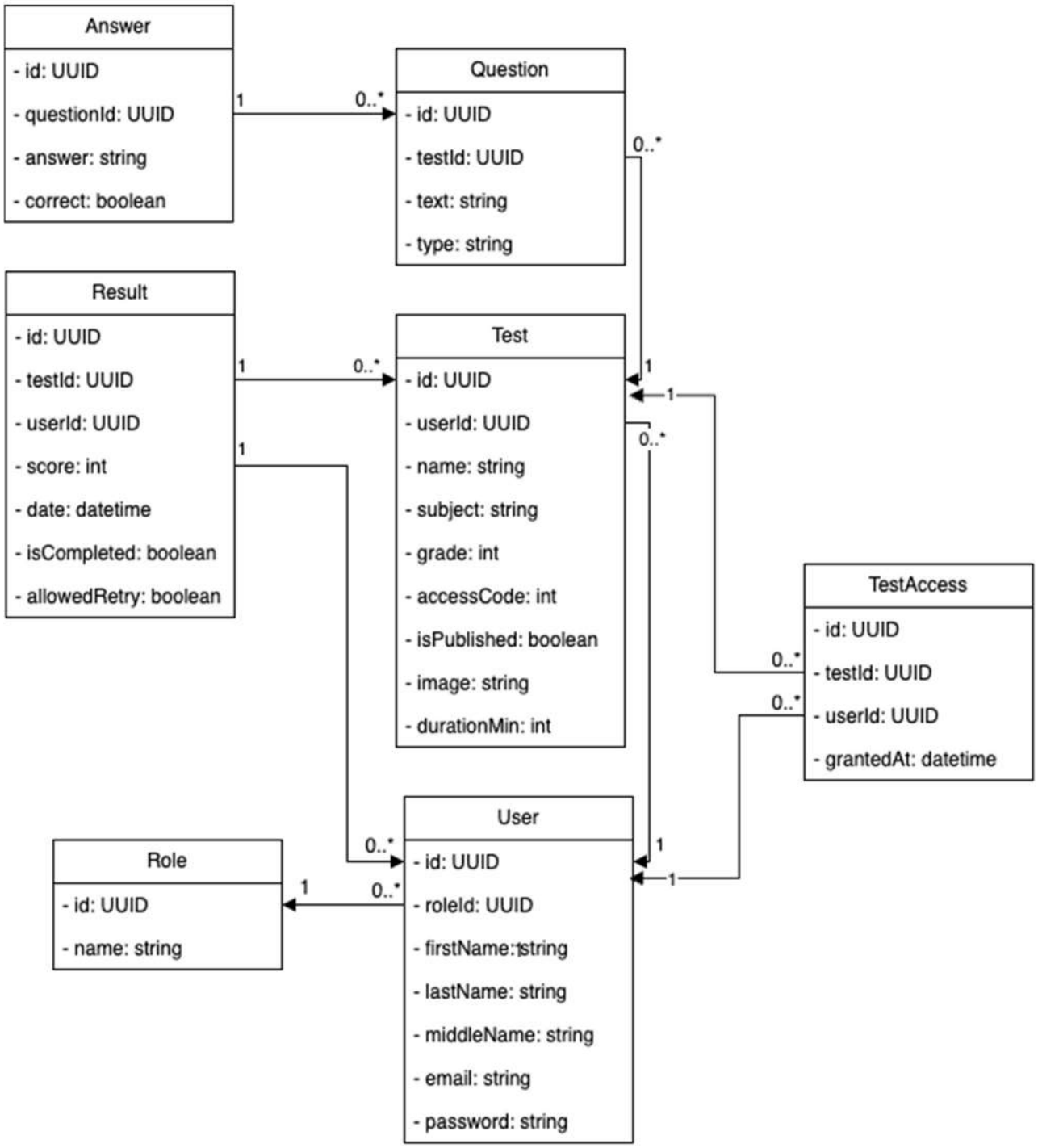

Рис. 2.2. Діаграма класів (Class Diagram)

На рис. 2.3 представлено розроблену діаграму діяльності, яка ілюструє послідовність дій користувача під час проходження тесту. Діаграма демонструє основні етапи процесу, включаючи вибір тесту, надання відповідей на запитання, а також завершення тесту з подальшим збереженням результату в базу даних.

Рис. 2.3. Діаграма активності процесу проходження тесту користувачем

2.3 Проєктування бази даних Web-ресурсу

Проєктування бази даних є важливим етапом у процесі розробки Web-ресурсу, оскільки саме на цьому етапі закладається фундамент для ефективного зберігання, обробки та доступу до інформації. Зважаючи на специфіку розроблюваного Web-ресурсу, для реалізації було обрано реляційну базу даних PostgreSQL, яка відповідає сучасним стандартам, забезпечує високу надійність, продуктивність та гнучкість в роботі з даними. Підхід на основі реляційної моделі дозволяє чітко структурувати інформацію у вигляді взаємопов'язаних таблиць, що значно спрощує управління та аналіз даних.

Реляційна модель була обрана через низку переваг, важливих для даного проєкту. Зокрема, вона дозволяє легко реалізовувати складні взаємозв’язки між сутностями, забезпечує цілісність та узгодженість даних, підтримує транзакції та складні запити з вибірками й агрегацією інформації. Це важливо для реалізації функціоналу, пов'язаного з тестуванням знань, де необхідно зберігати і оперативно отримувати дані про користувачів, створені тести, запитання, відповіді та результати.

Для того, щоб Web-ресурс ефективно виконував усі свої функції, необхідно чітко визначити основні сутності (entity), які відображатимуться у структурі бази даних. На основі сформульованих у попередньому пірозділі моделей системи було визначено такі ключові сутності:

- Користувач – містить інформацію про зареєстрованих учасників системи (учні, вчителі, адміністратори), включаючи особисті дані, контактну інформацію, авторизаційні дані (логін, зашифрований пароль) та роль користувача.
- Роль – визначає права доступу користувача до різних функціональних можливостей системи (учень, вчитель, адміністратор).
- Тест – містить дані про створені викладачем тестові завдання: назву, опис, предмет, клас, дату створення, тривалість проходження, унікальний код доступу та прив’язку до автора.

- Запитання – інформація про конкретні питання тестів, включаючи текст питання, тип (одиночний вибір, множинний вибір), можливі відповіді і правильні варіанти.
- Відповідь – варіанти відповідей, які користувач може обирати при проходженні тестів.
- Результат – інформація про результати проходження тесту конкретним учнем, включаючи кількість правильних відповідей, дату і час проходження, та зв’язок з користувачем і тестом.

Між визначеними сутностями реалізовано декілька типів зв’язків: один-до-багатьох, багато-до-багатьох та один-до-одного. Наприклад, один користувач (User) може мати багато результатів тестування (Result), але кожен результат належить лише одному користувачу. Аналогічно, один тест (Test) містить багато питань (Question), тоді як кожне питання належить конкретному тесту. Для реалізації доступу до тестів кількома користувачами застосовується зв’язок багато-до-багатьох між User та Test, який реалізовано через допоміжну таблицю TestAccess. Таке структурування дозволяє ефективно організувати зберігання даних і забезпечити логічну цілісність бази.

Для кожного атрибута сутності було обрано відповідний тип даних з урахуванням його призначення, обсягу інформації та можливості подальшої обробки. Наприклад:

- id (у всіх таблицях): UUID – унікальний ідентифікатор;
- email (таблиця User): VARCHAR(255) – рядок із максимальною довжиною 255 символів;
- passwordHash: TEXT – для зберігання хешованого пароля;
- title (у Test): VARCHAR(255) – назва тесту;
- durationMin: INTEGER – тривалість тесту у хвилинах;
- createdAt: TIMESTAMP – дата й час створення запису;
- isCorrect (у Answer): BOOLEAN – логічне значення правильності відповіді.

Така типізація забезпечує правильну інтерпретацію даних, оптимальне використання ресурсів СКБД та полегшує подальшу валідацію й перевірку інформації.

У структурі бази даних реалізовано первинні та зовнішні ключі, що забезпечують цілісність зв’язків між сутностями. Первинний ключ (PRIMARY KEY) є унікальним ідентифікатором кожного запису в таблиці - зазвичай це поле id з типом UUID. Зовнішні ключі (FOREIGN KEY) використовуються для встановлення зв’язків між таблицями. Наприклад, поле userId у таблиці Result є зовнішнім ключем, що посилається на поле id таблиці User. Аналогічно, testId у таблицях Question та Result пов’язує ці записи з конкретним тестом.

Для підвищення продуктивності запитів до бази даних створено індекси на найбільш часто використовуваних полях. Зокрема, індекси створено для полів email у таблиці User та testId у таблиці Question, що значно пришвидшує пошук і фільтрацію даних.

Перераховані сутності забезпечують повне охоплення функціональних вимог, чітку структуру інформації, що зберігатиметься, і зручну взаємодію між компонентами системи. Структуру бази даних та взаємозв’язки між її основними сутностями представлено на ER-діаграмі (рисунок 2.4). Діаграма демонструє типи зв’язків між таблицями, а також ключові атрибути, які забезпечують реалізацію функціональних вимог системи.

Користувач (User) - одна з ключових сутностей системи, яка містить інформацію про всіх зареєстрованих учасників: учнів, вчителів та адміністраторів. Таблиця містить такі атрибути, як ім’я, прізвище, по батькові, електронна адреса (унікальна), пароль (у зашифрованому вигляді), а також зовнішній ключ role_id, що визначає роль користувача в системі. Первинним ключем таблиці є унікальний ідентифікатор id типу UUID.

