Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«МОДЕЛЬ ВЕБЗАСТОСУНКА ДЛЯ АДМІНІСТРУВАННЯ БАНКУ ТЕСТОВИХ ЗАВДАНЬ І АВТОМАТИЗОВАНОГО КОНСТРУЮВАННЯ ТЕСТІВ»

Спеціальність ***123 – Комп'ютерна інженерія.***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Дмитро ПРОСКУРІН**
(підпис)
Науковий керівник _____ **Сергій ПРИКОЛОТІН**
(підпис)

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку джерел та 5 додатків. Загальний обсяг роботи : 73 сторінки (50 - основний текст), 10 рисунків, 4 таблиці, 20 найменувань у списку використаних джерел.

Мета - оптимізація процесу створення та адміністрування тестових завдань в електронному навчанні шляхом розробки вебзастосунка з інтеграцією методів штучного інтелекту.

Об'єкт дослідження - процес автоматизації створення та адміністрування тестових завдань в електронному навчанні.

Предмет дослідження - методи, моделі та алгоритми штучного інтелекту для автоматизованого конструювання тестових завдань, а також засоби програмної підтримки ведення та адміністрування банку тестів у веборієнтованих системах.

Область застосування - заклади освіти, платформи дистанційного навчання та спеціальність 123 «Комп'ютерна інженерія».

Застосунок QuizApp має ієрархічну структуру контенту та RBAC-авторизацію. Функціонал викладача охоплює керування курсами та масовий JSON-імпорт тестів. Студентська частина включає каталог курсів із захищеним зарахуванням та інтерактивний модуль проходження тестів з автоперевіркою.

Ключовий результат - інтеграція Google Gemini API для автогенерації тестів із налаштуванням теми, мови та розподілу за 6 когнітивними рівнями Таксономії Блума.

Наукова новизна полягає у комплексному підході до інтеграції великих мовних моделей у конструювання тестів у межах структурованих курсів. Практична значущість - скорочення часу викладача на підготовку тестів до 80%.

Ключові слова: комп'ютерна інженерія, вебзастосунок, банк тестових завдань, штучний інтелект, автоматизована генерація тестів, великі мовні моделі, next.js, typescript, prisma ORM, електронне навчання, google gemini API.

ABSTRACT

The explanatory note consists of an introduction, 4 chapters, conclusions, a list of references, and 5 appendices. Total volume: 73 pages (50 - main text), 10 figures, 4 tables, 20 references.

The purpose - to optimise the process of creating and administering test items in e-learning by developing a client-server web application with integrated artificial intelligence methods.

The object of research - the automation of creating and administering test items in e-learning systems.

The subject of research - methods, models, and algorithms of artificial intelligence for automated test construction, as well as software tools for maintaining and administering the test item bank in web-oriented systems.

Application area - educational institutions and distance learning platforms.

The QuizApp implements a hierarchical content structure and RBAC authorisation. The teacher manages courses and performs bulk JSON test import; the student browses the catalogue, enrols in courses, and takes tests with automatic answer checking.

Key outcome - Google Gemini API integration for automated test generation configurable by topic, language, and distribution across 6 cognitive levels of Bloom's Taxonomy.

Scientific novelty - a comprehensive approach to integrating large language models into test construction within structured course systems. Practical significance - reducing instructor test preparation time by up to 80 %.

Keywords: computer engineering, web application, test item bank, artificial intelligence, automated test generation, large language models, next.js, typescript, prisma ORM, e-learning, google gemini API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ **6**
ВСТУП **7**
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ **11**
1.1. Актуальність автоматизації процесу тестування в системі електронного навчання **11**
1.2. Огляд та порівняльний аналіз існуючих платформ для тестування **12**
1.3. Порівняльний аналіз технологій розробки **14**
1.4. Обґрунтування вибору технологічного стеку **16**
Висновки за розділом 1 ..... **17**
РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ 19
2.1. Загальна архітектура вебзастосунка **19**
2.2. Проєктування моделі бази даних **20**
2.3. Модель авторизації та розмежування ролей **23**
2.4. Проєктування API та серверних дій **25**
Висновки за розділом 2 ..... **28**
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ВЕБЗАСТОСУНКА 30
3.1. Реалізація функціоналу вчителя **30**
3.2. Реалізація функціоналу учня **34**
3.3. Система оцінювання та рейтингування студентів **39**
3.4. Інтерфейс користувача та бібліотека компонентів shadcn/ui **41**
Висновки за розділом 3 ..... **43**
РОЗДІЛ 4 ІНТЕГРАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА ТЕСТУВАННЯ СИСТЕМИ 44
4.1. Автоматизована генерація тестових завдань через Google Gemini API **44**
4.2. Параметризація генерації та оцінка якості ШІ-завдань **47**
4.3. Тестування програмного забезпечення та аналіз результатів **50**
4.3.1. Автоматизоване модульне тестування **50**

4.3.2. Тестування модуля інтеграції ШІ (Google Gemini API)..... 51
Висновки до розділу 4..... 52
ВИСНОВКИ..... 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 56
ДОДАТКИ..... 58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI (Artificial Intelligence) - штучний інтелект

API (Application Programming Interface) - інтерфейс програмування застосунків

CRUD (Create, Read, Update, Delete) - базові операції з даними: створення, читання, оновлення, видалення

EdTech (Educational Technology) - освітні технології

JSON (JavaScript Object Notation) - легкий формат обміну даними

JWT (JSON Web Token) - відкритий стандарт для передачі інформації у вигляді JSON-об'єкта

LLM (Large Language Model) - велика мовна модель

LMS (Learning Management System) - система управління навчанням

ORM (Object-Relational Mapping) - об'єктно-реляційне відображення

RBAC (Role-Based Access Control) - керування доступом на основі ролей

REST (Representational State Transfer) - архітектурний стиль побудови API

SaaS (Software as a Service) - програмне забезпечення як послуга

SSR (Server-Side Rendering) - серверний рендеринг

SSG (Static Site Generation) - статична генерація сайту

UI (User Interface) - інтерфейс користувача

UX (User Experience) - досвід користувача

RSC (React Server Components) - серверні компоненти React

СУБД (Система управління базами даних) - програмне забезпечення для створення, зберігання та управління базами даних

ВСТУП

Сучасна система освіти активно переходить до цифрового формату, що зумовлено швидким розвитком інформаційних технологій та потребами сьогодення. Пандемія COVID-19 стала каталізатором прискореного переходу до дистанційних форм навчання: за даними UNESCO [1], у 2020-2021 роках понад 1,5 мільярда учнів у світі були змушені перейти на онлайн-навчання, що виявило критичну потребу в якісних цифрових інструментах для організації освітнього процесу. Навіть після завершення пандемії гібридні та дистанційні форми навчання залишаються невід'ємною складовою сучасної освіти, а ринок освітніх технологій продовжує стабільно зростати: станом на 2026 рік його глобальний обсяг оцінюється у понад 214 мільярдів доларів, а до 2030 року прогнозується зростання до 340-400 мільярдів доларів [4].

Одним із ключових елементів як дистанційної, так і традиційної освіти є система тестування, яка дозволяє здійснювати об'єктивне оцінювання знань, забезпечувати зворотний зв'язок між викладачем та студентом, а також адаптувати навчальний процес до потреб кожного здобувача освіти [3]. Традиційні підходи до створення тестових завдань вимагають значних часових витрат від викладача: розробка якісного тесту з 20–40 питань різного рівня складності може займати від 4 до 8 годин. При цьому необхідно забезпечити різноманітність типів завдань, належний рівень складності, валідність питань та їхню відповідність навчальній програмі.

Автоматизація процесу створення тестових завдань за допомогою сучасних технологій штучного інтелекту, зокрема великих мовних моделей, є актуальним науковим та практичним завданням. Сучасні LLM-системи, такі як Google Gemini, GPT-5 та інші, демонструють здатність до генерації змістовних, контекстно-залежних тестових завдань різного рівня складності, що відкриває принципово нові можливості для автоматизації педагогічної діяльності.

Актуальність дослідження визначається необхідністю створення сучасного, зручного та ефективного вебінструменту, який поєднує функції адміністрування банку тестових завдань із можливостями автоматизованого конструювання тестів на основі технологій штучного інтелекту. Існуючі рішення (Google Forms, Kahoot, Moodle, Quizlet) мають суттєві обмеження: або надто спрощений функціонал, або надмірну складність налаштування та адміністрування, або відсутність інтеграції із сучасними ШІ-інструментами для автоматизованої генерації тестових завдань.

Зв'язок роботи з науковими програмами, планами, темами.

Робота виконана в межах спеціальності 123 «Комп'ютерна інженерія» та безпосередньо реалізує її ключові компетентності: проєктування клієнт-серверної архітектури, розробку вебзастосунка з використанням сучасного стеку (Next.js, TypeScript, Prisma ORM), реалізацію рольової моделі доступу (RBAC) та програмну інтеграцію зовнішнього AI-сервісу. Таким чином, QuizApp є прикладним програмно-інженерним рішенням у галузі інформаційних технологій, а не суто педагогічним або методичним дослідженням.

Об'єкт дослідження - процес автоматизації створення та адміністрування тестових завдань в електронному навчанні.

Предмет дослідження - методи, моделі та алгоритми штучного інтелекту для автоматизованого конструювання тестових завдань, а також засоби програмної підтримки ведення та адміністрування банку тестів у веборієнтованих системах.

Мета роботи - оптимізація процесу створення та адміністрування тестових завдань в електронному навчанні шляхом розробки вебзастосунка з інтеграцією методів штучного інтелекту.

Для досягнення поставленої мети визначено такі завдання:

1. Проаналізувати сучасні підходи та існуючі рішення для адміністрування банків тестових завдань і автоматизованого конструювання тестів.

2. Спроектувати архітектуру вебзастосунка та розробити модель реляційної бази даних з використанням ORM Prisma та SQLite.

3. Реалізувати функціонал викладача: створення та керування курсами, структурування контенту за темами, імпорт тестових завдань через JSON-парсер.

4. Дослідити та інтегрувати методи штучного інтелекту для автоматизованої генерації тестових завдань різного рівня складності.

5. Реалізувати функціонал учня: каталог курсів із «живим» пошуком, механізм зарахування з парольним захистом, інтерфейс проходження курсів та тестів.

6. Розробити клієнтську частину застосунка з використанням Next.js (App Router), TypeScript та бібліотеки компонентів shadcn/ui.

7. Провести тестування розробленого вебзастосунка та оцінити якість ШІ-генерації тестових завдань.

8. Надати рекомендації щодо подальшого вдосконалення системи та можливих напрямків її розвитку.

Методи дослідження. Методологія дослідження базується на комплексному підході, що включає:

– аналіз та синтез - для вивчення існуючих рішень у сфері автоматизованого конструювання тестів та визначення їхніх переваг і недоліків;

– метод моделювання - для проєктування архітектури застосунка та моделі бази даних;

– порівняльний аналіз - для вибору оптимального технологічного стеку та оцінки ефективності запропонованого рішення;

– метод експерименту - для тестування системи та оцінки якості ШІ-генерації тестових завдань.

Наукова новизна роботи полягає в розробці комплексного підходу до інтеграції великих мовних моделей у процес автоматизованого конструювання тестових завдань різного рівня складності. На відміну від існуючих рішень, запропонований підхід дозволяє генерувати питання з урахуванням тематики

конкретного навчального курсу, заданого розподілу за когнітивними рівнями Таксономії Блума та мови генерації.

Практична значущість результатів полягає у створенні готового до впровадження вебзастосунка, який може бути використаний у навчальних закладах різного рівня для організації процесу тестування. Застосунок суттєво зменшує часові витрати викладача на підготовку тестових матеріалів - орієнтовно на 80% порівняно з повністю ручним створенням завдань. Зазначений відсоток обґрунтовується таким чином: ручне створення якісного тесту з 20 - 40 питань займає 4 - 8 годин; при використанні ШІ-генерації сам процес створення чернетки тесту скорочується до 10 - 30 секунд, однак викладач додатково витрачає час на рецензування, коригування формулювань та верифікацію правильності відповідей, що в сукупності становить орієнтовно 1 - 1,5 години - тобто близько 20% від початкових витрат. Результати роботи пройшли апробацію у Всеукраїнському конкурсі студентських наукових робіт з галузей знань і спеціальностей у 2025-2026 навчальному році (спеціальність «Комп'ютерна інженерія»), за результатами якого автора нагороджено дипломом І ступеня (копію диплома наведено у Додатку Д).

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів з висновками до кожного з них, загальних висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 69 сторінок, з них основного тексту - 40 сторінок. Робота містить 10 рисунків, 4 таблиці. Список використаних джерел налічує 21 найменування.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Актуальність автоматизації процесу тестування в системі електронного навчання

Система освіти активно змінюється під впливом цифрових технологій. Електронне навчання давно перестало бути чимось новим - сьогодні воно є частиною освіти на всіх рівнях: від шкільної до вищої та корпоративної. Невід'ємною складовою будь-якого навчального процесу є оцінювання знань - воно дозволяє визначити, наскільки добре засвоєно матеріал, знайти прогалини та за потреби скоригувати підхід до навчання.

Тестування закріпило за собою статус одного з найефективніших інструментів оцінювання в умовах масового дистанційного навчання. Проте етап підготовки якісного тестового матеріалу залишається найбільш ресурсомістким завданням для викладачів. Розробка одного якісного тесту з 20–40 питань різного рівня складності може займати від 4 до 8 годин робочого часу викладача [2]. При цьому необхідно дотримуватись вимог до якості тесту:

– змістовної валідності - питання мають охоплювати весь навчальний матеріал рівномірно;

– когнітивного балансу - розподіл завдань має відбуватися за різними рівнями розумової діяльності (від простого відтворення фактів до аналізу та синтезу), що є значно ефективнішим підходом, ніж класичний та застарілий поділ на «легкі» й «складні»;

– формату питань - різноманітність типів (одиночний вибір, множинний вибір, встановлення відповідності, есе) для комплексної перевірки знань;

– мовної коректності - чіткість та однозначність формулювань.

Автоматизація процесу генерації тестових завдань за допомогою штучного інтелекту є одним із найперспективніших напрямків розвитку. Сучасні великі мовні моделі, зокрема Gemini, OpenAI GPT-5 та Claude, демонструють здатність

до генерації змістовних, контекстно-залежних тестових завдань різних типів і рівнів складності. Проте більшість існуючих освітніх платформ або не інтегрують такі можливості взагалі, або роблять це поверхово - без прив'язки до конкретного навчального курсу та його тематики.

Таким чином, існує об'єктивна потреба у вебзастосунку, який поєднує:

– структуроване адміністрування банку тестових завдань (за курсами, темами, квізами);

– масовий імпорт питань у машиночитаному форматі;

– автоматизовану ШІ-генерацію тестових завдань з налаштуванням параметрів;

– зручний інтерфейс для студентів з миттєвим відображенням результатів;

– систему рейтингування та відстеження прогресу навчання.

Саме ця потреба стала основою для розробки вебзастосунка - предмету даної кваліфікаційної роботи.

1.2. Огляд та порівняльний аналіз існуючих платформ для тестування

Ринок освітніх технологій продовжує демонструвати стабільне зростання. За актуальними даними провідних аналітичних компаній (зокрема HolonIQ), станом на 2026 рік глобальний обсяг ринку оцінюється у понад 214 мільярдів доларів, а до 2030 року прогнозується його подальше зростання до 340–400 мільярдів доларів [4]. При цьому загальні світові витрати на освіту до кінця десятиліття сягнуть 10 трильйонів доларів. У сегменті систем тестування функціонує значна кількість рішень - розглянемо найбільш поширені з них.

Google Forms - безкоштовний інструмент від Google, який дозволяє створювати опитування та прості тести. Переваги: доступність, простота використання, інтеграція з Google Workspace, можливість збору відповідей у Google Sheets. Недоліки: обмежені можливості для створення складних тестів, відсутність структурування за курсами та темами, відсутність системи рейтингування та відстеження прогресу, неможливість імпорту питань у структурованому форматі, відсутність ШІ-генерації завдань [5].

Kahoot - інтерактивна платформа для створення квізів у ігровій формі. Основний акцент зроблено на залучення та мотивацію студентів через ігрові елементи. Переваги: висока залученість студентів, простота створення квізів, підтримка режиму реального часу. Проте платформа має обмежений безкоштовний функціонал, не підтримує ієрархічну структуру курсів, не надає можливостей для автоматизованої генерації тестів та масового імпорту питань [6].

