Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ СТУДЕНТІВ

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Костишин П. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ
Гармаш В. В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Костишину Павлу Віталійовичу

1. Тема роботи: Розробка WEB-ресурсу для тестування знань студентів.
керівник роботи: Крилик Людмила Вікторівна, к.т.н., доц.
затверджені наказом вищого навчального закладу «20» березня 2025 року № 97

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи: мова програмування – об'єктно-орієнтована; кількість підтримуваних платформ не менше 3; типи питань – множинний вибір, відкриті запитання, відповідність; інтерфейс користувача має бути інтуїтивно зрозумілий, проста авторизація та аутентифікація для користувачів.

4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності розробки WEB-ресурсу для тестування знань студентів; проектування WEB-ресурсу для тестування знань студентів; програмна реалізація WEB-ресурсу для тестування знань студентів; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): загальна структурна схема системи; UML-діаграма класів клієнтської частини WEB-ресурсу для тестування знань студентів; UML-діаграма класів серверної частини WEB-ресурсу для тестування знань студентів; ER-модель бази даних; схема алгоритму роботи WEB-ресурсу для тестування знань студентів; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Крилик Л. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки WEB-ресурсу для тестування знань студентів	05.05.25	07.05.25	
2	Проектування WEB-ресурсу для тестування знань студентів	08.05.25	11.05.25	
3	Програмна реалізація WEB-ресурсу для тестування знань студентів	12.05.25	15.05.25	
4	Висновки та аналіз отриманих результатів	16.05.25	26.05.25	
5	Оформлення додатків	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Костишин П. В**
(підпис) (прізвище та ініціали

Керівник роботи _____ **Крилик Л. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 101 сторінок формату А4, які включають 27 рисунків і 7 таблиць. У списку використаних джерел зазначено 24 найменування.

Метою роботи є розширення функціональних можливостей WEB-ресурсу для тестування знань студентів.

У загальній частині роботи на основі аналізу існуючих платформ обґрунтовано доцільність розробки нового рішення з покращеною функціональністю. Спроєктовано архітектуру WEB-ресурсу, розроблено UML-діаграми клієнтської і серверної частин, алгоритм роботи системи та ER-модель бази даних.

WEB-ресурс реалізований з використанням технологій Next.js, React, PostgreSQL, Sequelize, Tailwind CSS, JWT та інших бібліотек. Програмний продукт протестовано, проведено порівняльний аналіз з аналогами та доведено його переваги: підтримка динамічного формування тестів, гнучка конфігурація варіантів, адаптивний інтерфейс, безпечний захист даних.

Ключові слова: WEB-ресурс, тестування знань, Next.js, база даних, React.

ABSTRACT

The bachelor's thesis consists of 101 A4 pages, including 27 figures and 7 tables. The list of references includes 24 titles.

The purpose of the work is to expand the functionality of a WEB resource for testing students' knowledge.

In the general part of the work, based on the analysis of existing platforms, the expediency of developing a new solution with improved functionality is substantiated. The architecture of the WEB resource is designed, UML diagrams of the client and server parts, the system algorithm and the ER-model of the database are developed.

The WEB-resource was implemented using Next.js, React, PostgreSQL, Sequelize, Tailwind CSS, JWT and other libraries. The software product has been tested, a comparative analysis with analogues has been carried out and its advantages have been proved: support for dynamic test generation, flexible configuration of options, adaptive interface, secure data protection.

Keywords: WEB-resource, knowledge testing, Next.js, database, React.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Сучасний розвиток інформаційних технологій сприяє суттєвим змінам у традиційних підходах до освіти. Інтеграція WEB-технологій в освітні процеси відкриває нові можливості для організації навчання, особливо у сфері тестування та оцінювання знань студентів. Освітні установи прагнуть впроваджувати інноваційні цифрові рішення, які забезпечують доступність, автоматизацію та прозорість освітнього процесу. Використання WEB-ресурсів для проведення тестувань дозволяє викладачам і студентам взаємодіяти ефективніше, скорочуючи витрати часу та ресурсів.

Особливої уваги потребує розробка платформ, які не лише надають базові інструменти тестування, а й враховують сучасні вимоги до функціональності. Сьогодні викладачі потребують зручних інструментів для налаштування тестів, які дозволяють обирати типи запитань, задавати їх вагу, використовувати мультимедійні матеріали та забезпечувати підтримку математичних виразів. Для студентів важливими є інтуїтивно зрозумілий інтерфейс, доступність з різних пристроїв і можливість отримувати швидкий зворотний зв'язок за результатами тестування.

В умовах зростаючої потреби в дистанційній освіті, яка посилюється пандеміями та іншими глобальними викликами, створення платформ, що адаптовані до віддаленого навчання, стає вкрай актуальним. Це дозволяє організовувати якісне оцінювання знань незалежно від місця перебування учасників навчального процесу.

Отже, створення спеціалізованого WEB-ресурсу для тестування знань студентів є доцільним та актуальним завданням, яке відповідає сучасним потребам освітнього середовища. Такий ресурс дозволить підвищити ефективність оцінювання, забезпечить комфортну взаємодію для викладачів і студентів та сприятиме вдосконаленню освітнього процесу в умовах цифрової трансформації.

Метою дослідження є розширення функціональних можливостей WEB-ресурсу для тестування знань студентів.

Об'єктом дослідження є процес тестування знань студентів у навчальних закладах.

Предметом дослідження є програмні засоби реалізації WEB-ресурсу для тестування знань студентів

Задачі дослідження:

1. Обґрунтувати доцільність розробки WEB-ресурсу для тестування знань студентів.

2. Спроектувати WEB-ресурс для тестування знань студентів.

3. Програмно реалізувати WEB-ресурс для тестування знань студентів.

4. Виконати тестування розробки та провести аналіз отриманих результатів.

5. Розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) м. Вінниці у 2025 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1] та отримано свідоцтво про реєстрацію авторського права на твір у формі комп’ютерної програми – «WEB-ресурс для тестування знань студентів [2].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ СТУДЕНТІВ

У сучасному світі, де технології стрімко розвиваються, а цифровізація стає невід'ємною частиною освітнього процесу, розробка WEB-ресурсів для тестування знань студентів є не лише актуальною, але й надзвичайно корисною. Такий WEB-ресурс може значно покращити процес оцінювання знань, зробити його зручним та ефективним як для викладачів, так і для студентів. Розглянемо переваги, які забезпечує розробка такого ресурсу [3]:

1. Автоматизація процесу тестування: WEB-ресурс дозволяє автоматизувати рутинні завдання, пов’язані з підготовкою, проведенням та оцінюванням тестувань. Це значно скорочує час, необхідний викладачам для виконання цих завдань і мінімізує ймовірність людських помилок.

2. Гнучкість у налаштуванні тестувань: викладачі можуть самостійно обирати параметри тестування, такі як типи запитань (тести з вибором, відкриті запитання, запитання на відповідність тощо), обмеження за часом, кількість спроб, критерії оцінювання тощо, адаптуючи тест до конкретних потреб [4].

3. Можливість аналізу результатів: система дозволяє зберігати, аналізувати та відображати результати тестувань, що надає викладачам інструменти для детального аналізу успішності студентів і швидкої ідентифікації проблемних тем.

4. Доступність і мобільність: оскільки WEB-ресурс доступний через Інтернет, він дозволяє проводити тестування з будь-якого місця, забезпечуючи гнучкість у часі та просторі як для викладачів, так і для студентів.

5. Інтерактивність і зручність: інтуїтивно зрозумілий інтерфейс робить ресурс зручним у використанні, сприяючи позитивному досвіду як викладачів, так і студентів [5].

6. Екологічність та економія ресурсів: використання цифрового формату тестувань значно зменшує потребу у паперових матеріалах, що відповідає принципам сталого розвитку.

Таким чином, розробка WEB-ресурсу для тестування знань студентів є ефективним рішенням, яке враховує сучасні виклики і потреби освітньої системи, підвищуючи її продуктивність, доступність та якість.

1.1 Огляд середовищ для реалізації програмного продукту

Вибір середовища для реалізації WEB-ресурсу є ключовим етапом у процесі розробки. З урахуванням різноманітності платформ та їхніх можливостей, рішення має ґрунтуватися на потребах користувачів, особливостях майбутнього продукту та функціональних вимогах до нього.

У таблиці 1.1 представлена порівняльна характеристика платформ, які можуть бути використані для розробки WEB-ресурсу для тестування знань студентів.

Таблиця 1.1 – Порівняльна характеристика платформ для розробки WEB-ресурсу для тестування знань студентів

Характеристика	ПК	Мобільні пристрої	WEB-застосунки
Доступність	Зручний доступ на робочому місці	Мобільний доступ із будь-якого місця	Зручний доступ з будь-якого пристрою з Інтернетом
Операційні системи	Windows, macOS, Linux	Android, iOS	Сумісність із будь-яким сучасним WEB-браузером
Охоплення користувачів	Менше охоплення порівняно з телефонами	Велике охоплення через популярність мобільних пристроїв	Велике охоплення через універсальність доступу
Встановлення	Потребує встановлення окремого додатку	Потребує встановлення мобільного додатку	Не потребує встановлення, доступ через WEB-браузер

Продовження таблиці 1.1

Характеристика	ПК	Мобільні пристрої	WEB-застосунки
Зручність використання	Обмежена мобільність, залежність від ОС	Компактність, портативність, залежність від ОС	Універсальність доступу
Екосистема	Різноманітна, залежить від платформи з фреймворками, такими як .NET Core, Electron, Qt	Розділена між Android і iOS, обмеженіша	Велика кількість бібліотек, таких як React, Angular, Vue.js, Django, Flask, Laravel
Мови програмування	Не обмежений (Java, C++, C#, Python тощо)	Java, Kotlin (Android), Swift, Objective-C (iOS)	Обмежений для FrontEnd (JavaScript/TypeScript, WASM). Необмежений для BackEnd (Python, PHP тощо)

WEB-застосунки є оптимальним вибором для розробки WEB-ресурсу для тестування знань студентів завдяки їхній універсальності, доступності та простоті використання.

WEB-застосунки працюють на різних операційних системах та пристроях, починаючи від комп’ютерів і до смартфонів, що забезпечує зручний доступ до ресурсу незалежно від типу обладнання. Для роботи з ними достатньо мати WEB-браузер, який зазвичай є на більшості сучасних пристроїв, що значно спрощує доступ до тестувань із будь-якого місця.

WEB-застосунки також не потребують додаткового встановлення програмного забезпечення, що зменшує поріг входу для користувачів, роблячи їх більш доступними [6].

Завдяки цим перевагам, WEB-застосунки забезпечують ефективну взаємодію між викладачами та студентами, дозволяючи створювати і проводити тестування максимально зручно та гнучко.

Аналіз даних з таблиці 1.1 підтверджує, що розробка WEB-ресурсу у форматі WEB-застосунку є найбільш раціональним вибором для забезпечення якісного та ефективного процесу оцінювання знань [7].

1.2 Огляд систем-аналогів

Існує багато програмних продуктів схожого функціоналу і кожен з них має свої переваги та недоліки. Варто провести аналіз систем-аналогів для визначення перспектив розробки та покращення функціональних можливостей WEB-ресурсу для тестування знань студентів.

Для порівняння розглянемо такі ресурси, як «Майстер-Тест», «Google Classroom», «Всеосвіта» [8 − 11].

«Майстер-Тест» − це безкоштовний онлайн-сервіс, призначений для створення та проведення тестувань у навчальних цілях. Він дозволяє викладачам розробляти різнорівневі тести з різними типами запитань, такими як вибір однієї або кількох правильних відповідей, встановлення відповідності тощо (рис. 1.1) [8].

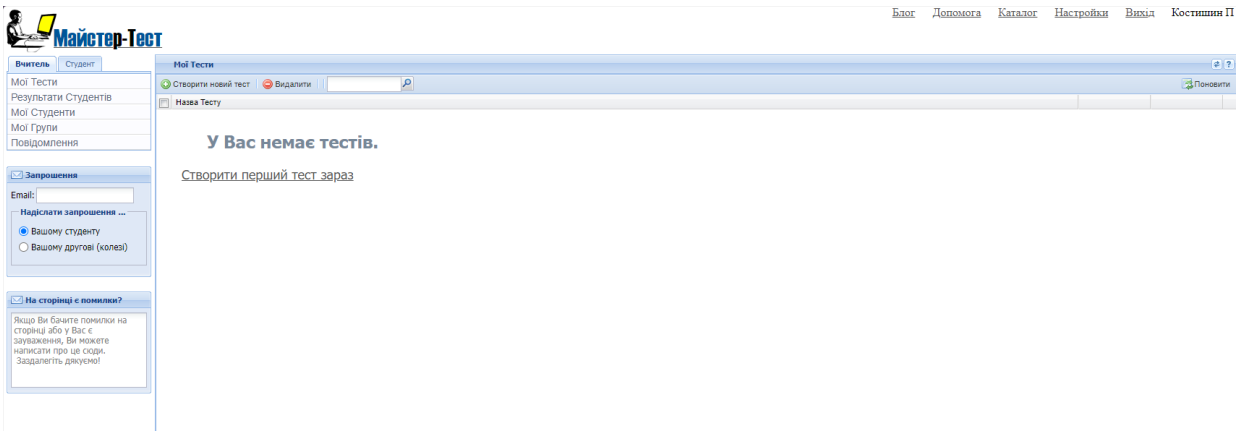

Рисунок 1.1 – Загальний вигляд сторінки з тестами онлайн-сервісу «Майстер-Тест»

Переваги «Майстер-Тест»:

- доступність: сервіс є безкоштовним і доступним онлайн, що дозволяє використовувати його без необхідності встановлення додаткового програмного забезпечення;

- різноманітність запитань: підтримка різних типів запитань, включаючи вибір однієї або кількох правильних відповідей, встановлення відповідності, що відповідає вимогам різних навчальних програм;

- мультимедійна підтримка: можливість додавання зображень, аудіо та відео до тестових завдань, що сприяє підвищенню інтересу студентів та кращому засвоєнню матеріалу.

Недоліки «Майстер-Тест»:

- збої в роботі: деякі користувачі відзначають, що під час роботи з сервісом можуть виникати збої, що впливає на загальну зручність використання;

- обмежена функціональність: порівняно з іншими платформами, «Майстер-Тест» може мати менше можливостей для налаштування тестів та інтеграції з іншими освітніми ресурсами;

- інтерфейс користувача: деякі користувачі вважають інтерфейс сервісу не досить інтуїтивним, що може ускладнювати процес створення та проведення тестів.

Google Classroom − це безкоштовна платформа від Google, призначена для організації навчального процесу в освітніх закладах [9]. Для доцільності порівняння – будемо порівнювати тільки складову створення тестів, а саме – Google Forms (рис. 1.2) [10].

Переваги платформи «Google Forms»:

- простота використання: інтуїтивно зрозумілий інтерфейс дозволяє швидко створювати форми без необхідності спеціальних технічних знань;

- автоматичний збір та аналіз результатів: відповіді респондентів автоматично збираються та можуть бути переглянуті у вигляді зведених даних або експортовані в Google Sheets для детальнішого аналізу;

- інтеграція з Google Classroom: дозволяє легко розповсюджувати тести серед студентів та отримувати їхні результати в режимі реального часу;

- доступність: форми можна створювати та редагувати з будь-якого пристрою з доступом до інтернету, що забезпечує гнучкість у роботі.

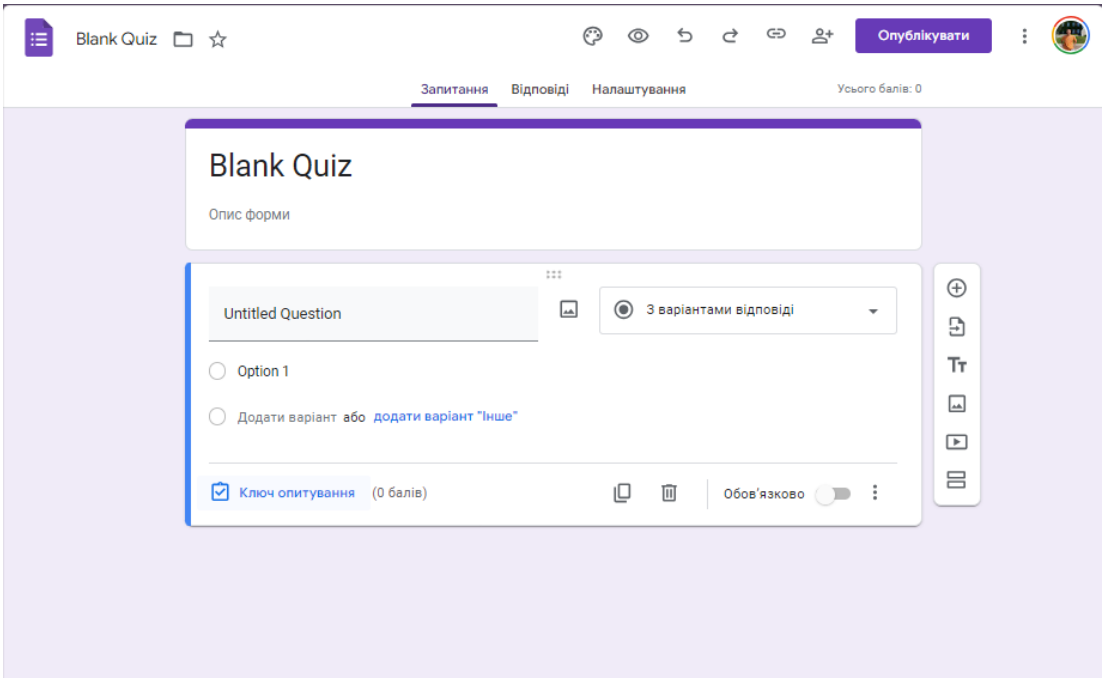

Рисунок 1.2 – Загальний вигляд сторінки платформи «Google Forms»

Недоліки платформи «Google Forms»:

- обмежений вибір типів питань: порівняно з деякими спеціалізованими платформами для тестування, Google Forms пропонує меншу різноманітність типів запитань, що може обмежувати можливості створення складніших тестів;

- відсутність підтримки математичних формул: для тестів, що містять математичні вирази, відсутня можливість безпосереднього введення формул, що може ускладнити створення таких тестів.

«Всеосвіта» – це українська освітня онлайн-платформа, яка надає інструменти для дистанційного навчання. Сервіс пропонує можливості для створення та проведення онлайн-тестувань (рис. 1.3) [11].

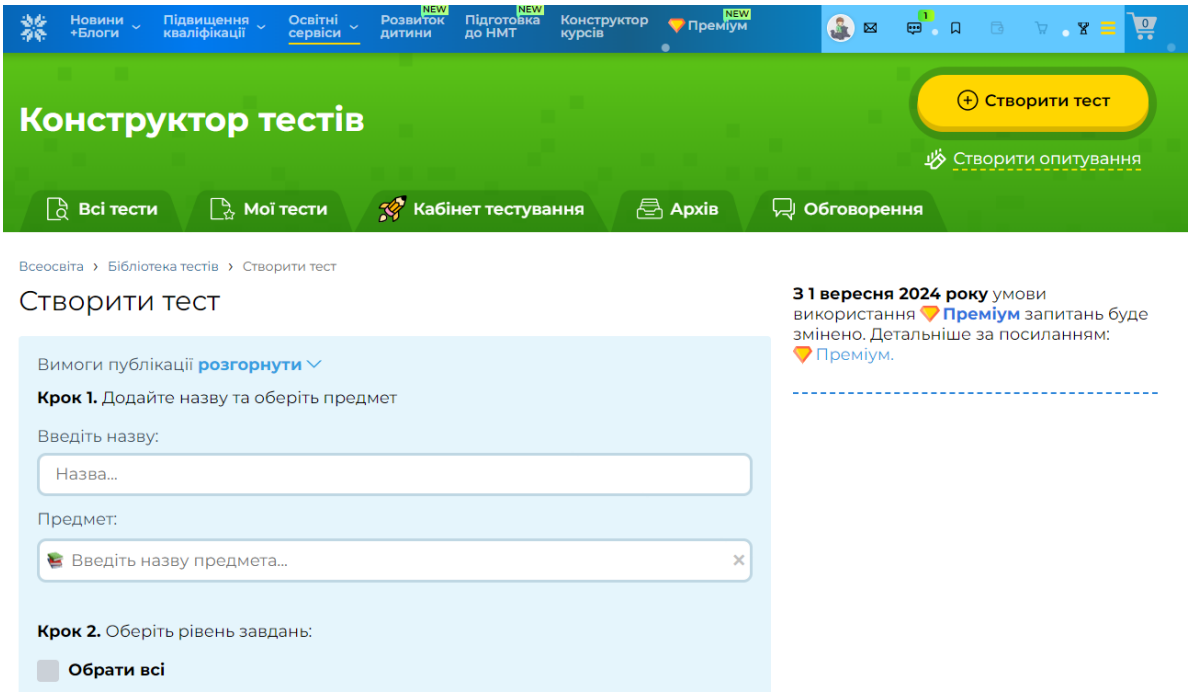

Рисунок 1.3 – Загальний вигляд сторінки платформи «Всеосвіта»

Переваги онлайн-платформи «Всеосвіта»:

- зручний конструктор тестів: інтуїтивно зрозумілий інтерфейс дозволяє швидко створювати тести, обираючи предмет, клас та додаючи запитання різних типів;

- різноманітність типів запитань: платформа підтримує різні формати запитань, що дозволяє створювати тести з урахуванням специфіки навчального матеріалу;

- автоматичне оцінювання: система автоматично перевіряє відповіді студентів, що зменшує навантаження на викладача та забезпечує оперативний зворотний зв'язок;

- статистика та аналітика: платформа надає детальні звіти про результати тестування, що допомагає аналізувати успішність студентів та коригувати навчальний процес;

- доступність: тести можна проходити з будь-якого пристрою з доступом до інтернету, що забезпечує гнучкість у навчанні.