Роль (Role) - допоміжна сутність, яка містить перелік можливих ролей у системі: «учень», «вчитель» та «адміністратор». Кожен користувач має лише одну роль, зв’язок між таблицями реалізовано через зовнішній ключ role_id у таблиці User.

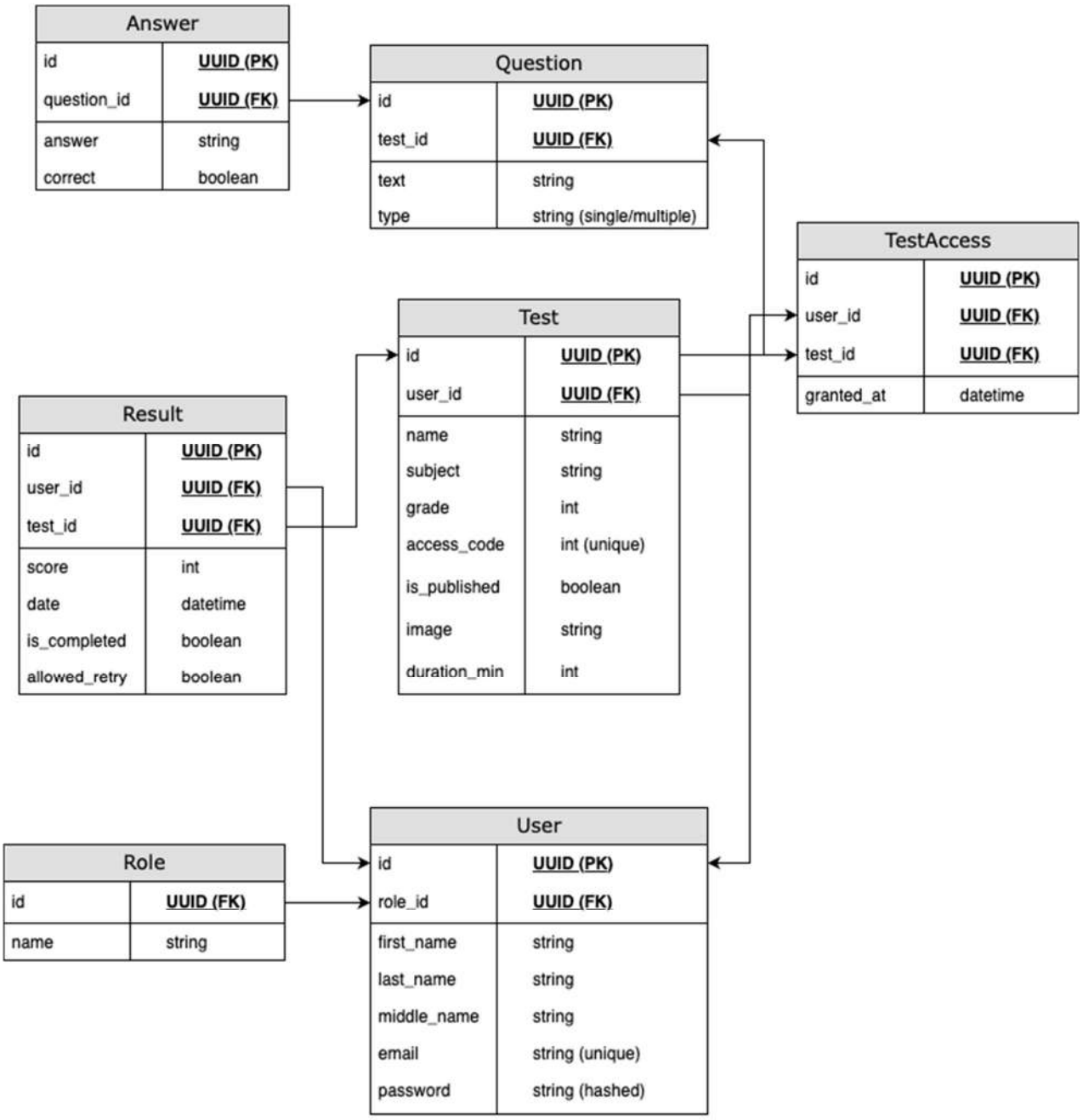

Рис. 2.4. ER-діаграма бази даних Web-застосунку

Тест (Test) - сутність, що зберігає основну інформацію про створені вчителями тести. Кожен тест має назву, предмет, клас (grade), унікальний код доступу (access_code), посилання на зображення (image), статус публікації (is_published) і максимальну тривалість проходження в хвилинах (duration_min). Кожен тест прив’язується до вчителя-автора за допомогою зовнішнього ключа user_id, що посилається на таблицю User. Первинний ключ - id (UUID).

Питання (Question) - таблиця, що зберігає перелік питань для кожного тесту. Кожне питання містить текстове формулювання (text) та тип (type), що визначає формат відповіді - одиночний або множинний вибір. Зовнішній ключ test_id вказує на тест, до якого належить питання. Один тест може містити багато питань.

Відповідь (Answer) - допоміжна сутність, яка зберігає варіанти відповідей до кожного запитання. Поле answer містить текст відповіді, а поле correct визначає, чи є відповідь правильною. Зв’язок із таблицею Question реалізується через зовнішній ключ question_id. Таким чином, одне питання може мати декілька відповідей.

Результат (Result) - сутність, що фіксує результати проходження тестів користувачами. Містить інформацію про кількість набраних балів (score), дату проходження (date), статус завершення (is_completed) та можливість повторного проходження (allowed_retry). Пов’язується з таблицями User і Test через зовнішні ключі user_id та test_id.

Доступ до тесту (TestAccess) - таблиця, яка реалізує механізм відкриття доступу до тестів для конкретних користувачів. Вона містить зовнішні ключі user_id та test_id, а також дату й час надання доступу (granted_at). Ця сутність дозволяє зберігати інформацію про те, хто і коли отримав доступ до проходження певного тесту, що особливо актуально у випадках введення коду тесту вручну.

Таким чином спроєктовано структуру бази даних для Web-ресурсу створення та проходження тестів. Зокрема, визначено основні сутності, їх атрибути та типи даних, а також побудовано ER-діаграму, яка наочно демонструє взаємозв’язки між таблицями. У структурі бази даних реалізовано первинні та зовнішні ключі, що забезпечують логічну цілісність і структурованість збережених даних. При цьому, індексація полів, які найчастіше використовуються у запитах (наприклад, email, testId), сприяє підвищенню продуктивності при масштабуванні. Розроблена модель даних повністю відповідає функціональним вимогам системи, забезпечує ефективне зберігання

та обробку інформації, а також закладає надійну основу для реалізації логіки взаємодії між компонентами системи на наступних етапах розробки.

Висновки до розділу 2

Розроблена архітектура інформаційної системи передбачає чітке розмежування на клієнтську та серверну частини з урахуванням ролей користувачів. При цьому створена інформаційна модель системи (у вигляді діаграми сутностей і зв’язків, яка охоплює всі основні об’єкти предметної області, їх атрибути та взаємозв’язки) представляє собою якісне підґрунтя для подальшої розробки. Зокрема, формалізація основних сценаріїв взаємодії користувачів із системою за допомогою діаграми варіантів використання та створена діаграма класів, дозволяють досить повно візуалізувати структуру даних і логіку їх взаємозв’язку. При цьому побудована діаграма активності описує динаміку процесу проходження тесту з урахуванням таймера, перевірки відповідей і збереження результатів.

Результати проєктування послугують якісною основою для подальшої реалізації інформаційної системи, забезпечивши чітке розуміння її логіки, структури та функціональності.

РОЗДІЛ 3
РОЗРОБКА ОКРЕМИХ МОДУЛІВ WEB-РЕСУРСУ ДЛЯ СТВОРЕННЯ ТЕСТІВ

3.1 Розробка серверної частини Web-ресурсу

Серверна частина Web-застосунку реалізована з використанням середовища виконання Node.js та фреймворку Express.js, що забезпечує обробку HTTP-запитів, маршрутизацію, авторизацію, перевірку прав доступу, а також взаємодію з базою даних. Уся бізнес-логіка згрупована в маршрутах (routes) і окремих модулях, що відповідає принципам структурованого REST-підходу.

Взаємодія із реляційною базою даних PostgreSQL реалізована за допомогою ORM Prisma, що дозволяє працювати з моделями на високому рівні абстракції. Основні сутності, що використовуються у серверній частині: User, Test, Question, Answer, TestResult. Усі зв’язки між ними описані в окремому файлі schema.prisma. Наприклад, користувач має роль (UserRole), а також пов’язаний із результатами проходження (TestResult[]) і тестами, які він створив (Test[]).

Авторизація реалізована через JWT (JSON Web Token). Після входу в систему сервер генерує токен, що містить ID користувача та його роль. Токен зберігається на клієнті та передається у кожному запиті до захищених маршрутів. На сервері використовується middleware для перевірки дійсності токена, що дозволяє обмежити доступ до певного функціоналу (наприклад, редагування ролей користувачів або створення тестів).

Для захисту паролів використовується бібліотека bcrypt. Перед збереженням у базу пароль хешується з використанням сільового значення, а при вході перевіряється відповідність за допомогою методу compare.

Сервер містить окремі маршрути для:

– реєстрації та входу в систему (з перевіркою унікальності email та шифруванням пароля);

– керування тестами (створення, отримання за ID або кодом, фільтрація за класом);

– керування питаннями та відповідями до кожного тесту;

– обробки результатів проходження тестів та їх збереження в таблицю TestResult;

– роботи з ролями та профілями користувачів (доступно лише адміністратору).