Moodle - потужна система управління навчанням з відкритим кодом. Має розвинений модуль тестування з підтримкою різних типів питань, банку завдань та адаптивного тестування. Переваги: широкі можливості налаштування, підтримка різних типів питань, велика спільнота розробників. Однак Moodle характеризується високою складністю налаштування та адміністрування, вимагає окремої серверної інфраструктури та кваліфікованого технічного персоналу, а також не має нативної інтеграції з сучасними LLM для ШІ-генерації [7].

Quizlet - платформа для створення навчальних карток та тестів. Зручна для запам'ятовування фактичного матеріалу, має приємний інтерфейс та мобільні застосунки. Проте Quizlet має обмежені можливості для створення комплексних тестів з розгалуженою логікою, не підтримує курсову структуру та ієрархічне структурування контенту [8].

Для наочного порівняння розглянутих платформ з розробленим застосунком QuizApp складено таблицю 1.1.

Таблиця 1.1

Порівняльний аналіз платформ для тестування

Критерій	Google Forms	Kahoot	Moodle	Quizlet	QuizApp
Безкоштовний доступ	Так	Частково	Так	Частково	Так
Ієрархічна структура курсів	Ні	Ні	Так	Ні	Так

Продовження таблиці 1.1

Масовий імпорт питань (JSON)	Ні	Ні	Частково	Ні	Так
AI-генерація тестів	Ні	Ні	Ні	Ні	Так
Система рейтингування	Ні	Частково	Так	Ні	Так
Простотаналаштування	Так	Так	Ні	Так	Так
Парольний захист курсів	Ні	Ні	Так	Ні	Так
Живий пошук курсів	Ні	Ні	Частково	Ні	Так

Умовні позначення: Так - підтримується; Частково - обмежена підтримка; Ні - не підтримується

Проведений аналіз виявив низку спільних недоліків існуючих рішень:

– відсутність або обмежена підтримка автоматизованої генерації тестів із використанням ШІ;

– складність або неможливість масового імпорту питань у структурованому форматі;

– недостатня гнучкість у структуруванні навчального контенту за ієрархічним принципом;

– обмежені можливості аналітики, рейтингування та відстеження прогресу студентів.

Виявлені недоліки підтверджують доцільність розробки нового вебзастосунка, який усуває зазначені обмеження.

1.3. Порівняльний аналіз технологій розробки

Для розробки сучасного вебзастосунка необхідно обрати оптимальний технологічний стек, який забезпечить високу продуктивність розробки, масштабованість та якість кінцевого продукту. Розглянемо основні альтернативи за ключовими компонентами стеку.

Фронтенд-фреймворки. У сфері фронтенд-розробки продовжують домінувати три ключові інструменти: React, Angular та Vue.js. React, створений і підтримуваний компанією Meta, залишається найпопулярнішою бібліотекою для побудови інтерфейсів користувача. За даними опитування Stack Overflow Developer Survey (2025 рік), React використовують 44,7% розробників [9]. Компонентний підхід React, використання віртуального DOM та надзвичайно розвинена екосистема роблять його найоптимальнішим вибором для проєктів середньої та великої складності у 2026 році.

Angular - повнофункціональний фреймворк від Google з вбудованими засобами для маршрутизації, форм та HTTP-запитів. Відрізняється строгою архітектурою та використанням TypeScript за замовчуванням. Однак Angular характеризується вищим порогом входження для розробників через обов'язкове знання декораторів, модулів, механізму впровадження залежностей та бібліотеки реактивного програмування, а також надмірною архітектурною складністю для проєктів середнього масштабу, що збільшує час початкової розробки.

Vue.js - прогресивний фреймворк з простим синтаксисом та гарною документацією. Популярний серед початківців, але має меншу екосистему порівняно з React та Angular.

Повностековий фреймворк. Next.js - фреймворк на базі React, розроблений компанією Vercel, який надає потужні можливості серверного рендерингу (SSR), статичної генерації (SSG) та React Server Components у версії 14 з App Router. Next.js суттєво спрощує маршрутизацію, оптимізацію продуктивності та розгортання застосунків [10]. Альтернативою є Remix (також на базі React) та SvelteKit (на базі Svelte), однак вони мають меншу екосистему та спільноту підтримки.

Мова програмування. TypeScript - надмножина JavaScript, розроблена Microsoft, яка додає статичну типізацію. Використання TypeScript зменшує кількість помилок на етапі розробки на 15-20% порівняно з чистим JavaScript [11] та значно покращує досвід розробки завдяки автодоповненню та рефакторингу в

IDE. У 2026 році TypeScript став загальноприйнятим стандартом для промислових вебпроєктів середнього та великого масштабу.

ORM та база даних. Для роботи з базою даних розглянуто два основних підходи: використання ORM та прямі SQL-запити. ORM Prisma є сучасним інструментом для Node.js та TypeScript, який забезпечує типобезпечний доступ до бази даних, автоматичну генерацію міграцій та зручний API для CRUD-операцій [12]. Альтернативою є Drizzle ORM - більш легкий інструмент, проте з меншою документацією та екосистемою.

Як СУБД на етапі розробки та прототипування обрано SQLite - легку вбудовану базу даних, що не потребує окремого серверного процесу. SQLite забезпечує швидке розгортання без необхідності окремого серверного процесу та є доцільним вибором для прототипування і навчальних проєктів з можливістю подальшої міграції на PostgreSQL при масштабуванні.

UI-бібліотека. Для побудови інтерфейсу користувача обрано бібліотеку компонентів shadcn/ui - колекцію повторно використовуваних компонентів, побудованих на основі Radix UI та стилізованих за допомогою Tailwind CSS. На відміну від традиційних бібліотек компонентів (Material UI, Ant Design), shadcn/ui копіює код компонентів безпосередньо в проєкт, що забезпечує повний контроль над стилями та поведінкою [13].

1.4. Обґрунтування вибору технологічного стеку

На основі проведеного порівняльного аналізу для розробки вебзастосунка QuizApp обрано такий технологічний стек:

- Next.js 14 (App Router) - як основний повностековий фреймворк, що забезпечує SSR, React Server Components, маршрутизацію та Server Actions. Дозволяє реалізувати як клієнтський інтерфейс, так і серверну логіку в єдиній кодовій базі;

- TypeScript - для забезпечення типобезпеки, зменшення кількості помилок та підвищення якості коду;

- React - як бібліотека побудови компонентного інтерфейсу користувача з підтримкою серверних та клієнтських компонентів;

- Prisma ORM - для типобезпечної роботи з реляційною базою даних, автоматичної генерації міграцій та зручного CRUD API;

- SQLite - як легка вбудована СУБД для зберігання даних без необхідності окремого серверного процесу;

- Tailwind CSS - для утилітарної стилізації компонентів без написання окремих CSS-файлів;

– shadcn/ui - як бібліотека готових UI-компонентів з повним контролем над кодом;

- NextAuth.js - для реалізації автентифікації та авторизації користувачів на основі Credentials Provider [16] та JWT-сесій;

- Google Gemini API - для інтеграції штучного інтелекту та автоматизованої генерації тестових завдань;

- Lucide React - для сучасних SVG-іконок у інтерфейсі.

Обраний стек забезпечує повний цикл розробки - від серверної логіки до клієнтського інтерфейсу - в єдиній кодовій базі на TypeScript, що суттєво підвищує продуктивність розробки та зменшує кількість потенційних помилок на стику технологій. Використання виключно безкоштовних або відкритих програмних засобів мінімізує фінансові витрати на розробку.

Висновки за розділом 1

У першому розділі проведено комплексний аналіз предметної області та існуючих рішень у сфері електронного тестування. За результатами аналізу зроблено такі висновки:

1. Автоматизація процесу створення тестових завдань є актуальним завданням, яке дозволяє суттєво скоротити часові витрати викладача та підвищити якість освітнього процесу. Скорочення оцінюється на рівні 80%: сама ШІ-генерація тесту з 20 - 30 питань займає 10 - 30 секунд, проте повний цикл підготовки включає також рецензування згенерованих питань, редагування

формулювань та перевірку відповідності навчальній програмі, що в сукупності потребує орієнтовно 1 - 1,5 години замість 4 - 8 годин при повністю ручному створенні.

2. Порівняльний аналіз існуючих платформ виявив їхні спільні недоліки: відсутність AI-генерації тестів, обмежену підтримку масового імпорту питань, недостатню гнучкість структурування контенту та обмежені засоби аналітики й рейтингування.

3. Для розробки вебзастосунка обрано сучасний технологічний стек на основі Next.js, TypeScript, Prisma ORM та Google Gemini API. Такий вибір забезпечує повний цикл розробки в єдиній кодовій базі, типобезпечність та можливість інтеграції сучасних ШІ-інструментів.

4. Обраний технологічний стек дозволяє реалізувати всі заявлені функціональні вимоги до застосунка та усунути виявлені недоліки існуючих рішень.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Загальна архітектура вебзастосунка

Проєктування архітектури є ключовим етапом розробки будь-якої інформаційної системи, оскільки архітектурні рішення, прийняті на цьому етапі, визначають масштабованість, надійність та зручність подальшої підтримки застосунка. При виборі архітектурного підходу для вебзастосунка враховувались такі вимоги: єдина кодова база для клієнтської та серверної частин, типобезпечність на всіх рівнях, простота розгортання та мінімізація операційних витрат.

Архітектура розробленого вебзастосунка базується на концепції повностекового фреймворка Next.js з використанням нового підходу App Router. Ця архітектура реалізує модель React Server Components (RSC), яка дозволяє виконувати рендеринг компонентів безпосередньо на сервері, мінімізуючи обсяг JavaScript-коду, що передається клієнту, та суттєво покращуючи початкову швидкість завантаження сторінок [10].

Застосунок побудовано за трирівневою архітектурою:

1. Рівень представлення - React-компоненти, які відповідають за відображення інтерфейсу користувача. Використано як серверні компоненти (Server Components, виконуються на сервері та не потребують передачі JavaScript клієнту), так і клієнтські компоненти (з директивою «use client») залежно від потреби в інтерактивності та доступу до браузерних API.

2. Рівень бізнес-логіки - Server Actions та серверні функції, що реалізують основну логіку застосунка: створення курсів, перевірку вхідних даних, обробку відповідей на тести, розрахунок оцінок та генерацію тестів засобами ШІ. Server Actions - це асинхронні серверні функції, що викликаються безпосередньо з клієнтських компонентів, забезпечуючи безпечну взаємодію між клієнтом і сервером без необхідності створення окремих мережевих маршрутів вручну.

3. Рівень доступу до даних - Prisma Client, який забезпечує типобезпечний доступ до бази даних SQLite через ORM-інтерфейс з автоматично згенерованими типами на основі схеми бази даних.

Маршрутизація застосунка організована за файловою системою, де кожна директорія у папці «app/» відповідає маршруту URL. Застосовано групування маршрутів для розмежування функціоналу за ролями:

– «/auth» - сторінки автентифікації (вхід, реєстрація);

– «/teacher/courses» - панель управління курсами для викладача;

– «/teacher/courses/[id]» - редактор контенту курсу;

– «/course/new» - форма створення нового курсу;

– «/course/[id]» - сторінка проходження курсу для учня;

– «/quiz/[id]» - інтерфейс проходження тесту.

Безпека застосунку та розмежування прав доступу забезпечуються перевірками безпосередньо на рівні серверних функцій та маршрутизації. Перед виконанням будь-якої операції з базою даних система перевіряє наявність активної сесії та роль користувача - наприклад, студент не може отримати доступ до функціоналу викладача. Це унеможливлює несанкціонований доступ до даних навіть у разі прямого звернення до серверних функцій в обхід інтерфейсу.

Ключовою перевагою обраної архітектури є єдина кодова база для серверної та клієнтської частин на TypeScript. Це спрощує підтримку коду, зменшує кількість потенційних помилок та підвищує загальну надійність системи.

2.2. Проєктування моделі бази даних

Модель даних вебзастосунка спроєктовано на основі реляційного підходу та реалізовано із застосуванням ORM Prisma і СУБД SQLite. Основною метою проєктування було забезпечення:

- підтримки ієрархічної структури навчального контенту (курс → теми → квізи → питання → варіанти відповідей);

- коректного розмежування ролей користувачів (викладач/студент) і зв'язування даних із конкретним автором або виконавцем;

- можливості безпечного зарахування студентів на курси та фіксації їх прогресу;

- цілісності даних за рахунок зовнішніх ключів, обмежень NOT NULL та каскадних правил видалення.

Основні сутності та їх призначення

У межах системи визначено такі базові сутності.

User - обліковий запис користувача. Сутність містить ідентифікаційні дані (зокрема, email), атрибути профілю та поле role, що задає роль у системі (TEACHER або STUDENT). Через зв'язки User забезпечується авторство курсів, участь у курсах через зарахування та належність результатів тестів конкретному студенту.

Course - навчальний курс, базова одиниця контенту верхнього рівня. Курс містить загальні метадані (назва, опис, зображення), час створення та посилання на автора (creatorId). Для реалізації закритого доступу передбачено код зарахування. Курс об'єднує теми та зберігає інформацію про зарахування студентів.

Topic - тема курсу. Сутність є другим рівнем ієрархії та прив'язана до конкретного курсу через courseId. Тема дозволяє логічно групувати навчальні квізи за розділами та підтемами.

Quiz - квіз (тест). Сутність прив'язана до теми через topicId і містить метадані (назва, опис), параметри проходження (дедлайн, максимальна кількість спроб) та зв'язки з питаннями й результатами. Додатково зберігається ідентифікатор автора (creatorId).

Question - питання квізу. Питання належить до одного квізу (quizId) і містить текст, тип (одиночний або множинний вибір) та атрибут bloomLevel для позначення когнітивного рівня за таксономією Блума.

Option - варіант відповіді. Сутність належить до одного питання (questionId) та містить текст варіанту й ознаку правильності isCorrect, яка використовується для автоматичної перевірки.

Enrollment - зарахування на курс. Проміжна сутність, що реалізує зв'язок «багато-до-багатьох» між студентами (User) і курсами (Course). Включає зовнішні ключі userId та courseId, а також часову позначку joinedAt.

QuizResult - результат проходження тесту. Сутність фіксує підсумок спроби: відсоток правильних відповідей (score), кількість правильних відповідей (correctAnswers), загальну кількість питань (totalQuestions), час проходження (createdAt) та зв'язки зі студентом і квізом. На основі цих даних формується статистика прогресу та рейтинг.

Зв'язки між сутностями

Зв'язки між сутностями відповідають ієрархічній структурі навчального контенту: User створює Course, курс містить теми, теми - квізи, квізи - питання з варіантами відповідей. Участь студентів у курсах фіксується через проміжну сутність Enrollment, а результати проходження тестів - через QuizResult. Усі дочірні сутності підтримують каскадне видалення при видаленні батьківського запису.

Забезпечення цілісності даних

Для зменшення ризику появи некоректних даних у моделі передбачено первинні ключі для кожної сутності, зовнішні ключі для фіксації залежностей, обов'язкові поля для критичних атрибутів, а також каскадні правила видалення для ієрархічного контенту — при видаленні курсу автоматично видаляються всі пов'язані теми, квізи, питання, варіанти, записи зарахувань і результатів. Це спрощує адміністрування та запобігає записам без батьківського об'єкта.

Окрему увагу приділено сутності Enrollment як точці контролю доступу: наявність відповідного запису дозволяє визначити, чи має студент право переглядати матеріали курсу та проходити квізи.

Повну структуру бази даних та зв'язки між сутностями наведено на рисунку 2.1.