Недоліки онлайн-платформи «Всеосвіта»:

- обмежена інтеграція з іншими платформами: відсутність повної інтеграції з деякими популярними освітніми платформами може ускладнювати використання «Всеосвіти» у комплексних навчальних середовищах;

- можливі технічні обмеження: як і будь-яка онлайн-платформа, «Всеосвіта» може стикатися з технічними проблемами або обмеженнями, що впливають на доступність сервісу;

- потреба в інтернет-з'єднанні: для створення та проходження тестів необхідний постійний доступ до інтернету, що може бути недоліком у регіонах з нестабільним з'єднанням.

1.3 Формування вимог та постановка задачі

Щороку роль технологій у освітньому процесі стрімко зростає і автоматизація тестування знань студентів стає невід'ємною частиною сучасної освіти. Використання цифрових інструментів для оцінювання дозволяє забезпечити ефективність і прозорість процесу, зменшити навантаження на викладачів, а також надати студентам більш зручний спосіб взаємодії з тестовими завданнями.

Існуючі платформи, такі як «Майстер-Тест», «Google Forms» і «Всеосвіта», значною мірою спрощують процес створення та проведення тестувань. Кожна з цих систем має свої переваги, але недоліки обмежують їх використання в певних умовах. Наприклад, «Майстер-Тест» пропонує розширені можливості налаштування тестів, однак має застарілий інтерфейс, що ускладнює взаємодію для сучасного користувача. «Google Forms» забезпечує легкість у створенні тестів і доступність, проте не підтримує повного спектра функцій, таких як обробка математичних формул чи створення варіантів тестів з урахуванням ваги питань. Платформа «Всеосвіта» має потужний функціонал для аналітики, але обмежена у гнучкості налаштувань і адаптації під мобільні пристрої.

Враховуючи аналіз цих платформ, можна виділити основні виклики, які постають перед освітянами при використанні існуючих рішень: незручність

налаштування параметрів тестування, обмежена підтримка мультимедіа та математичних формул, недостатня адаптація для мобільних пристроїв, відсутність можливості створення складних варіантів тестів, що враховують вагу запитань.

На основі проведеного аналізу сформульовано вимоги до функціоналу та задачі для розробки нового WEB-ресурсу для тестування знань студентів. Цей ресурс покликаний вирішити існуючі проблеми, запропонувавши сучасний інструмент, який поєднує інтуїтивно зрозумілий інтерфейс, розширений функціонал і високу доступність для всіх категорій користувачів.

Загальні вимоги:

- легка авторизація для студентів: студенти повинні мати змогу швидко проходити авторизацію, використовуючи мінімум даних;
- легкий спосіб поділитися тестом: платформа має забезпечувати можливість швидко розповсюджувати посилання на тест;
- можливість обмеження доступу до тесту: викладач повинен мати змогу закрити доступ до тесту після завершення періоду тестування;
- аналіз результатів тестування: система має генерувати аналітичні звіти з графіками, середнім балом, частотами правильних і неправильних відповідей.

Функціональні вимоги:

- підтримка різних типів питань: текстові, варіанти відповідей, багатоваріантні відповіді, відкриті запитання, відповідність, перетягування елементів;
- мультимедійна підтримка: можливість додавання зображень, аудіо, відео та інших медіа для створення інтерактивних запитань;
- підтримка математичних формул: інтеграція з математичними редакторами;
- сучасний інтерфейс: простий, інтуїтивно зрозумілий, з адаптацією до різних розмірів екранів;
- доступність для мобільних пристроїв: інтерфейс має бути повністю адаптованим для смартфонів та планшетів;

- можливість вказати вагу питання: викладач може призначати різні бали для різних запитань залежно від їхньої складності;

- генерація варіантів тестів: інструмент для створення кількох варіантів одного тесту з автоматичною ротацією питань із заданих блоків. Наприклад, кожен студент отримує обов’язкові питання з першого, другого і третього блоку відповідно до заданих умов.

Основне завдання полягає в розробці WEB-ресурсу для тестування знань студентів, який забезпечить:

- зручність роботи для викладача при створенні, розповсюдженні та оцінюванні тестів;

- адаптацію до сучасних освітніх потреб, включаючи мультимедійні запитання та підтримку математичних виразів;

- високий рівень доступності та інтерактивності для студентів через мобільні пристрої;

- функціонал аналізу результатів тестування для викладачів.

Реалізація цих вимог дозволить створити конкурентоспроможний продукт, який враховує сучасні освітні тенденції та подолання недоліків аналогічних платформ.

1.4 Висновки до розділу 1

У цьому розділі було проаналізовано середовища для реалізації програмного продукту, а також проведено огляд існуючих систем-аналогів для тестування знань студентів. Рішення про розробку WEB-ресурсу для тестування знань було обґрунтовано на основі універсальності та доступності таких середовищ, які забезпечують зручність використання WEB-ресурсу для студентів і викладачів.

Аналіз існуючих платформ показав, що жодна з них не задовольняє повністю потреби користувачів, оскільки більшість фокусується лише на базових функціях створення тестів без врахування вимог до мультимедійної підтримки,

математичних формул або розширених аналітичних можливостей та створення рівноцінних варіантів.

Таким чином, розробка WEB-ресурсу для тестування знань є актуальною, оскільки вона дозволить вирішити існуючі проблеми і забезпечити комфортне середовище оцінювання знань для викладачів і студентів. Платформа має бути гнучкою, з можливістю налаштування тестів під різні потреби, підтримкою мультимедійних матеріалів, математичних виразів і сучасних функцій аналізу результатів.

У розділі також було сформульовано вимоги до системи та поставлено завдання для подальшої розробки програмного продукту, що дозволить створити конкурентоспроможний та ефективний інструмент для тестування знань студентів.

2 ПРОЕКТУВАННЯ WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ СТУДЕНТІВ

2.1 Розробка структури WEB-ресурсу для тестування знань студентів

WEB-ресурс для тестування знань студентів є сучасним програмним засобом, призначеним для організації процесу оцінювання знань студентів. Основними користувачами системи є дві категорії: студенти та викладачі, кожна з яких має окремі права доступу та функціональні можливості.

Головна мета системи – надати простий, доступний і масштабований інструмент для:

- створення та редагування тестів викладачами;
- проходження тестування студентами;
- збору та аналізу результатів з можливістю перегляду статистики;
- автоматизованого зберігання інформації в єдиній базі даних.

Архітектурно система реалізована за моделлю «клієнт-сервер»:

- клієнтська частина створена з використанням фреймворку Next.js;
- серверна частина реалізована на основі вбудованих API-роутів Next.js;
- база даних – PostgreSQL, керується через ORM Sequelize;
- система слідує шаблону MVC (Model-View-Controller) для логічного поділу структури коду [12].

WEB-ресурс має чітко визначену модульну структуру, яка забезпечує логічне розділення функціональності між компонентами системи. Центральним елементом взаємодії є користувач, який через WEB-інтерфейс звертається до функціоналу системи відповідно до своєї ролі.

На рисунку 2.1 зображено загальну структурну схему системи. WEB-ресурс має чітко визначену модульну структуру, яка забезпечує логічне розділення функціональності між компонентами системи. Центральним елементом взаємодії

є користувач, який через WEB-інтерфейс звертається до функціоналу системи відповідно до своєї ролі.

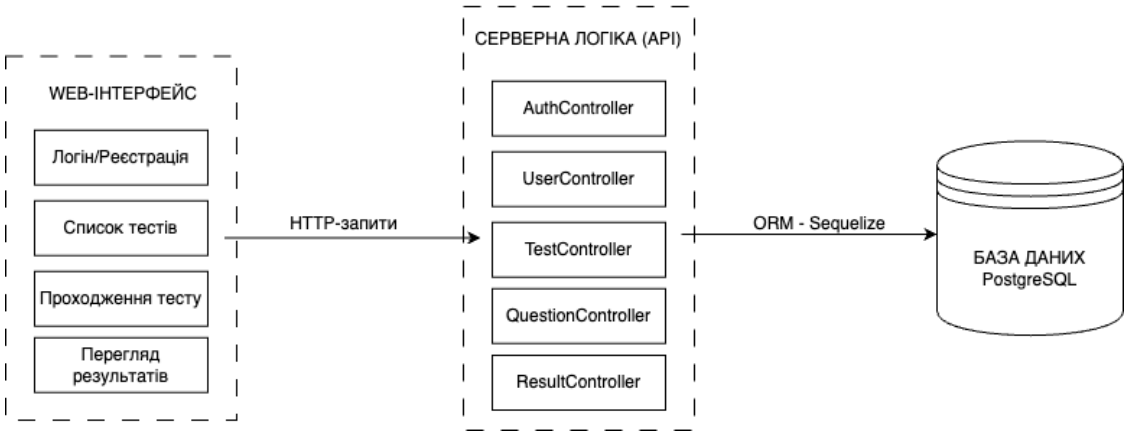

Рисунок 2.1 – Загальна структурна схема система

Після запуску ресурсу користувач потрапляє на сторінку автентифікації, де вводить логін і пароль. Система перевіряє облікові дані та, у разі успішного входу, перенаправляє до особистого кабінету:

- студента – на панель з доступними тестами;
- викладача – до інтерфейсу управління тестами;

Система складається з трьох основних частин:

- Клієнтська частина – реалізована на базі Next.js: формує інтерфейс користувача; обробляє події (натискання, введення, переходи); взаємодіє з API через HTTP-запити; використовує бібліотеку Formik для валідації форм [13];

- Серверна частина – реалізована на Next.js: обробляє запити від клієнта; перевіряє права доступу; взаємодіє з базою даних через ORM Sequelize; реалізує маршрути для операцій з тестами, питаннями, результатами;

- База даних (PostgreSQL): зберігає дані про користувачів, тести, запитання, відповіді, результати; гарантує цілісність зв’язків через зовнішні ключі;

- має окремі таблиці для студентів і викладачів.

Загальний потік взаємодії:

- Користувач виконує дію у браузері;
- Дані надсилаються на сервер;
- Сервер обробляє запит, звертається до бази даних;
- Результат повертається у вигляді JSON;
- Інтерфейс оновлюється на основі отриманих даних.

WEB-застосунок реалізований за модульною архітектурою, відповідно до якої кожна ключова функція винесена в окремий логічний блок. Такий підхід забезпечує структурованість програмного коду, спрощує його тестування, супровід та подальший розвиток. Основні функціональні модулі системи наведено нижче:

1. Модуль автентифікації (Auth): призначений для забезпечення процесів реєстрації, входу в систему та контролю доступу. Модуль реалізує перевірку облікових даних користувача, формування токена доступу та розмежування прав відповідно до ролі (студент або викладач).

2. Модуль керування тестами (Test): цей модуль доступний викладачам та забезпечує функціональність створення, редагування, видалення та перегляду тестів. Передбачено можливість додавання запитань із кількома варіантами відповідей, зазначенням правильних варіантів, а також структурування тестів за темами або навчальними дисциплінами.

3. Модуль проходження тестів (Pass Test): функціональність цього модуля орієнтована на студентів. Він надає інтерфейс для вибору доступного тесту, послідовного проходження запитань та надсилання відповідей на сервер. Після завершення тестування відповіді перевіряються, а результат автоматично розраховується та зберігається.

4. Модуль перегляду результатів (Results) : забезпечує відображення результатів тестування. Для студентів доступна інформація про кількість набраних балів та відсоток правильних відповідей. Викладачі мають змогу переглядати аналітичні дані за тестами.

Функціональні модулі WEB-ресурсу взаємодіють між собою відповідно до архітектурної моделі «клієнт-сервер» через стандартизовані REST API-запити.

Комунікація забезпечується за допомогою протоколу HTTPS з використанням токенів авторизації (JWT) для перевірки прав доступу.

Клієнтська частина реалізована з використанням Next.js та відповідає за:

- формування запитів до серверу (отримання тестів, надсилання відповідей, реєстрація користувача тощо);
- обробку відповідей від API;
- рендеринг інтерфейсу на основі отриманих даних.

Серверна частина, побудована на Next.js, виконує роль посередника між користувачем та базою даних. Кожен модуль серверної частини має відповідний набір маршрутів, які реалізують повний набір CRUD-операцій:

- POST /api/auth/register – реєстрація користувача;
- POST /api/auth/login – вхід до системи;
- GET /api/tests – отримання доступних тестів;
- POST /api/tests/:id/pass – надсилання відповідей;
- GET /api/results/:userId – перегляд результатів.

База даних реалізована на PostgreSQL та містить зв’язні таблиці:

- users – інформація про зареєстрованих користувачів;
- tests – мета-дані тестів;
- questions, answers – структура запитань і варіантів відповідей;
- results, user_answers – результати проходження та обрані відповіді.

Комунікація між клієнтською і серверною частинами відбувається через HTTP-запити з використанням стандартних методів: GET, POST, PUT, DELETE. Усі запити, що стосуються захищених ресурсів, супроводжуються токеном авторизації.

Запропонована структура WEB-застосунку для тестування знань студентів, побудована за принципами модульної архітектури, має низку ключових переваг, що забезпечують її ефективність, зручність експлуатації та подальший розвиток. Завдяки поділу системи на окремі функціональні модулі, такі як автентифікація, створення та проходження тестів, обробка результатів тощо, досягається висока

модульність. Це спрощує супровід та дозволяє оперативно оновлювати або доповнювати окремі частини системи без ризику порушення її загальної цілісності.

Архітектура проєкту є масштабованою, що дозволяє у подальшому без значних зусиль розширювати функціонал – наприклад, додавати нові типи тестових завдань, імпортувати результати, інтегрувати аналітичні модулі або реалізувати зв’язок з іншими освітніми платформами. Важливим аспектом є безпечність системи: реалізований механізм авторизації з розподілом прав доступу (через JWT) гарантує захист персональних даних та обмежує доступ до функціоналу згідно з роллю користувача.

Крім того, особлива увага приділена зручності інтерфейсу, який адаптований до різних пристроїв та відповідає принципам доступності. Це дозволяє ефективно використовувати систему як студентам, так і викладачам незалежно від рівня цифрової компетентності. Дотримання шаблону архітектури MVC забезпечує логічну організацію коду, що є перевагою при розробці, тестуванні й подальшій підтримці проєкту. Завдяки використанню стандартних протоколів обміну даними, структура системи також відкрита до майбутньої інтеграції із зовнішніми сервісами, такими як системи дистанційного навчання чи аналітики.

У сукупності ці переваги формують надійну основу для стабільного функціонування та розвитку системи в умовах реального освітнього процесу.

2.2 Розробка UML-діаграми класів

Розробка WEB-ресурсу для тестування знань студентів базується на об’єктно-орієнтованому підході, реалізованому мовою програмування JavaScript із використанням фреймворку Next.js. Хоча JavaScript не має повноцінної підтримки класів у традиційному розумінні, як це властиво мовам типу Java чи

C++, концепції об’єктно-орієнтованого програмування (ООП) можуть бути реалізовані за допомогою класів, компонентів та функцій [14].

Об’єктно-орієнтоване програмування передбачає логічну організацію коду у вигляді структурованих сутностей, які мають власні методи, поля та зони відповідальності. Це значно спрощує підтримку, повторне використання та масштабування проєкту.

У проєкті реалізовано основні принципи ООП:

Інкапсуляція – поділ логіки на окремі компоненти React, які взаємодіють лише через публічні інтерфейси (пропси).

Наслідування – не використовується прямо (через класи), але реалізується через повторне використання логіки у вигляді кастомних хуків і компонентів вищого порядку.

Абстракція – винесення спільної логіки в сервіси, утиліти, контексти.

Поліморфізм – реалізується на рівні пропсів і поведінки компонентів залежно від переданих параметрів.

У клієнтській частині системи реалізовано набір React-компонентів, згрупованих за призначенням. На рисунку 2.2 зображена UML-діаграма класів клієнтської частини WEB-ресурсу для тестування знань студентів.

Розглянемо опис основних компонентів, які беруть участь у логіці створення, налаштування і проходження тестів:

1. CreateTestPage.

Методи (функції):

- handleSubmit() – надсилання форми створення тесту на сервер;
- handleAddPool() – додавання нового пулу запитань.

Призначення: створення нового тесту викладачем.

2. QuestionPoolBlock.

Методи:

- addQuestion(), removeQuestion(index) – керування запитаннями в пулі;

- setDistribution(count) – визначення кількості запитань, які будуть вибрані з пулу.

Призначення: формування пулів запитань із можливістю керування вибіркою.

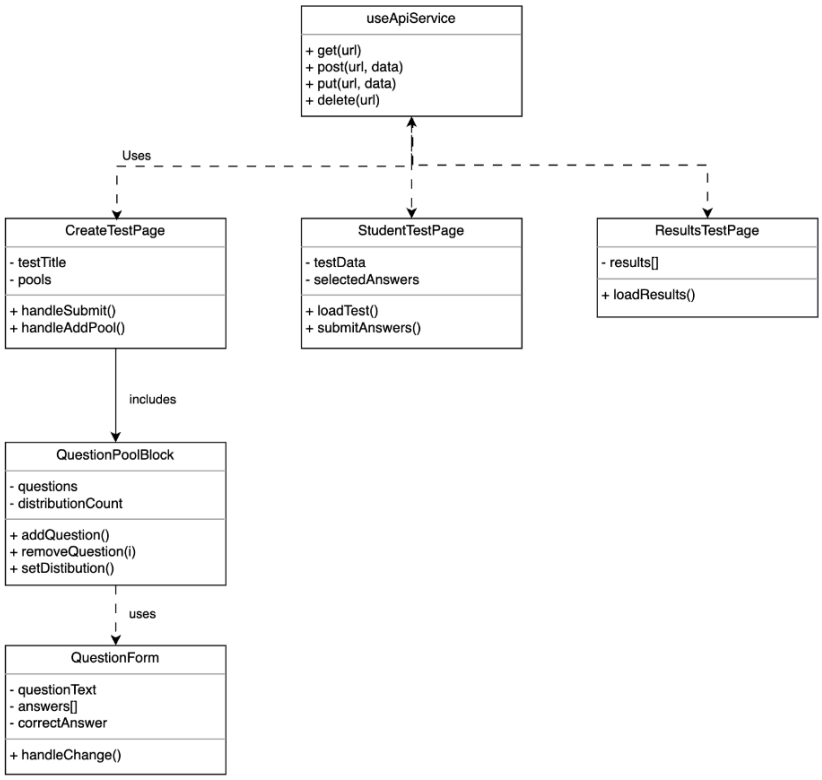

Рисунок 2.2 – UML-діаграма класів клієнтської частини WEB-ресурсу для тестування знань студентів

3. QuestionForm.

Поля:

- questionText, answers, correctAnswer.

Призначення: інтерфейс для створення одного запитання.

4. StudentTestPage.

Методи:

- loadTest() – завантаження структури тесту;
- submitAnswers() – надсилання відповідей студента.

Призначення: проходження тесту студентом.

5. ResultsPage.

Методи:

- loadResults() – отримання результатів з сервера.

Призначення: відображення результатів студенту або викладачу.

6. useApiService (кастомний хук).

Методи:

- get(), post(), put(), delete() – універсальні методи для роботи з API.

Призначення: забезпечення взаємодії між клієнтом і сервером.

Серверна логіка WEB-ресурсу реалізована за допомогою вбудованих API-роутів фреймворку Next.js. Кожен API-роут є окремим файлом, який експортує функцію-обробник, що викликається під час відповідного HTTP-запиту (GET, POST, PUT, DELETE). Це дозволяє реалізувати серверну функціональність без використання зовнішніх серверних фреймворків, таких як Express [15].

На рисунку 2.3 зображена UML-діаграма класів серверної частини WEB-ресурсу для тестування знань студентів.

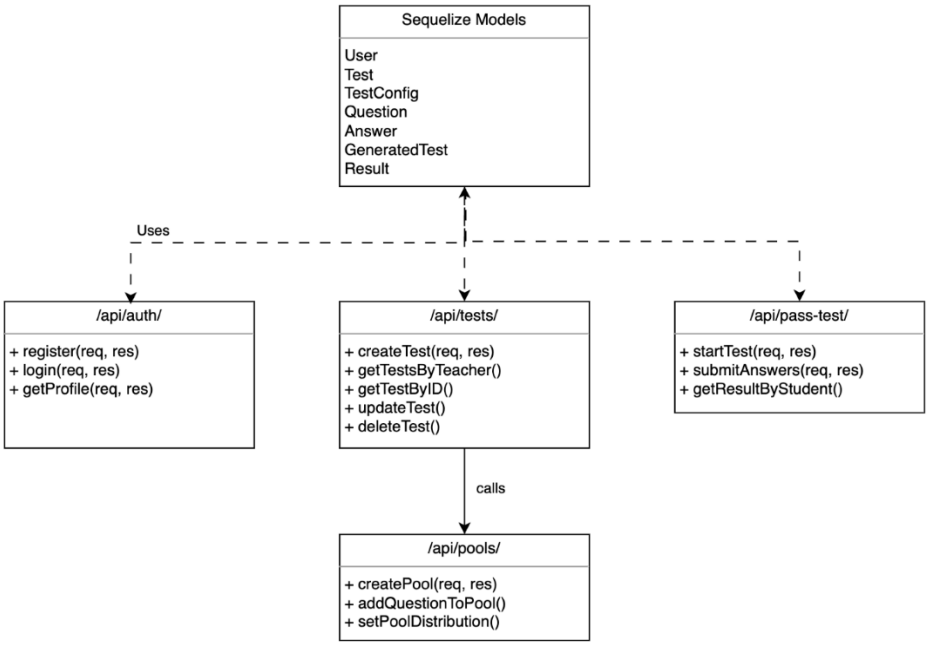

Рисунок 2.3 – UML-діаграма класів серверної частини WEB-ресурсу для тестування знань студентів

Кожен API-роут відповідає за обробку певного типу запитів та виконує одну із CRUD-операцій, взаємодіючи з базою даних через ORM Sequelize [16].