Усі запити йдуть у форматі JSON через HTTP, структура відповідає загальноприйнятим стандартам REST API. У разі помилок сервер повертає відповідний статус (наприклад, 401 - неавторизовано, 403 - заборонено, 404 - не знайдено). Уся серверна логіка поділена на окремі файли й організована за принципом розділення відповідальностей, що забезпечує масштабованість та зручність підтримки проєкту.

Прикладом ключового запиту, який обробляється на сервері, є збереження результату тесту після його проходження учнем. Коли користувач завершує тест, на клієнтській стороні формується об’єкт результату з такими даними:

– ідентифікатор тесту (testId),

– кількість правильних відповідей (score),

– кількість загальних питань (total),

– дата проходження,

– список ID обраних відповідей (або заповнені поля, залежно від реалізації).

Ці дані надсилаються на сервер POST-запитом до маршруту /api/results, де попередньо проходять перевірку через middleware JWT (чи авторизований користувач). Після перевірки токена та ролі система створює новий запис у таблиці TestResult, зв’язуючи його з користувачем та відповідним тестом. Завдяки ORM Prisma цей процес реалізовано у вигляді однієї транзакції.

Такий підхід гарантує узгодженість даних, автоматичну фіксацію дати та унеможливлює дублювання результатів для одного й того ж тесту. У разі

повторного надсилання система повідомляє про помилку, що забезпечує додаткову стабільність бізнес-логіки.

Збереження, обробка та доступ до структурованих даних у системі реалізовано з використанням реляційної бази даних PostgreSQL. Робота з БД здійснюється через ORM Prisma, яка забезпечує автоматичну генерацію SQL-запитів, міграції та типобезпеку на рівні серверного коду [31].

Структуру бази даних описано в окремому файлі schema.prisma, де задано сутності, типи полів, зв’язки між таблицями та інші обмеження. Після створення або зміни схеми система формує відповідні міграції, що дозволяє підтримувати актуальний стан бази на різних етапах розробки.

Ключові сутності бази даних:

– User - містить особисті дані користувача, електронну пошту, хеш пароля та роль;

– Test - тест, створений вчителем, з назвою, предметом, кодом доступу, кольором, тривалістю;

– Question - окреме питання до тесту;

– Answer - варіанти відповідей з позначенням правильності;

– TestResult - результат проходження тесту користувачем (бал, дата, зв’язок з тестом і користувачем).

Для підвищення продуктивності та уникнення дублювання реалізовано індексацію на полях email (у таблиці User) та accessCode (у Test). Ідентифікатори зберігаються у форматі UUID, що гарантує унікальність навіть при масштабуванні.

Усі запити до бази даних (на збереження, читання, оновлення чи видалення) реалізовані з використанням Prisma-методів (findMany, create, update, delete, include тощо). Приклад:

– отримання всіх тестів для певного класу (where: { grade: 5 });

– створення нового тесту з вкладеними питаннями та відповідями (create({ data: { ... } }));

– збереження результату (create TestResult з автоматичною фіксацією дати).

У разі критичних помилок або некоректного доступу сервер обробляє виключення й повертає відповідний статус клієнту, що дозволяє зберігати стабільність роботи системи навіть за наявності великої кількості одночасних запитів.

3.2 Реалізація клієнтської частини Web-ресурсу

Клієнтська частина розроблена з використанням бібліотеки React, що дозволяє створювати компонентну структуру інтерфейсу. Для маршрутизації між сторінками використано React Router, а для оформлення інтерфейсу - Tailwind CSS, який забезпечує зручне застосування готових CSS-класів. Обмін даними між клієнтом та сервером здійснюється за допомогою бібліотеки Axios.

У неавторизованих користувачів (гостей) є доступ до головної сторінки Web-застосунку, де можна переглядати доступні тести за класами, а також шукати тест за кодом. Інтерфейс для гостей є інтуїтивно зрозумілим, мінімалістичним та реалізований українською мовою.

Вибір класу реалізовано у вигляді горизонтального ряду кнопок з номерами класів (4, 5, 6 тощо). Після вибору конкретного класу користувач бачить перелік доступних тестів. Якщо тести для вибраного класу відсутні, з’являється відповідне повідомлення. На рис. 3.1 зображено головну сторінку для неавторизованого користувача з можливістю вибору класу.

У разі відсутності тестів для обраного класу на сторінці з’являється повідомлення «Для вибраного класу тестів не знайдено». Приклад відображення повідомлення наведено на рис. 3.2.

Кожен тест відображається у вигляді окремої картки з зазначенням основної інформації: назви, класу, предмета, тривалості тесту та кількості питань. У верхній частині картки розміщено кольорову смужку, що візуально відрізняє кожен тест. Вибір кольору реалізовано з попередньо визначеного

списку. При натисканні на картку тесту відкривається інформаційне вікно, у якому відображається додаткова інформація: ім’я та прізвище автора тесту, унікальний код доступу, а також кнопка «Пройти тест» або «Увійдіть, щоб пройти тест» залежно від статусу користувача.

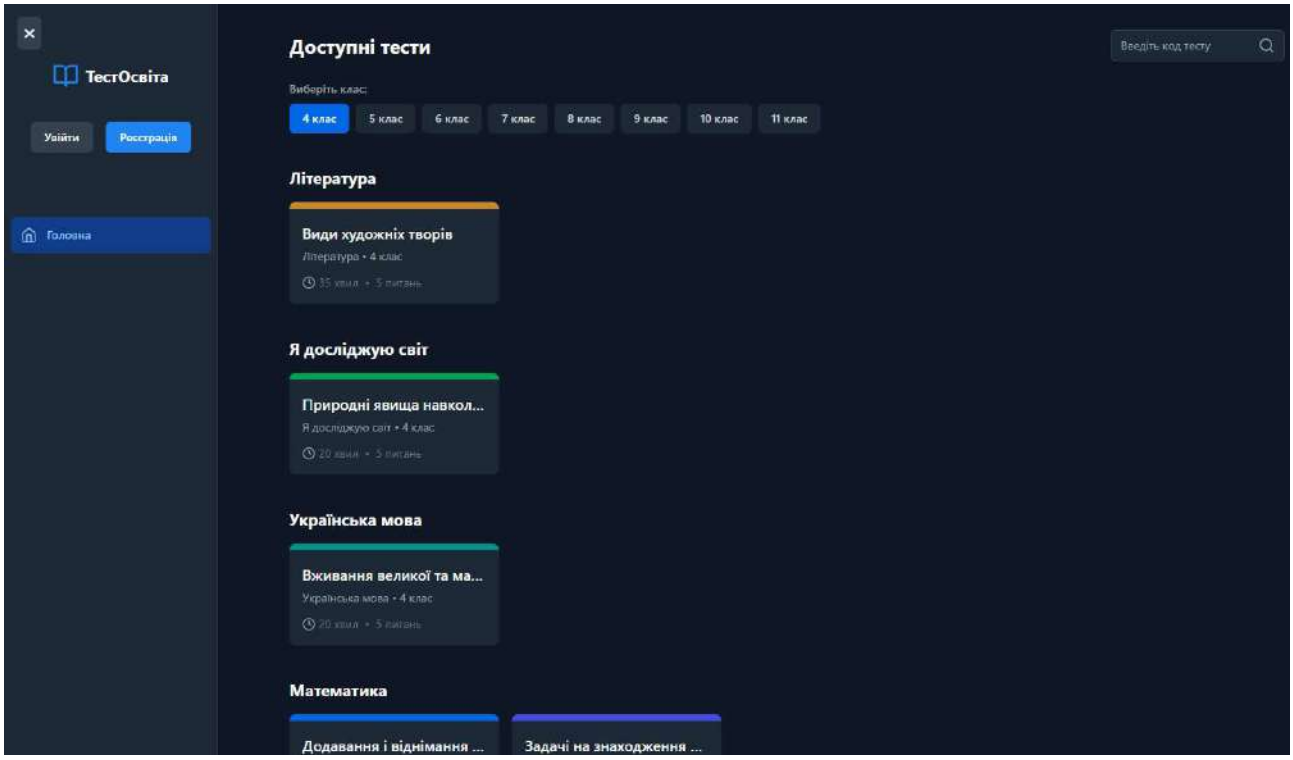

Рис. 3.1. Головна сторінка для гостя з вибором класу

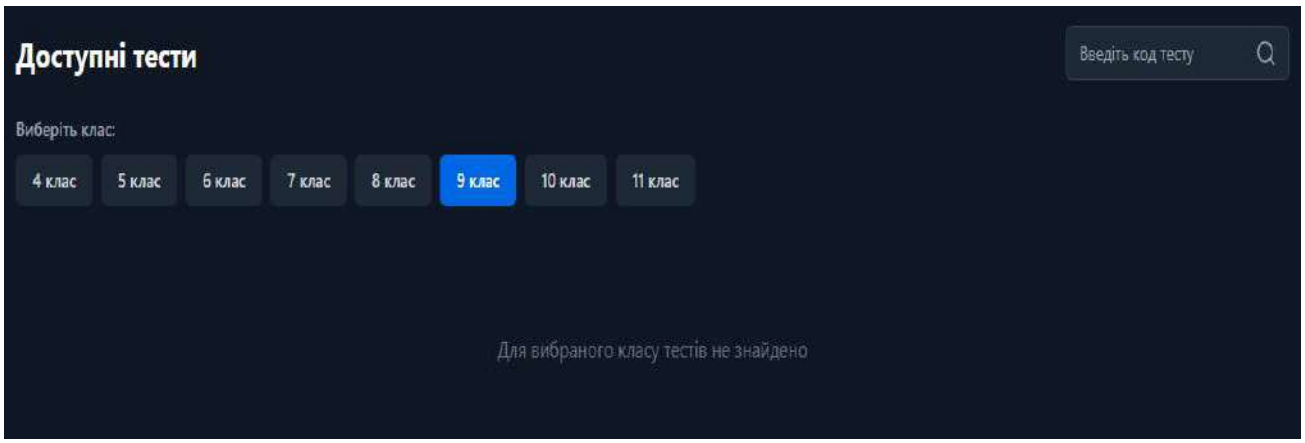

Рис. 3.2. Повідомлення про відсутність тестів для вибраного класу

На рис. 3.3 показано інформаційне вікно тесту для неавторизованого користувача.