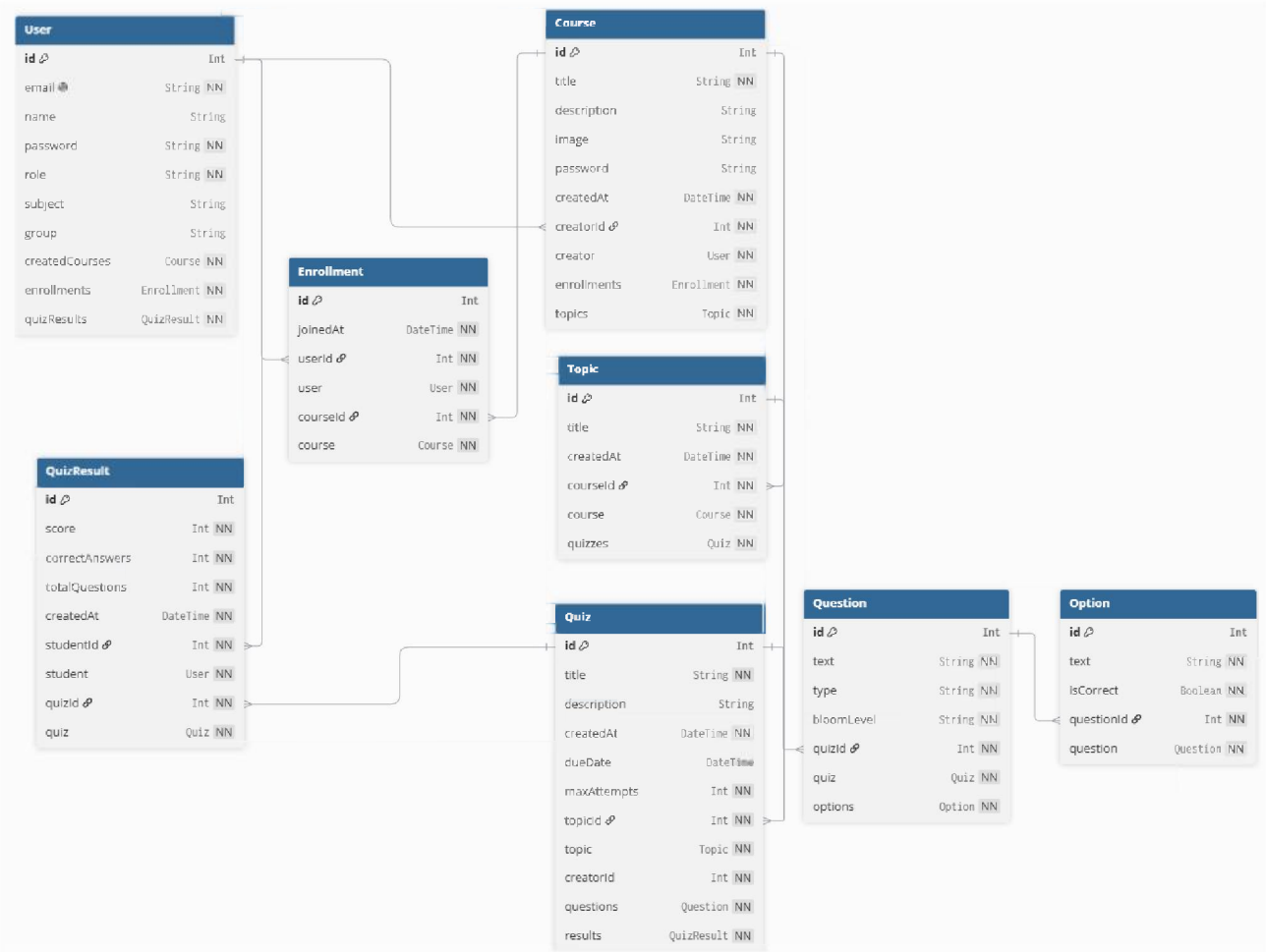


Рисунок 2.1 - ER-діаграма бази даних QuizApp.

2.3. Модель авторизації та розмежування ролей

Система авторизації є критично важливою складовою будь-якого вебзастосунка, що обробляє персональні дані та розмежовує доступ до функціоналу. В QuizApp реалізовано повноцінну власну систему автентифікації та авторизації з використанням безпечних серверних сесій (HTTP-only cookies).

Процес реєстрації та автентифікації включає такі етапи:

1. Користувач заповнює форму реєстрації, вказуючи ім'я, електронну пошту, пароль та роль (TEACHER або STUDENT). Студенти додатково вказують навчальну групу. Слід зазначити, що в поточній реалізації вибір ролі здійснюється самим користувачем, що є спрощенням прототипу. У подальших версіях системи доцільно реалізувати верифікацію ролі викладача (наприклад, через код-запрошення від адміністратора або підтвердження електронною поштою інституційного домену), щоб унеможливити несанкціоноване отримання привілейованого доступу студентами.

2. Server Action виконує перевірку введених даних: перевірка формату email, мінімальна довжина пароля, перевірка унікальності email у базі даних.

3. Пароль хешується за допомогою алгоритму bcrypt з коефіцієнтом складності 12 («saltRounds = 12»). Алгоритм bcrypt також характеризується високою стійкістю до брут-форс атак завдяки навмисно сповільненій функції хешування. Це унеможливлює відновлення оригінального пароля навіть у разі витоку бази даних.

4. Створюється запис User у базі даних із хешованим паролем.

5. При наступному вході система перевіряє введений пароль через «bcrypt.compare()» та у разі успіху створює безпечну серверну сесію. Ідентифікатор користувача зберігається у зашифрованих HTTP-only cookies, що захищає систему від XSS-атак, а інформація про роль («role») динамічно перевіряється в базі даних при кожному захищеному запиті.

Модель розмежування доступу (RBAC - Role-Based Access Control) передбачає дві ролі з чітко визначеними правами:

– TEACHER (Викладач) - має доступ до: панелі управління курсами («/teacher/courses»), створення та редагування курсів і контенту, перегляду списку зарахованих студентів, відрахування студентів, перегляду статистики по курсах.

– STUDENT (Студент) - має доступ до: каталогу курсів, зарахування на курси за кодом доступу, перегляду матеріалів та проходження тестів зарахованих курсів, перегляду власного прогресу та рейтингу.

Безпека застосунку та розмежування прав доступу забезпечуються на двох взаємопов'язаних рівнях. **Перший** рівень захисту діє на етапі рендерингу сторінок (компонентний рівень). Під час спроби доступу до захищених розділів (наприклад, панелі керування курсами) система перевіряє наявність сесійного ідентифікатора у файлах cookie та поточну роль користувача. Якщо сесія відсутня або користувач із роллю STUDENT намагається відкрити маршрут викладача, виконується автоматичне перенаправлення на головну сторінку або сторінку авторизації. Водночас викладач має право редагувати та видаляти лише власні курси; доступ до контенту інших викладачів обмежений режимом перегляду.

Другий, основний рівень захисту реалізований безпосередньо всередині кожної серверної функції (Server Action). Перед виконанням будь-якої операції з даними сервер незалежно зчитує ідентифікатор сесії, виконує запит до бази даних для перевірки статусу користувача та перевіряє наявність у нього відповідних прав доступу (наприклад, ролі TEACHER для створення курсу). Цей рівень є критично важливим, оскільки він повністю унеможливлює несанкціонований доступ до даних шляхом прямого звернення до серверних функцій в обхід користувацького інтерфейсу.

Парольний захист курсів є додатковим механізмом контролю доступу: при зарахуванні студент повинен ввести код доступу, встановлений викладачем для конкретного курсу. Перевірка коду виконується на сервері перед створенням запису про зарахування. У разі невірного пароля запис не створюється, а користувач отримує повідомлення про помилку.

2.4. Проєктування API та серверних дій

Основним способом обміну даними між інтерфейсом користувача (клієнтом) та сервером у розробленому вебзастосунку є механізм Server Actions.

Це сучасний інструмент фреймворку Next.js, який дозволяє безпечно викликати серверні функції безпосередньо з компонентів сторінки або вебформ.

Головна перевага цього підходу полягає у його простоті та ефективності. На відміну від класичного підходу (REST API), розробнику не потрібно вручну створювати окремі мережеві адреси для запитів, налаштовувати заголовки чи перетворювати формати даних. Усі зазначені технічні процеси взаємодії абстрагуються фреймворком Next.js і виконуються автоматично у фоновому режимі.

Завдяки цьому вся ключова бізнес-логіка системи - від базового збереження курсів у базу даних до складних запитів до штучного інтелекту Google Gemini API - реалізована через єдиний та зрозумілий механізм Server Actions. Таке рішення робить архітектуру програми більш цілісною, безпечною та легкою для подальшої підтримки.

Для забезпечення модульності та зручності підтримки коду, усі серверні дії логічно згруповано за функціональними доменами:

- Управління курсами та контентом: забезпечення повного циклу операцій (CRUD) для викладачів щодо створення та ієрархічного структурування матеріалів (теми, квізи), а також механізми пошуку та безпечного зарахування для студентів.
- Обробка тестових завдань: реалізація алгоритмів генерації тестів (через ШІ-модель, аналіз тексту або JSON-імпорт), а також збереження результатів проходження з автоматичним підрахунком балів.
- Робота з користувачами та аналітика: процеси реєстрації облікових записів, формування індивідуальної статистики прогресу та агрегація даних для загальної таблиці лідерів.

Виконання кожної серверної дії відбувається за єдиною стандартизованою схемою. Схему виконання серверної дії наведено на рисунку 2.2.

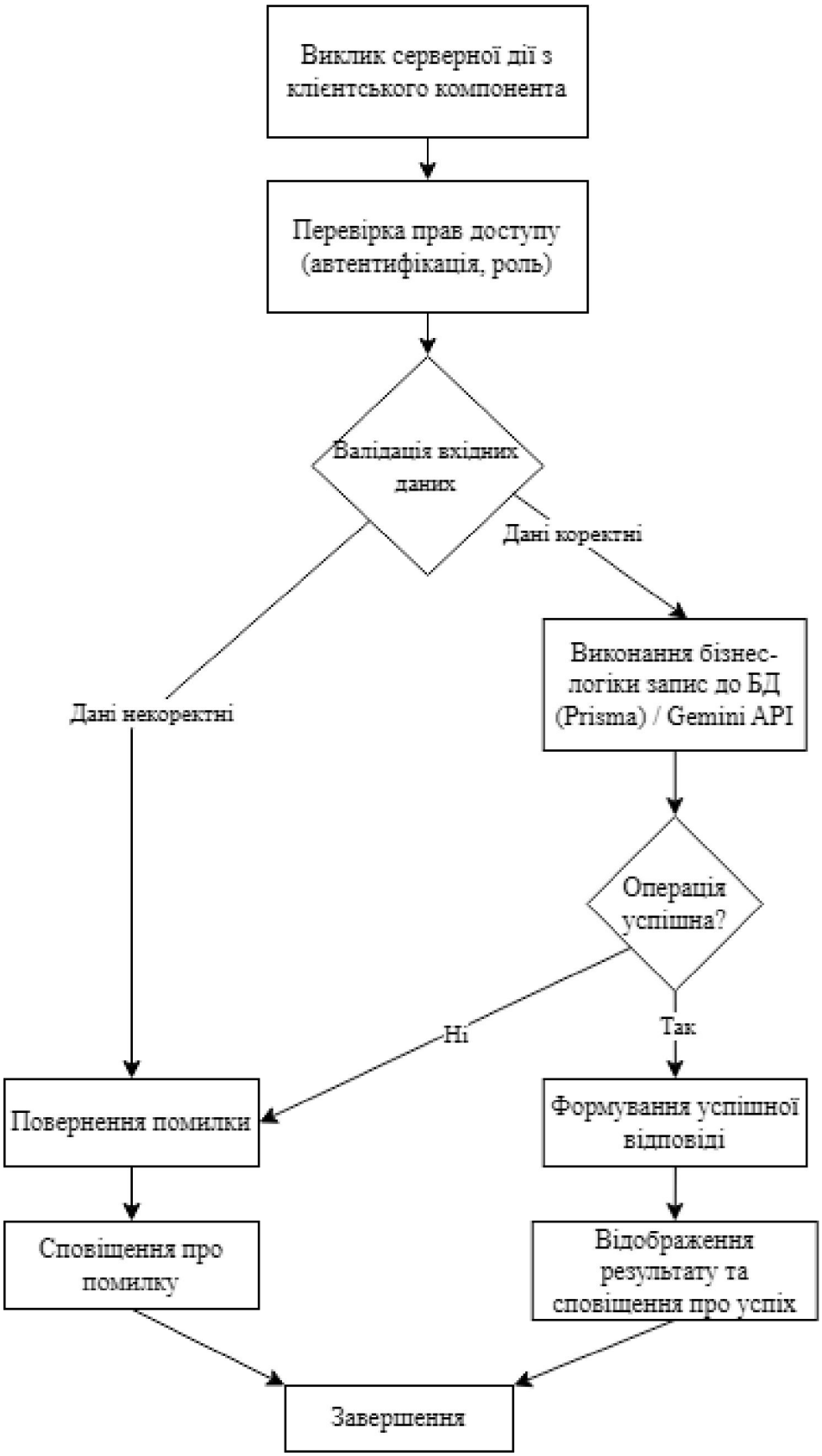


Рисунок 2.2 - Схема взаємодії клієнта та Server Actions.

Перевірка вхідних даних є обов'язковим етапом у кожній серверній дії. Вхідні дані перевіряються на наявність усіх обов'язкових полів, коректність форматів (електронна пошта, URL), а також відповідність правилам роботи застосунку - наприклад, кількість питань у тесті має бути від 5 до 40.

При імпорті з JSON-файлу додатково перевіряється структура даних: наявність поля назви тесту, масиву питань, а для кожного питання - тексту, когнітивного рівня за таксономією Блума та масиву варіантів відповідей, серед яких щонайменше один має бути позначений як правильний.

Обробка помилок реалізована на двох взаємопов'язаних рівнях. На рівні серверних дій використовується механізм перехоплення винятків із поверненням структурованого об'єкта відповіді, який містить ознаку успішності операції, повідомлення про помилку та дані результату. На рівні клієнтських компонентів отримана відповідь асинхронно обробляється внутрішніми механізмами управління станом, після чого користувач отримує сповіщення про результат виконаної операції у вигляді спливаючого повідомлення (засобами бібліотеки Sonner).

Оптимізація запитів до бази даних досягається завдяки вибірковому завантаженню лише необхідних полів та пов'язаних даних засобами Prisma Client. Це дозволяє уникнути надмірної кількості запитів до бази даних та зменшити обсяг даних, що передаються між рівнями застосунку. Наприклад, при завантаженні каталогу курсів для студента запит містить лише ім'я та електронну пошту автора, а також статус зарахування поточного користувача - без завантаження повного вмісту курсу.

Кешування даних реалізовано на рівні Next.js: однакові запити до бази даних у межах одного циклу обробки сторінки виконуються лише один раз. Для сторінок, де важлива актуальність даних - таких як каталог курсів та результати тестів - кешування вимкнено, що забезпечує отримання оновлених даних при кожному зверненні до сервера.

Висновки за розділом 2

У другому розділі спроєктовано архітектуру вебзастосунка QuizApp та визначено ключові структурні рішення системи. За результатами проєктування зроблено такі висновки:

1. Обрана трирівнева архітектура на основі Next.js App Router з моделлю React Server Components забезпечує чітке розмежування відповідальності між рівнями представлення, бізнес-логіки та доступу до даних, мінімізує обсяг JavaScript-коду на клієнті та спрощує підтримку застосунка.

2. Спроєктована реляційна модель бази даних з восьми взаємопов'язаних сутностей (User, Course, Topic, Quiz, Question, Option, Enrollment, QuizResult) повністю відображає ієрархічну структуру навчального контенту та забезпечує цілісність даних через зовнішні ключі та каскадне видалення.

3. Реалізована дворівнева система безпеки (захист клієнтських компонентів та серверних функцій) у поєднанні з RBAC-моделлю та парольним захистом курсів забезпечує надійне розмежування доступу між ролями TEACHER та STUDENT і унеможливлює несанкціонований доступ до даних.

4. Використання Server Actions замість традиційного REST API суттєво спрощує архітектуру застосунка, забезпечує надійність взаємодії між клієнтом і сервером та усуває необхідність у ручному написанні мережевих маршрутів, що підвищує продуктивність розробки.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ВЕБЗАСТОСУНКА

3.1. Реалізація функціоналу викладача

Функціонал викладача є ключовою частиною системи, яка забезпечує повний цикл створення та управління навчальним контентом. Відповідно до вимог технічного завдання реалізовано три основних блоки: керування курсами, редактор контенту та створення тестів.

Керування курсами. Після входу в систему викладач потрапляє на панель управління («/teacher/courses»), де відображається перелік усіх створених ним курсів у вигляді карток. Кожна картка містить обкладинку курсу, назву, кількість зарахованих студентів та набір дій: перехід до редактора, перегляд студентів, видалення.

Для кожного курсу реалізовано повний набір CRUD-операцій:

– Створення (Create) - окрема форма за маршрутом «/course/new» з перевіркою полів: назва (обов'язкове), опис, URL обкладинки, код доступу для студентів. Server Action «createCourse» приймає дані форми, виконує серверну валідацію та створює запис у базі даних із прив'язкою до ідентифікатора авторизованого викладача.

– Читання (Read) - завантаження списку курсів із фільтрацією за «authorId» поточного користувача. Використовується Server Component для прямого звернення до бази даних без додаткових проміжних запитів.

– Оновлення (Update) - модальне вікно (компонент «Dialog» з shadcn/ui) дозволяє змінити назву та код доступу до курсу без перезавантаження сторінки.