1. /api/auth/* – обробка автентифікації.

Функції:

- register(req, res) – створює нового користувача (роль: студент або викладач), зберігає облікові дані після валідації;
- login(req, res) – перевіряє дані входу, формує та повертає JWT-токен;
- getProfile(req, res) – повертає профіль авторизованого користувача.

Призначення: керування доступом до системи, аутентифікація користувачів.

2. /api/tests/* – управління тестами.

Функції:

- createTest(req, res) – зберігає новий тест із загальними параметрами: назва, опис, автор;
- getTestsByTeacher(req, res) – повертає тести, створені певним викладачем;
- getTestById(req, res) – завантажує повну структуру тесту, включно з питаннями і пулами;
- updateTest(req, res) – оновлює тест;
- deleteTest(req, res) – видаляє тест.

Призначення: створення, редагування, перегляд і видалення тестів викладачами.

3. /api/pools/* – керування пулами запитань.

Функції:

- createPool(req, res) – створює новий тематичний або логічний пул запитань;
- addQuestionToPool(req, res) –додає питання до вказаного пулу;
- setPoolDistribution(req, res) – задає кількість запитань, які мають обиратись із пулу при генерації тесту.

Призначення: формування гнучкої логіки генерації тестів на основі структурованих блоків запитань.

4. /api/pass-test/* – проходження тестів студентами.

Функції:

- startTest(req, res) – формує для студента тест із випадковими запитаннями відповідно до заданих параметрів;
- submitAnswers(req, res) – приймає відповіді, перевіряє правильність, обчислює результат, зберігає у таблицях results і user_answers;
- getResultByStudent(req, res) – повертає результати для конкретного студента або аналітику для викладача.

Призначення: реалізація бізнес-логіки проходження тесту та формування результатів.

Усі API-обробники використовують ORM Sequelize для доступу до бази даних PostgreSQL. Основні сутності (моделі):

- User – дані користувача, включно з роллю;
- Test – опис тесту та його властивості;
- QuestionPool – пул запитань, зв’язаний з тестом;
- Question, Answer – зміст запитань і варіанти відповідей;
- Result, UserAnswer – результати тестування та збережені відповіді.

2.3 Розробка ER-моделі бази даних

У межах системи тестування знань до універсального відношення включено атрибути, що описують такі сутності: користувачі, тести, запитання, відповіді, результати тестування, конфігурації тестів та їх генерації. Універсальне відношення виглядає так: R (ID користувача, ім’я користувача, email, пароль, роль, ID тесту, назва тесту, опис тесту, активність, ID запитання, текст запитання, тип запитання, варіанти, правильні відповіді, ID відповіді, відповідь студента,

правильність, ID результату, оцінка, дата завершення, ID конфігурації, кількість запитань, ID генерації тесту).

Ступінь відношення – 23.

У таблиці 2.1 наведено типи зв’язків між основними сутностями системи.

Таблиця 2.1 – Характеристики відношень між сутностями WEB-ресурсу для тестування знань студентів

Ім’я сутності 1	Ім’я сутності 2	Тип зв’язку	Клас належності
Users	Tests	1:Б	Обов’язковий
Users	TestResults	1:Б	Обов’язковий
Users	Answers	1:Б	Обов’язковий
Tests	Questions	1:Б	Обов’язковий
Tests	TestConfigs	1:Б	Обов’язковий
Tests	GeneratedTests	1:Б	Обов’язковий
Tests	TestResults	1:Б	Обов’язковий
Questions	Answers	1:Б	Обов’язковий
Questions	Tests	Б:1	Обов’язковий
Answers	Users	Б:1	Обов’язковий
Answers	Questions	Б:1	Обов’язковий
TestConfigs	Tests	Б:1	Обов’язковий
GeneratedTests	Tests	Б:1	Обов’язковий
TestResults	Users	Б:1	Обов’язковий
TestResults	Tests	Б:1	Обов’язковий

ER-модель відображає такі основні сутності:

- Users – зберігає дані про користувачів, включаючи роль (студент/викладач);
- Tests – містить інформацію про тести (назва, активність, автор);
- Questions – зберігає формулювання запитань, типи та правильні відповіді;
- Answers – відповіді студентів, подані під час проходження тестів;
- TestResults – підсумкові результати, бали, спроби;
- TestConfigs – параметри конфігурації тестів за пулами;
- GeneratedTests – унікальні згенеровані копії тестів з конкретними запитаннями.

На рисунку 2.4 зображено ER-модель бази даних WEB-ресурсу для тестування знань студентів.

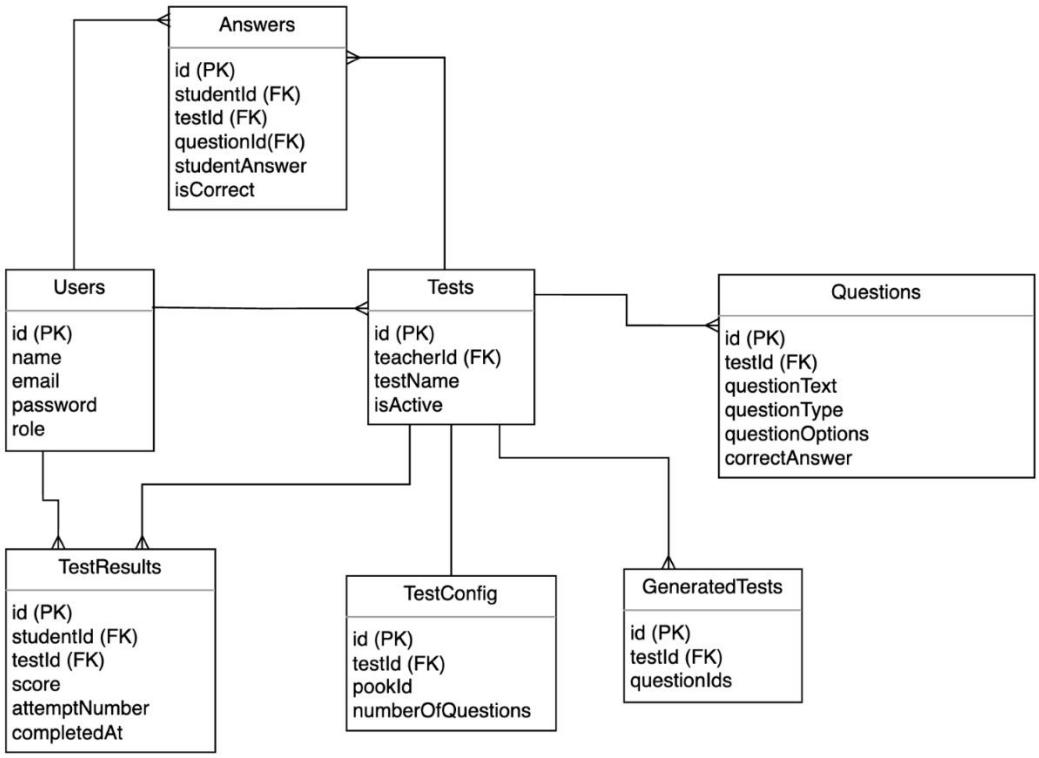

Рисунок 2.4 – ER-модель бази даних

Розроблена ER-модель бази даних WEB-ресурсу для тестування знань студентів включає сім основних сутностей: Users, Tests, Questions, Answers, TestResults, TestConfig та GeneratedTests. Кожна з цих сутностей відповідає за зберігання конкретних типів даних та має визначені зв’язки з іншими елементами системи, що забезпечує логічну цілісність, узгодженість та масштабованість бази даних.

- Users – зберігає дані зареєстрованих користувачів системи: ім’я, електронну пошту, пароль та роль (викладач або студент). Один користувач може створювати декілька тестів, а також проходити їх, що реалізується через зв’язки 1:Б з таблицями Tests, TestResults та Answers.

- Tests – містить основну інформацію про тести, зокрема назву, активність та ідентифікатор викладача, що його створив. Тест може містити декілька

запитань (Questions), результати проходження (TestResults), конфігурацій (TestConfig) та варіантів генерації (GeneratedTests), що забезпечується зв’язками 1:Б.

- Questions – описує запитання, які належать до певного тесту, з полями для тексту запитання, типу, набору варіантів відповіді (у форматі JSONB) та правильної відповіді. Кожне запитання може мати декілька відповідей студентів, збережених у таблиці Answers.

- Answers – зберігає відповіді студентів на конкретні запитання в межах певного тесту. Ключові поля включають ідентифікатори студента, тесту, запитання, саму відповідь (у форматі JSONB) та ознаку правильності. Зв’язки з таблицями Users, Tests та Questions реалізовані за допомогою зовнішніх ключів.

- TestResults – містить інформацію про проходження тесту студентом, включаючи оцінку, номер спроби та дату завершення. Кожен запис пов’язаний із конкретним студентом та тестом (зв’язки Б:1).

- TestConfig – зберігає налаштування, що визначають, скільки запитань потрібно обрати з конкретного пулу під час генерації тесту. Забезпечує зв’язок 1:Б із таблицею Tests.

- GeneratedTests – містить унікальні згенеровані копії тестів, які зберігають список ідентифікаторів запитань (у форматі JSONB). Забезпечує можливість створення динамічних варіантів тестів для кожного проходження.

Усі зв’язки в ER-моделі реалізовані через зовнішні ключі, що дозволяє забезпечити узгодженість даних та підтримку складної логіки проходження, перевірки й аналізу тестів.

2.4 Розробка схеми алгоритму роботи WEB-ресурсу для тестувань знань студентів

Алгоритм – це чітко впорядкована послідовність дій, яка забезпечує досягнення певної мети при виконанні конкретного завдання. У сфері розробки програмного забезпечення алгоритм визначає логіку функціонування модуля або всієї системи, а його блок-схема – наочне графічне представлення цієї логіки.

Для зручності розуміння логіки роботи WEB-ресурсу для тестування знань студентів розроблено кілька алгоритмів, кожен з яких описує один із основних сценаріїв використання системи: від реєстрації користувача до проходження тесту та перегляду результатів. Візуалізація таких процесів у вигляді блок-схем дає змогу краще структурувати програмну логіку, спростити розробку та забезпечити узгодженість дій між клієнтською та серверною частинами застосунку.

Для опису алгоритмів у цьому підрозділі використовується класичний підхід до побудови блок-схем із загальноприйнятими умовними позначеннями. Це дозволяє чітко передати послідовність дій, що виконуються користувачем і системою на кожному етапі взаємодії з WEB-ресурсом.

Для опису алгоритму роботи WEB-ресурсу для тестування знань студентів було вибрано представлення алгоритму у вигляді блоків схеми. Схему алгоритму роботи WEB-ресурсу для тестування знань студентів зображено на рисунку 2.5.

І випадок

Алгоритм реєстрації користувача. Процес реєстрації користувача є початковим етапом взаємодії з системою. Він передбачає створення нового облікового запису в базі даних та ініціалізацію сесії за допомогою JWT токенів. Це дозволяє користувачеві одразу після реєстрації перейти до використання WEB-ресурсу без повторної авторизації.

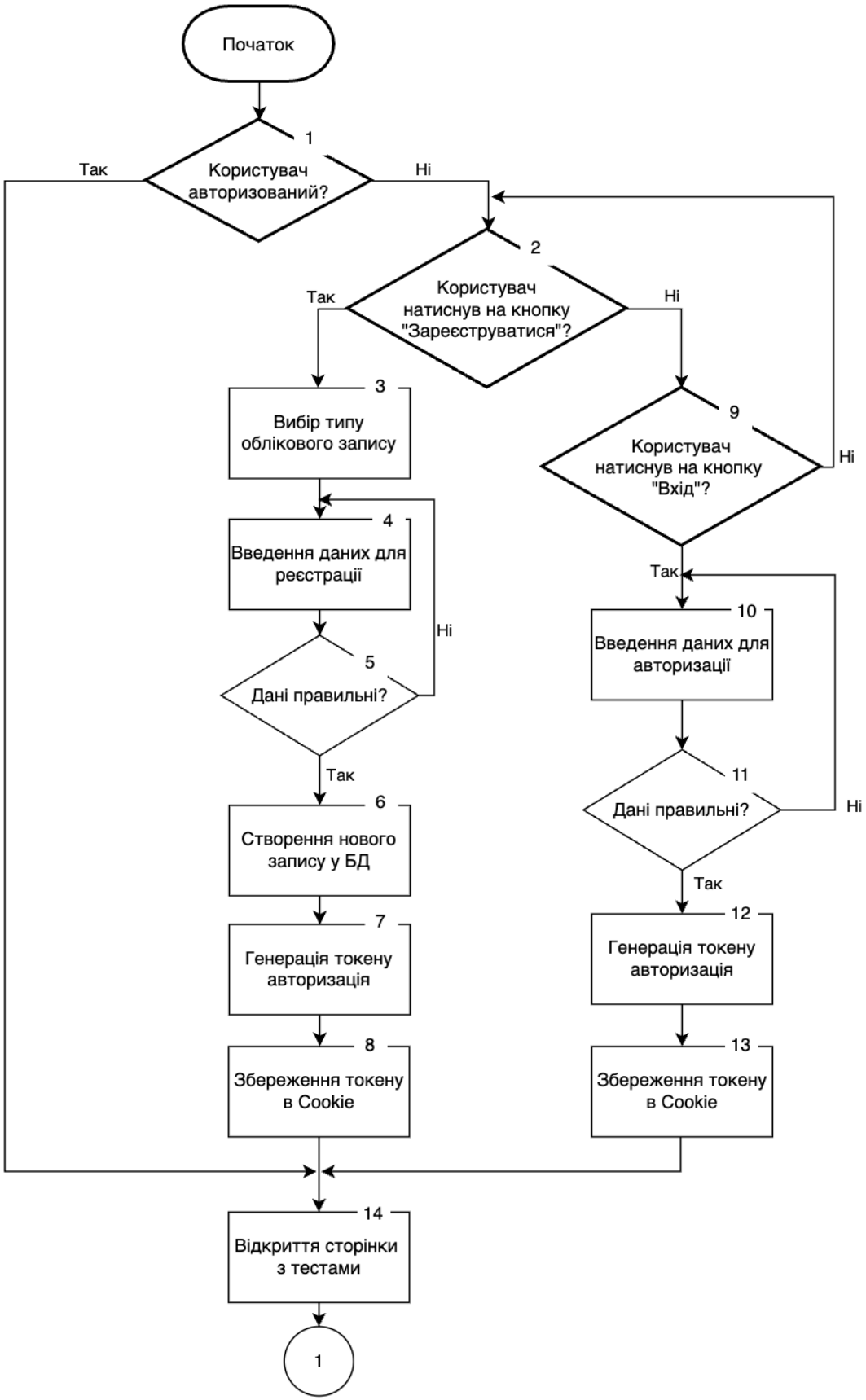

Рисунок 2.5 – Схема алгоритму роботи WEB-ресурсу для тестування знань студентів

Рисунок 2.5, аркуш 2

Алгоритм складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку і якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 2. Користувач натискає на кнопку «Зареєструватися».

Крок 3. Користувач обирає тип облікового запису (студент або викладач).

Крок 4. Користувач вводить необхідні дані для реєстрації: ім’я, електронну пошту, пароль.

Крок 5. Відбувається перевірка коректності даних.

Крок 6. Якщо дані правильні – відбувається створення нового запису у базу даних.

Крок 7. Сервер генерує токен для авторизованої сесії.

Крок 8. Токен зберігається на стороні клієнта – в HttpOnly Cookie.

Крок 14. Авторизований користувач потрапляє на сторінку з тестами.

II випадок

Алгоритм авторизації користувача. Авторизація користувача – це процес входу в систему за допомогою раніше створеного облікового запису. У цій системі після успішної авторизації користувач отримує токен, який дозволяє йому взаємодіяти з WEB-ресурсом протягом сесії. Токен зберігається у Cookie, що забезпечує автоматичну перевірку прав доступу до закритих маршрутів та захищену роботу з особистими даними.

Алгоритм складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку і якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 9. Користувач натискає на кнопку «Вхід».

Крок 10. Користувач вводить необхідні дані для авторизації: електронну пошту, пароль.

Крок 11. Відбувається перевірка коректності даних.

Крок 12. Сервер генерує токен для авторизованої сесії.

Крок 13. Токен зберігаюється на стороні клієнта – в HttpOnly Cookie.

Крок 14. Авторизований користувач потрапляє на сторінку з тестами.

III випадок

Алгоритм створення тесту викладачем.

Кроки алгоритму створення тесту:

Крок 1. Користувач відкриває WEB-сторінку і якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 14. Авторизований користувач потрапляє на сторінку з тестами.

Крок 15. Відбувається перевірка ролі користувача.

Крок 16. Викладачу доступна кнопка створення нового тесту та список раніше створених тестів.

Крок 17. Якщо викладач натиснув на кнопку створення нового тесту, користувач вводить основну інформацію про тест (назву тесту, короткий опис).

Крок 18. Користувач обирає конфігурація тесту (кількість пулів запитань та кількість питань в кожному з пулів).

Крок 19. Користувач наповнює кожен пул питаннями різного типу.

Крок 20. Користувач натискає кнопку «generate test» (Згенерувати тест).

Крок 21. Система надсилає дані на сервер, де створюється запис тесту в базі даних.

Крок 21. Система повертає посилання за яким можна пройти цей тест.

IV Випадок

Алгоритм проходження тесту студентом.

Кроки алгоритму проходження тесту:

Крок 1. Користувач відкриває WEB-сторінку та якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 14. Авторизований користувач потрапляє на сторінку з тестами.

Крок 15. Відбувається перевірка ролі користувача.

Крок 23. Студенту відображаються список його пройдених тестів.

Крок 24. Студент має можливість переглядати результати пройдених тестів.

Крок 25. Якщо студент має посилання на проходження нового тесту – переходить за посилання для проходження тесту.

Крок 26. Система формує індивідуальний варіант тесту, генеруючи випадкові запитання відповідно до конфігурації пулів.

Крок 27. Студент по черзі відповідає на запитання різних типів (одна/декілька правильних відповідей, відкриті відповіді тощо).

Крок 28. Після завершення натискає кнопку «Submit Test» (Завершити тест).

Крок 29. Відповіді надсилаються на сервер для збереження та перевірки.

Крок 30. Система обчислює результат та зберігає його в базі даних.

Крок 31. Студенту відображається повідомлення про завершення тестування та результати.

V Випадок

Алгоритм перегляду результатів тесту та можливість редагування тесту викладачем.

Кроки алгоритму:

Крок 1. Користувач відкриває WEB-сторінку і якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 14. Авторизований користувач потрапляє на сторінку з тестами.

Крок 15. Відбувається перевірка ролі користувача.

Крок 16. Викладачу доступна кнопка створення нового тесту та список раніше створених тестів.

Крок 32. Серед списку існуючих тестів, викладач обирає один і натискає на нього.

Крок 33. Система відображає інформацію про тест.

Крок 34. Викладачу доступна інформація про тест і можливість редагування тесту.

Крок 35. Якщо викладач обирає опцію змінити наповнення тесту, система відображає інтерфейс для редагування/відображення нових питань.

Крок 36. Якщо користувач не обирає опцію змінити наповнення тесту, система відображає викладачу результати проходження тесту (ім’я студента, дата проходження, кількість правильних відповідей, підсумковий бал).

Алгоритм всіх випадків зображено на рисунку 2.5.

2.5 Висновки до розділу 2

У цьому розділі було спроєктовано ключові компоненти WEB-ресурсу для тестування знань студентів. Було визначено загальну структуру застосунку, яка включає окрему функціональність для різних типів користувачів – студентів і викладачів. Такий підхід забезпечує зручність використання та масштабованість системи в процесі подальшого розвитку.

Створені UML-діаграми класів клієнтської та серверної частин системи дали змогу візуально описати логіку програмного забезпечення, представити взаємозв'язки між об'єктами, а також деталізувати методи, які забезпечують бізнес-логіку застосунку. UML-діаграми слугують важливою технічною документацією, що полегшує підтримку та доопрацювання коду.

Створено ER-модель бази даних, яка охоплює сутності користувачів, тестів, запитань, відповідей, результатів, конфігурацій і генерацій. Модель підтримує як рольову диференціацію (викладач/студент), так і динамічне формування тестів, забезпечуючи масштабовану і структуровану архітектуру.

Особливу увагу приділено побудові схем алгоритмів роботи системи. Було описано п'ять основних процесів: реєстрація користувача, авторизація, створення тесту викладачем, проходження тесту студентом та перегляд результатів. Для кожного алгоритму визначено послідовність дій користувача та реакцію системи, а також сформовано інтегровану схему алгоритму, яка демонструє узгоджену логіку всієї системи. Це дозволяє наочно зрозуміти бізнес-процеси та мінімізує ризик помилок під час реалізації.