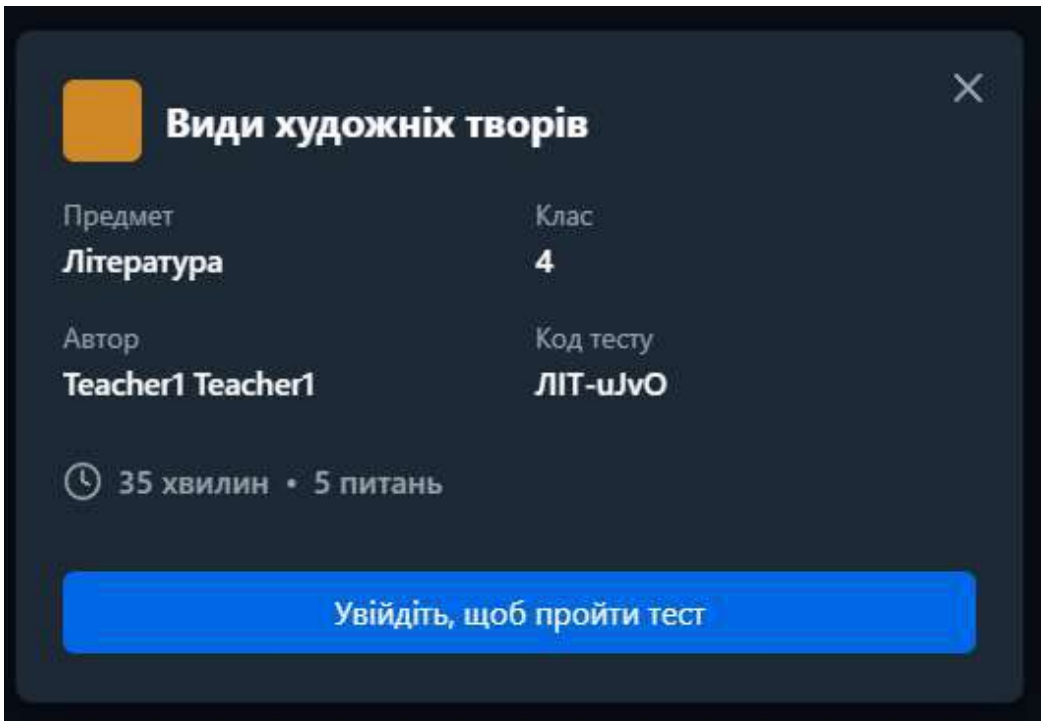

Рис. 3.3. Інформаційне вікно тесту у гостьовому режимі

У правому верхньому куті інтерфейсу також реалізовано поле для пошуку тесту за кодом. У разі введення коректного коду відкривається те саме інформаційне вікно, що й при натисканні на картку тесту. Зліва на сторінці знаходиться навігаційне меню, яке дозволяє перемикатися між основними розділами сайту. У режимі гостьового доступу в меню відображається лише одна вкладка - «Головна». Меню реалізовано у вигляді бічної панелі, яка може згортатися та розгортатися для зручності користувача, що дозволяє заощадити простір на екрані та сфокусувати увагу на вмісті головної частини сторінки.

Таким чином, інтерфейс для неавторизованого користувача надає базові можливості для ознайомлення з тестами: вибір класу, перегляд інформації про тести, пошук за кодом та ознайомлення з автором. Незважаючи на обмежений функціонал, гостьовий режим дозволяє користувачам швидко оцінити можливості платформи та за потреби здійснити реєстрацію для подальшого проходження тестів.

У Web-застосунку реалізовано окремі форми для реєстрації та авторизації користувачів. Після натискання відповідної кнопки користувач переходить на одну з цих форм, де вводить необхідні дані. Інтерфейс побудований у зручній та

зрозумілій формі: усі поля мають українське позначення, елементи розташовані логічно, а кнопки дій - інтуїтивно зрозумілі.

Форма реєстрації передбачає заповнення таких полів: ім’я, прізвище, по батькові, електронна пошта та пароль. Всі поля є обов’язковими для заповнення. За замовчуванням, кожен новий користувач після реєстрації отримує роль "учень". На рис. 3.4 зображено інтерфейс форми реєстрації нового користувача.

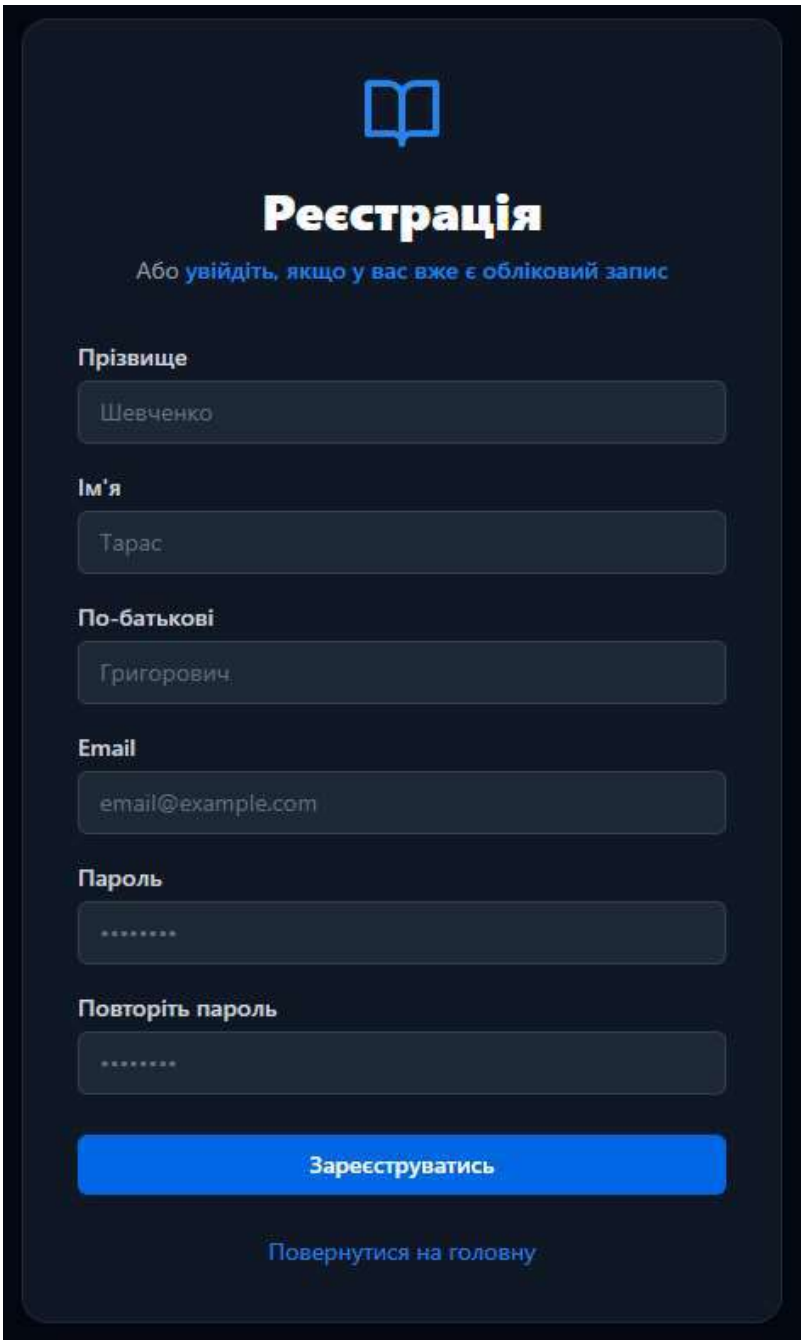

Рис. 3.4. Форма реєстрації користувача

Після успішної реєстрації користувач автоматично перенаправляється на головну сторінку сайту та отримує доступ до функціоналу, передбаченого для зареєстрованих користувачів. Авторизація передбачає введення електронної пошти та пароля. При коректному введенні даних користувач входить у систему, при цьому в навігаційному меню змінюється статус на авторизований. Інтерфейс авторизації показано на рис. 3.5.