– Видалення (Delete) - каскадне видалення курсу з усіма пов'язаними даними: темами, квізами, питаннями, варіантами відповідей, записами зарахування та результатами тестів. Операція виконується в транзакції Prisma для гарантії атомарності.

Керування студентами. Викладач може переглядати список зарахованих студентів для кожного курсу. Інформація включає ім'я, навчальну групу та дату зарахування. Реалізовано можливість відрахування студента з курсу (видалення запису «Enrollment»), що автоматично скасовує доступ до матеріалів курсу та тестів.

Редактор контенту курсу. Сторінка «/teacher/courses/[id]» надає викладачу інтерфейс для структурування навчального контенту відповідно до ієрархії «Курс → Теми → Квізи». Викладач може створювати теми («Topic») для логічного розподілу матеріалу - кожна тема отримує власну назву і відображається як окрема секція. Усередині кожної теми розміщуються квізи.

Створення квізів реалізовано трьома методами, доступ до яких надається через модальне вікно з візуальними картками вибору:

1. Текст у Тест - викладач вставляє довільний текст (лекцію, статтю, навчальний матеріал) у текстове поле. Серверна функція передає текст до Google Gemini API, який автоматично аналізує зміст і генерує тестові питання на основі ключових понять матеріалу. Метод дозволяє швидко перетворити наявний навчальний контент на готовий квіз без ручного формулювання питань.

2. JSON-імпорт - викладач вставляє JSON-рядок зі структурованими даними тесту у текстове поле. Спеціальна серверна функція виконує парсинг JSON, валідацію структури (перевірка наявності полів «title», «questions», для кожного питання - «text», «bloomLevel», масиву «options» з принаймні одним «isCorrect: true») та атомарне створення квізу з усіма питаннями та варіантами відповідей одним вкладеним запитом до бази.

3. Генерація за Блумом - викладач вказує тему тесту та точну кількість питань для кожного з 6 когнітивних рівнів Таксономії Блума (Знання, Розуміння, Застосування, Аналіз, Оцінка, Створення), після чого система автоматично генерує збалансований тест за допомогою інтеграції з Google Gemini API. Детальний опис механізму ШІ-генерації наведено у розділі 4.

Після успішного створення, квіз відображається у відповідній темі, а викладач отримує спливаюче повідомлення про успішне створення квізу.

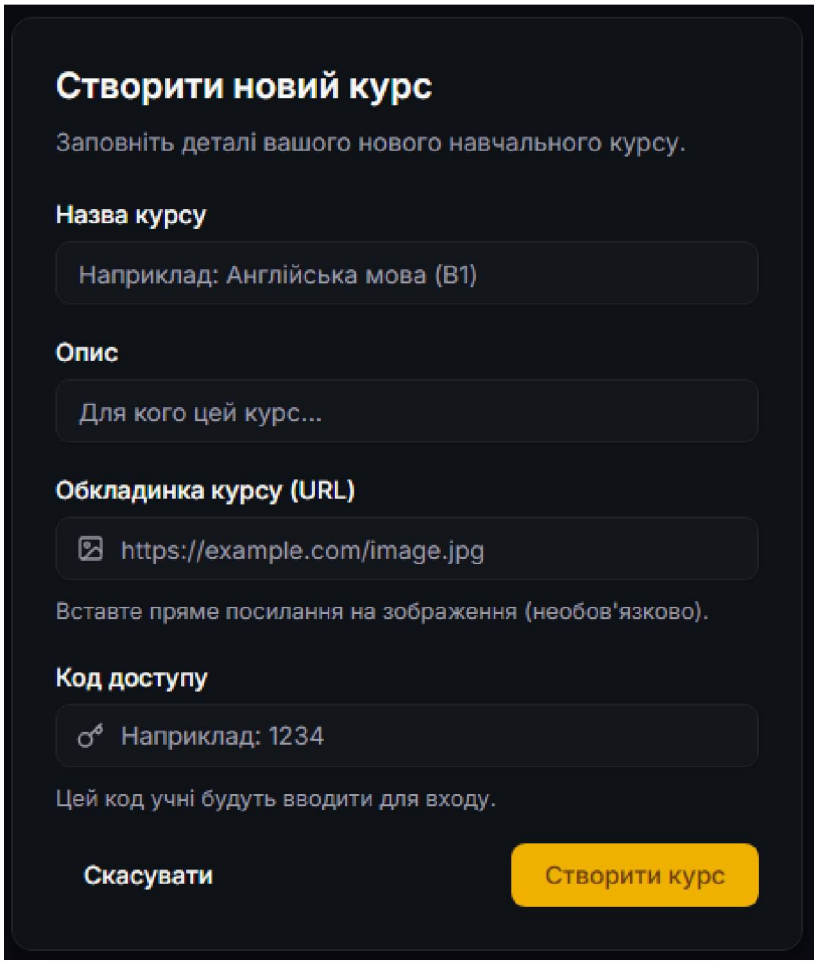


Рисунок 3.1 - Інтерфейс створення курсу (роль TEACHER).



Рисунок 3.2 - Модальне вікно вибору методу створення квізу.

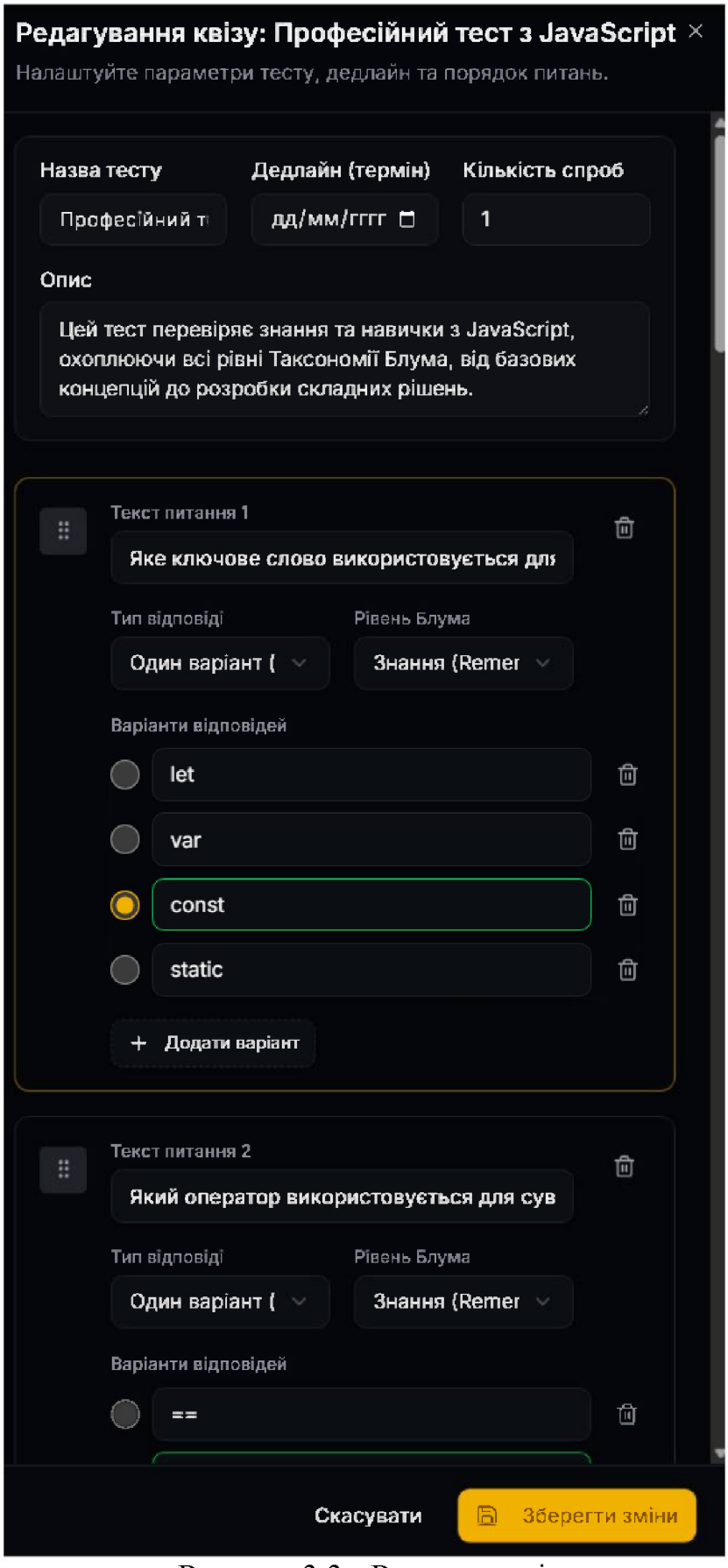


Рисунок 3.3 - Редактор квізу.

Таблиця 3.1 узагальнює реалізований функціонал викладача та відповідні технічні механізми.

Таблиця 3.1

Функціонал викладача та реалізовані механізми

Функція	Маршрут / компонент	Технічний механізм
Перегляд курсів	«/teacher/courses»	Server Component + Prisma «findMany»
Створення курсу	«/course/new»	Server Action «createCourse»
Редагування курсу	«Dialog» (модальне вікно)	Server Action «updateCourse»
Видалення курсу	«AlertDialog»	Server Action «deleteCourse» + транзакція
Управління темами	«/teacher/courses/[id]»	Server Actions «createTopic», «deleteTopic»
JSON-імпорт квізу	«Dialog» → textarea	Server Action «importQuizFromJSON»
AI-генерація квізу	«Dialog» → форма	Server Action «generateQuizWithAI» + Gemini API
Перегляд студентів	«/teacher/courses/[id]/students»	Server Component + Prisma «include: Enrollment»
Відрахування студента	Кнопка у списку	Server Action «unenrollStudent»

3.2. Реалізація функціоналу учня

Функціонал учня спрямовано на забезпечення зручного та інтуїтивного доступу до навчальних матеріалів, проходження тестів та відстеження власного прогресу. Усі сторінки для студента захищені на рівні серверних компонентів та функцій маршрутизації: спроба незареєстрованого або незарахованого на курс користувача отримати доступ до захищених навчальних матеріалів призводить до автоматичного перенаправлення на відповідну сторінку доступу.

Каталог курсів. Головна сторінка для студента відображає каталог усіх доступних курсів. Реалізовано функцію «живого» пошуку за назвою курсу або іменем викладача. Пошук здійснюється на стороні клієнта без перезавантаження сторінки: під час введення тексту внутрішній стан компонента миттєво

оновлюється, а відфільтрований масив курсів одразу відображається на екрані, забезпечуючи зручний інтерфейс без затримок мережевих запитів.

Кожен курс відображається у вигляді картки з обкладинкою, назвою, описом та ім'ям автора. Візуальний стан картки залежить від статусу зарахування поточного студента:

- Незарахований - відображається кнопка «Записатися». При натисканні відкривається діалогове вікно з полем введення коду доступу.

- Зарахований - картка позначається зеленою іконкою. Кнопка змінюється на «Перейти до навчання» з посиланням на сторінку курсу.

Зарахування на курс. Процес зарахування реалізовано через захищену серверну дію. Алгоритм передбачає пошук відповідного курсу в базі даних та криптографічне порівняння введеного студентом коду з хешованим еталоном. У разі успішної перевірки система створює новий запис про зарахування та надає доступ до матеріалів. Якщо ж введено хибний код, транзакція скасовується, а студент отримує інформативне спливаюче сповіщення про помилку.

Інтерфейс проходження курсу. Сторінка курсу реалізована за принципом інтерактивного навчального плеєра. Компоновка сторінки включає:

- Інформаційний блок - містить обкладинку, назву, ім'я викладача та загальну статистику успішності: кількість пройдених тестів відносно загальної їх кількості, а також загальний прогрес у відсотках (обчислюється на основі агрегованих даних бази).

- Структура курсу - відображається у вигляді інтерактивного списку-акордеону: кожна тема розкривається при натисканні та демонструє вкладені тести з їхнім поточним статусом («не розпочато», «у процесі» або «завершено» з відповідним балом).

Інтерфейс проходження тесту. Сторінка конкретного тесту реалізує повноцінний інтерактивний модуль для перевірки знань. Поточний стан відповідей студента тимчасово зберігається у локальній пам'яті компонента до

моменту фінального відправлення на сервер. Інтерфейс сторінки містить такі елементи:

- Карта питань - візуальна навігаційна панель у вигляді нумерованих кнопок. Кольорова індикація стану: жовтий - поточне питання, сірий - не відвідане, зелений - відповідь надана.

- Блок питання відображає порядковий номер, текст завдання та варіанти відповідей. Тип вибору - одиночний або множинний - визначається автоматично на основі кількості правильних відповідей для кожного питання.

- Прогрес-бар - відображає відсоток відповідей, що оновлюється в реальному часі при виборі відповідей.

- Навігація - кнопки «Назад» та «Далі» для послідовного переходу між питаннями. Кнопка «Завершити тест» стає доступною після відповіді на всі питання.

- Навігація - кнопки «Назад» та «Далі» для послідовного переходу між питаннями. Кнопка «Завершити тест» стає доступною після відповіді на всі питання.

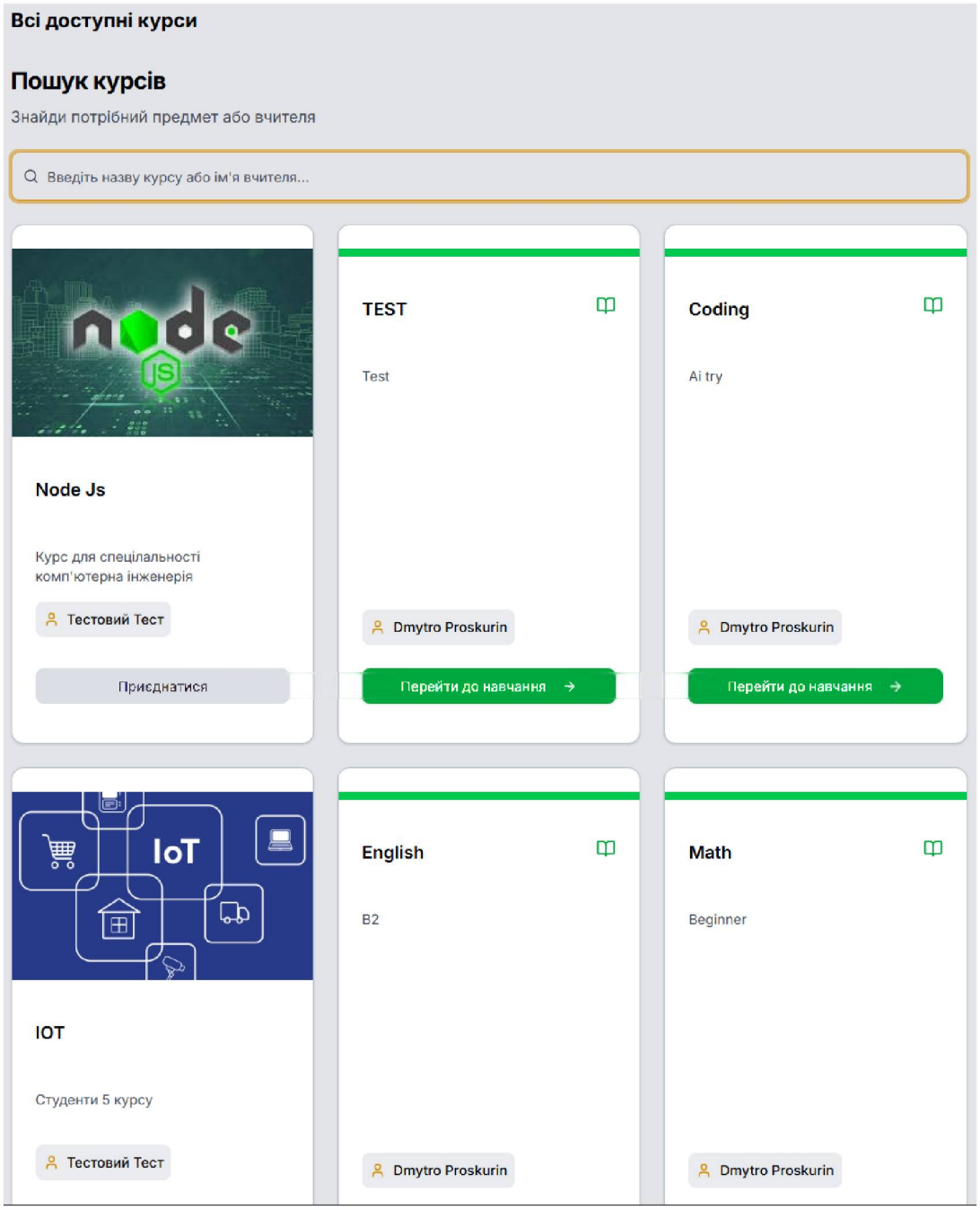


Рисунок 3.4 - Каталог курсів із живим пошуком (роль STUDENT).