Комплексна розробка структури, діаграм класів та алгоритмів роботи дає змогу забезпечити надійність, зрозумілість та ефективність функціонування системи. Вона створює основу для успішної реалізації всіх технічних вимог та забезпечує легку адаптацію проєкту в майбутньому.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ СТУДЕНТІВ

3.1 Обґрунтування вибору мови та бібліотек програмування

Розробка WEB-ресурсу для тестування знань студентів передбачає створення як клієнтської, так і серверної частини. Для реалізації цього програмного продукту було обрано React для клієнта та Next.js як фреймворк, що об’єднує клієнтську і серверну логіку в єдиному середовищі. Також застосовано мови програмування JavaScript і TypeScript у гібридному режимі [17].

У цьому розділі буде обґрунтовано вибір технологій для обох частин системи, а також проведено їх порівняння з популярними альтернативами.

Основним фреймворком для реалізації клієнтської частини було обрано React, який забезпечує компонентно-орієнтовану архітектуру, гнучкість та активну підтримку спільноти. Частина компонентів і сторінок написана на JavaScript, інша – з використанням TypeScript, що дозволяє поступово впроваджувати статичну типізацію без потреби повної міграції проєкту [18].

Порівняємо React з іншими фрейморками, такими як Vua та Angular.

React, Vue та Angular – три найпопулярніші фреймворки для фронтенд-розробки. У таблиці 3.1 наведено їх порівняння за ключовими параметрами [19].

Як видно з таблиці 3.1, React забезпечує ідеальний баланс між гнучкістю, продуктивністю та масштабованістю. Його гнучка архітектура дозволила інтегрувати з ним Next.js – фреймворк для створення повноцінних React-застосунків із підтримкою SSR (server-side rendering) та API [20].

Для реалізації серверної частини було обрано Next.js API Routes, що дозволяє обробляти запити без потреби створення окремого серверу. У поєднанні з PostgreSQL та ORM Sequelize, це дозволило реалізувати повноцінну серверну логіку всередині одного фреймворку.

Таблиця 3.1 – Порівняння фронтенд-фреймворків

Характеристика	**React**	**Vue**	**Angular**
Парадигма	Бібліотека для UI	Повноцінний фреймворк	Повноцінний фреймворк
Простота у вивченні	+	++	-
Документація	++	+	+
Гнучкість архітектури	++	+	-
Типізація	За бажанням (TS)	За бажанням (TS)	Вбудована (TS)
Спільнота та підтримка	++	+	+
Швидкість рендерингу	++	++	+

Порівняємо Next.js з Express.js та Node.js [21].

У таблиці 3.2 наведено порівняння Next.js з класичними підходами розробки серверної частини на основі Node.js та Express.js.

Таблиця 3.2 – Порівняння фреймворків для серверної логіки

Характеристика	**Next.js**	**Express.js**	**Node.js**
SSR (Server-Side Rendering)	+	-	-
Побудова UI-клієнта	+	-	-
Підтримка API	+	+	+
Статична генерація сторінок	+	-	-
Інтеграція з React	+	-	-
Простота налаштування	+	+	-
Підтримка full-stack архітектури	+	-	-

Next.js поєднує функції клієнтського та серверного рендерингу, що робить його ідеальним для розробки повноцінних WEB-застосунків без поділу на окремі

репозиторії для клієнтської і серверної розробки. Такий підхід значно спрощує розгортання, масштабування та підтримку проєкту.

У проєкті також використовуються такі бібліотеки:

- TailwindCSS – для швидкого й адаптивного UI;
- Formik + Yup – для обробки форм і валідації;
- Framer Motion – для анімацій;
- Lucide React, React Icons – для SVG-іконок;
- Sequelize – для ORM-роботи з базою даних PostgreSQL;
- dotenv, bcryptjs, jsonwebtoken – для роботи з безпекою, сесіями та авторизацією.

Обрана комбінація React + Next.js + PostgreSQL + TypeScript/JavaScript є ідеальною для створення сучасного, масштабованого та підтримуваного WEB-застосунку. Такий стек дозволив об’єднати клієнтську і серверну частини в одному фреймворку, спростити архітектуру, зменшити час на розробку та забезпечити високу якість продукту.

3.2 Обґрунтування вибору мови програмування для управління організованою базою даних

Для управління організованими базами даних у цьому проєкті використовувалась мова програмування SQL (Structured Query Language), що є галузевим стандартом для взаємодії з реляційними базами даних. У якості системи управління базами даних було обрано PostgreSQL, а для взаємодії з нею – ORM-бібліотека Sequelize, яка дозволяє зручно будувати SQL-запити за допомогою JavaScript/TypeScript.

Розглянемо дві популярні мови програмування для управління організованими базами даних: SQL (Structured Query Language) та OQL (Object Query Language).

SQL – це мова структурованих запитів, яка широко використовується для роботи з реляційними базами даних, де дані представлені у вигляді таблиць. Вона дозволяє виконувати базові операції над даними: створення, читання, оновлення та видалення (CRUD). Мова має зрозумілий синтаксис, який базується на англомовних ключових словах (SELECT, INSERT, UPDATE, DELETE), що полегшує її вивчення і використання. SQL добре підтримується у більшості мов програмування, включно з JavaScript і TypeScript [22].

OQL, або об’єктна мова запитів, була розроблена як розширення SQL для роботи з об’єктно-орієнтованими базами даних. Основною перевагою є можливість працювати з об’єктами напряму – без потреби перетворювати їх у таблиці. Однак, на відміну від SQL, OQL рідко підтримується стандартними системами управління БД, а її використання потребує застосування специфічних об’єктно-орієнтованих БД, які не мають такої популярності, стабільності та інструментальної підтримки, як реляційні [23].

У таблиці 3.3 наведено порівняння основних характеристик SQL та OQL.

Таблиця 3.3 – Порівняння SQL та OQL мов програмування для управління організованою базою даних

Характеристика	**SQL**	**OQL**
Модель даних	Реляційна	Об’єктна
Синтаксис	Простий і декларативний	Більш складний, підтримка виразів
Запити	CRUD-операції з таблицями	Робота з об’єктами
Призначення	Реляційні СУБД	Об’єктно-орієнтовані СУБД
Нормалізація даних	Необхідна	Частково непотрібна (зв’язки)
Відношення між сутностями	Через зовнішні ключі	Через посилання між об’єктами

Враховуючи, що в застосунку зберігаються структури даних, притаманні класичним реляційним базам (користувачі, тести, відповіді, результати),

найдоцільнішим вибором є реляційна модель з використанням SQL. У зв’язку з цим було обрано PostgreSQL як надійну, масштабовану, безкоштовну систему управління базами даних з відкритим кодом.

Для зручної взаємодії з БД у коді було використано ORM Sequelize, яка дозволяє:

- створювати моделі з чіткою типізацією (через TypeScript);
- виконувати запити за допомогою методів;
- легко проводити міграції та seed-операції;
- уникати ручного написання SQL там, де це не потрібно.

Це значно спрощує управління структурованими даними та підвищує гнучкість і безпечність взаємодії з базою.

3.3 Обґрунтування вибору середовища розробки

Інтегроване середовище розробки (IDE – Integrated Development Environment) – це спеціалізоване програмне забезпечення, яке надає розробникам комплекс інструментів для написання, перевірки, налагодження та тестування програмного коду. Зазвичай IDE включає в себе текстовий редактор, систему контролю версій, налагоджувач, термінал, підтримку розширень та інтеграцію з інструментами для роботи з базами даних і API.

Під час реалізації проєкту було проаналізовано такі популярні середовища розробки: Visual Studio Code, WebStorm, Atom.

Visual Studio Code (VS Code) – сучасний безкоштовний редактор коду від Microsoft. Підтримує JavaScript, TypeScript, Node.js, а також має розвинену систему розширень. У цьому проєкті використовувались розширення для підтримки Next.js, PostgreSQL, ESLint, TailwindCSS та інші.

WebStorm – платне IDE від JetBrains для роботи з JavaScript і TypeScript. Має вбудовані інструменти аналізу коду, тестування та налагодження, але потребує ліцензії.

Atom – редактор від GitHub, який працює на основі Electron. Підтримує кастомізацію через плагіни, але поступається іншим IDE за стабільністю й продуктивністю.

Порівняння їх характеристик наведено в таблиці 3.4.

Таблиця 3.4 – Порівняння середовищ розробки Visual Studio Code, WebStorm, Atom

Характеристика	Visual Studio Code (VS Code)	WebStorm	Atom
Вартість	Безкоштовний	Платний	Безкоштовний
Продуктивність	Легкий і швидкий	Висока, але «важчий»	Повільніший із багатьма плагінами
Інтерфейс користувача	Гнучкий, мінімалістичний	Насичений, але складніший	Простий, обмежена гнучкість
Підтримка TypeScript	Відмінна	Відмінна	Посередня (через плагіни)
Підтримка Node.js	Повна	Повна	Часткова
Розширення та плагіни	10 000+	Вбудовано	Базова підтримка
Інструменти для Next.js	Так (через плагіни)	Так (вбудовано)	Немає прямої підтримки
Інтеграція з Git	Вбудована	Вбудована	Через плагіни

Враховуючи безкоштовну ліцензію, гарну підтримку TypeScript, Node.js, PostgreSQL, а також наявність розширень для роботи з Next.js, було обрано Visual Studio Code як основне середовище розробки для цього проєкту.

3.4 Розробка клієнтської та серверної частин

Для реалізації WEB-ресурсу тестування знань студентів було обрано Next.js як фреймворк для клієнтської та серверної частин одночасно (Full Stack), що дозволяє ефективно поєднувати серверний рендеринг, REST API та

інтерактивний інтерфейс користувача в єдиному середовищі. Додатково у проєкті використовуються React, Tailwind CSS, Sequelize, PostgreSQL, JWT та інші сучасні бібліотеки.

Серверна логіка WEB-ресурсу реалізована з використанням Next.js API Routes, які дозволяють створювати серверні функції у форматі REST-контролерів без необхідності окремого сервера. Всі API-роути розташовано у директорії /pages/api/ і обробляють запити згідно з HTTP-методами (GET, POST тощо).

Серверна частина реалізована з використанням таких ключових технологій:

- Sequelize – ORM для роботи з PostgreSQL;
- bcryptjs – для хешування паролів і перевірки автентичності;
- jsonwebtoken – для створення та перевірки JWT токенів;
- dotenv – для роботи з конфігураційними змінними середовища;
- cookie – для зберігання токена у HTTP-only cookie;
- middleware – для обробки авторизації в запитах.

Розглянемо фрагменти які відповідають за аторизацію та автентифікацію користувача із серверної сторони WEB-ресурсу.

Модуль login.js виконує перевірку облікових даних користувача та створює сесію у вигляді JWT:

```
const user = await User.findOne({ where: { email }, attributes: ["id", "email", "password", "role"] });
const isMatch = await bcrypt.compare(password, user.getDataValue("password"));
const token = jwt.sign({ id, email, role }, process.env.JWT_SECRET, { expiresIn: "10h" });
res.setHeader("Set-Cookie", serialize("token", token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production",
  sameSite: "strict",
  path: "/",
  maxAge: 3600,
```

```
}));
```

Токен зберігається у cookie, захищеній від доступу з JavaScript (httpOnly), що мінімізує ризик XSS-атак.

У папці utils реалізовано модуль authMiddleware, який використовується для захисту приватних маршрутів:

```
import jwt from "jsonwebtoken";
import { parse } from "cookie";
export default function authMiddleware(req, res, next) {
  const cookies = parse(req.headers.cookie || "");
  const token = cookies.token;
  if (!token) {
    return res.status(401).json({ message: "Unauthorized: No token" });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ message: "Unauthorized: Invalid token" });
  }
}
```

Цей middleware додає до запиту об’єкт user, що містить id, email, role – витягнуті з JWT. Він використовується в маршрутах, таких як /api/profile, щоб обмежити доступ лише авторизованим користувачам.

Уся взаємодія з базою PostgreSQL виконується через ORM Sequelize. З’єднання з базою ініціалізується у файлі db.ts, де також імпортуються й описуються моделі, зокрема:

```
import { Sequelize } from "sequelize";
```

```
import dotenv from "dotenv";
dotenv.config();
export const sequelize = new Sequelize(process.env.DB_NAME,
process.env.DB_USER, process.env.DB_PASSWORD, {
  host: process.env.DB_HOST,
  dialect: "postgres",
});
```

Модель користувача user.model.ts описує таблицю Users:

```
import { Model, DataTypes } from "sequelize";
import { sequelize } from "./db";
export class User extends Model {}
User.init({
  name: DataTypes.STRING,
  email: DataTypes.STRING,
  password: DataTypes.STRING,
  role: DataTypes.STRING,
}, { sequelize, modelName: "user" });
```

Усі ендпоінти серверної частини згруповано у папці pages/api/. Вони виконують функції автентифікації, управління користувачами, зчитування профілю, обробки відповідей і результатів тестів. Кожен файл у папці api представляє окремий маршрут, який обробляє запити згідно з HTTP-методами.

Крім зазначених маршрутів, у системі також типово присутні API-обробники, які можна реалізовувати у вигляді таких файлів:

- /api/register – для реєстрації користувачів;
- /api/tests – створення тестів викладачем;
- /api/questions – додавання запитань до тесту;
- /api/submit – передача відповідей студентом;
- /api/results – обчислення та повернення результату.

Кожен з цих маршрутів зазвичай має обробку POST або GET запитів, залежно від типу взаємодії.

Розглянемо обробник, який дозволяє створити новий тест викладачу, ідентифікатор якого отримується з JWT-токена:

```
export default async function handler(req, res) {
  if (req.method === "POST") {
    authMiddleware(req, res, async () => {
      const { name, description } = req.body;
      const test = await Test.create({ name, description, teacherId: req.user.id });
      res.status(201).json(test);
    });
  }
}
```

Наступний обробник відповідає за обробку та зберігання відповідей користувача. Відповіді студента передаються на сервер у вигляді масиву об’єктів та зберігаються у таблиці Answers:

```
export default async function handler(req, res) {
  if (req.method === "POST") {
    authMiddleware(req, res, async () => {
      const { testId, answers } = req.body;
      await Promise.all(answers.map(ans => Answer.create({ ...ans, userId: req.user.id })));
      res.status(200).json({ message: "Answers submitted" });
    });
  }
}
```

Папка api/ у застосунку виконує роль вхідної точки для взаємодії клієнта з сервером. Кожен маршрут структуровано відповідно до REST-парадигми, з підтримкою авторизації, валідації та захисту сесій. Використання authMiddleware як загального механізму перевірки дозволяє централізовано контролювати доступ до даних. Така реалізація сприяє безпечній, масштабованій та логічно впорядкованій обробці всіх запитів.

Клієнтська частина WEB-ресурсу для тестування знань студентів реалізована з використанням Next.js 15 та React 19, що забезпечує гнучку архітектуру, високий рівень інтерактивності та підтримку SSR (серверного рендерингу). Стилізація інтерфейсу здійснюється за допомогою Tailwind CSS, а керування станом форм і валідацією – через Formik та Yup.

Проєкт побудований відповідно до файлової структури Next.js:

- index.jsx – головна сторінка (наприклад, привітання, вхід);
- login.jsx – авторизація;
- profile.jsx – перегляд особистих даних;
- [id].jsx – сторінка конкретного тесту;
- [testCode].jsx – результати проходження тесту.

Маршрутизація відбувається динамічно, на основі параметрів URL, що дозволяє створювати багато унікальних сторінок без дублювання коду.

Авторизація реалізована через JWT-токени, які зберігаються у cookie після виклику API /api/login. Валідація даних здійснюється бібліотекою Yup, а збирання та обробка – за допомогою Formik:

```
<Formik
  initialValues={{ email: "", password: "" }}
  validationSchema={Yup.object({
    email: Yup.string().email("Невірна пошта").required("Обов’язково"),
    password: Yup.string().min(6, "Мінімум 6 символів").required("Обов’язково"),
  })}
  onSubmit={handleLogin}
>
```

У разі успішного входу відбувається перенаправлення на захищену сторінку profile.

Взаємодія з сервером здійснюється через стандартний fetch або обгорнуті виклики axios (за наявності):

```
useEffect(() => {
```

```
  fetch("/api/profile")
    .then((res) => res.json())
    .then(setUser)
    .catch(() => router.push("/login"));
}, []);
```

У клієнтській частині передбачено модальні діалоги, що використовуються для підтвердження важливих дій користувача – наприклад, видалення тесту, виходу з профілю, завершення проходження тесту.

Модальні вікна реалізовані через умовне рендерення з використанням useState:

```
const [showModal, setShowModal] = useState(false);
{showModal && (
  <div className="fixed inset-0 bg-black/40 flex items-center justify-center">
    <div className="bg-white rounded-xl p-6">
      <p>Ви впевнені, що хочете завершити тест?</p>
      <div className="flex justify-end mt-4">
        <button onClick={() => setShowModal(false)}>Скасувати</button>
        <button onClick={confirmFinish}>Підтвердити</button>
      </div>
    </div>
  </div>
)}
```

Такі компоненти дозволяють: уникнути випадкових дій, централізувати логіку підтвердження, покращити UX завдяки фокусованій взаємодії.

Для збереження стану авторизованого користувача та повторного використання його в різних компонентах застосовано React Context API. Це дозволяє уникнути «prop drilling» (передачі даних через багато рівнів компонентів) і централізувати доступ до даних сесії:

```
export const AuthContext = createContext();
export const AuthProvider = ({ children }) => {
```

```
  const [user, setUser] = useState(null);
  const login = (userData) => setUser(userData);
  const logout = () => setUser(null);
  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

У будь-якому компоненті можна отримати доступ до контексту через useContext(AuthContext):