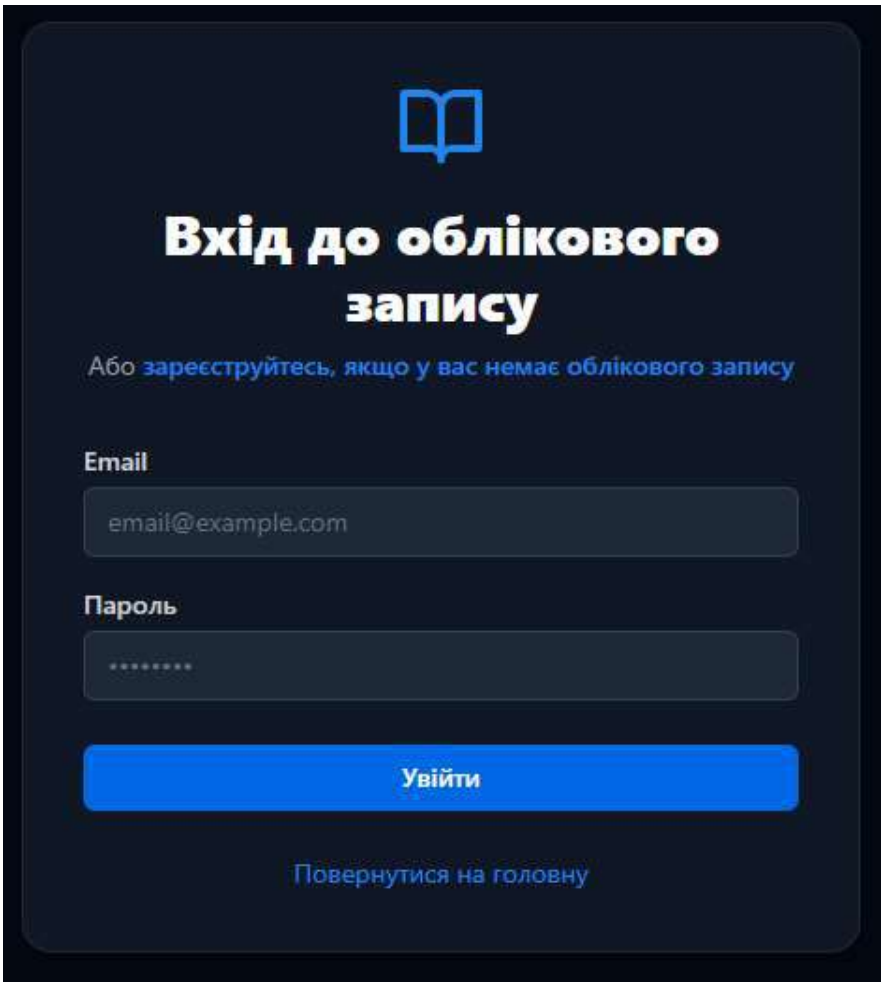

Рис. 3.5. Форма авторизації користувача

Варто зазначити, що в поточній версії застосунку не реалізовано механізм зміни пароля користувачем. Також відсутня клієнтська валідація полів пароля - користувач має змогу ввести пароль будь-якої довжини без повідомлень про помилки. Незважаючи на це, паролі на сервері зберігаються у захешованому вигляді з використанням алгоритму bcrypt, що забезпечує безпеку даних.

Таким чином, процеси реєстрації та авторизації реалізовані у відповідності до базових вимог до системи: нові користувачі мають змогу створити обліковий запис, увійти до системи та отримати початкову роль "учень". Незважаючи на відсутність валідації пароля на стороні клієнта та неможливість його зміни у поточній версії, система забезпечує безпечне зберігання даних завдяки використанню сучасного хешування та авторизаційних токенів.

Після успішної авторизації користувач з роллю "учень" отримує доступ до додаткового функціоналу. У навігаційному меню з’являються нові розділи: «Головна», «Пройдені тести» та «Стати вчителем». Основна взаємодія з системою для учня відбувається через головну сторінку, де відображаються тести, доступні для проходження. Після натискання кнопки, на картці тесту, «Пройти тест», відкривається інтерфейс тестування. У верхній частині відображається назва тесту, предмет, клас, а також таймер зворотного відліку, який автоматично розпочинається. Нижче послідовно відображаються запитання з відповідними варіантами відповідей. Тип запитання визначає, чи можна обрати одну правильну відповідь або кілька. Під кожним запитанням розміщено кнопки навігації - «Назад» та «Далі». Також реалізовано візуальний індикатор прогресу проходження (кількість запитань і поточний номер). Усі відповіді зберігаються локально до моменту завершення тесту. Інтерфейс проходження тесту з таймером та варіантами відповідей наведено на рис. 3.6.

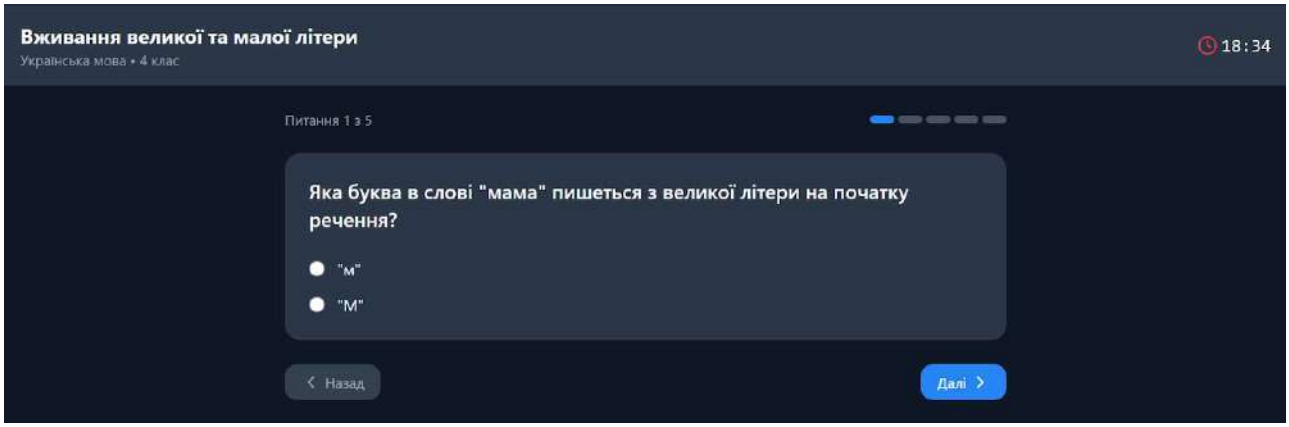

Рис. 3.6. Інтерфейс проходження тесту (з таймером та варіантами відповідей)

Якщо учень вже проходив певний тест, картка цього тесту на головній сторінці змінюється - додається текст із результатом (наприклад, "7/10"), а також змінюється стан кнопки при відкритті тесту. На рис. 3.7 представлено приклад головної сторінки, де один із тестів уже був пройденим.

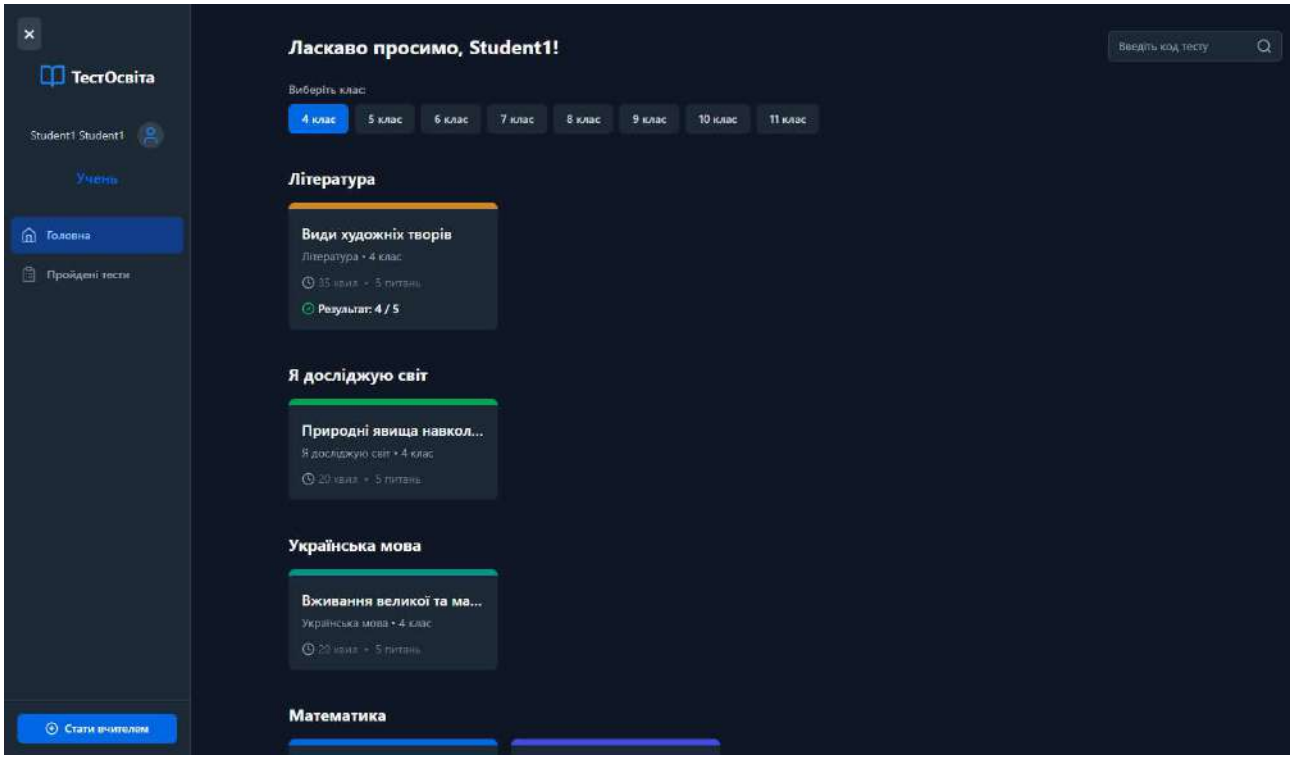

Рис. 3.7. Картка вже пройденого тесту на головній сторінці

При відкритті такої картки користувач бачить повідомлення про дату проходження тесту та свій результат. Кнопка «Пройти тест» замінюється на повідомлення «Тест вже пройдено», що унеможливлює повторну спробу. Відображення цього інтерфейсу показано на рис. 3.8.