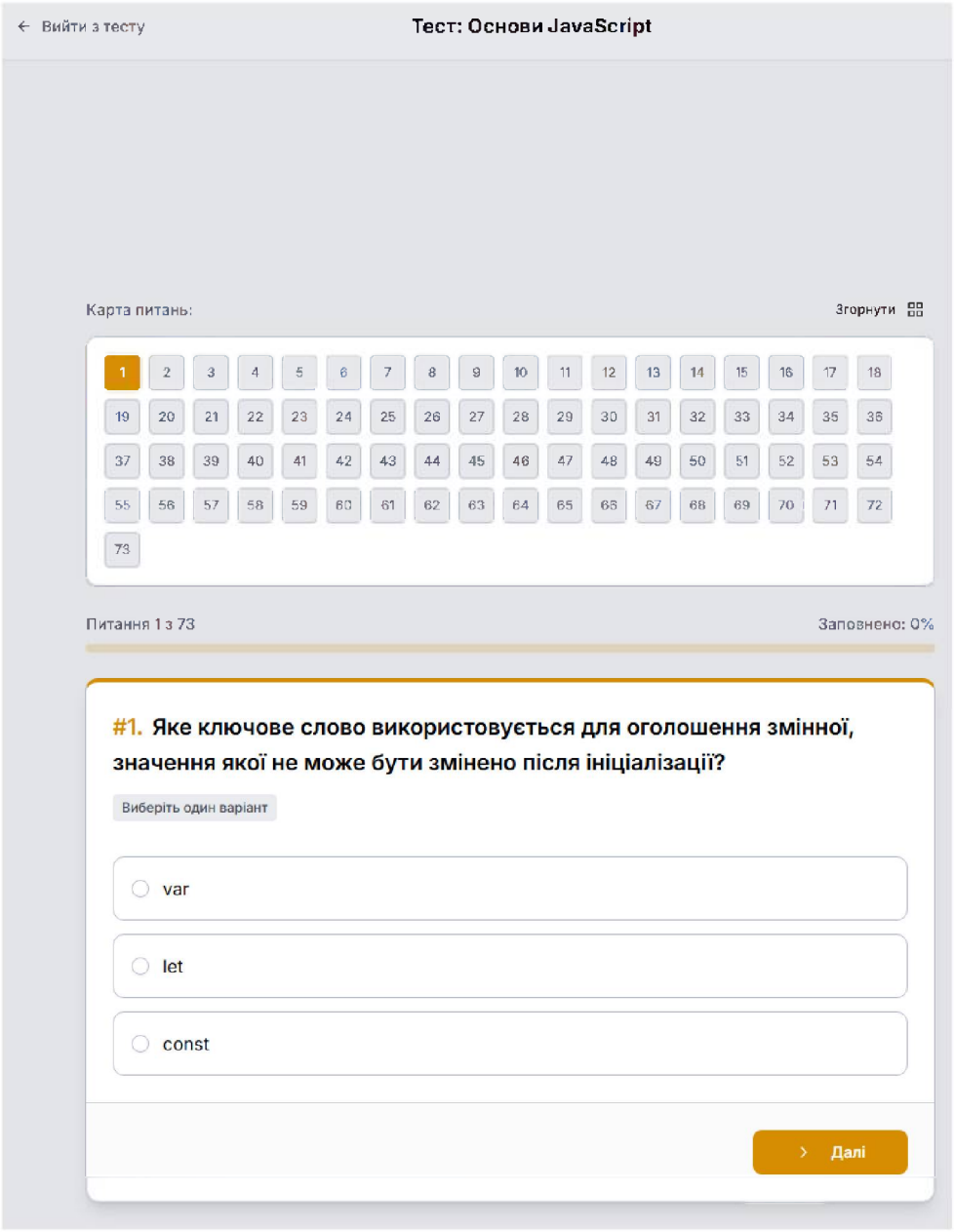


Рисунок 3.5 - Інтерфейс проходження тесту з картою питань.

Після натискання кнопки завершення Server Action «submitQuizResult» отримує об'єкт відповідей студента, порівнює їх з правильними варіантами («isCorrect = true») з бази даних та розраховує результат. Результат зберігається у моделі «QuizResult», після чого студент перенаправляється на сторінку результатів.

3.3. Система оцінювання та рейтингування студентів

Система оцінювання і рейтингування є важливою мотиваційною складовою освітнього процесу. В QuizApp реалізовано автоматичний підрахунок результатів, детальну статистику прогресу та таблицю лідерів.

Підрахунок результатів тесту. Після завершення тесту серверна функція завантажує всі питання квізу разом із правильними відповідями, після чого порівнює їх з відповідями студента. Для кожного питання перевіряється точний збіг обраних варіантів з правильними. На основі підрахованої кількості правильних відповідей обчислюється відсоток успішності за формулою: score = (correctAnswers / totalQuestions) × 100. Отриманий результат зберігається у базі даних із зазначенням ідентифікаторів студента та квізу, кількості правильних відповідей, загальної кількості питань та дати проходження, після чого повертається клієнту для відображення на сторінці підсумків.

Сторінка прогресу навчання. Окрема сторінка агрегує інформацію про прогрес студента по всіх курсах, на які він зарахований. Для кожного курсу відображається: загальна кількість квізів та кількість складених, прогрес-бар у відсотках, детальний список квізів із зазначенням статусу та результату кожного.

Таблиця лідерів (Leaderboard). Окрема сторінка відображає рейтинг серед усіх студентів платформи. Алгоритм формування рейтингу:

1. Для кожного студента збираються всі результати тестів студента.

2. Обчислюється середній відсоток успішності: «avgScore = сума score / кількість результатів».

3. Студенти ранжуються за «avgScore» у спадному порядку.

Візуалізація рейтингу включає два елементи:

– П'єдестал - виділений блок для трійки лідерів з аватарами, іменами та середнім відсотком успішності. Перше місце відображається на вищій позиції з золотим акцентом, друге та третє - зі срібним та бронзовим відповідно.

– Загальна таблиця - повний список учасників рейтингу з позиціями, іменами та середнім відсотком. Рядок поточного авторизованого студента виділяється кольором для зручної орієнтації.

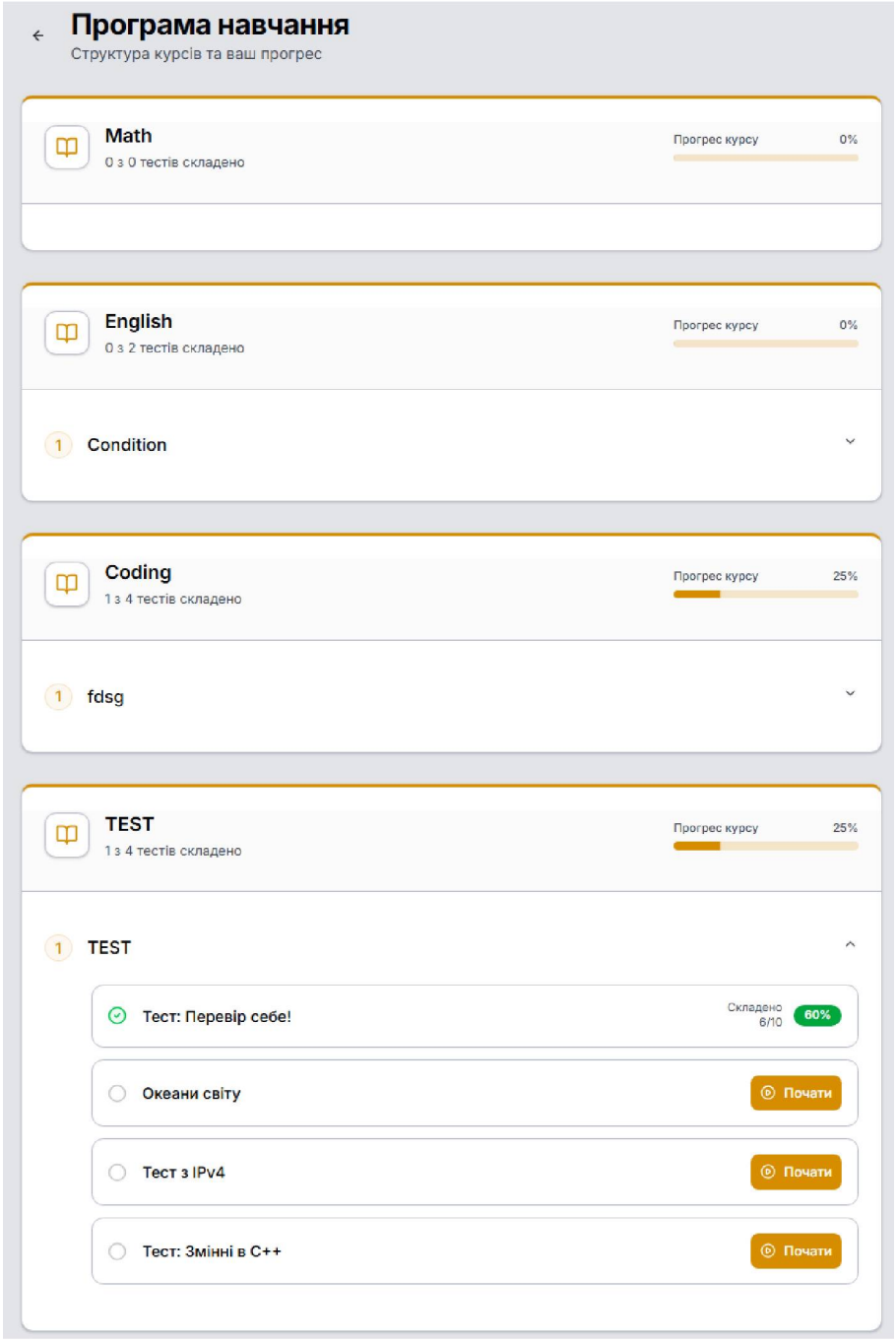


Рисунок 3.6 - Сторінка результатів тесту.

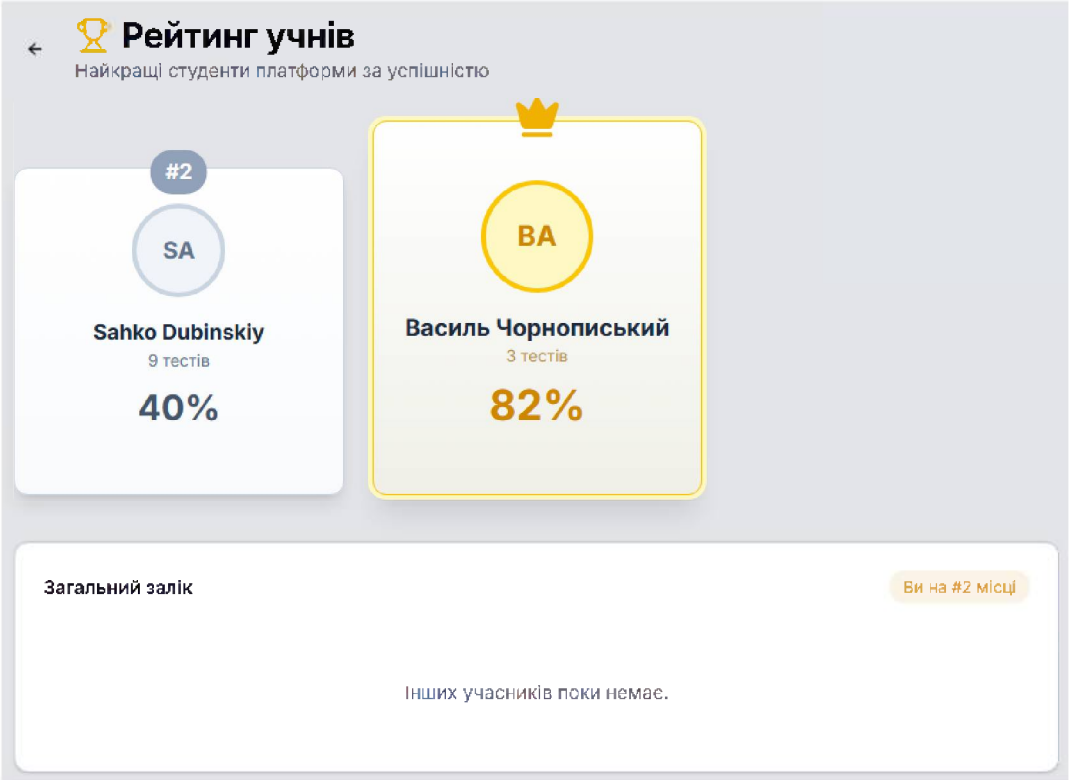


Рисунок 3.7 - Таблиця лідерів (Leaderboard).

Рейтингова система сприяє мотивації студентів через елементи ігрові елементи та здорової конкуренції, що є доведеним педагогічним інструментом підвищення залученості учнів до навчального процесу [20].

3.4. Інтерфейс користувача та бібліотека компонентів shadcn/ui

Якість інтерфейсу користувача безпосередньо впливає на зручність використання застосунка та рівень залученості кінцевих користувачів - викладачів і студентів. При розробці UI застосовано бібліотеку shadcn/ui як основу компонентної системи у поєднанні з утилітарним CSS-фреймворком Tailwind CSS [17].

shadcn/ui відрізняється від традиційних UI-бібліотек підходом до дистрибуції: компоненти не встановлюються як npm-залежність, а копіюються безпосередньо в кодову базу проєкту. Це забезпечує повний контроль над архітектурою та стилями: розробник може без обмежень модифікувати поведінку, зовнішній вигляд та доступність компонентів [13]. В основі

компонентів лежать примітиви Radix UI - бібліотеки доступних стилістично нейтральних елементів [18].

У застосунку QuizApp використано такі компоненти shadcn/ui:

- «Button» - уніфікована кнопка з різними візуальними варіантами та розмірами.

- «Card» - контейнер для карток курсів та квізів із секціями заголовка, контенту та підвалу.

- «Dialog» / «AlertDialog» - модальні вікна для форм створення та підтверджень видалення.

- «Accordion» - розкладні секції для відображення тем у курсі.

- «RadioGroup» / «Checkbox» - елементи для одиничного та множинного вибору варіантів відповідей у тесті.

- «Progress» - графічна шкала для відображення прогресу проходження тесту.

- «Avatar» - аватари користувачів у таблиці лідерів.

- «Badge» - візуальні мітки для маркування когнітивного рівня питань за Таксономією Блума (наприклад, «REMEMBER», «ANALYZE», «EVALUATE»).

- «Sonner» - спливаючі повідомлення для інформування про результат операцій.

Інтерфейс реалізовано як вебверсію з фіксованим мінімальним розміром області перегляду. Компоненти побудовані на адаптивних утилітах Tailwind CSS, що забезпечує їх коректне відображення на типових роздільній здатності екрана (від 1280×720 і вище).

Тематичне оформлення застосунка базується на CSS-змінних із підтримкою системної теми (світла/темна) через бібліотеку «next-themes». Кольорова схема визначається у глобальних стилях та автоматично поширюється на всі компоненти, що гарантує єдиний надлишкового вигляд у будь-якому режимі відображення.

Іконки у застосунку реалізовано через бібліотеку Lucide React - сучасний набір SVG-іконок з відкритим кодом [19]. Використано понад 20 різних іконок для візуального позначення дій та станів: додавання, видалення, налаштування, успішне зарахування, ШІ-генерація, рейтинг тощо.

Управління станом завантаження під час виконання асинхронних серверних дій (Server Actions) реалізовано за допомогою внутрішніх механізмів React. Під час очікування відповіді від сервера інтерактивні елементи тимчасово блокуються та відображають анімований індикатор процесу. Це інформує користувача про виконання операції та надійно запобігає дублюванню мережевих запитів.

Висновки за розділом 3

У третьому розділі описано повну реалізацію функціоналу вебзастосунка QuizApp. За результатами роботи зроблено такі висновки:

1. Реалізовано повний цикл функціоналу викладача: CRUD-операції з курсами, ієрархічне структурування контенту (Курс → Теми → Квізи), три методи створення тестів («Текст у тест», JSON-імпорт та «Генерація за Блумом») та управління списком студентів, що повністю відповідає вимогам технічного завдання.

2. Функціонал студента забезпечує зручний доступ до навчальних матеріалів: каталог курсів із динамічним пошуком у реальному часі, захищений механізм зарахування та повноцінний інтерактивний плеєр проходження тестів із картою питань, індикатором прогресу і автоматичним підрахунком результатів.