```
const { user, logout } = useContext(AuthContext);
```

Цей підхід дозволяє зберігати стан сесії між сторінками, оновлювати його в разі входу/виходу та обмежувати доступ до певних маршрутів.

Клієнтська частина застосунку побудована з використанням сучасних інструментів і патернів, що забезпечують: логічне розділення сторінок, безпечне зберігання токенів, динамічне оновлення інтерфейсу, зручну взаємодію з сервером, захищені маршрути.

Використання модальних вікон підвищує якість UX, а застосування Context API дозволяє організувати ефективне управління глобальним станом авторизації. Це створює стабільну основу для масштабування функціоналу, в тому числі – додавання нових ролей, сторінок або механізмів доступу.

3.5 Тестування роботи програми та аналіз результатів

Тестування програмного продукту є критичним етапом у процесі розробки, оскільки воно дозволяє переконатися у правильності реалованої функціональності, стабільності, безпеці та відповідності вимогам користувача. Основна мета тестування – виявлення можливих помилок і дефектів до моменту впровадження системи в експлуатацію.

Для перевірки працездатності WEB-ресурсу для тестування знань студентів було реалізовано сценарії ручного функціонального тестування, які охоплюють основні ланки взаємодії користувача з системою: реєстрацію, авторизацію, проходження тесту, відображення результату та захист доступу.

Розроблений WEB-ресурс був ретельно протестований, із проведенням понад 50 тестових запусків. Після запуску WEB-ресурсу для тестування знань студентів користувач потрапляє на стартову сторінку. Вигляд стартової сторінки зображено на рисунку 3.1. На стартовій сторінці розміщено навігаційну панель з логотипом продукту, навігацією та кнопками для авторизації та автентифікації. Задній фон стартової сторінки – це зациклене відео для затримання користувачів на даному етапі.

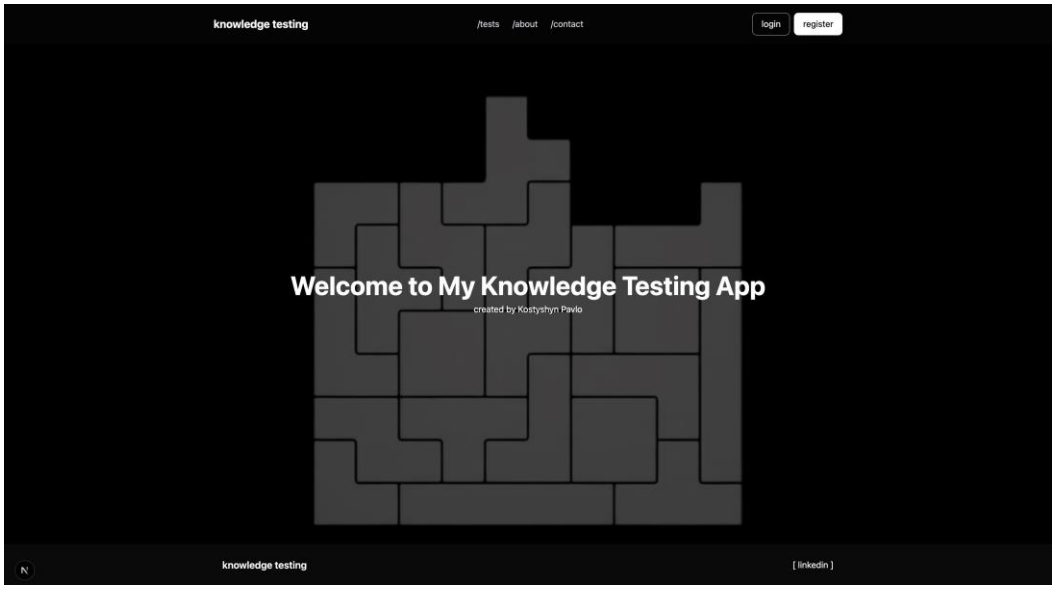

Рисунок 3.1 – Загальний вигляд інтерфейсу стартової сторінки WEB-ресурсу для тестування знань студентів

Одразу можна перевірити захищеність сторінок, спробувавши відвідати сторінку з тестами, коли користувач ще не авторизований. Відповідь системи зображена на рисунку 3.2.

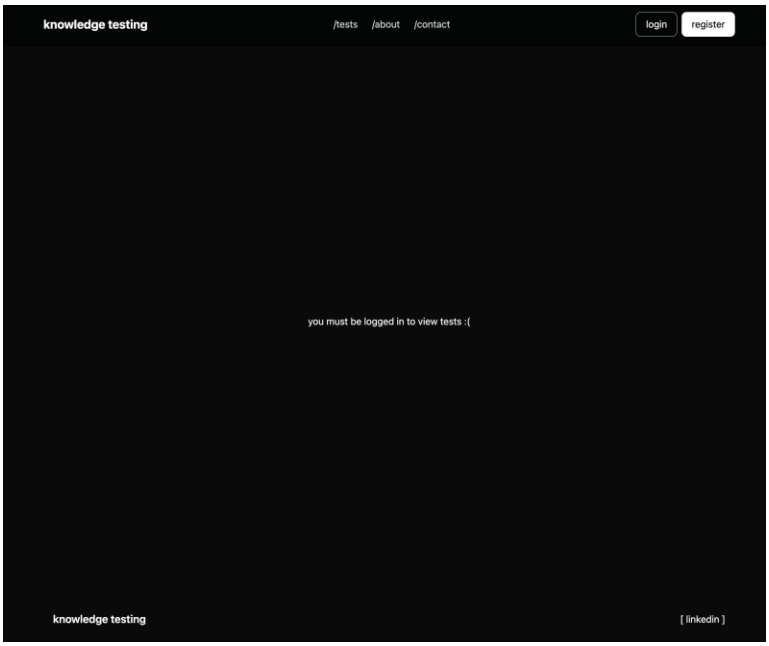

Рисунок 3.2 – Вигляд інтерфейсу відповіді системи
на неавторизований вхід

Наступним етапом для тестування – реєстрація користувача. На рисунку 3.3 зображено перший крок реєстрації – вибір ролі користувача.

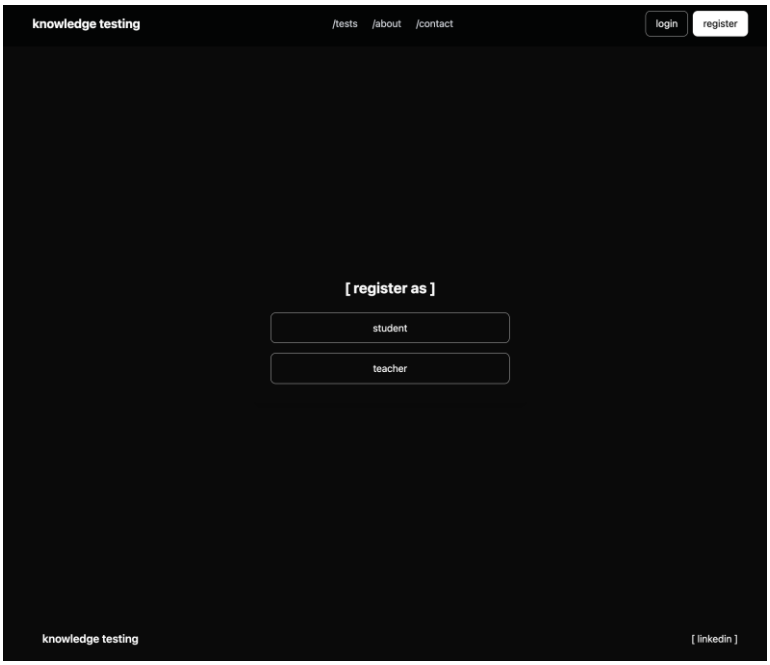

Рисунок 3.3 – Вигляд інтерфейсу вибору ролі
користувача при реєстрації

Після обрання відповідної ролі користувач має ввести дані для реєстрації. На рисунку 3.4 зображено форму для даних користувача.

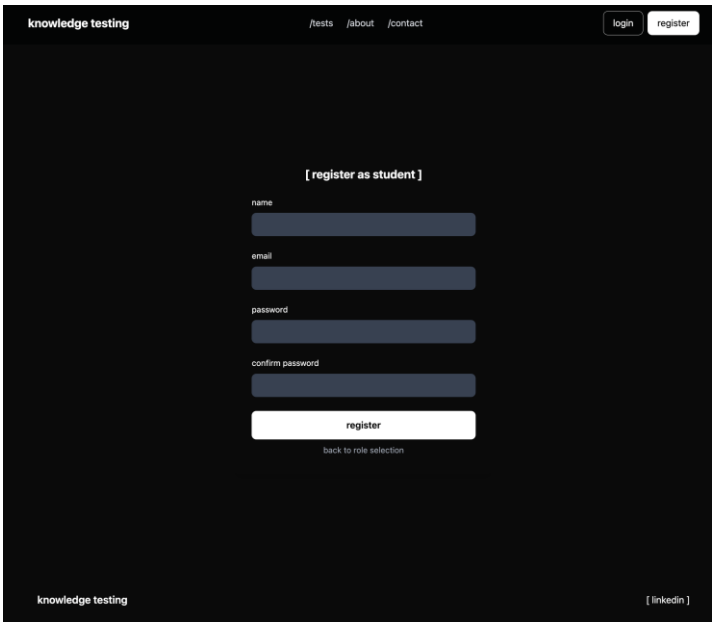

Рисунок 3.4 – Вигляд інтерфейсу форми для даних користувача при реєстрації

На цьому етапі протестовано відповідь системи на введення даних у некоректному форматі. На рисунку 3.5 зображено відповідь системи на некоректні дані.

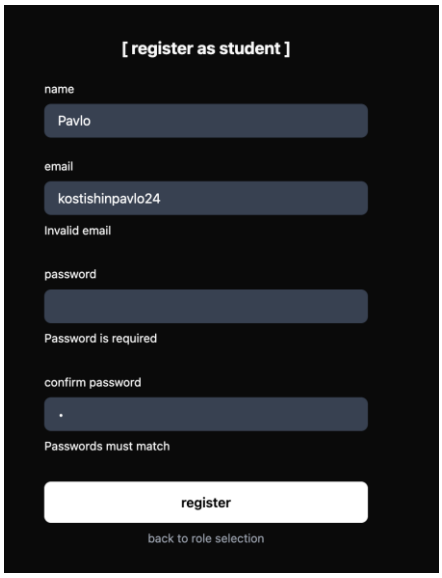

Рисунок 3.5 – Вигляд інтерфейсу відповіді системи на некоректні дані

Після успішної реєстрація користувача перенаправляє на сторінку profile, де користувач може отримати інформацію про свій профіль. Цей інтерфейс зображено на рисунку 3.6.

Рисунок 3.6 – Вигляд інтерфейсу сторінки користувача

Розглянемо процес створення тесту. На рисунку 3.7 зображено загальний вигляд сторінки з тестами для викладача.

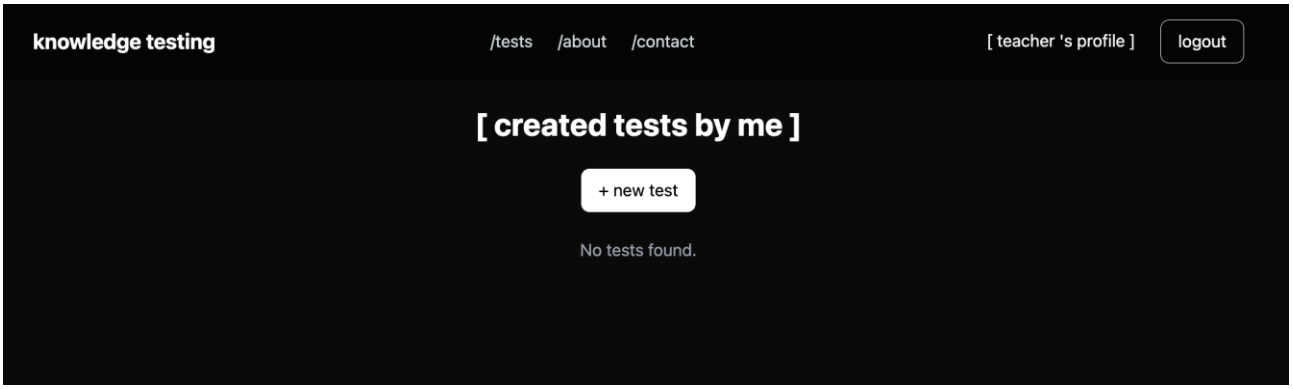

Рисунок 3.7 – Вигляд інтерфейсу сторінки з тестами для викладача

При натисканні на кнопку для створення нового тесту, користувача переадресовує на інтерфейс заповнення базової інформації про тест. Такий інтерфейс зображено на рисунку 3.8.

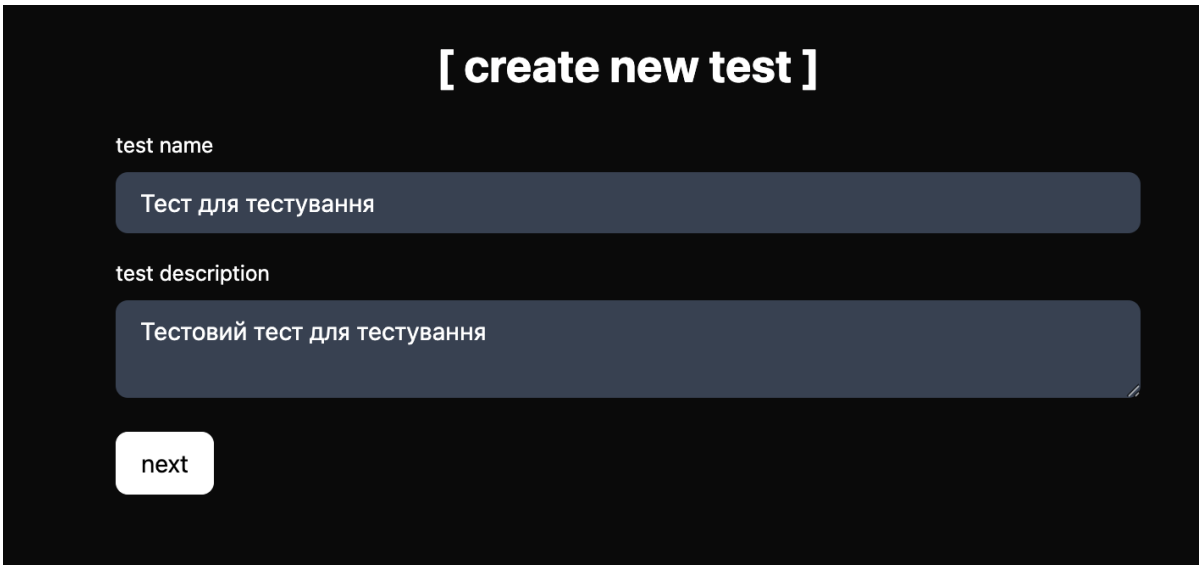

Рисунок 3.8 – Вигляд інтерфейсу заповнення базової інформації про тест

Після заповнення інформації про тест, система переадресовує на наступний етап, налаштування пулів тесту та додавання питань. Цей стан зображено на рисунку 3.9.

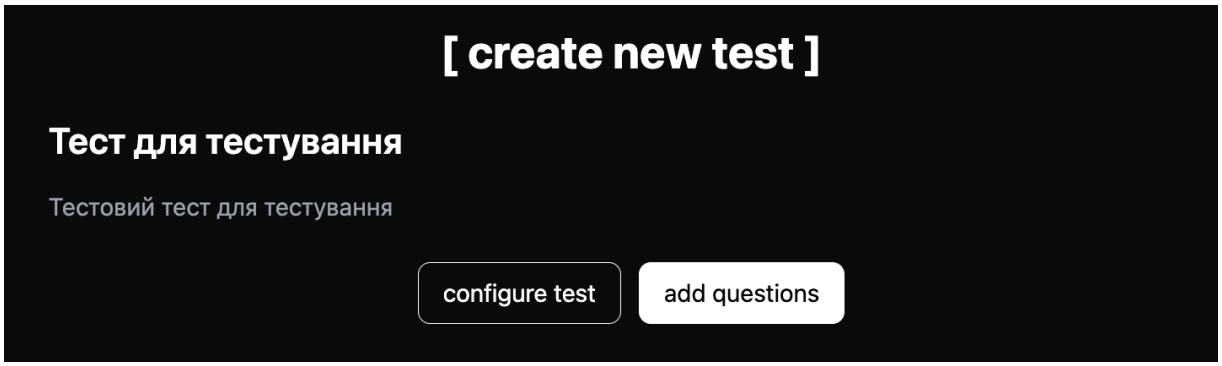

Рисунок 3.9 – Вигляд інтерфейсу налаштування
пулів тесту та додавання питань

При спробі налаштувати тест, система відповідає модальним вікном, що дає змогу ввести відповідну кількість пулів та кількість питань з кожного пулу. Вигляд інтерфейсу модального вікна зображене на рисунку 3.10.

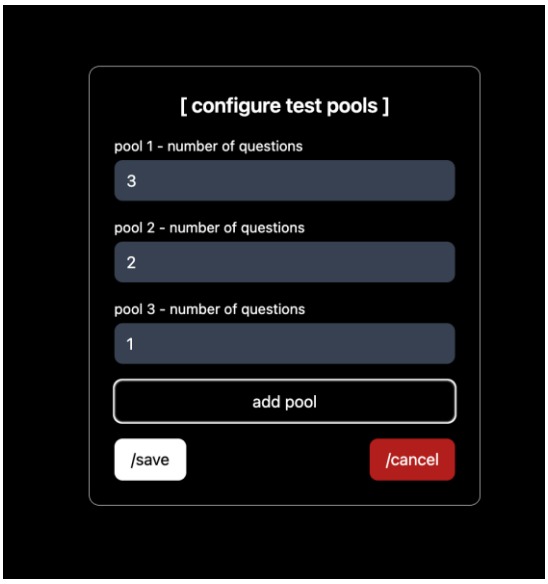

Рисунок 3.10 – Вигляд інтерфейсу модального вікна для налаштування тесту

Наступний крок – додавання питань до тесту. Додавання питань реалізовано також у вигляді модального вікна, його вигляд зображено на рисунку 3.11.

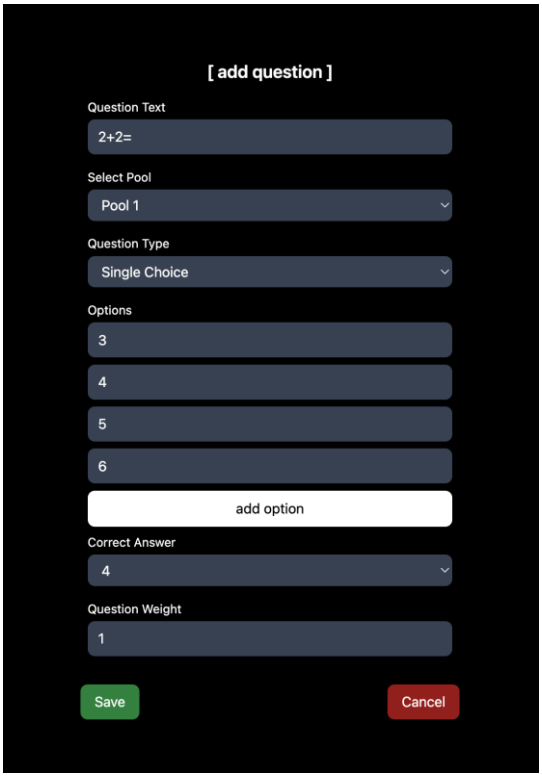

Рисунок 3.11 – Вигляд інтерфейсу модального вікна для додавання питань з однією відповіддю

В цьому модальному вікні у користувача є можливість обрати до якого пулу належить питання, якого типу це питання, вагу питання і відповідно правильну відповідь. На рисунках 3.12 – 3.14 зображені модальні вікна для різних типів питань.

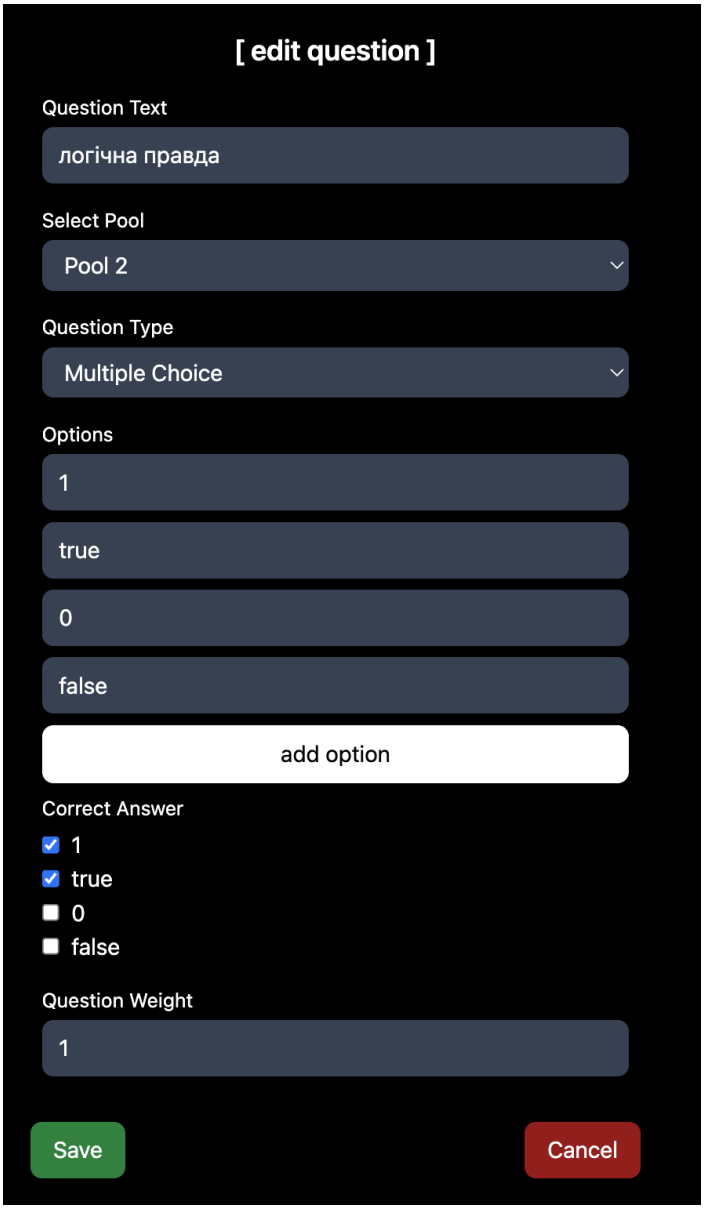

Рисунок 3.12 – Вигляд інтерфейсу модального вікна для додавання питань з кількома відповідями

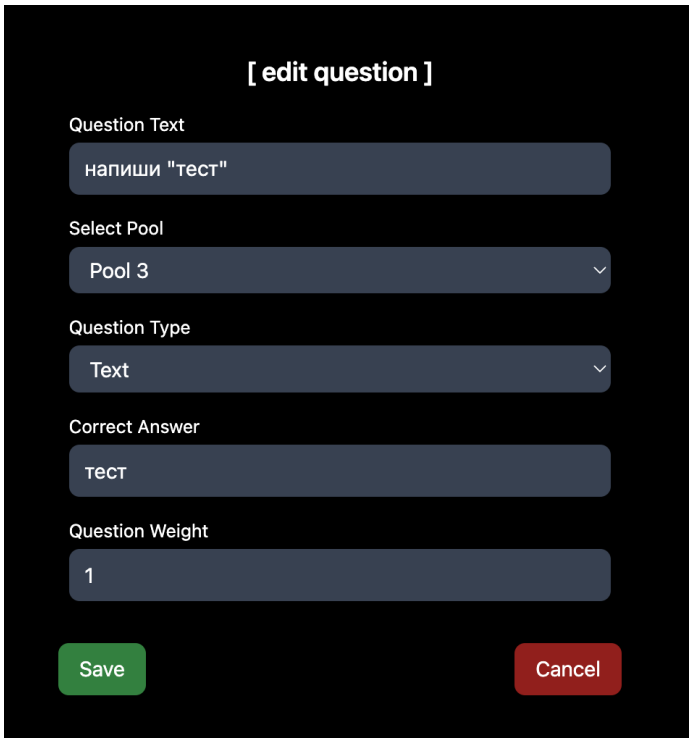

Рисунок 3.13 – Вигляд інтерфейсу модального вікна для додавання питань з відповіддю у вигляді тексту

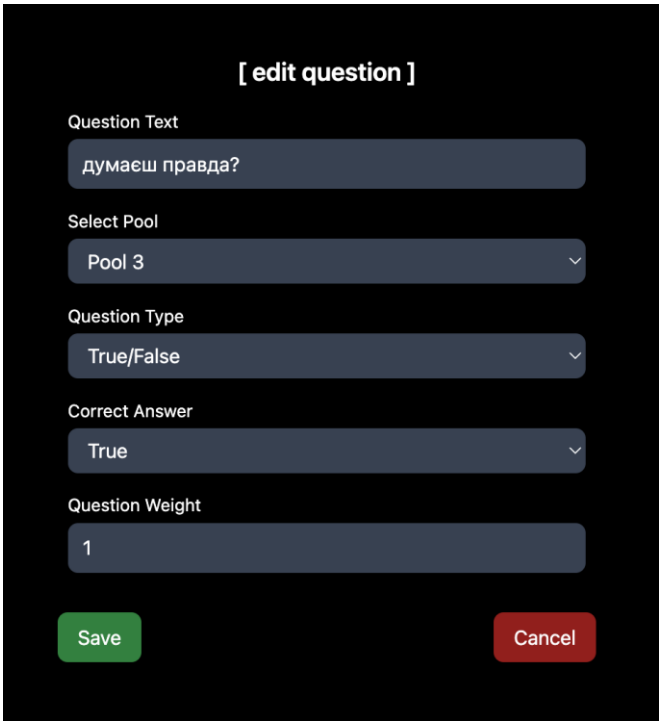

Рисунок 3.14 – Вигляд інтерфейсу модального вікна для додавання питань з відповіддю правда/брехня

Після того як користувач наповнив тест необхідною кількістю питань, можна генерувати тест, натиснувши відповідну кнопку. Зведену інформацію про тест зображено на рисунку 3.15.

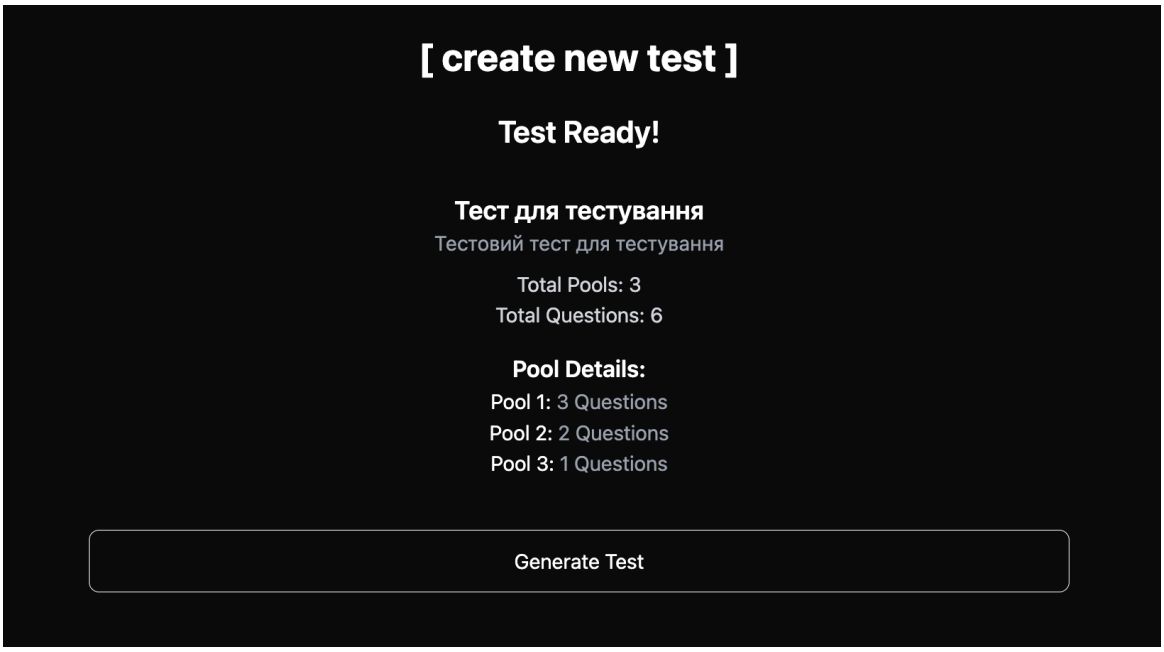

Рисунок 3.15 – Вигляд інтерфейсу зведеної інформація про тест

Після чого система згенерує тест, повернувши користувачу посилання на проходження цього тесту. Цю відповідь зображено на рисунку 3.16.

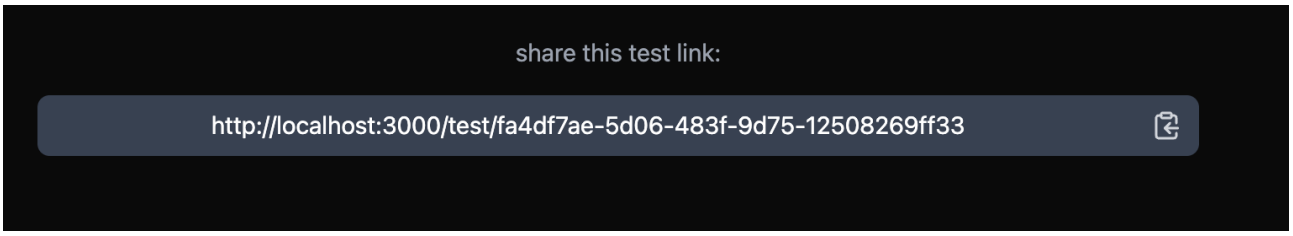

Рисунок 3.16 – Вигляд інтерфейсу відповіді системи з посилання для поширення

Після цього викладач може поділитися посиланням з користувачами. Коли новий користувач відкриє це посилання він спершу має авторизуватися після чого він зможе отримати доступ до проходження тесту. Вигляд тесту зображено на рисунку 3.17.

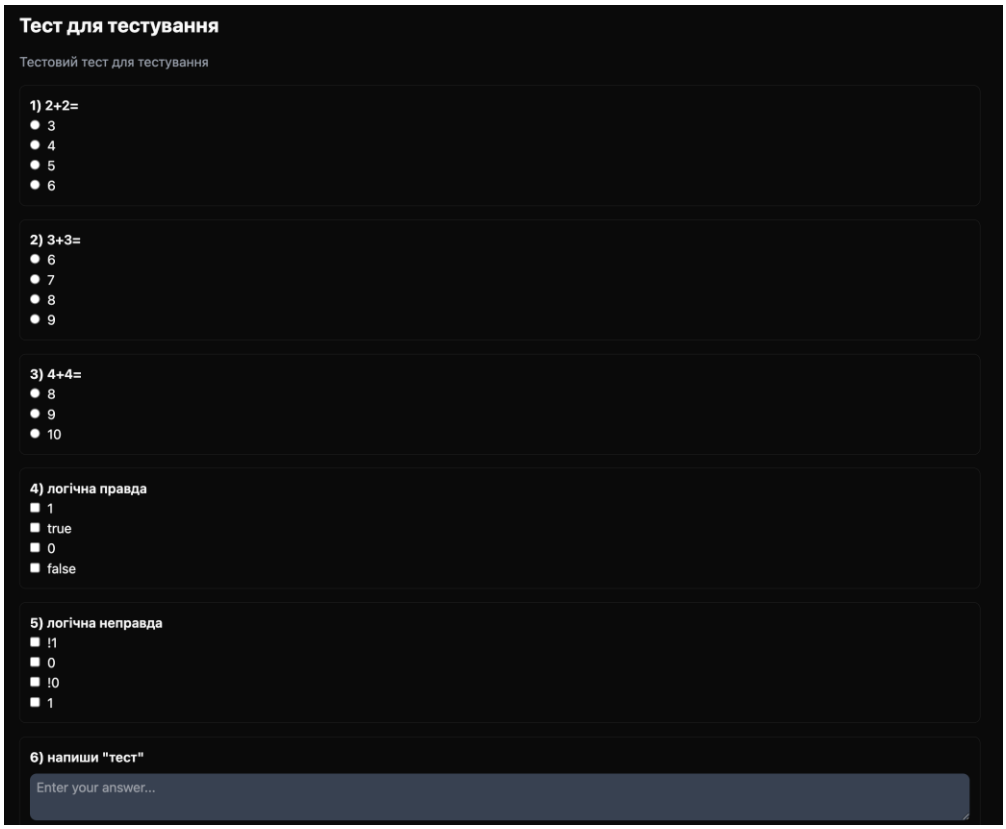

Рисунок 3.17 – Загальний вигляд інтерфейсу проходження тесту

Після проходження тесту і відправлення результатів користувач отримує відповідну оцінку за цей тест. Оцінка зображена на рисунку 3.18.

Рисунок 3.18 – Вигляд інтерфейсу оцінки за пройдений тест

Викладач може переглядати результати проходження тесту у вкладці «тести» обравши потрібний. Результати тесту, який був створений для тестування зображені на рисунку 3.19.

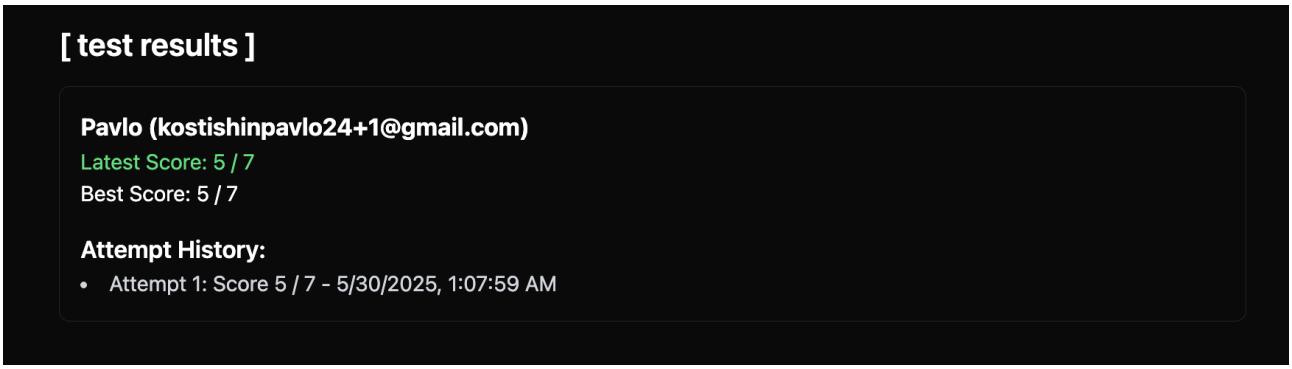

Рисунок 3.19 – Вигляд інтерфейсу результату проходження тесту

Розроблений WEB-ресурс пройшов усі ключові етапи тестування. Було перевірено:

- коректність авторизації та захисту;
- стабільність проходження та завершення тесту;
- відображення результатів;
- безпечність маршрутів і доступу;
- адаптивність до різних пристроїв.

В таблиці 3.5 подано результати тестування розробленого програмного продукту та аналогів.

Таблиця 3.5 – Результати тестування розробленого програмного продукту та аналогів

Функціональна можливість	Розроблений ресурс	Майстер-Тест	Google Classroom	Всеосвіта
Авторизація з поділом на ролі	+	+	+	+
Створення динамічно згенерованих тестів	+	–	–	–

Продовження таблиці 3.5.

Функціональна можливість	Розроблений ресурс	Майстер-Тест	Google Classroom	Всеосвіта
Миттєве відображення результату	+	+	+	+
Зберігання відповідей у структурованому вигляді (JSON)	+	–	–	–
Перегляд статистики проходження тестів	+	+	+	+
Мобільна адаптивність	+	+	+	+
Простота та інтуїтивність інтерфейсу	+	–	+	–
Модальні вікна для підтвердження дій	+	–	–	–

На основі даних таблиці 3.5 можна зробити висновок, що розроблений WEB-ресурс має 4 ключові переваги порівняно з аналогами, зокрема:

- підтримка динамічного формування тестів;
- гнучкий вибір формування однакової складності варіантів;
- інтуїтивний інтерфейс користувача;
- захищений доступ від несанкціонованого доступу до відповідей.

Також варто зауважити, що через використання модальних вікон до взаємодії з мобільними пристроями WEB-ресурс добре адаптується. Ці особливості роблять систему сучасною, адаптивною та конкурентоспроможною на фоні популярних освітніх рішень.

3.6 Висновки до розділу 3

У цьому розділі було проведено комплексне обґрунтування вибору технологій та середовища для розробки WEB-ресурсу для тестування знань студентів, а також здійснено повноцінне проєктування, реалізацію та тестування системи.

Виконано порівняльний аналіз сучасних фреймворків для розробки клієнтської та серверної частин – React, Angular, Vue (на боці клієнта) та Node.js, Express, Next.js (на боці сервера). За результатами аналізу було обрано Next.js, що дозволяє об’єднати клієнтську і серверну логіку в одному середовищі, забезпечує SSR, автоматичну маршрутизацію та гарну масштабованість.

Розглянуто мови запитів до баз даних – SQL та OQL. Зважаючи на переваги реляційної моделі для освітнього середовища, обрано SQL, а саме – у зв’язці з ORM Sequelize, що забезпечує зручну обробку моделей, автоматичну генерацію таблиць та підтримку зв’язків між сутностями.

Проаналізовано середовища розробки – Visual Studio Code, WebStorm, Atom. Вибір зупинено на Visual Studio Code завдяки його легкості, швидкості, розширюваності, інтеграції з Git та підтримці TypeScript/JavaScript.

Описано реалізацію клієнтської та серверної частин. Серверна логіка побудована на основі API-роутів Next.js, з використанням Sequelize, JWT, bcryptjs та middleware для захисту маршрутів. Клієнтська частина побудована на React 19 із застосуванням Formik, Yup, Tailwind CSS, React Context API та модальних вікон, що покращують UX та безпеку взаємодії.

Проведено тестування основних сценаріїв: реєстрації, авторизації, проходження тесту, підтвердження дій через модальні вікна, перегляду результатів, перевірки захисту маршрутів. Здійснено порівняльний аналіз розробленого продукту з аналогами – «Майстер-Тест», Google Classroom та «Всеосвіта». Розроблений WEB-ресурс має розширений функціонал на 4 можливості порівняно з аналогами, зокрема: підтримка динамічного формування тестів, гнучкий вибір формування однакової складності варіантів, інтуїтивний

інтерфейс користувача та захищений доступ від несанкціонованого доступу до відповідей.

Таким чином, усі поставлені цілі цього етапу було досягнуто: обґрунтовано вибір ефективного стеку технологій, реалізовано повноцінний WEB-ресурс, перевірено його функціональність та доведено переваги над існуючими аналогами.

ВИСНОВКИ

Всі задачі, поставлені для реалізації WEB-ресурсу для тестування знань студентів, були виконані в повному обсязі, а саме: обґрунтовано доцільність розробки WEB-ресурсу для тестування знань студентів; спроєктовано WEB-ресурс для тестування знань студентів; програмно реалізовано WEB-ресурс для тестування знань студентів; виконано тестування програми та проведено аналіз отриманих результатів; розроблено інструкцію користувача.

Для розробки WEB-ресурсу були використані мови програмування TypeScript та JavaScript, а для управління базою даних – SQL з використанням ORM Sequelize. Як платформу реалізації обрано WEB-браузери, що забезпечує універсальний доступ до системи на різних пристроях.

Під час реалізації було створено структуру програмного забезпечення для тестування знань студентів, розроблено UML-діаграму та ER-модель бази даних, яка охоплює користувачів, тести, запитання, відповіді та результати..

Розроблений продукт успішно пройшов тестування коректності роботи, а порівняльний аналіз із відомими аналогами показав розширений функціонал на 4 можливості порівняно з аналогами.

Мета роботи, яка полягала в розширенні функціональних можливостей WEB-ресурсу для тестування знань студентів досягнута за рахунок введення таких 4 додаткових компонентів функціоналу: підтримка динамічного формування тестів, гнучкий вибір формування однакової складності варіантів, інтуїтивний інтерфейс користувача та захищений доступ від несанкціонованого доступу до відповідей.

Робота виконана у повному обсязі відповідно до поставлених цілей, вимог та методичних рекомендацій щодо підготовки бакалаврських кваліфікаційних робіт. Усі етапи були реалізовані належним чином, що підтверджує якість, завершеність і практичну цінність розробленого рішення [24].

Розроблений програмний продукт відповідає вимогам сучасної розробки WEB-ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Костишин П. В., Крилик Л. В. Перспективи використання Next.js у розробці WEB-ресурсу для тестування знань студентів. *LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації*. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23170/19166 (дата звернення: 12.05.2025).

2. Свідоцтво про реєстрацію авторського права на твір № 136024. Комп'ютерна програма «WEB-ресурс для тестування знань студентів» / Крилик Л. В., Костишин П. В., дата реєстрації 12.05.2025 р.

3. Designing a complete Online Examination Portal from scratch. URL: https://medium.com/@diksha.saxena78/designing-a-complete-online-examination-portal-from-scratch-ui-ux-case-study-c58fda5764d4 (дата звернення 14.05.2025).

4. 8 Ways To Design Online. URL: https://elearningindustry.com/ways-to-design-engaging-online-assessments (дата звернення 14.05.2025).

5. Революціонізація Навчання та Розвитку за допомогою Мобільних eLearning-додатків. URL: https://cluelabs.com/blog/%d1%80%d0%b5%d0%b2%d0%be%d0%bb%d1%8e%d1%86%d1%96%d0%be%d0%bd%d1%96%d0%b7%d0%b0%d1%86%d1%96%d1%8f-%d0%bd%d0%b0%d0%b2%d1%87%d0%b0%d0%bd%d0%bd%d1%8f-%d1%82%d0%b0-%d1%80%d0%be%d0%b7%d0%b2%d0%b8%d1%82/ (дата звернення 14.05.2025).

6. Онлайн тестування. URL: https://myownconference.com/blog/uk/onlain-testuvannia-myownconference/ (дата звернення 16.05.2025).

7. Переваги та недоліки онлайн навчання. URL: https://myownconference.com/blog/uk/onlajn-navchannya-plyusy-ta-minusy/ (дата звернення 16.05.2025).

8. Майстер-Тест. URL: https://master-test.net/uk (дата звернення 16.05.2025).

9. Google Classroom. URL: https://classroom.google.com (дата звернення 16.05.2025).

10. Google Forms. URL: https://docs.google.com/forms (дата звернення 16.05.2025).

11. Всеосвіта. URL: https://vseosvita.ua (дата звернення 16.05.2025).

12. MVC. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення 18.05.2025).

13. Next.js Docs. URL: https://nextjs.org/docs (дата звернення 18.05.2025).

14. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web (дата звернення 18.05.2025).

15. Express.js. URL: https://expressjs.com/ (дата звернення 18.05.2025).

16. Sequelize Docs. URL: https://sequelize.org/docs/v6/ (дата звернення 18.05.2025).

17. TypeScript Docs. URL: https://www.typescriptlang.org/ (дата звернення 18.05.2025).

18. React Docs. URL: https://react.dev/learn (дата звернення 18.05.2025).

19. Angular Docs. URL: https://angular.dev/overview (дата звернення 18.05.2025).

20. Vue.js. URL: https://vuejs.org/guide/introduction.html (дата звернення 18.05.2025).

21. Node.js. URL: https://nodejs.org/docs/latest/api/ (дата звернення 18.05.2025).

22. SQL. URL: https://www.w3schools.com/sql/sql_intro.asp (дата звернення 18.05.2025).

23. OQL. URL: https://www.ibm.com/docs/en/networkmanager/4.2.0?topic=reference-object-query-language (дата звернення 18.05.2025).

24. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця : ВНТУ, 2023. (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-ресурсу для тестування знань студентів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,15 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А. А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О. К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В. С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Крилик Л. В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Костишин П. В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

Лістинг програми