Окрему вкладку навігаційного меню - «Пройдені тести» - реалізовано для зручного перегляду результатів усіх завершених тестів. Якщо користувач уже проходив тести, вони відображаються у вигляді окремих блоків з інформацією: назва тесту, предмет, клас, результат (кількість правильних відповідей) та дата проходження.

У разі, якщо жоден тест ще не був пройдений, нижче з’являється повідомлення: «Немає пройдених тестів» разом із додатковим поясненням, яке

мотивує учня скористатися доступними тестами на головній сторінці. Таким чином, у межах одного інтерфейсу система динамічно відображає як завершені тести, так і відсутність таких (при першому вході або з новим акаунтом). Обидва стани вкладки «Пройдені тести» (наявність результатів і відсутність) зображено на рис. 3.9.

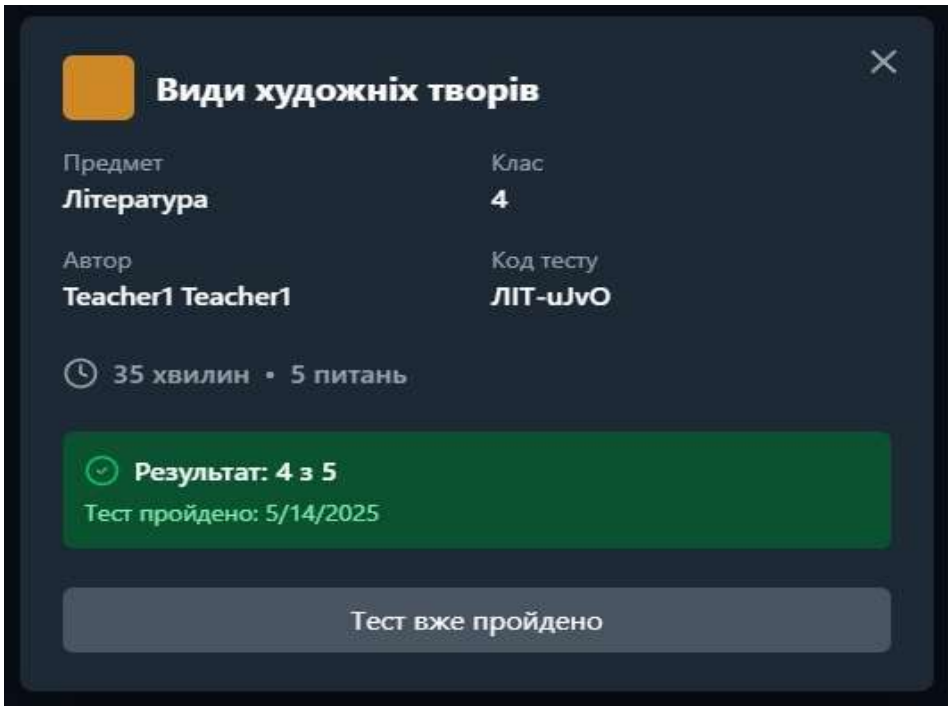

Рис. 3.8. Вікно тесту після його проходження

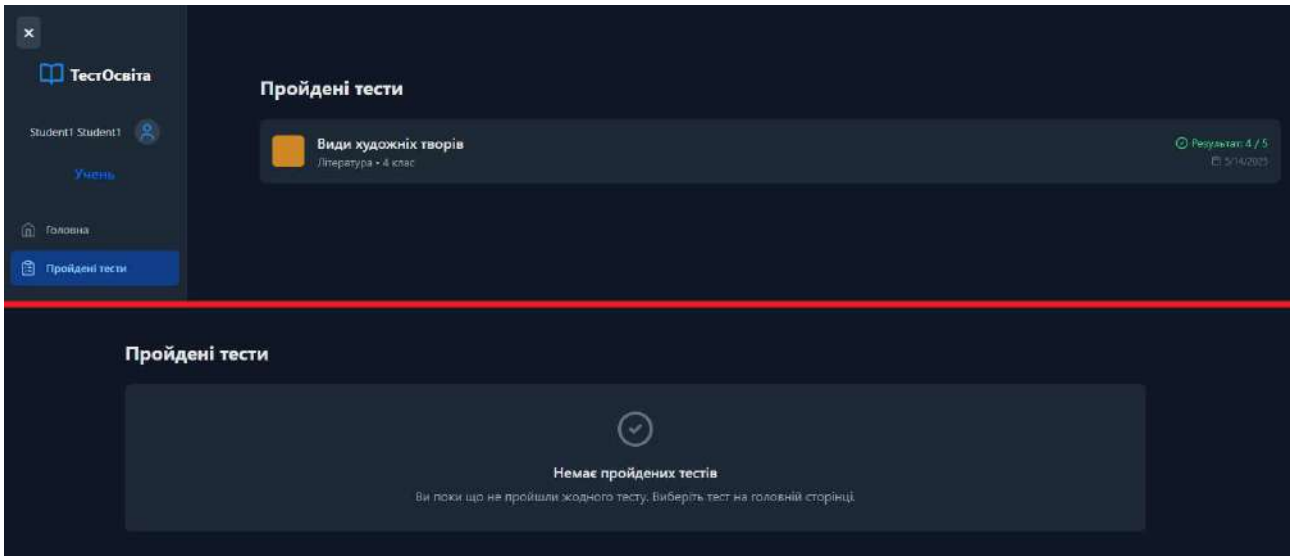

Рис. 3.9 Вкладка «Пройдені тести» з результатами та повідомленням про їх відсутність

Додатково у навігаційному меню учень бачить пункт «Стати вчителем», який дозволяє подати запит на зміну ролі. При натисканні з’являється інформаційне повідомлення з контактами адміністратора, з яким необхідно зв’язатися для підтвердження особи. Після спілкування адміністратор може надати роль «вчитель», якщо користувач справді має відповідну кваліфікацію. У поточній версії системи реалізовано лише інформаційне повідомлення (заглушку), без реального відправлення запиту, що є потенційним напрямом розвитку функціоналу.

Отже, інтерфейс користувача з роллю «учень» забезпечує повноцінну взаємодію з тестовим середовищем: від перегляду доступних завдань до фіксації результатів та їх подальшого аналізу. Візуальні підказки, відображення статусу проходження тесту та наявність вкладки з історією дозволяють учням контролювати свій прогрес. Водночас окремі аспекти, як-от обробка пошуку пройдених тестів або повторне проходження, залишаються потенційними напрямами для подальшого вдосконалення платформи.

Після надання ролі «вчитель» (адміністратором) користувач отримує доступ до додаткового функціоналу, що дозволяє створювати, публікувати та переглядати створені тести. У навігаційному меню з’являються нові вкладки - «Мої тести» та «Створити тест». При переході до розділу «Мої тести» користувач бачить таблицю з переліком власних тестів, у якій наведено такі дані: назва тесту, клас, предмет, тривалість, код доступу, дата створення та кольорова позначка. Приклад вигляду вкладки «Мої тести» наведено на рис. 3.10.

Для створення нового тесту вчитель може скористатися кнопкою «Створити тест», яка доступна як у розділі «Мої тести», так і в окремій вкладці меню. В обох випадках відкривається модальне вікно з майстером створення, який складається з кількох вкладок. У вкладці «Тест» користувач заповнює основну інформацію: назву тесту, клас, предмет, тривалість (із кроком у 5 хвилин), а також обирає колір, який візуально вирізнятиме тест серед інших.

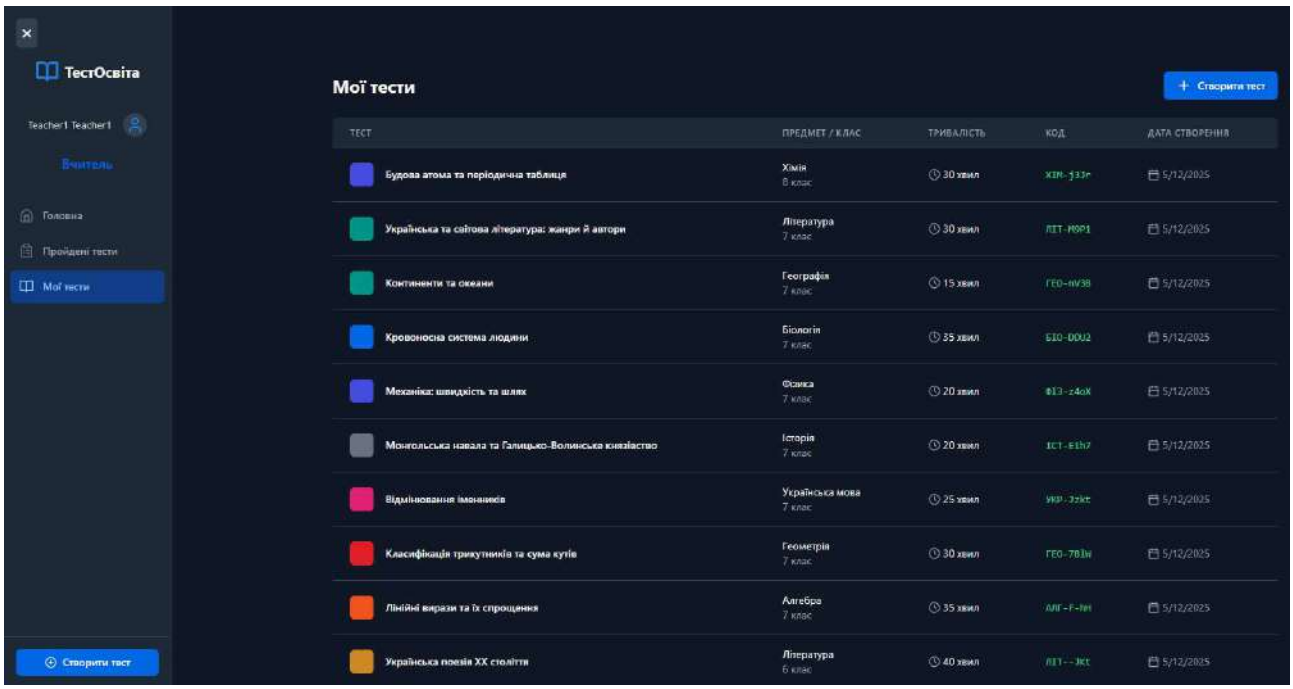

Рис. 3.10. Перелік створених тестів у вкладці «Мої тести»

Інтерфейс створення тесту реалізовано у вигляді покрокового майстра, що містить кілька вкладок. На першій вкладці заповнюється основна інформація про тест: назва, клас, предмет, тривалість (у хвилинах із кроком у 5), а також вибирається колір, який відображатиметься у картці тесту. На наступній вкладці вчитель має змогу ввести текст запитання, обрати його тип (одна або кілька правильних відповідей), а також додати варіанти відповідей (від 2 до 8). Кожен варіант можна редагувати або видалити. Об’єднаний інтерфейс створення тесту з обома вкладками показано на рис. 3.11.