3. Система оцінювання автоматично аналізує та зберігає результати проходження тестів, а таблиця лідерів на основі середнього відсотка успішності забезпечує мотиваційний ефект через елементи ігор.

4. Використання shadcn/ui та Tailwind CSS дозволило створити консистентний, адаптивний та доступний інтерфейс користувача з підтримкою світлої та темної тем без написання зайвого коду стилів.

РОЗДІЛ 4
ІНТЕГРАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1. Автоматизована генерація тестових завдань через Gemini API

Інтеграція штучного інтелекту є ключовою інноваційною складовою розробленого вебзастосунка та головним чинником, що вирізняє QuizApp серед існуючих рішень. Автоматизована генерація тестових завдань, як зазначалось у вступі, дозволяє суттєво скоротити час підготовки навчальних матеріалів для викладача.

Вибір моделі. Для автоматизованої генерації тестових завдань використано Google Gemini API з моделлю **Gemini 2.5 Flash**. Вибір обумовлений такими факторами: висока якість генерації структурованого JSON-контенту, підтримка параметра «response_mime_type: «application/json» для фіксованого формату виводу, безкоштовний рівень API, достатній для навчального проєкту, а також провідні показники порівняльних тестів продуктивності з генерації освітнього контенту серед доступних мовних моделей [14].

Архітектура інтеграції. Взаємодія з мовною моделлю реалізована через серверну дію з використанням офіційного інструментарію розробника (SDK) від Google. Вибір саме цього архітектурного підходу замість побудови класичних мережевих запитів зумовлений двома вагомими факторами. По-перше, він забезпечує максимальний рівень безпеки: секретний ключ доступу до API зберігається виключно у захищеному серверному середовищі і за жодних обставин не потрапляє до клієнтського браузера. По-друге, гарантується надійність та цілісність передачі даних між інтерфейсом і сервером без необхідності написання додаткового коду для перетворення їх форматів.

Процес генерації тесту включає такі послідовні етапи:

1. Викладач у модальному вікні заповнює форму параметрів: тема тесту (текстове поле), детальний розподіл кількості питань за 6 когнітивними рівнями

Таксономії Блума (числові поля) та мова генерації. Загальна кількість питань підраховується системою динамічно.

2. Клієнтський компонент викликає серверну функцію, передаючи параметри.

3. Server Action формує системну інструкцію для мовної моделі з детальними інструкціями щодо формату відповіді та педагогічних вимог до якості питань.

4. Запит відправляється до API. Для забезпечення стійкості системи реалізовано механізм автоматичного перемикання між моделями: у разі недоступності основної моделі («gemini-2.5-flash») система послідовно звертається до резервних. Модель отримує чіткі інструкції щодо формату відповіді у вигляді JSON. Крім того, отриманий результат проходить через спеціальний алгоритм санітизації, який очищає відповідь від можливих побічних текстових артефактів (наприклад, Markdown-розмітки), залишаючи лише чисту структуру даних.

5. Отримана від моделі відповідь перевіряється на коректність: система переконується, що дані мають правильний формат, містять усі необхідні поля, а кожне запитання має хоча б одну правильну відповідь.

6. Якщо перевірка пройшла успішно, дані зберігаються у базі даних - квіз, запитання та відповіді записуються одночасно одним запитом, що виключає можливість часткового збереження. Якщо ж перевірку не пройдено (відсутня правильна відповідь, порожнє формулювання тощо), операція скасовується повністю, а користувач отримує повідомлення з переліком виявлених помилок.

7. Клієнтський інтерфейс отримує підтвердження від сервера та відображає візуальне сповіщення про успішне створення нового навчального матеріалу.

Формування запитів для штучного інтелекту. Якість згенерованих тестових завдань безпосередньо залежить від того, наскільки точно та грамотно сформульована базова інструкція для мовної моделі. Розроблена інструкція включає такі ключові компоненти:

- Формат відповіді - чіткий опис очікуваної структури даних у форматі JSON з прикладом, що містить назву тесту, масив питань із зазначенням тексту, когнітивного рівня та варіантів відповідей.

- Вимоги до питань - формулювання мають бути чіткими та однозначними, без підказок у тексті; питання повинні відповідати заявленій темі та когнітивному рівню за Таксономією Блума [15].

- Вимоги до варіантів відповідей - неправильні варіанти мають бути правдоподібними та концептуально близькими до правильного; типова кількість варіантів - чотири.

- Розподіл за когнітивними рівнями — інструкція містить чіткі вказівки щодо генерації питань за кожним із 6 рівнів: від відтворення фактів (Знання) до синтезу нових ідей (Створення).

- Мовні інструкції - явна вказівка мови генерації (українська або англійська) для всіх полів відповіді.

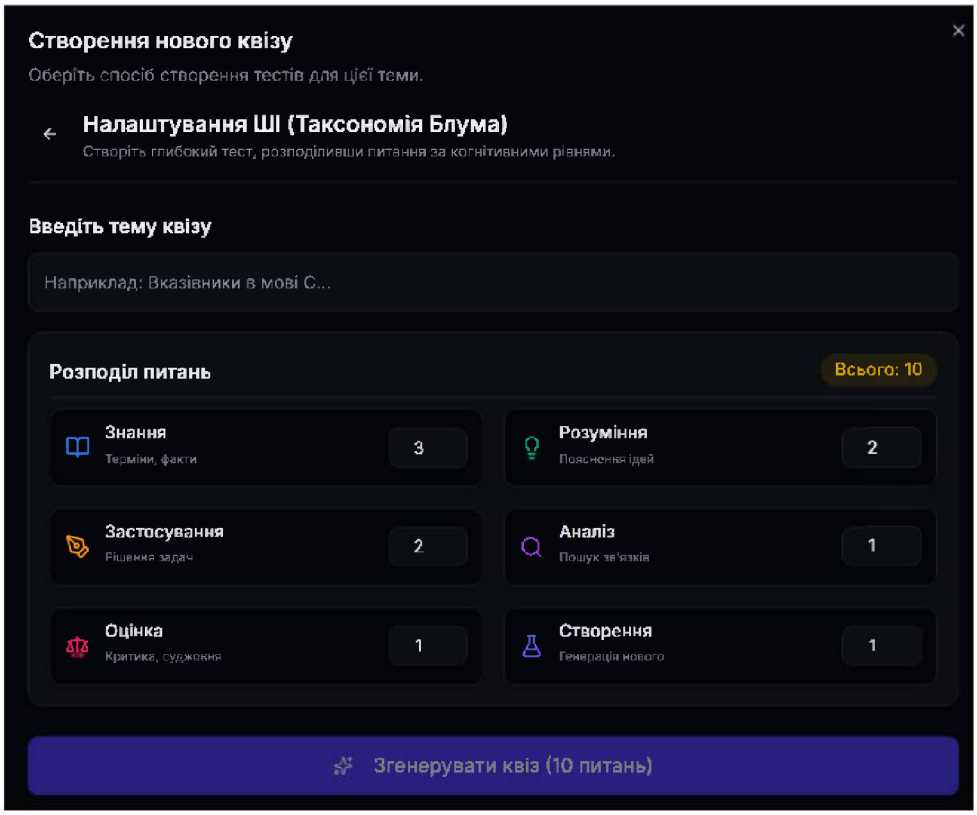


Рисунок 4.1 - Інтерфейс AI-генерації тесту з параметрами.

Обробка помилок реалізована з урахуванням специфіки роботи із зовнішніми API:

- Таймаут запиту (30 секунд) - якщо API не відповідає протягом встановленого часу, операція скасовується, а користувач отримує повідомлення про необхідність повторної спроби.

- Помилки мережі - автоматична повторна спроба при тимчасових мережевих помилках.

- Некоректна відповідь API - якщо згенерований JSON не відповідає очікуваній структурі, виконується повторний запит з розширеною інструкцією щодо структури JSON-відповіді, яка явно вказує на виявлену невідповідність формату.

- Недоступність API - користувач отримує рекомендацію скористатися альтернативним методом створення квізу (JSON-імпорт).

4.2. Параметризація генерації та оцінка якості ШІ-завдань

Параметризація генерації є ключовим механізмом, що дозволяє викладачу отримувати тестові завдання, максимально адаптовані до конкретних навчальних цілей. Система підтримує чотири незалежних параметри, які спільно визначають характеристики генерованого тесту.

Параметр «тема» є найважливішим для того, щоб згенеровані питання відповідали потрібному навчальному матеріалу. Викладач може вказати як загальну тематику (наприклад, «Алгоритми сортування»), так і конкретну підтему або навіть точне формулювання навчальної мети (наприклад, «Порівняння складності алгоритмів bubble sort та quick sort»). Чим точніше сформульована тема, тим більше згенеровані питання відповідають очікуваному змісту.

Параметр «розподіл питань» дозволяє викладачу вказати точну кількість завдань окремо для кожного з шести когнітивних рівнів Таксономії Блума. Загальна кількість питань підраховується системою динамічно на основі введених значень. Рекомендований максимум становить 40 питань за один запит,

що зумовлено технічними обмеженнями вікна контексту мовної моделі на одну генерацію. При потребі створення об'ємнішого тесту викладач може генерувати питання кількома ітераціями та додавати їх до одного квізу. Тест із 40 завдань збалансованого когнітивного навантаження студент проходить орієнтовно за 15-20 хвилин. На рівні серверу та клієнта обов'язково виконується перевірка вхідних даних: загальна сума питань має бути більшою за нуль, інакше операція генерації блокується з виведенням відповідного повідомлення про помилку.

Параметр «Таксономія Блума» замінює примітивне поняття «складності» та дозволяє розподілити питання за 6 рівнями когнітивної діяльності:

- Знання (REMEMBER) - перевірка пам'яті, фактів та термінів;
- Розуміння (UNDERSTAND) - пояснення ідей та класифікація;
- Застосування (APPLY) - використання знань та розв'язання задач;
- Аналіз (ANALYZE) - пошук зв'язків та деконструкція;
- Оцінка (EVALUATE) - критика, аргументація та судження;
- Створення (CREATE) - генерація нових рішень та синтез ідей.

Параметр «мова» дозволяє генерувати тестові завдання українською або англійською мовою. Це забезпечує можливість використання застосунка як у вітчизняних навчальних закладах, так і для підготовки до міжнародних сертифікаційних іспитів.

Оцінка якості генерації завдань штучним інтелектом. Для перевірки якості створених тестових завдань було проведено практичне тестування системи з використанням різних тем та налаштувань. Аналіз результатів виявив такі особливості:

– **Відповідність змісту** - згенеровані питання у переважній більшості випадків чітко відповідають заданій темі та обраному когнітивному рівню. Певні відхилення спостерігаються лише при дуже вузькоспеціалізованих темах, де штучний інтелект схильний формулювати надто узагальнені питання.

– **Якість хибних варіантів** - невірні відповіді формулюються як логічно обґрунтовані та концептуально близькі до правильного варіанта. Це підвищує об'єктивність перевірки знань та мінімізує ймовірність випадкового вгадування.

– **Мовна грамотність** - створення завдань як українською, так і англійською мовами демонструє високу граматичну та термінологічну точність завдяки здатності мовних моделей розуміти різні мови.

– **Технічна правильність** - завдяки чіткому опису необхідного формату (JSON) у базовій інструкції для ШІ та спеціальним налаштуванням підключення, технічні помилки у структурі отриманих даних трапляються вкрай рідко.

Таблиця 4.1 відображає результати тестування генерації за різними параметрами.

Таблиця 4.1

Оцінка якості AI-генерації за параметрами

Тема	Рівень Блума	К-сть питань	Релевантність	Структурна коректність	Час генерації
Алгоритми сортування	UNDERSTAND	10	100%	100%	~8 с
Бази даних SQL	APPLY	15	93%	100%	~12 с
Мережеві протоколи	ANALYZE	20	90%	100%	~18 с
Вебтехнології	REMEMBER	10	95%	100%	~9 с
Машинне навчання	CREATE	15	87%	100%	~26 с

Примітка: релевантність оцінювалась суб'єктивно за шкалою відповідності теми питань заявленій тематиці

Атрибут когнітивного навантаження у системі. Кожне питання в системі має атрибут «bloomLevel», який використовується при AI-генерації та ручному редагуванні. Цей рівень впливає на такі аспекти:

- Аналітика прогресу - статистика успішності може розбиватися за когнітивними рівнями, що дозволить викладачу глибоко оцінити, на якому етапі засвоєння матеріалу (наприклад, студент чудово пам'ятає теорію, але має прогалини у застосуванні) виникають труднощі.

- Адаптивне тестування - подальший розвиток системи передбачає автоматичне коригування рівня питань під час самопідготовки студента.

4.3. Тестування програмного забезпечення та аналіз результатів

На завершальному етапі розробки було проведено комплексне тестування вебзастосунка для підтвердження його надійності, безпеки та відповідності технічному завданню. Стратегія тестування включала два основні рівні: автоматизоване модульне тестування (Unit Testing) серверної логіки та функціональне тестування інтеграції з ШІ.

4.3.1. Автоматизоване модульне тестування

Для перевірки критично важливої логіки застосунку - автентифікації, розмежування прав доступу та перевірки вхідних даних - було використано фреймворк Jest. Щоб забезпечити незалежність тестів та виключити будь-які зміни у реальній базі даних, для клієнта Prisma та методів серверного рендерингу Next.js застосовувалися імітаційні об'єкти (mock-об'єкти).

У таблиці 4.2 наведено зведені результати виконання розроблених тестових сценаріїв.

Таблиця 4.2

Зведені результати модульного тестування (Unit Tests)

Назва модуля (Тестовий набір)	Кількість тестів	Опис перевірених сценаріїв	Статус проходження
auth.test.ts	2	Автентифікація: створення користувача з роллю за замовчуванням (STUDENT); коректна обробка та блокування реєстрації дублікатів e-mail.	Успішно (100%)

Продовження таблиці 4.2

rbac.test.ts	3	Авторизація та безпека: блокування неавторизованих сесій; перевірка обмеження прав (роль STUDENT не має доступу до створення курсів); успішна авторизація ролі TEACHER.	Успішно (100%)
scoring.test.ts	4	Математичний апарат: розрахунок відсотків успішності; коректність округлення; перевірка генерації винятків при спробі ділення на нуль (тест без питань).	Успішно (100%)
quiz-import.test.ts	5	Перевірка даних: парсинг структури JSON при імпорті; перевірка обов'язкових атрибутів (title, bloomLevel, наявність хоча б однієї правильної відповіді).	Успішно (100%)

Загалом було розроблено та успішно виконано 14 модульних тестів. Жоден із них не виявив критичних помилок, що свідчить про стабільну роботу серверної частини застосунку.

4.3.2. Тестування модуля інтеграції ШІ (Google Gemini API)

Окрему увагу було приділено тестуванню модуля автоматизованої генерації тестових завдань. Оскільки відповіді великих мовних моделей можуть бути непередбачуваними, перевірялася стійкість розробленого алгоритму до збоїв:

1. **Тестування очищення даних:** алгоритм успішно виокремлював коректний JSON-об'єкт із тексту, видаляючи службові символи й теги, що запобігало помилкам обробки.
2. **Тестування резервних каналів:** Перевірено логіку перемикання між моделями (від gemini-2.5-flash до gemini-pro). У разі збою першої моделі система переходила на резервну, забезпечуючи безперервну роботу.

Висновки до розділу 4

Результати тестування підтверджують, що розроблений вебзастосунок є стабільним, захищеним від несанкціонованого доступу та готовим до обробки реальних даних. Інтеграція сучасних практик тестування дозволила створити надійний програмний продукт, що відповідає вимогам якості.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання в галузі інформаційних технологій (спеціальність 123 «Комп'ютерна інженерія») – розроблено повнофункціональний вебзастосунок QuizApp для адміністрування банку тестових завдань та автоматизованого конструювання тестів з інтеграцією технологій штучного інтелекту. За результатами виконаної роботи зроблено такі висновки:

1. Проведено комплексний аналіз галузі освітніх технологій та існуючих рішень. Дослідження платформ Google Forms, Kahoot, Moodle та Quizlet виявило їхні критичні обмеження: відсутність гнучкого структурування контенту, відсутність вбудованих механізмів конструювання тестів на базі LLM та надмірну складність адміністрування. Виявлені недоліки обґрунтували потребу у спеціалізованому програмному рішенні.

2. Спроєктовано трирівневу клієнт-серверну архітектуру вебзастосунка на основі Next.js 14 (App Router) та React Server Components, що відповідає сучасним вимогам до веборієнтованих інформаційних систем. Побудовано нормалізовану реляційну модель бази даних із восьми сутностей засобами Prisma ORM та SQLite. Реалізовано рольову модель доступу (RBAC) з розподілом на викладачів і студентів, серверними перевірками HTTP-only cookie та bcrypt-хешуванням паролів.