```
export default async function authMiddleware(req, res, next) {
  try {
    const cookie = parse(req.headers.cookie || "");
    const token = cookie.token;

    if (!token) {
      return res.status(401).json({ message: "Unauthorized: No token provided" });
    }

    const verified = jwt.verify(token, process.env.JWT_SECRET);

    req.user = verified;
    return next();
  } catch {
    return res.status(401).json({ message: "Unauthorized: Invalid token" });
  }
}

import { createContext, useContext, useState, useEffect } from "react";

const AuthContext = createContext();

export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchUser = async () => {
      try {
        const response = await fetch("/api/user/profile", {
          method: "GET",
          credentials: "include",
        });

        if (!response.ok) throw new Error("Not authenticated");

        const userData = await response.json();
        setUser(userData);
      } catch {
        setUser(null);
      } finally {
        setLoading(false);
      }
    };

    fetchUser();
  }, []);
```

```
  const login = async (email, password) => {
    try {
      const response = await fetch("/api/auth/login", {
        method: "POST",
        credentials: "include",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
      });

      if (!response.ok) throw new Error("Invalid credentials");

      const userProfile = await fetch("/api/user/profile", {
        method: "GET",
        credentials: "include",
      });

      if (!userProfile.ok) throw new Error("Failed to get user profile");

      const userData = await userProfile.json();
      setUser(userData);
      return { success: true };
    } catch (error) {
      return { success: false, message: error.message };
    }
  };

  const logout = async () => {
    await fetch("/api/auth/logout", { method: "POST", credentials: "include" });
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, loading, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

import { Navbar } from "@/components/NavBar";
import "../styles/globals.css";
import { useEffect } from "react";
import { Footer } from "@/components/Footer";
import { AuthProvider } from "@/contexts/AuthContext";
import Head from "next/head";

export default function MyApp({ Component, pageProps }) {
  useEffect(() => {
    document.documentElement.classList.add("dark");
  }, []);

  return (
    <>
      <Head>
        <link rel="icon" type='image/png' href="/favicon.ico" />
        <title>test your brain</title>
```

```
      </Head>

      <AuthProvider>
        <div className="flex flex-col min-h-screen justify-between bg-background text-foreground">
          <Navbar />
          <Component {...pageProps} />
          <Footer />
        </div>
      </AuthProvider>
    </>
  );
}

import { useState } from "react";
import { useRouter } from "next/router";
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as Yup from "yup";
import { useAuth } from "@/contexts/AuthContext";

export default function Login() {
  const { login } = useAuth();
  const [errorMessage, setErrorMessage] = useState("");
  const router = useRouter();
  const { redirectTo } = router.query;

  const validationSchema = Yup.object().shape({
    email: Yup.string().email("Invalid email").required("Email is required"),
    password: Yup.string()
      .min(6, "Password must be at least 6 characters")
      .required("Password is required"),
  });

  const handleSubmit = async (values, { setSubmitting }) => {
    setErrorMessage("");

    const result = await login(values.email, values.password);

    if (result.success) {
      router.push(redirectTo || "/profile");
    } else {
      setErrorMessage(result.message);
    }

    setSubmitting(false);
  };

  return (
    <div className="flex items-center justify-center p-6">
      <div className="p-8 rounded-lg shadow-lg w-full max-w-md">
        <h2 className="text-2xl font-bold text-center mb-4">[ login ]</h2>