Реалізований функціонал надає вчителям можливість повноцінно керувати власними тестами - від створення до публікації. Завдяки зручному інтерфейсу та покроковому майстру створення, навіть користувачі без глибоких технічних знань можуть швидко формувати завдання для учнів. Отриманий функціонал відповідає основним вимогам до ролі вчителя та може бути розширений у майбутньому шляхом додавання статистики проходження або редагування вже опублікованих тестів.

Користувач із роллю «адміністратор» має найширші повноваження в системі. Окрім функціоналу учня та вчителя, йому відкривається додаткова

вкладка «Користувачі» у навігаційному меню. У цьому розділі адміністратор має змогу переглядати повний список зареєстрованих користувачів системи та редагувати їхні дані.

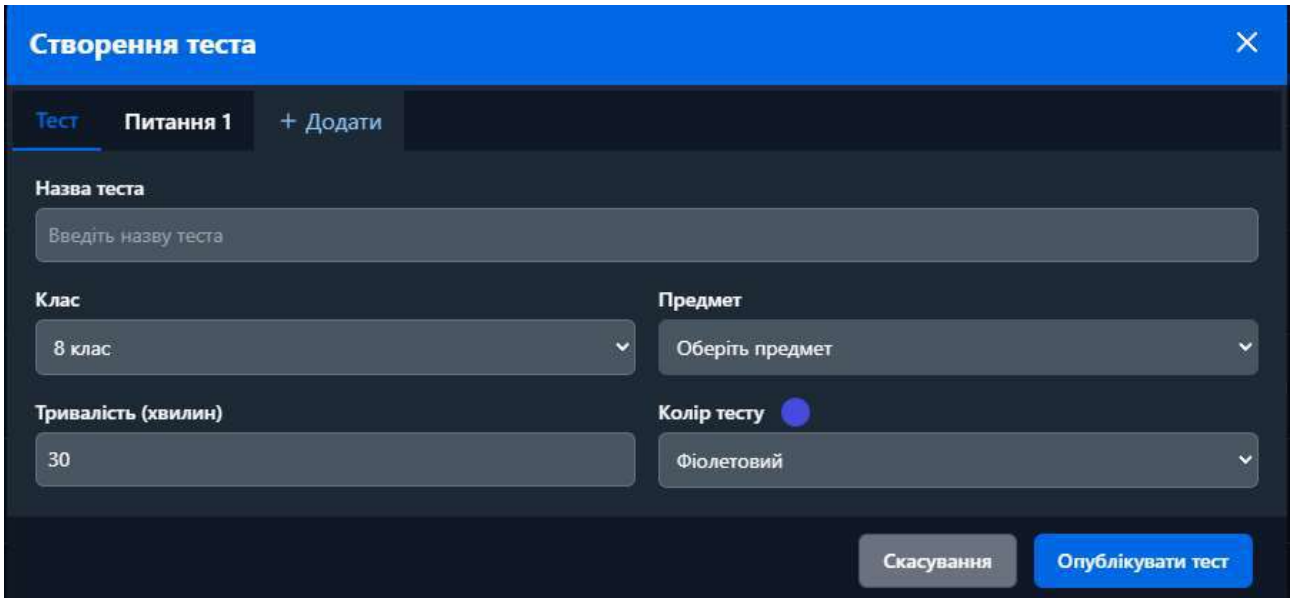

Рис. 3.11. Інтерфейс створення тесту

На рис. 3.12 показано вигляд навігаційного меню адміністратора з додатковою вкладкою.

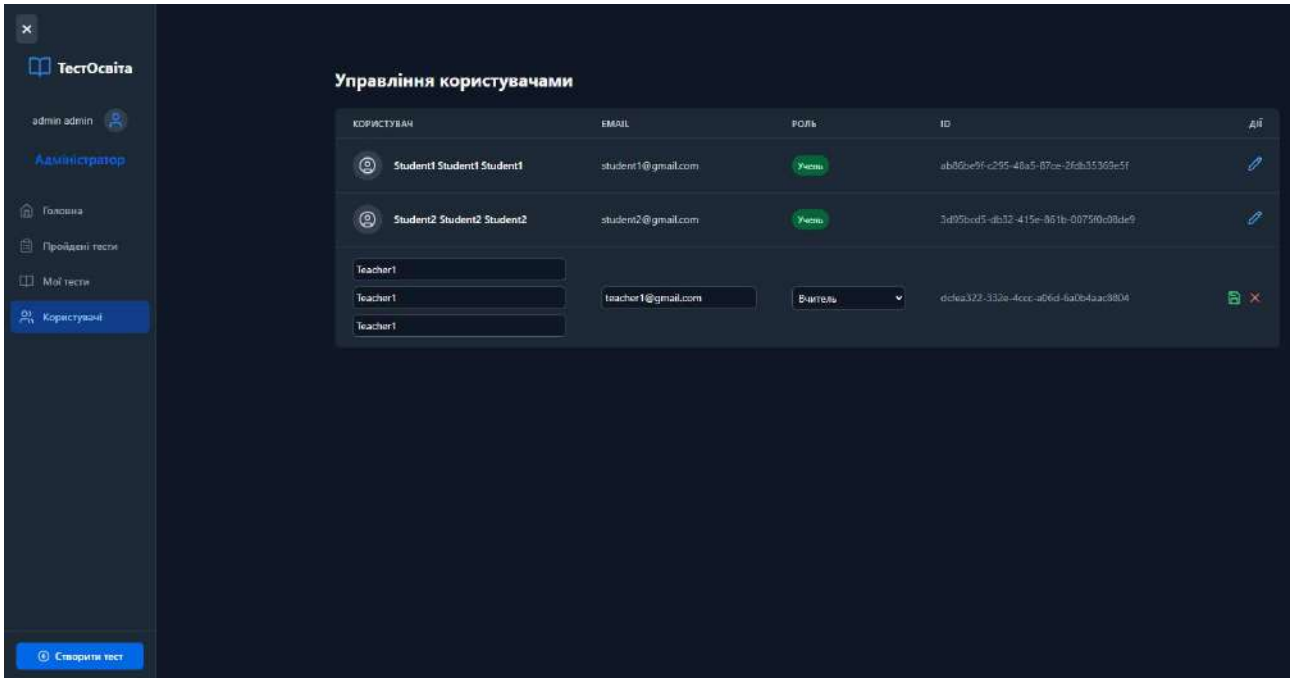

Рис. 3.12 Вкладка «Користувачі» у меню адміністратора

Таблиця користувачів містить такі дані: ім’я, прізвище, по батькові, email, роль, унікальний ідентифікатор користувача (ID), а також кнопки для редагування. При натисканні на кнопку редагування (позначену піктограмою олівця) відкривається вікно редагування, де адміністратор може змінити основні персональні дані та призначити іншу роль.

Варто зазначити, що у поточній версії системи адміністратор не має можливості змінювати пароль користувача - це обмеження пов’язане з відсутністю прямого доступу до відкритого значення пароля, оскільки всі паролі зберігаються в хешованому вигляді. Зміна пароля потребує окремої реалізації, яка може бути додана в подальших версіях платформи.

Завдяки реалізованому інтерфейсу адміністратор має змогу оперативно переглядати дані користувачів, змінювати їхні ролі та актуалізувати особисту інформацію. Незважаючи на обмеження щодо редагування паролів, представлений функціонал охоплює основні завдання адміністративного управління системою та створює надійну основу для її подальшого розширення.

3.3 Аналіз розробленого Web-застосунку та перспективи його подальшого розвитку

Під час реалізації Web-застосунку вдалося створити стабільно працюючу систему, яка охоплює ключові сценарії використання: реєстрацію та авторизацію користувачів, створення і проходження тестів, збереження результатів та розмежування прав доступу відповідно до ролей. Проте в процесі розробки деякі функціональні можливості, передбачені на етапі планування, не були реалізовані повністю або були спрощені з огляду на часові та технічні обмеження. Водночас вони розглядаються як потенційні напрями розвитку системи.

До функцій, які можуть бути реалізовані в майбутньому, належать:

– Повторне проходження тестів - наразі відсутній механізм дозволу на перескладання тесту для конкретного учня вчителем. Такий функціонал потребує додаткової логіки на сервері та інтерфейсу для вчителя.

– Запит на роль вчителя - поточна реалізація передбачає лише інформаційне повідомлення без можливості зв’язку з адміністратором. Надалі можливе додавання контактної форми або внутрішньої системи сповіщень.