3. Реалізовано повний функціонал для обох ролей системи. Викладачі отримали інструментарій для управління курсами (CRUD-операції з каскадним видаленням) та швидкого наповнення бази завдань через форматований JSON-імпорт. Для студентів реалізовано захищений каталог курсів, інтерактивний навчальний модуль із автоперевіркою відповідей та рейтингову таблицю лідерів.

4. Інтегровано модуль штучного інтелекту на базі Google Gemini API. Реалізовано алгоритм генерації тестових завдань із гнучкою параметризацією за темою, мовою та розподілом когнітивного навантаження відповідно до шести

рівнів Таксономії Блума. Передбачено механізми обробки мережевих помилок, тайм-аутів та некоректних відповідей API.

5. Проведено комплексне тестування системи: модульне тестування серверної логіки та функціональне тестування модуля інтеграції ШІ. Результати підтвердили коректність розмежування доступу, надійність серверних операцій, відповідність згенерованого контенту заданим параметрам та структурну цілісність JSON-відповідей.

6. Підтверджено наукову новизну роботи: розроблено комплексний підхід до інтеграції великих мовних моделей у процес педагогічного конструювання тестів у межах структурованих курсів. Запропонований підхід переводить генерацію завдань із площини випадкового підбору у площину цілеспрямованого створення контекстно-залежних тестів за строгими методичними критеріями.

7. Доведено практичну цінність системи: впровадження автоматизації дозволяє скоротити час підготовки тестових матеріалів орієнтовно на 80 % (з 4–8 годин ручної роботи до 1–1,5 години з урахуванням рецензування ШІ-результату). Застосунок пройшов апробацію у Всеукраїнському конкурсі студентських наукових робіт 2025–2026 навчального року (спеціальність «Комп'ютерна інженерія»), за результатами якого автора нагороджено дипломом І ступеня.

Рекомендації щодо подальшого вдосконалення

На основі результатів розробки та тестування системи визначено такі перспективні напрямки її розвитку:

1. Впровадження адаптивного тестування на основі моделей Item Response Theory (IRT) - динамічний підбір питань відповідно до поточного рівня знань студента для підвищення ефективності оцінювання.

2. Інтелектуальна обробка відкритих питань (есе-типу) із застосуванням LLM для автоматичної перевірки та оцінювання розгорнутих відповідей студентів.

3. Інтеграція з корпоративними системами управління навчанням (LMS) через протокол LTI 1.3 для можливості використання QuizApp як зовнішнього інструменту у Moodle, Canvas та подібних платформах.

4. Розширення моделі бази даних: виділення навчальної групи студента в окрему сутність Group та реалізація зв'язку через зовнішній ключ для забезпечення відповідності вимогам нормалізації реляційної бази даних.

5. Розгортання застосунка у хмарній інфраструктурі (Vercel, AWS або аналоги) та розробка REST API для підтримки мобільних клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UNESCO Institute for Statistics. Education: From disruption to recovery [Електронний ресурс]. – 2021. – Режим доступу: URL: https://en.unesco.org/covid19/educationresponse (дата звернення – 15.02.2026).
2. Haladyna T. M., Rodriguez M. C. Developing and Validating Test Items. – New York : Routledge, 2013. – 448 p.
3. Alsobhi A. Y., Abeywickrama K. Adaptive E-Learning Systems: A Systematic Review // Education and Information Technologies. – 2023. – Vol. 28. – P. 3115–3145.
4. HolonIQ Research. Global EdTech Market to reach $404B by 2025 [Електронний ресурс]. – 2024. – Режим доступу: URL: https://www.holoniq.com/edtech (дата звернення – 20.02.2026).
5. Google LLC. Google Forms documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://support.google.com/docs/answer/6281888 (дата звернення – 10.01.2026).
6. Wang A. I., Tahir R. The effect of using Kahoot! for learning – A literature review // Computers & Education. – 2020. – Vol. 149. – Art. 103818.
7. Dougiamas M. Moodle: A Virtual Learning Environment for the Rest of Us // TESL-EJ. – 2004. – Vol. 8, No. 2. – P. 1–8.
8. Quizlet Inc. About Quizlet [Електронний ресурс]. – 2026. – Режим доступу: URL: https://quizlet.com/mission (дата звернення – 10.01.2026).
9. Stack Overflow. Developer Survey 2024 [Електронний ресурс]. – 2024. – Режим доступу: URL: https://survey.stackoverflow.co/2024 (дата звернення – 01.03.2026).
10. Vercel Inc. Next.js Documentation. Version 14 [Електронний ресурс]. – 2026. – Режим доступу: URL: https://nextjs.org/docs (дата звернення – 05.01.2026).

11. Gao Z., Bird C., Barr E. T. To Type or Not to Type: Quantifying Detectable Bugs in JavaScript // IEEE/ACM 39th International Conference on Software Engineering. – Buenos Aires : IEEE, 2017. – P. 758–769.
12. Prisma. Prisma ORM Documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://www.prisma.io/docs (дата звернення – 05.01.2026).
13. shadcn. shadcn/ui Documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://ui.shadcn.com (дата звернення – 05.01.2026).
14. Google DeepMind. Gemini 2.5 Pro and Flash: Highly-capable Multimodal Models [Електронний ресурс]. – 2025. – Режим доступу: URL: https://arxiv.org/abs/2503.19786 (дата звернення – 10.03.2026).
15. Anderson L. W., Krathwohl D. R. A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. – New York : Longman, 2001. – 352 p.
16. npm. bcrypt: A library to help you hash passwords [Електронний ресурс]. – 2026. – Режим доступу: URL: https://www.npmjs.com/package/bcrypt (дата звернення – 05.01.2026).
17. Tailwind CSS. Tailwind CSS Documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://tailwindcss.com/docs (дата звернення – 05.01.2026).
18. Radix UI. Radix Primitives Documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://www.radix-ui.com (дата звернення – 05.01.2026).
19. Lucide. Lucide React Icons Documentation [Електронний ресурс]. – 2026. – Режим доступу: URL: https://lucide.dev (дата звернення – 05.01.2026).
20. Deterding S. From game design elements to gamefulness: defining gamification // Proceedings of the 15th International Academic MindTrek Conference. – Tampere : ACM, 2011. – P. 9–15.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
Підпис
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

ПРОСКУРІНА ДМИТРА СЕРГІЙОВИЧА
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **МОДЕЛЬ ВЕБ-ЗАСТОСУНКА ДЛЯ АДМІНІСТРУВАННЯ БАНКУ ТЕСТОВИХ ЗАВДАНЬ І АВТОМАТИЗОВАНОГО КОНСТРУЮВАННЯ ТЕСТІВ**
керівник роботи **Приколотін Сергій Анатолійович к.ф.-м. н, доцент ЗВО**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
1) вимоги до функціоналу та структури банку тестових завдань для освітніх платформ;
2) методи та моделі штучного інтелекту для автоматизованої генерації тексту та тестових завдань;
3) вимоги до архітектури веб-застосунку для адміністрування та автоматизованого конструювання тестів;
4) персональний комп'ютер для розробки програмного забезпечення, інтеграції ШІ-моделей та тестування системи;
5) критерії оцінки якості ШІ-генерації та продуктивності веб-застосунку;
6) обмеження щодо часу відгуку системи та обсягу бази даних.
4. Перелік питань, які потрібно розробити:
1) Теоретичний аналіз сучасних підходів та існуючих рішень для адміністрування банків тестових завдань і автоматизованого конструювання тестів.
2) Дослідження та інтеграція методів штучного інтелекту для автоматизованої генерації тестових завдань різного рівня складності.
3) Проєктування архітектури веб-застосунка та розробка моделі бази даних із використанням ORM Prisma та SQLite.
4) Програмна реалізація клієнтської та серверної частин веб-застосунка, включаючи модулі адміністрування для вчителя та інтерфейс проходження курсів для учня.
5) Тестування працездатності розробленого веб-застосунка та експериментальна оцінка якості ШІ-генерації тестових завдань.
6) Аналіз отриманих результатів, надання практичних рекомендацій щодо вдосконалення системи та оформлення пояснювальної записки кваліфікаційної роботи.

5. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Аналіз предметної області та огляд існуючих рішень	13.10.2025 – 15.11.2025
2	Проєктування архітектури застосунка та моделі бази даних	16.11.2025 – 14.01.2026
3	Реалізація серверної логіки: API, Server Actions, авторизація та безпека маршрутів	15.01.2026 – 31.01.2026
4	Реалізація функціоналу викладача: курси, теми, імпорт завдань JSON	01.02.2026 – 14.02.2026
5	Підключення та організація тестування ШІ для автоматизованої генерації	15.02.2026 – 10.03.2026
6	Реалізація функціоналу студента: каталог, зарахування, проходження тестів	11.03.2026 – 24.03.2026
7	Тестування системи та аналіз якості ШІ-генерації	25.03.2026 – 14.04.2026
8	Оформлення пояснювальної записки кваліфікаційної роботи	15.04.2026 – 30.05.2026
9	Оформлення звіту про преддипломну практику	17.05.2026 – 20.05.2026
10	Предзахист кваліфікаційної роботи	21.05.2026 – 24.05.2026
11	Остаточне оформлення пояснювальної записки кваліфікаційної роботи для проходження перевірки на антиплагіат	до 27.05.2026
12	Отримання відгуку керівника, підготовка висновків та подання роботи на рецензування	до 30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Проскурін Д.С.
Підпис ініціали, прізвище

Керівник роботи _____ Приколотін С.А.
Підпис ініціали, прізвище

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання

на розробку програмного виробу «МОДЕЛЬ ВЕБЗАСТОСУНКА ДЛЯ АДМІНІСТРУВАННЯ БАНКУ ТЕСТОВИХ ЗАВДАНЬ І АВТОМАТИЗОВАНОГО КОНСТРУЮВАННЯ ТЕСТІВ»

1.	Введення	1.1. Назва: Модель вебзастосунка для адміністрування банку тестових завдань і автоматизованого конструювання тестів (QuizApp). 1.2. Галузь застосування: освітні технології (EdTech), електронне та дистанційне навчання, комп'ютерна інженерія.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 «Комп'ютерна інженерія». 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/225 від «30» травня 2026 р. (Додаток А до пояснювальної записки).
3.	Призначення розробки	3.1. Мета розробки: оптимізація процесу створення та адміністрування тестових завдань в електронному навчанні шляхом розробки вебзастосунка з інтеграцією методів штучного інтелекту. 3.2. Призначення: автоматизація процесів створення, структурування та управління банком тестових завдань; ШІ-генерація тестів за допомогою LLM; забезпечення зручного інтерфейсу проходження тестів та ведення рейтингу. 3.3. Вихідні дані: фреймворк Next.js (App Router), TypeScript, Prisma ORM, бази даних (SQLite), Google Gemini API.

<table>
<tr><td>4.</td><td>Технічні вимоги до програмного виробу</td><td>4.1. Вимоги до функціональних характеристик: автентифікація та авторизація (RBAC); CRUD-операції для курсів; ієрархічне структурування контенту; масовий імпорт тестів (JSON); автоматична генерація тестів через Google Gemini API (параметри: тема, розподіл питань за 6 рівнями Таксономії Блума, мова); інтерфейс проходження з автопідрахунком; система рейтингу.
4.2. Вимоги до надійності: серверна перевірка даних, атомарність операцій через транзакції бази даних, обробка помилок та таймаутів API.
4.3. Вимоги до умов експлуатації: сучасний веббраузер адаптивний дизайн.
4.4. Вимоги до складу і параметрів технічних засобів: сервер, клієнт.
4.5. Вимоги до інформаційної та програмної сумісності: формат JSON для імпорту/експорту.
4.6. Вимоги до маркування та упаковки: немає.
4.7. Вимоги до транспортування і зберігання: хмарний хостинг або локальні носії інформації.
4.8. Спеціальні вимоги: немає.</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td>Програмною документацією до виробу вважати:
1) Справжнє Технічне завдання (Додаток Б);
2) Опис архітектури та функціоналу (розділи 2–3 пояснювальної записки);
3) Програму та методику випробувань (Додаток В);
4) Фрагменти тексту програми (Додаток Г).</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td>1) Зменшення часових витрат викладача на підготовку тестових матеріалів до 80% завдяки AI-генерації.
2) Тривалість розробки - 2–4 місяці; використовуються безкоштовні та відкриті програмні засоби.
3) Очікується підвищення ефективності навчального процесу та об'єктивності оцінювання.</td></tr>
<tr><td>7.</td><td>Стадії і етапи розробки</td><td><table><tr><th>Дата</th><th>Назва етапу</th></tr><tr><td>01.10.2025 - 18.10.2025</td><td>Аналіз предметної області та існуючих освітніх платформ.</td></tr><tr><td>18.10.2025 - 02.12.2025</td><td>Проєктування архітектури вебзастосунка та моделі бази даних.</td></tr><tr><td>02.12.2025 - 10.01.2025</td><td>Реалізація серверної логіки, моделі авторизації (RBAC) та функціоналу</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td><table>
<tr><td></td><td>викладача.</td></tr>
<tr><td>10.01.2025 - 08.02.2026</td><td>Розробка інтерфейсу студента та механізмів тестування.</td></tr>
<tr><td>10.03.2026 - 21.03.2026</td><td>Інтеграція Google Gemini API для генерації тестів за Таксономією Блума.</td></tr>
<tr><td>21.03.2026 - 30.03.2026</td><td>Тестування системи та оцінка якості ШІ-генерації.</td></tr>
<tr><td>08.04.2026 - 30.05.2026</td><td>Оформлення пояснювальної записки та передача роботи керівнику.</td></tr>
</table></td></tr>
<tr><td>8.</td><td>Порядок контролю і приймання</td><td>1. Контроль виконання згідно календарного плану;
2. Захист на засіданні Екзаменаційної комісії;
3. Подання пояснювальної записки у паперовому та електронному вигляді.</td></tr>
</table>

Виконавець
студент групи КІ- 41
ПРОСКУРІН Д. С.
_____

Замовник
к. ф. -м. н., доцент.
ПРИКОЛОТІН С. А.
_____

Додаток В

Програма і методика випробувань програмного виробу

«МОДЕЛЬ ВЕБЗАСТОСУНКА ДЛЯ АДМІНІСТРУВАННЯ БАНКУ ТЕСТОВИХ ЗАВДАНЬ І АВТОМАТИЗОВАНОГО КОНСТРУЮВАННЯ ТЕСТІВ»

1. Об'єкт випробувань

1.1. Найменування програмного виробу: Модель вебзастосунка для адміністрування банку тестових завдань і автоматизованого конструювання тестів (QuizApp).

1.2. Галузь застосування: освітні інформаційні технології (EdTech). Застосунок може використовуватися у вищих навчальних закладах, загальноосвітніх школах та на платформах дистанційного навчання для автоматизації створення тестів, адміністрування банку питань, структурування контенту за курсами та темами, інтерактивного проходження тестів студентами та рейтингування учасників.

1.3. Умовне призначення: автоматизація процесів створення, структурування та управління тестовими завданнями з можливістю AI-генерації тестів через LLM. Розроблено для використання в освітніх закладах.

2. Мета випробувань

Метою випробувань є перевірка відповідності функціональних характеристик розробленого вебзастосунка QuizApp вимогам, зазначеним у Технічному завданні (Додаток Б). Зокрема, перевіряється: система автентифікації та авторизації з RBAC; функціонал створення та управління навчальним контентом; JSON-імпорт та AI-генерація тестів; інтерфейс проходження тестів та підрахунок результатів; поведінка системи при граничних випадках (некоректні дані, збої API, несанкціонований доступ).

3. Загальні положення

3.1. Підстави для проведення: наказ ХНУ №4101-5/1245 від 28.05.2026 про затвердження тем кваліфікаційних робіт бакалаврів та Технічне завдання відповідно до ДЕСТ 19.201-78.

3.2. Місце та тривалість: комп'ютерний клас кафедри теоретичної та прикладної системотехніки ХНУ імені В. Н. Каразіна в період роботи ДАК. Тривалість - не більше одного робочого дня.

3.3. Обсяг: визначається цією Програмою та методикою; перевіряються ключові функціональні можливості, коректність збереження даних та відповідність документації вимогам ЄСПД.

3.4. Учасники: Замовник - доцент кафедри Приколотін С.А.; Виконавець - студент групи КІ-41 Проскурін Д.С.