        {errorMessage && (
          <p className="text-red-500 text-sm text-center mb-4">
            {errorMessage}
          </p>
        )}
```

```
      <Formik
        initialValues={{ email: "", password: "" }}
        validationSchema={validationSchema}
        onSubmit={handleSubmit}
      >
        {({ isSubmitting }) => (
          <Form className="flex flex-col space-y-4">
            <div>
              <label className="block text-sm my-2">email</label>
              <Field
                type="email"
                name="email"
                className="w-full px-4 py-2 bg-gray-700 rounded-md text-white"
              />
              <ErrorMessage
                name="email"
                component="div"
                className="text-red-500 text-sm my-1"
              />
            </div>

            <div>
              <label className="block text-sm my-2">password</label>
              <Field
                type="password"
                name="password"
                className="w-full px-4 py-2 bg-gray-700 rounded-md text-white"
              />
              <ErrorMessage
                name="password"
                component="div"
                className="text-red-500 text-sm my-1"
              />
            </div>

            <button
              type="submit"
              disabled={isSubmitting}
              className="my-2 cursor-pointer bg-white text-black border border-white font-semibold px-6 py-3
rounded-md hover:bg-black hover:text-white disabled:bg-gray-600"
            >
              {isSubmitting ? "logging in..." : "login"}
            </button>

            <p className="text-sm text-gray-400 text-center">
              don't have an account?{" "}
              <a href="/register" className="text-white hover:underline">
                register here
              </a>
            </p>
          </Form>
        )}
      </Formik>
    </div>
  </div>
```

```
  );
}

import Link from "next/link";
import { useEffect, useState } from "react";

export default function Profile() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    const fetchUserData = async () => {
      try {
        const response = await fetch("/api/user/profile", {
          method: "GET",
          headers: {
            "Content-Type": "application/json",
          },
        });

        if (!response.ok) {
          throw new Error("Failed to fetch user data");
        }

        const data = await response.json();
        setUser(data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchUserData();
  }, []);

  return (
    <div className="flex items-center justify-center p-6">
      <div className="p-8 rounded-lg shadow-lg w-full max-w-md">
        {loading ? (
          <p className="text-center text-gray-400">Loading user data...</p>
        ) : error ? (
          <div className="flex flex-col items-center space-y-4">
            <p className="text-center text-red-500">{error}</p>
            <div className="flex space-x-2">
              <Link
                href="/login"
                className="px-4 py-2 border border-white text-white rounded-md hover:bg-white hover:text-black"
              >
                login
              </Link>
              <Link
                href="/register"
                className="px-4 py-2 bg-white border border-white text-black rounded-md hover:bg-black
hover:text-white"
```

```
            >
              register
            </Link>
          </div>
        </div>
      ) : (
        <>
          <h2 className="text-2xl font-bold text-center mb-4">[ profile ]</h2>
          <div className="space-y-3">
            <p className="flex justify-between">
              <span className="font-semibold">name: </span>
              <span>{user.name}</span>
            </p>
            <p className="flex justify-between">
              <span className="font-semibold">email: </span>
              <span>{user.email}</span>
            </p>
            <p className="flex justify-between">
              <span className="font-semibold">role: </span>
              <span>{user.role}</span>
            </p>
            <p className="flex justify-between">
              <span className="font-semibold">joined: </span>
              <span>{new Date(user.createdAt).toLocaleDateString()}</span>
            </p>
          </div>
        </>
      )}
    </div>
  </div>
 );
}

import { User } from "@/utils/db";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
import { serialize } from "cookie";

dotenv.config();

export default async function handler(req, res) {
 if (req.method !== "POST") {
  return res.status(405).json({ message: "Method Not Allowed" });
 }

 try {
  const { email, password } = req.body;

  const user = await User.findOne({
   where: { email },
   attributes: ["id", "email", "password", "role"],
  });

  if (!user) {
   return res.status(400).json({ message: "User not found" });
```

```
    }

    const isMatch = await bcrypt.compare(
      password,
      user.getDataValue("password")
    );

    if (!isMatch) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    const tokenPayload = {
      id: user.getDataValue("id"),
      email: user.getDataValue("email"),
      role: user.getDataValue("role"),
    };

    const token = jwt.sign(tokenPayload, process.env.JWT_SECRET, {
      expiresIn: "10h",
    });

    res.setHeader(
      "Set-Cookie",
      serialize("token", token, {
        httpOnly: true,
        secure: process.env.NODE_ENV === "production",
        sameSite: "strict",
        path: "/",
        maxAge: 3600,
      })
    );

    res.json({ message: "Login successful", token });
  } catch (error) {
    res.status(500).json({ message: "Error logging in", error: error.message });
  }
}

import authMiddleware from "@/utils/authMiddleware";
import { TestConfig, Question, GeneratedTest, sequelize } from "@/utils/db";
import { autorize } from "@/utils/roleMiddleware";
import { v4 as uuidv4 } from "uuid";

export default async function handler(req, res) {
  if (req.method !== "POST") {
    return res.status(405).json({ message: "Method not allowed" });
  }

  try {
    await new Promise((resolve, reject) => {
      authMiddleware(req, res, (err) => (err ? reject(err) : resolve()));
    });

    await new Promise((resolve, reject) => {
      autorize(["teacher"])(req, res, (err) =>
        err ? reject(err) : resolve()
```

```
    );
  });

  const { testId } = req.body;
  const transaction = await sequelize.transaction();

  try {
    // Fetch test configuration
    const testConfig = await TestConfig.findAll({
      where: { testId },
      transaction,
    });

    if (!testConfig.length) {
      return res
        .status(400)
        .json({ message: "Test configuration not found" });
    }

    let selectedQuestions = [];
    let totalExpectedQuestions = testConfig.reduce(
      (sum, config) => sum + config.numberOfQuestions,
      0
    );

    for (const config of testConfig) {
      const { poolId, numberOfQuestions } = config;
      const questions = await Question.findAll({
        where: { testId, poolId },
        transaction,
      });

      if (questions.length < numberOfQuestions) {
        await transaction.rollback();
        return res
          .status(400)
          .json({ message: `Not enough questions in pool ${poolId}` });
      }

      // Randomly select required number of questions
      const shuffled = questions.sort(() => 0.5 - Math.random());
      selectedQuestions.push(...shuffled.slice(0, numberOfQuestions));
    }

    if (selectedQuestions.length !== totalExpectedQuestions) {
      await transaction.rollback();
      return res.status(400).json({
        message: `Configuration mismatch: Expected ${totalExpectedQuestions} questions, but got
${selectedQuestions.length}`,
      });
    }

    const generatedTestId = uuidv4();

    await GeneratedTest.create(
      {
```

```
          id: generatedTestId,
          testId,
          questionIds: selectedQuestions.map((q) => q.id),
        },
        { transaction }
      );

      await transaction.commit();
      res
        .status(201)
        .json({
          message: "Test generated successfully",
          testCode: generatedTestId,
          totalQuestions: selectedQuestions.length,
        });
    } catch (error) {
      await transaction.rollback();
      throw error;
    }
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error generating test", error: error.message });
  }
}

import { Test } from "@/utils/db";
import authMiddleware from "@/utils/authMiddleware";
import { autorize } from "@/utils/roleMiddleware";

export default async function handler(req, res) {
  if (req.method !== "POST") {
    return res.status(405).json({ message: "Method not allowed" });
  }

  try {
    await new Promise((resolve, reject) => {
      authMiddleware(req, res, (err) => (err ? reject(err) : resolve()));
    });

    await new Promise((resolve, reject) => {
      autorize(["teacher"])(req, res, (err) => (err ? reject(err) : resolve()));
    });

    const { testName, testDescription } = req.body;

    const test = await Test.create({
      teacherId: req.user.id,
      testName,
      testDescription,
    });

    res.status(201).json({ message: "Test created successfully", test });
  } catch (error) {
    res
      .status(500)
```

```
      .json({ message: "Error creating test", error: error.message });
  }
}

import authMiddleware from "@/utils/authMiddleware";
import { Question } from "@/utils/db";
import { autorize } from "@/utils/roleMiddleware";

export default async function handler(req, res) {
  if (req.method !== "DELETE") {
    return res.status(405).json({ message: "Method not allowed" });
  }

  try {
    await new Promise((resolve, reject) => {
      authMiddleware(req, res, (err) => (err ? reject(err) : resolve()));
    });

    await new Promise((resolve, reject) => {
      autorize(["teacher"])(req, res, (err) => (err ? reject(err) : resolve()));
    });

    const { id } = req.query;

    const question = await Question.findByPk(id);
    if (!question) {
      return res.status(404).json({ message: "Question not found" });
    }

    await question.destroy();

    res.status(200).json({ message: "Question deleted successfully" });
  } catch (error) {
    res.status(500).json({ message: "Error deleting question", error: error.message });
  }
}

import authMiddleware from "@/utils/authMiddleware";
import { TestResult } from "@/utils/db";
import { autorize } from "@/utils/roleMiddleware";

export default async function handler(req, res) {
  if (req.method !== "GET") {
    return res.status(405).json({ message: "Method Not Allowed" });
  }

  try {
    await new Promise((resolve, reject) => {
      authMiddleware(req, res, (err) => (err ? reject(err) : resolve()));
    });

    await new Promise((resolve, reject) => {
      autorize(["student"])(req, res, (err) =>
        err ? reject(err) : resolve()
      );
    });
```

```
    const result = await TestResult.findAll({
      where: { studentId: req.user.id },
    });

    res.json({ result });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error fetching results", error: error.message });
  }
}

import { DataTypes, Model, Optional, Sequelize } from "sequelize";
import bcrypt from "bcryptjs";

interface UserAttributes {
  id: number;
  name: string;
  email: string;
  password: string;
  role: "student" | "teacher";
}

class User extends Model<UserAttributes, Optional<UserAttributes, "id">> {
  static initModel(sequelize: Sequelize) {
    User.init(
      {
        id: {
          type: DataTypes.INTEGER,
          autoIncrement: true,
          primaryKey: true,
        },
        name: {
          type: DataTypes.STRING,
          allowNull: false,
        },
        email: {
          type: DataTypes.STRING,
          allowNull: false,
          unique: true,
          validate: {
            isEmail: true,
          },
        },
        password: {
          type: DataTypes.STRING,
          allowNull: false,
        },
        role: {
          type: DataTypes.ENUM("student", "teacher"),
          allowNull: false,
          defaultValue: "student",
        },
      },
      {
```

```
      sequelize,
      tableName: "Users",
      timestamps: true, // Adds createdAt & updatedAt
      hooks: {
        beforeCreate: async (user) => {
          if (!user.getDataValue("password")) {
            throw new Error("Password is required");
          }

          const salt = await bcrypt.genSalt(10);
          const hashedPassword = await bcrypt.hash(
            user.getDataValue("password"),
            salt
          );

          user.setDataValue("password", hashedPassword);
        },
      },
    }
  );
 }

 // eslint-disable-next-line @typescript-eslint/no-explicit-any
 static associate(models: any) {
   if (!models.Test || !(models.Test.prototype instanceof Model)) {
     throw new Error("Test model is not initialized");
   }
   User.hasMany(models.Test, { foreignKey: "teacherId" });
 }
}

const initializeUserModel = (sequelize: Sequelize) => {
 User.initModel(sequelize);

 return User;
};

export default initializeUserModel;
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«РОЗРОБКА WEB-РЕСУРСУ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ СТУДЕНТІВ»

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Костишин П. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

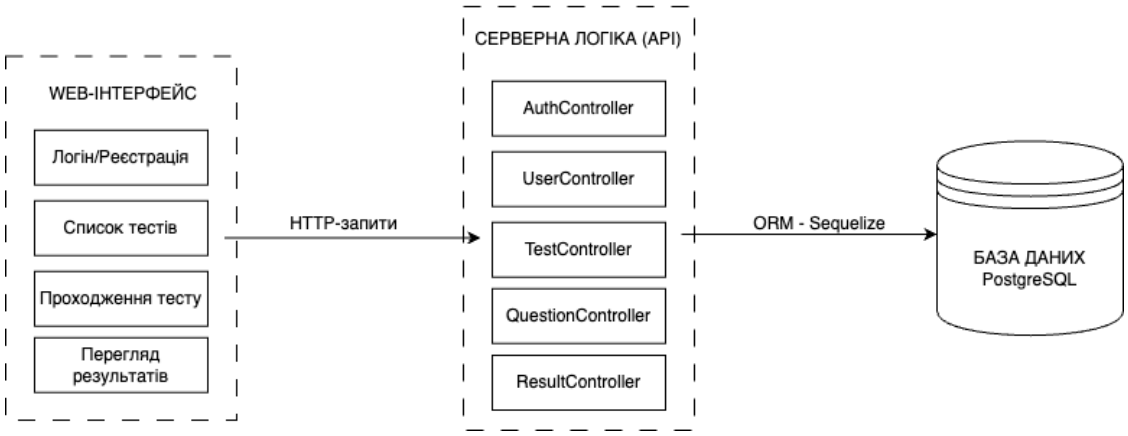

Рисунок В.1 – Загальна структурна схема системи

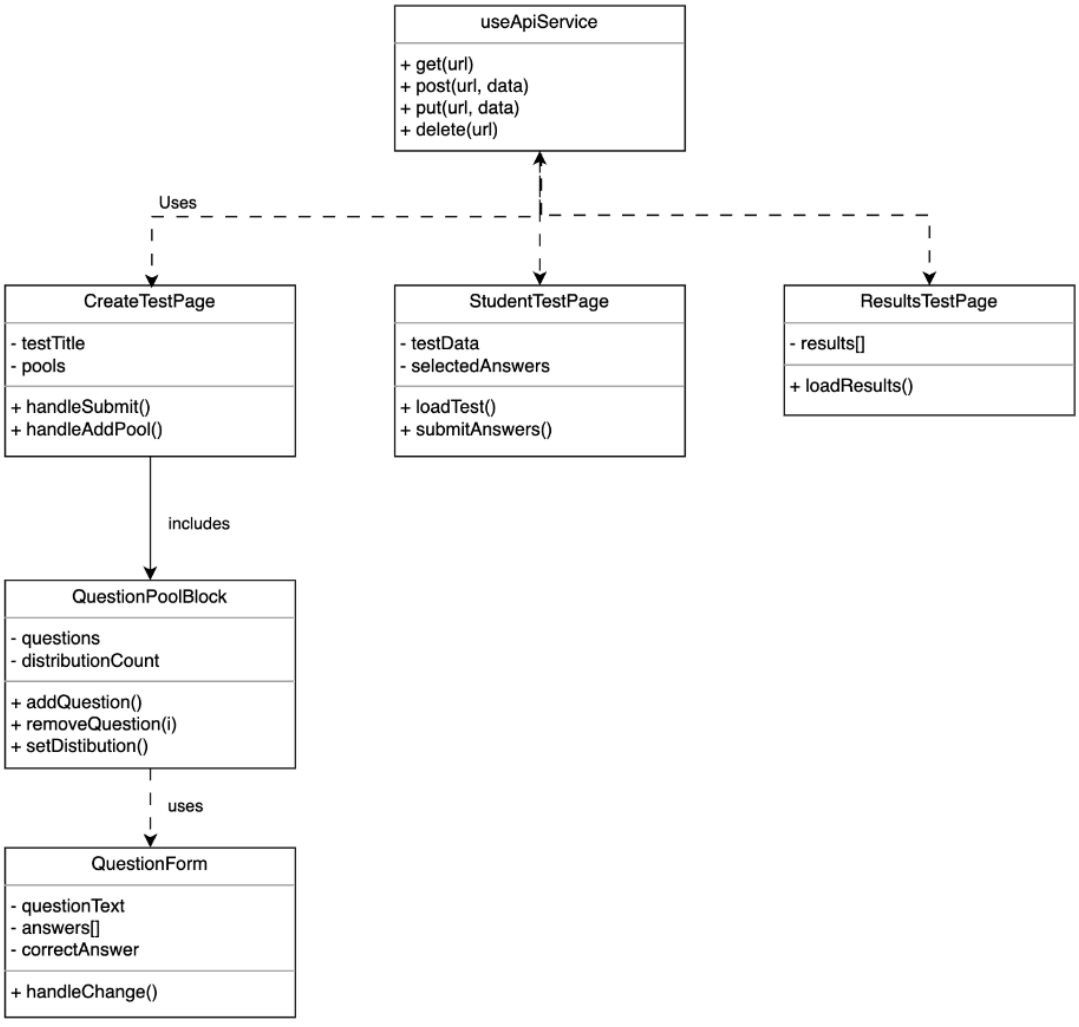

Рисунок В.2 – UML-діаграма класів клієнтської частини WEB-ресурсу для тестування знань студентів

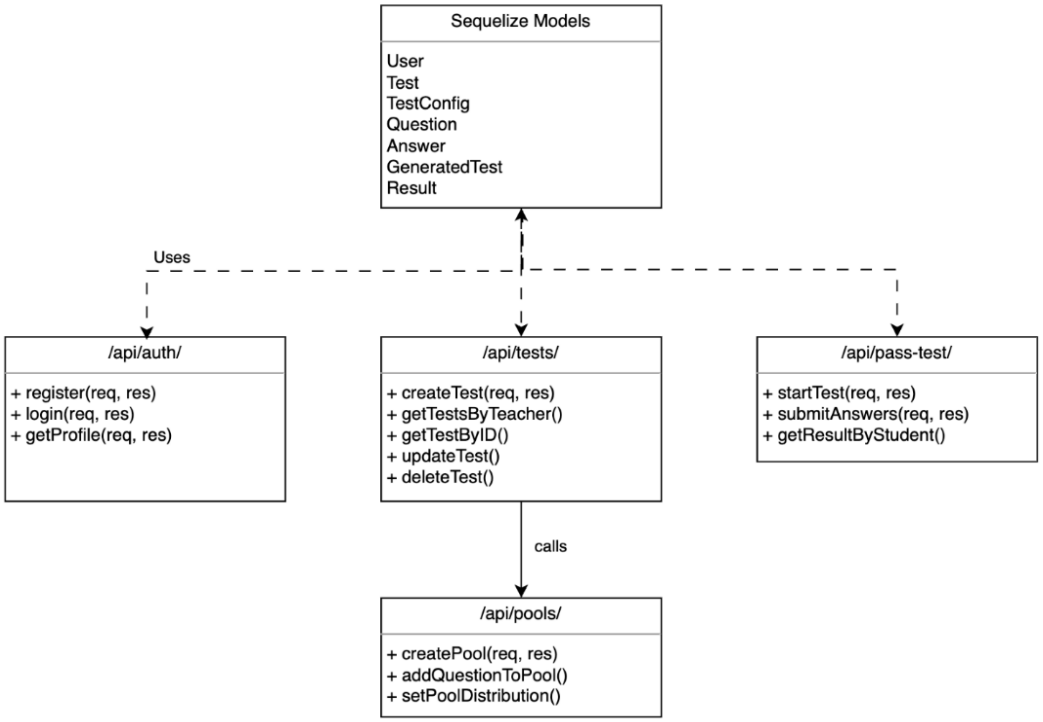

Рисунок В.3 – UML-діаграма класів серверної частини WEB-ресурсу для тестування знань студентів