– Зміна пароля користувачем - зараз така можливість відсутня через зберігання паролів у хешованому вигляді. Реалізація потребує створення механізму підтвердження особи та введення нового пароля.

– Повідомлення про помилки - при некоректних діях користувача система здебільшого не відображає повідомлень про помилку. Для покращення UX доцільно реалізувати вивід сповіщень про помилковий пароль, неіснуючий код тесту, відсутність прав доступу тощо.

– Фільтрація тестів за предметами - хоча реалізована фільтрація за класами, відбір за предметом відсутній. Її додавання дозволить зручніше шукати тести за навчальною дисципліною.

– Адаптивна верстка - візуальна частина системи не оптимізована для мобільних пристроїв, що обмежує зручність використання на різних екранах. Майбутнє вдосконалення може включати реалізацію повноцінної адаптивності.

Таким чином, описані напрями дозволяють поступово розширити функціонал застосунку, підвищити зручність використання та зробити систему більш гнучкою й масштабованою без суттєвих змін до вже реалізованої архітектури.

Висновки до розділу 3

Реалізований Web-застосунок забезпечує ключову функціональність системи тестування знань з урахуванням розмежування ролей користувачів, зручності інтерфейсу та безпечної роботи з даними. Структура застосунку побудована на сучасному технологічному стеку, що дозволяє ефективно

опрацьовувати запити, зберігати результати тестування та гарантувати стабільність взаємодії між клієнтом і сервером.

Логіка авторизації, керування тестами, обробки результатів і взаємодії з базою даних реалізована відповідно до вимог, сформованих на етапі проєктування. Використання JWT та bcrypt гарантує базовий рівень безпеки, а ORM Prisma забезпечує надійну та структуровану роботу з даними.

Незважаючи на відсутність деяких другорядних функцій, таких як повторне проходження тестів або зміна пароля, платформа виконує свою основну мету - створення, публікацію та проходження тестів з подальшим аналізом результатів. Архітектура системи дає змогу у майбутньому реалізувати додатковий функціонал без потреби у кардинальній зміні вже існуючої структури.

ВИСНОВКИ

Результатом виконання бакалаврської кваліфікаційної роботи став повнофункціональний Web-ресурс для створення тестів та перевірки знань, що дозволяє автоматизувати процес тестування, збереження результатів та взаємодії між учнями, вчителями й адміністраторами. У ході роботи проведено ґрунтовний аналіз предметної області, досліджено існуючі аналоги та виявлено їх ключові обмеження - зокрема, відсутність гнучкого управління ролями, недостатню адаптованість до української шкільної системи та обмеженість функціоналу в частині аналітики.

Запропоноване рішення реалізовано на основі сучасного клієнт-серверного підходу з використанням бібліотеки React для створення інтерфейсу, Express.js - для серверної логіки, та PostgreSQL у поєднанні з ORM Prisma - для збереження даних. Розроблений технологічний стек забезпечує високу продуктивність, масштабованість і безпечну роботу системи. Авторизація реалізована через JSON Web Token (JWT), а зберігання паролів - із використанням bcrypt, що відповідає вимогам сучасної кібербезпеки.

Система підтримує розмежування функціоналу для трьох типів користувачів (учень, вчитель, адміністратор), дозволяє створювати тести з питаннями різних типів, проходити їх у таймерному режимі, фіксувати результати й переглядати статистику. Інтерфейс користувача побудований з урахуванням зручності та інтуїтивності, а базові сценарії взаємодії змодельовано за допомогою UML-діаграм, що забезпечило зрозумілість та логічну послідовність реалізації.

Незважаючи на обмежений обсяг часу, реалізовано повний цикл розробки: від аналізу предметної області до створення працюючого прототипу. Водночас, окремі функціональні можливості, такі як повторне проходження тестів, зміна пароля або адаптивна верстка, залишились поза межами реалізації - проте визначені як потенційні напрями розвитку системи.

Отримані результати підтверджують доцільність обраного підходу до побудови системи онлайн-тестування та демонструють її практичну значущість для використання в закладах освіти. Обрана архітектура Web-додатку дозволяє легко масштабувати його функціонал у майбутньому, розширюючи можливості платформи відповідно до потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Forms – Офіційний сайт. URL: https://www.google.com/forms/ (дата звернення: 05.04.2025).
2. Moodle – Офіційний сайт. URL: https://moodle.org/ (дата звернення: 05.04.2025).
3. Платформа «На Урок». URL: https://naurok.ua/ (дата звернення: 06.04.2025).
4. Google Classroom – Офіційний сайт. URL: https://classroom.google.com/ (дата звернення: 06.04.2025).
5. Гейдарлі Є. О. Сучасні підходи до проєктування інформаційних систем: монографія. Київ: КНЕУ, 2020. 215 с.
6. Фаріонов В. Є. Інформаційні технології в управлінні. Київ: КНТ, 2018. 288 с.
7. Назаренко І. В., Шевчук А. В. Архітектура сучасних веб-застосунків: навч. посіб. Харків: ХНУРЕ, 2022. 190 с.
8. React – офіційна документація. URL: https://react.dev/ (дата звернення: 08.04.2025).
9. Vue.js – офіційна документація. URL: https://vuejs.org/ (дата звернення: 08.04.2025).
10. Angular – офіційна документація. URL: https://angular.io/ (дата звернення: 08.04.2025).
11. Node.js – офіційна документація. URL: https://nodejs.org/ (дата звернення: 08.04.2025).
12. Django – офіційна документація. URL: https://www.djangoproject.com/ (дата звернення: 08.04.2025).
13. Flask – офіційна документація. URL: https://flask.palletsprojects.com/ (дата звернення: 08.04.2025).
14. Spring Framework – офіційна документація. URL: https://spring.io/ (дата звернення: 08.04.2025).

15. PostgreSQL – офіційний сайт. URL: https://www.postgresql.org/ (дата звернення: 10.04.2025).
16. MySQL – офіційний сайт. URL: https://www.mysql.com/ (дата звернення: 10.04.2025).
17. Prisma ORM – офіційна документація. URL: https://www.prisma.io/ (дата звернення: 15.04.2025).
18. Sequelize – офіційна документація. URL: https://sequelize.org/ (дата звернення: 15.04.2025).
19. Tailwind CSS – офіційна документація. URL: https://tailwindcss.com/ (дата звернення: 17.04.2025).
20. Axios – офіційна документація. URL: https://axios-http.com/ (дата звернення: 17.04.2025).
21. React Router – офіційна документація. URL: https://reactrouter.com/ (дата звернення: 17.04.2025).
22. bcrypt – GitHub репозиторій. URL: https://github.com/kelektiv/node.bcrypt.js (дата звернення: 17.04.2025).
23. jsonwebtoken – офіційна сторінка. URL: https://github.com/auth0/node-jsonwebtoken (дата звернення: 17.04.2025).
24. nanoid – GitHub. URL: https://github.com/ai/nanoid (дата звернення: 17.04.2025).
25. OpenJS Foundation. Node.js Security Best Practices. URL: https://openjsf.org/ (дата звернення: 18.04.2025).
26. ISO/IEC 27001:2022 – Інформаційна безпека: Стандарти. URL: https://www.iso.org/ (дата звернення: 18.04.2025).
27. Microsoft. Secure Password Storage. URL: https://learn.microsoft.com/ (дата звернення: 18.04.2025).
28. REST API – Wikipedia. URL: https://en.wikipedia.org/wiki/Representational_state_transfer (дата звернення: 19.04.2025).

29. UML Diagrams Overview – Visual Paradigm. URL: https://www.visual-paradigm.com/ (дата звернення: 22.04.2025).
30. Філатов М. Ю. Основи побудови UML-діаграм: навч. посібник. Львів: Львівська політехніка, 2021. 124 с.
31. Prisma: Working with PostgreSQL. Офіційна документація. URL: https://www.prisma.io/docs/concepts/database-connectors/postgresql (дата звернення: 24.04.2025).
32. Swagger API Docs. URL: https://swagger.io/docs/ (дата звернення: 24.04.2025).
33. JWT.IO – JSON Web Tokens. URL: https://jwt.io/ (дата звернення: 26.04.2025).
34. Visual Studio Code – офіційний сайт. URL: https://code.visualstudio.com/ (дата звернення: 02.05.2025).
35. MySQL Workbench – Офіційний сайт. URL: https://www.mysql.com/products/workbench/ (дата звернення: 02.05.2025).
36. GitHub Copilot. URL: https://github.com/features/copilot (дата звернення: 05.05.2025).
37. React Hook Form – офіційна документація. URL: https://react-hook-form.com/ (дата звернення: 08.05.2025).
38. JWT Best Practices. Auth0 Blog. URL: https://auth0.com/blog/jwt-best-practices/ (дата звернення: 10.05.2025).
39. PostgreSQL Indexes – офіційна документація. URL: https://www.postgresql.org/docs/current/indexes.html (дата звернення: 12.05.2025).
40. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: Мінекономрозвитку України, 2015. 16 с.

ЗГОДА здобувача(чки) вищої освіти

Державного університету економіки і технологій про
перевірку кваліфікаційної роботи на прояви
академічного плагіату
та розміщення в Репозитарії Університету

Я, Камінський Ілля Миколайович (ПІП),

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська (магістерська) робота

Розробка веб-ресурсу для створення тестів та перевірки знань

(назва роботи повністю) виконана самостійно та не містить академічного плагіату. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений(а). Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований(на), що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений(на).

Дата	підпис	ініціали, прізвище (власноруч)
10.06.2025		І.М Камінський