4. Вимоги до програмного виробу

Під час проведення випробувань програмний виріб QuizApp повинен відповідати функціональним вимогам технічного завдання: автентифікація та авторизація з RBAC (TEACHER / STUDENT); повний цикл управління курсами, темами, квізами з каскадним видаленням; JSON-імпорт із серверною валідацією без часткового запису при помилках; AI-генерація через Gemini API з параметрами (тема, розподіл за рівнями Таксономії Блума, мова); інтерфейс проходження тестів (Question Map, прогрес-бар, автопідрахунок); каталог курсів з живим пошуком, зарахування за кодом доступу; перегляд прогресу та таблиці лідерів.

5. Вимоги до програмної документації

Документація повинна відповідати ДЕСТ 19.301-79 та включати: Технічне завдання (Додаток Б); Програму та методику випробувань (Додаток В); Опис програмного виробу (розділи 2–3); Текст програми (Додаток Г).

6. Програма та методика модульного тестування (Unit testing)

Для забезпечення надійності розробленого програмного забезпечення було застосовано методологію модульного тестування (Unit Testing) з використанням фреймворку Jest. Тестування проводилося в ізольованому середовищі з використанням технології Mocking (імітації) для бази даних Prisma та внутрішніх функцій Next.js (cookies, redirect, revalidatePath).

6.1. Тестування модуля автентифікації (auth.test.ts)

Перевірено серверну дію registerUser.

- *Сценарій 1:* Успішна реєстрація нового користувача з роллю за замовчуванням (STUDENT). Перевірено коректність передачі даних до ORM Prisma.
- *Сценарій 2:* Обробка виняткових ситуацій. Перевірено блокування реєстрації та генерацію помилки у випадку, якщо електронна пошта вже існує в базі даних.

6.2. Тестування рольової моделі доступу (RBAC) (rbac.test.ts) Перевірено серверну дію createCourse на предмет безпеки.

- *Сценарій 1:* Блокування доступу для неавторизованих користувачів (імітація відсутності токена/кукі) з перенаправленням на сторінку логіну.
- *Сценарій 2:* Блокування створення курсу для користувачів із роллю STUDENT (помилка доступу 403 Forbidden).
- *Сценарій 3:* Успішне створення курсу користувачем із роллю TEACHER із коректним записом creatorId.

6.3. Тестування математичної логіки підрахунку балів (scoring.test.ts) Перевірено чисту функцію calculateScore. Відпрацьовано граничні випадки: 100% правильних відповідей, 0% правильних відповідей, коректність математичного округлення дробових значень до цілого відсотка та генерація помилки (Exception) при діленні на нуль (тест без питань).

6.4. Тестування валідації імпорту JSON (quiz-import.test.ts). Перевірено механізм імпорту квізів. Система успішно пропускає валідні структури та блокує пошкоджені файли: відсутність обов'язкового поля title, некоректний або

відсутній когнітивний рівень Таксономії Блума (bloomLevel), відсутність масиву питань або відсутність хоча б одного правильного варіанту відповіді (isCorrect: true).

Загальний результат виконання модульних тестів (14 успішних перевірок із 14) наведено на рисунку В.1.

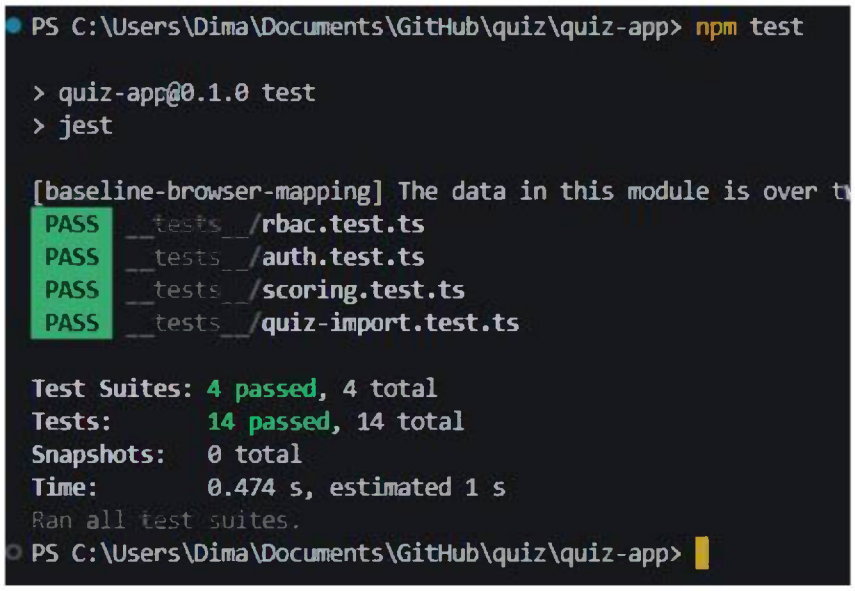


Рисунок В.1 - Результати автоматизованого модульного тестування компонентів вебзастосунка.

Виконавець

Студент групи КІ-41

Проскурін Д.С.

_____

Додаток Г

Приклади коду програми

Г.1. Схема бази даних (Prisma Schema)

```
generator client {
 provider = "prisma-client-js"
}

generator dbml {
 provider = "prisma-dbml-generator"
}

datasource db {
 provider = "sqlite"
 url    = env("DATABASE_URL")
}

model User {
 id          Int       @id @default(autoincrement())
 email       String    @unique
 name        String?
 password    String
 role        String    @default("STUDENT")
 subject     String?
 group       String?

 createdCourses Course[]   @relation("CreatedCourses")
 enrollments   Enrollment[]
 quizResults   QuizResult[]
}

model Course {
 id        Int     @id @default(autoincrement())
 title     String
 description String?
 image     String?
 password  String?
 createdAt  DateTime @default(now())

 creatorId  Int
 creator    User    @relation("CreatedCourses", fields: [creatorId], references: [id])

 enrollments Enrollment[]
```

```
 topics     Topic[]
}

model Topic {
 id       Int     @id @default(autoincrement())
 title    String
 createdAt DateTime @default(now())

 courseId  Int
 course    Course   @relation(fields: [courseId], references: [id], onDelete: Cascade)

 quizzes   Quiz[]
}

model Enrollment {
 id      Int     @id @default(autoincrement())
 joinedAt DateTime @default(now())

 userId   Int
 user     User    @relation(fields: [userId], references: [id])

 courseId Int
 course   Course   @relation(fields: [courseId], references: [id])

 @@unique([userId, courseId])
}

model Quiz {
 id        Int      @id @default(autoincrement())
 title     String
 description String?
 createdAt   DateTime  @default(now())
 dueDate     DateTime?
 maxAttempts Int      @default(1)

 topicId    Int
 topic      Topic    @relation(fields: [topicId], references: [id], onDelete: Cascade)

 creatorId   Int

 questions   Question[]
 results     QuizResult[]
}
model Question {
 id       Int     @id @default(autoincrement())
```

```
 text      String
 type      String   @default("SINGLE")

 bloomLevel String   @default("REMEMBER")

 quizId    Int
 quiz      Quiz    @relation(fields: [quizId], references: [id], onDelete: Cascade)

 options   Option[]
}

model Option {
 id        Int     @id @default(autoincrement())
 text      String
 isCorrect  Boolean  @default(false)

 questionId Int
 question   Question @relation(fields: [questionId], references: [id], onDelete: Cascade)
}

model QuizResult {
 id          Int     @id @default(autoincrement())
 score       Int
 correctAnswers Int
 totalQuestions Int
 createdAt     DateTime @default(now())

 studentId     Int
 student       User    @relation(fields: [studentId], references: [id], onDelete: Cascade)

 quizId       Int
 quiz         Quiz    @relation(fields: [quizId], references: [id], onDelete: Cascade)
}
```

Г.2. Server Action: AI-генерація тесту через Google Gemini API

```
"use server"

import { GoogleGenerativeAI } from "@google/generative-ai"

const API_KEY = process.env.GEMINI_API_KEY_TEST

if (!API_KEY) {
 console.error("УВАГА: API ключ не знайдено! Перевірте GEMINI_API_KEY_TEST у файлі .env")
}

const genAI = new GoogleGenerativeAI(API_KEY || "");

const MODELS_TO_TRY = [
```

```
  "gemini-2.5-flash",
  "gemini-pro",
  "gemini-1.5-pro"
 ]

 function cleanGeminiResponse(text: string): string {
  let clean = text.replace(/```json/g, "").replace(/```/g, "").trim();
  const firstBrace = clean.indexOf('{');
  const lastBrace = clean.lastIndexOf('}');
  if (firstBrace !== -1 && lastBrace !== -1) {
   clean = clean.substring(firstBrace, lastBrace + 1);
  }
  return clean;
 }

 export async function generateQuizWithAI(
  topic: string,
  remember: number,
  understand: number,
  apply: number,
  analyze: number,
  evaluate: number,
  create: number
 ) {
  let lastError = null;
  const totalCount = remember + understand + apply + analyze + evaluate + create;

  if (totalCount === 0) {
     return { error: "Виберіть хоча б одне питання для генерації." }
  }

  const prompt = `
     Ти експерт-методист. Створи професійний тест на тему: "${topic}".
     Мова: Українська.

     Твоя мета - згенерувати питання, суворо дотримуючись Таксономії Блума (6 рівнів).
     Кількість питань по кожному рівню:
     - Знання (REMEMBER): ${remember} шт. (Перевірка пам'яті: факти, терміни, визначення).
      - Розуміння (UNDERSTAND): ${understand} шт. (Пояснення ідей: класифікація, наведення прикладів, резюмування).
     - Застосування (APPLY): ${apply} шт. (Використання знань: розв'язання задач, застосування правил у нових ситуаціях).
     - Аналіз (ANALYZE): ${analyze} шт. (Пошук зв'язків: порівняння, деконструкція, пошук помилок або логіки).
      - Оцінка (EVALUATE): ${evaluate} шт. (Критика та судження: вибір найкращого рішення, аргументація, перевірка гіпотез).
     - Створення (CREATE): ${create} шт. (Генерація нового: дизайн, планування, синтез нових ідей з наявних).
     - Всього питань: ${totalCount}.

     Важливо:
     1. Поверни лише валідний JSON.
     2. Ніякого додаткового тексту до чи після JSON.
      3. Обов'язково вказуй поле "bloomLevel" для кожного питання ВЕЛИКИМИ англійськими літерами (REMEMBER, UNDERSTAND, APPLY, ANALYZE, EVALUATE, CREATE).

     Формат JSON:
     {
      "title": "Назва тесту",
      "description": "Опис",
      "questions": [
       {
        "text": "Питання",
        "bloomLevel": "REMEMBER",
        "options": [
         { "text": "Варіант 1", "isCorrect": true },
         { "text": "Варіант 2", "isCorrect": false },
         { "text": "Варіант 3", "isCorrect": false }
        ]
```

```
      }
     ]
    }
  `

  for (const modelName of MODELS_TO_TRY) {
   try {
    const model = genAI.getGenerativeModel({ model: modelName })
    const result = await model.generateContent(prompt)
    const response = await result.response
    const text = response.text()

    const cleanJson = cleanGeminiResponse(text)
    JSON.parse(cleanJson) // Валідація

    return { jsonString: cleanJson }

   } catch (error: any) {
    console.error(`Модель ${modelName} помилка: ${error.message}`)
    lastError = error.message
   }
  }

  return { error: `Не вдалося згенерувати квіз. Деталі: ${lastError}` }
}

// 2. Генерація з тексту (Text to Quiz)
export async function generateQuizFromText(sourceText: string) {
  let lastError = null;
  const cleanText = sourceText.slice(0, 20000);

  const prompt = `
    Ти професійний методист. Проаналізуй наданий текст і створи по ньому тест.

    Текст для аналізу:
    """
    ${cleanText}
    """

    Інструкції (дуже важливо):
    1. Проаналізуй текст.
    2. Якщо це вже список питань (тест):
      - Витягни усі питання (до 50 штук).
      - Збережи правильні відповіді.
      - Самостійно визнач рівень Блума (bloomLevel) для кожного питання.
    3. Якщо це просто лекція або стаття:
      - Створи 10-15 питань на основі тексту, використовуючи різні рівні Таксономії Блума.

    Вимоги до формату:
    - Мова: Українська (або така, як в тексті).
    - Поверни ТІЛЬКИ чистий JSON код.

    Структура JSON:
    {
     "title": "Назва тесту",
     "description": "Короткий опис",
     "questions": [
      {
       "text": "Питання",
       "bloomLevel": "UNDERSTAND",
       "options": [
        { "text": "Варіант 1", "isCorrect": true },
        { "text": "Варіант 2", "isCorrect": false }
       ]
      }
     ]
    }
```

```
`

 for (const modelName of MODELS_TO_TRY) {
  try {
   const model = genAI.getGenerativeModel({ model: modelName })
   const result = await model.generateContent(prompt)
   const response = await result.response
   const text = response.text()

   const cleanJson = cleanGeminiResponse(text)
   JSON.parse(cleanJson)

   return { jsonString: cleanJson }

  } catch (error: any) {
   console.error(` Модель ${modelName} помилка (Text-to-Quiz): ${error.message}`)
   lastError = error.message
  }
 }

 return { error: `Не вдалося обробити текст. Деталі: ${lastError}` }
}
```

Г.3. Server Action: створення курсу

```
import { hash, compare } from "bcryptjs";

export async function createCourse(formData: FormData) {
 const title = formData.get("title") as string;
 const description = formData.get("description") as string;
 const password = formData.get("password") as string;
 const image = formData.get("image") as string;

 const cookieStore = await cookies();
 const userId = cookieStore.get("userId")?.value;

 if (!userId) redirect("/login");

 const user = await prisma.user.findUnique({
  where: { id: parseInt(userId) },
  select: { role: true },
 });

 // Якщо роль не викладач - блокую
 if (user?.role !== "TEACHER" && user?.role !== "teacher") {
  return { error: "Тільки викладачі можуть створювати курси!" };
 }

 if (!title || title.trim().length < 3) {
  return;
 }
```

```
if (!password || password.trim().length < 4) {
 return;
}
// 2. Хешую пароль курсу
const hashedCoursePassword = await hash(password, 10);

await prisma.course.create({
 data: {
  title,
  description,
  password: hashedCoursePassword, // Записую захешований пароль
  image: image || null,
  creatorId: parseInt(userId),
 },
});
```

Г.4. Server Action: імпорт квізу через JSON

```
export async function createQuizFromJSON(topicId: number, courseId: number, jsonString: string) {
  try {
    console.log("Початок створення квізу")
    console.log("1. JSON від ШІ:", jsonString)

    const data: QuizJsonData = JSON.parse(jsonString)
    console.log("2. JSON успішно розпарсено. Кількість питань:", data.questions?.length)

    // Отримую автора (викладача) з кукі
    const cookieStore = await cookies()
    const userIdStr = cookieStore.get("userId")?.value
    const userId = userIdStr ? parseInt(userIdStr) : null

    if (!userId) {
      console.error("Помилка: Користувач не авторизований (немає userId в кукі)")
      return { error: "Не авторизовано" }
    }

    console.log("3. Спроба запису в БД для topicId:", topicId, "creatorId:", userId)

    await prisma.quiz.create({
     data: {
      title: data.title,
      description: data.description || "",
      dueDate: data.dueDate ? new Date(data.dueDate) : null,
      maxAttempts: data.maxAttempts || 1,
      topicId: topicId,
      creatorId: userId,
      questions: {
       create: data.questions.map((q) => ({
        text: q.text,
        type: q.type || "SINGLE",
        bloomLevel: q.bloomLevel || "REMEMBER",
        options: {
         create: q.options.map((o) => ({ text: o.text, isCorrect: o.isCorrect }))
        }
       }))
      }
     }
    })
```

Додаток Д

Апробація результатів кваліфікаційної роботи

Нижче наведено копію диплома І ступеня, яким нагороджено за результатами участі у Всеукраїнському конкурсі студентських наукових робіт з галузей знань і спеціальностей у 2025-2026 навчальному році за спеціальністю 123 «Комп'ютерна інженерія».



ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ імені В. Н. КАРАЗІНА

Диплом

I ступеня

нагороджується

Проскурін Дмитро Сергійович

здобувач 4 року навчання першого (бакалаврського) рівня вищої освіти навчально-наукового інституту комп'ютерних наук та штучного інтелекту за І місце у всеукраїнському конкурсі студентських наукових робіт з галузей знань і спеціальностей у 2025-2026 навчальному році (Спеціальність "Комп'ютерна інженерія")

Ректорка Тетяна КАГАНОВСЬКА

Харків 2026

Рисунок Д.1 - Копія диплома І ступеня Всеукраїнського конкурсу студентських наукових робіт (2025-2026 н. р.).