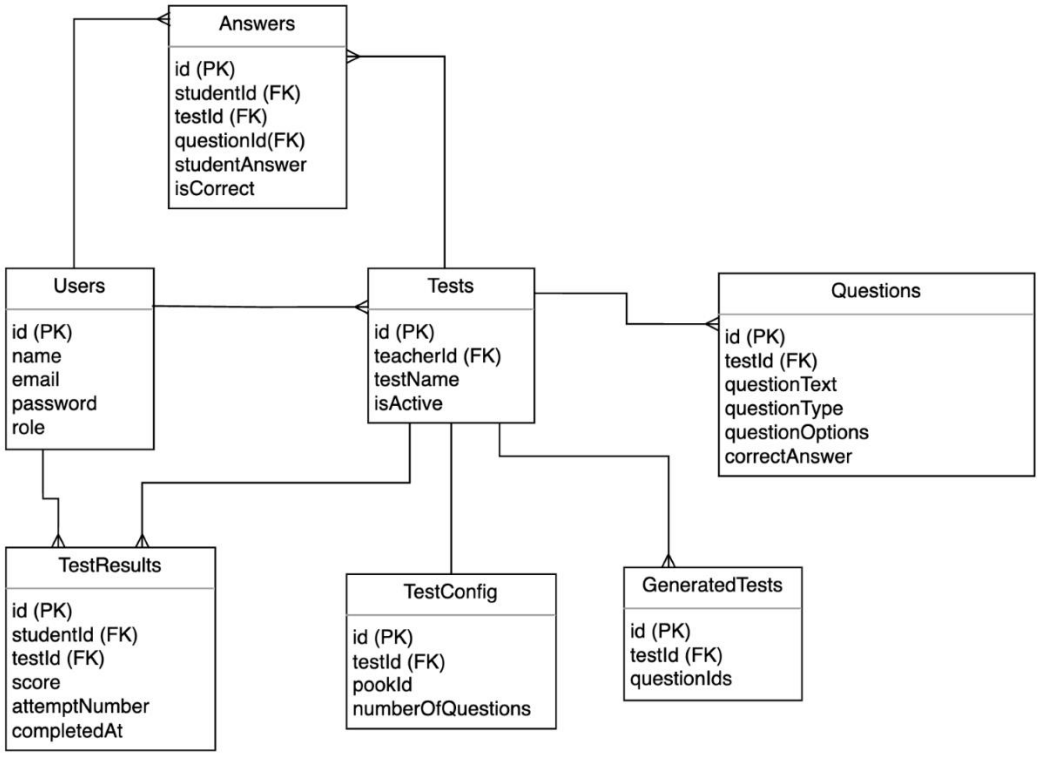

Рисунок В.4 – ER-модель бази даних

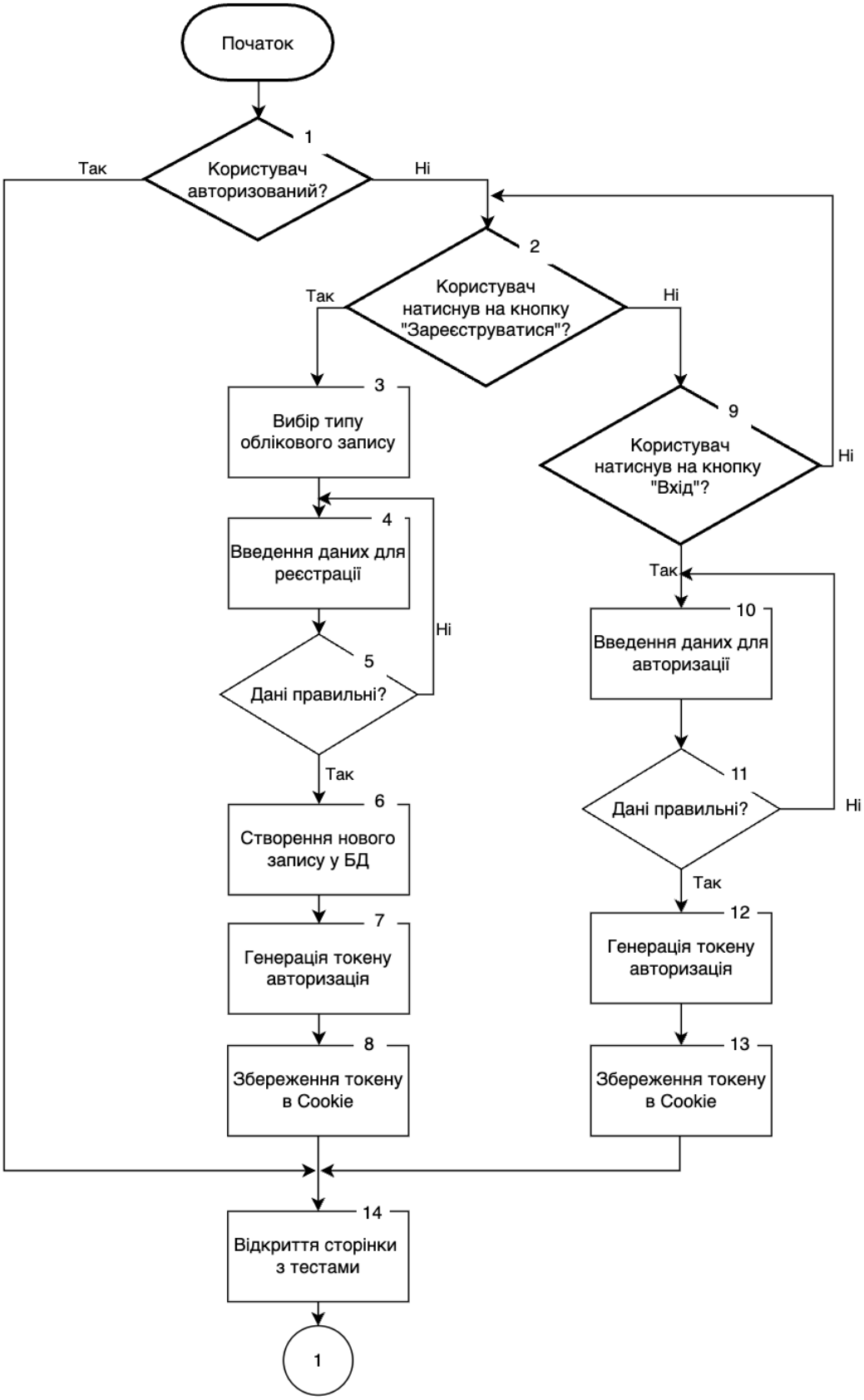

Рисунок В.5 – Схема алгоритму роботи WEB-ресурсу для тестування знань студентів

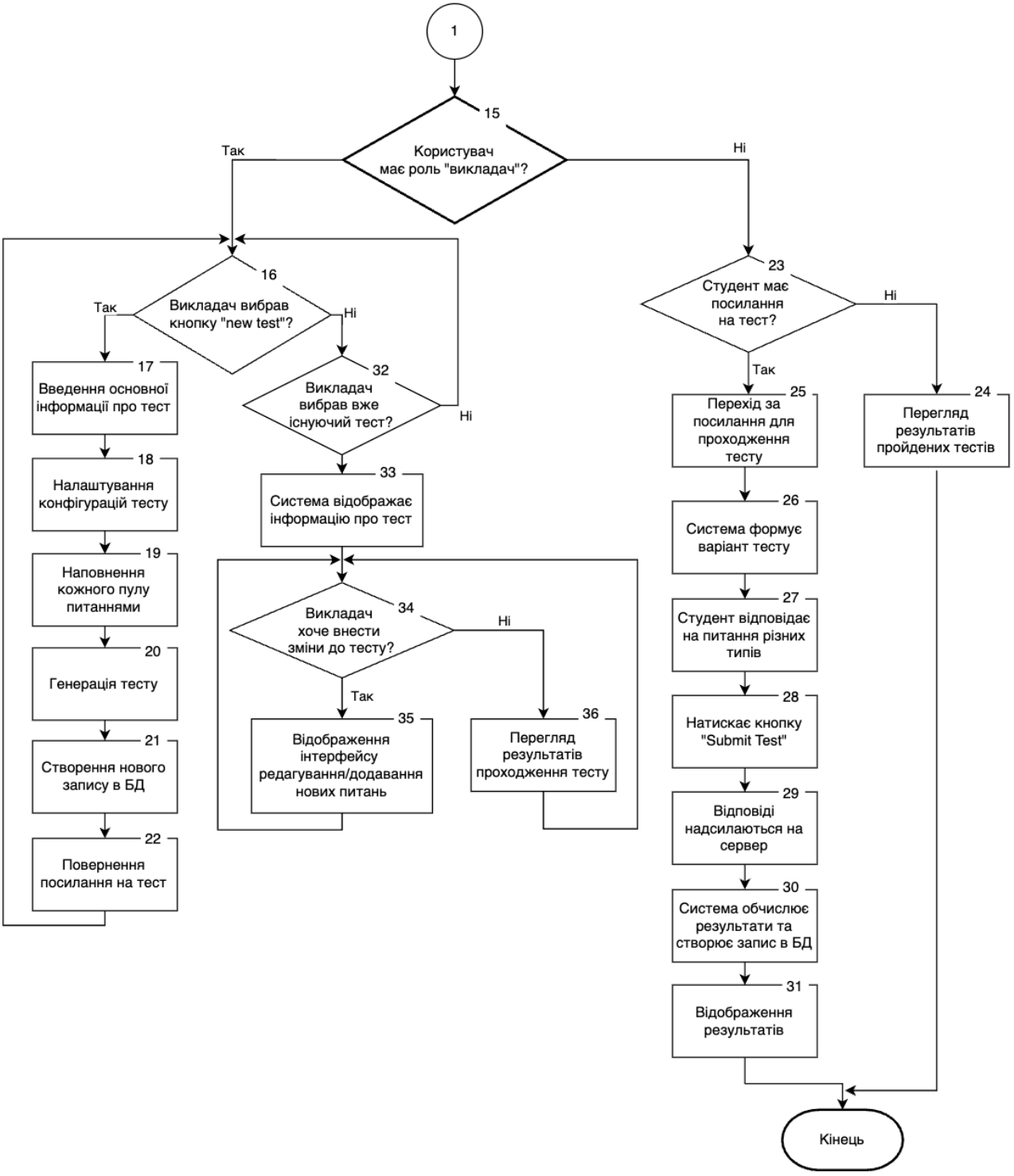

Рисунок В.5, аркуш 2

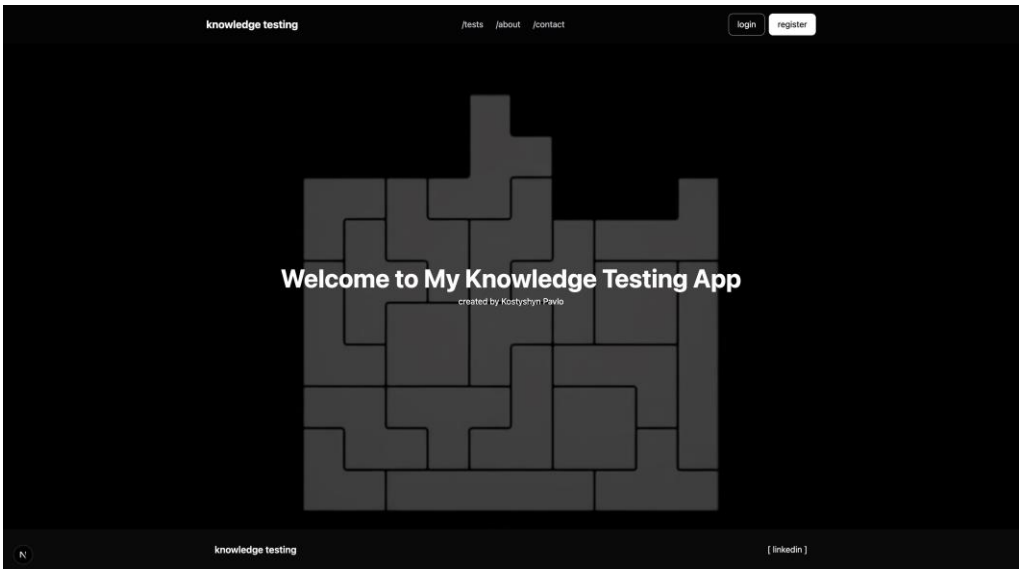

Рисунок В.6 – Загальний вигляд інтерфейсу стартової сторінки WEB-ресурсу для тестування знань студентів

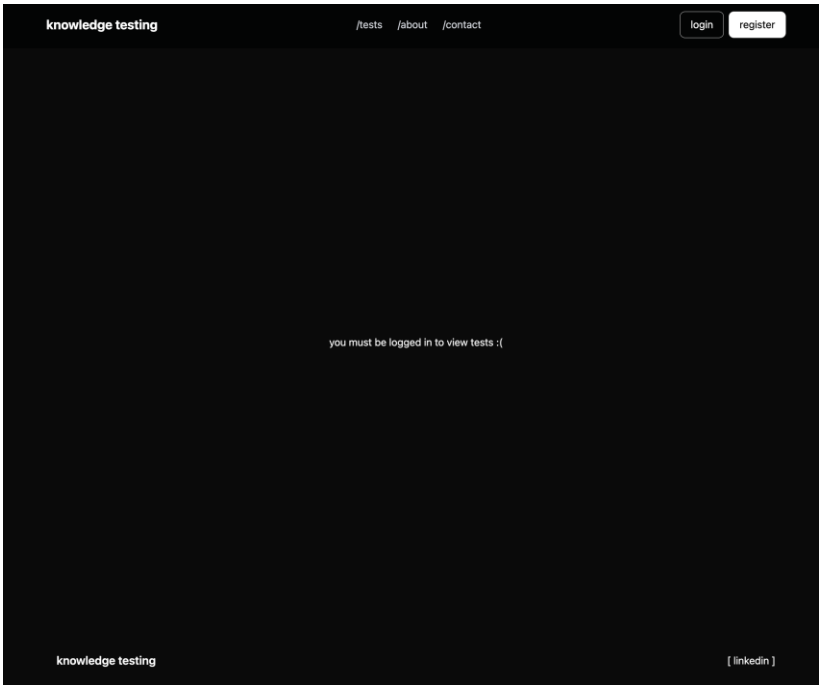

Рисунок В.7 – Вигляд інтерфейсу відповіді системи на неавторизований вхід

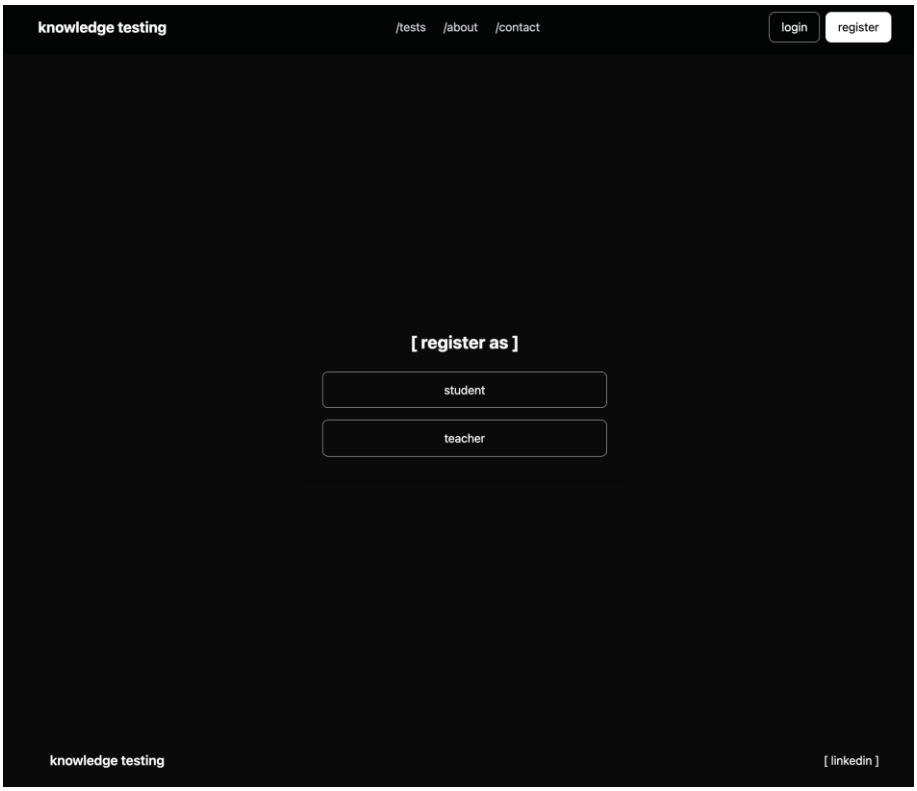

Рисунок В.8 – Вигляд інтерфейсу вибору ролі користувача при реєстрації

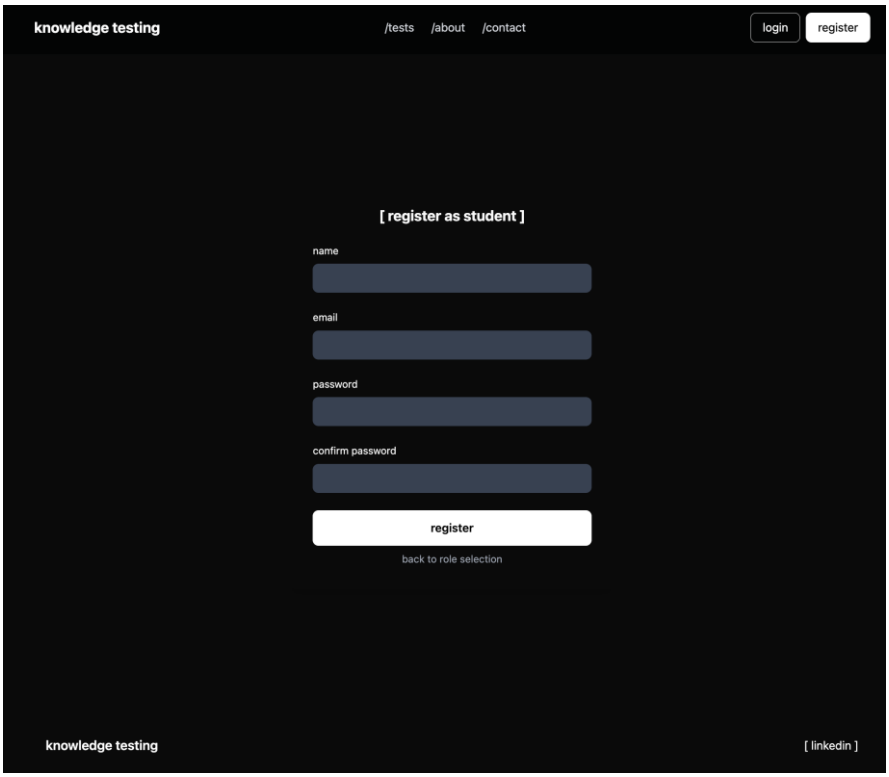

Рисунок В.9 – Вигляд інтерфейсу форми для даних користувача при реєстрації

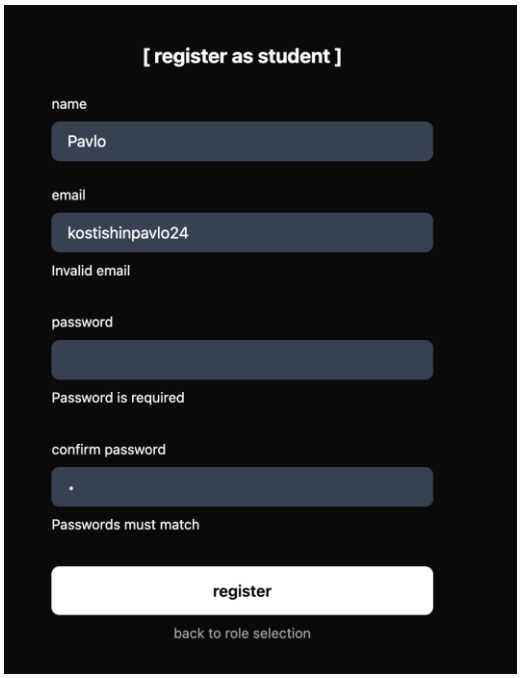

Рисунок В.10 – Вигляд інтерфейсу відповіді системи на некоректні дані

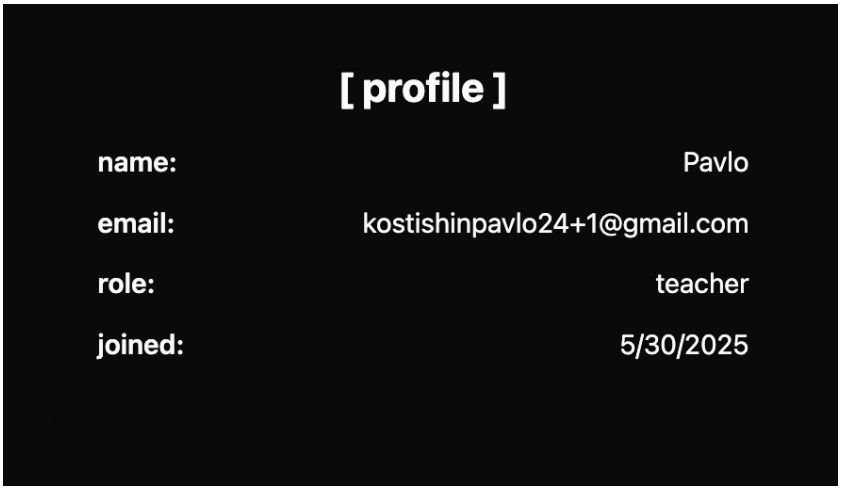

Рисунок В.11 – Вигляд інтерфейсу сторінки користувача

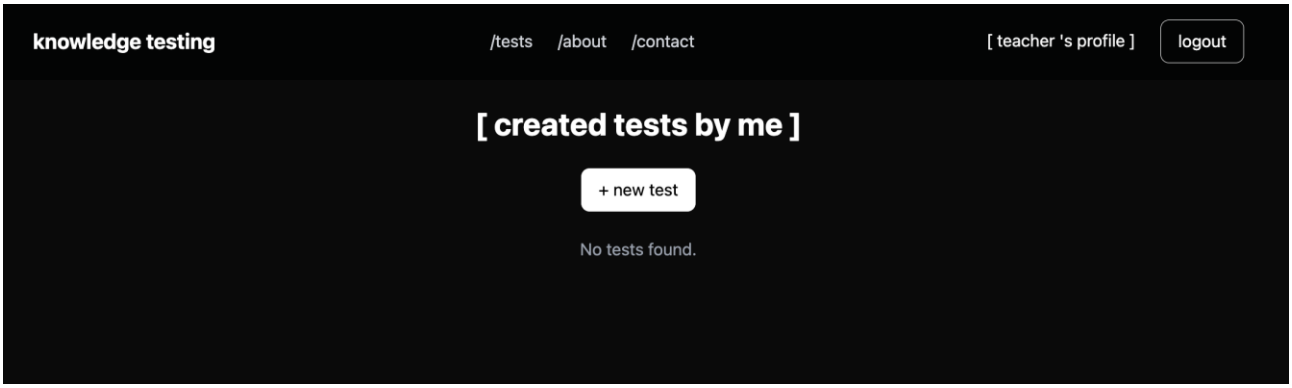

Рисунок В.12 – Вигляд інтерфейсу сторінки з тестами для викладача

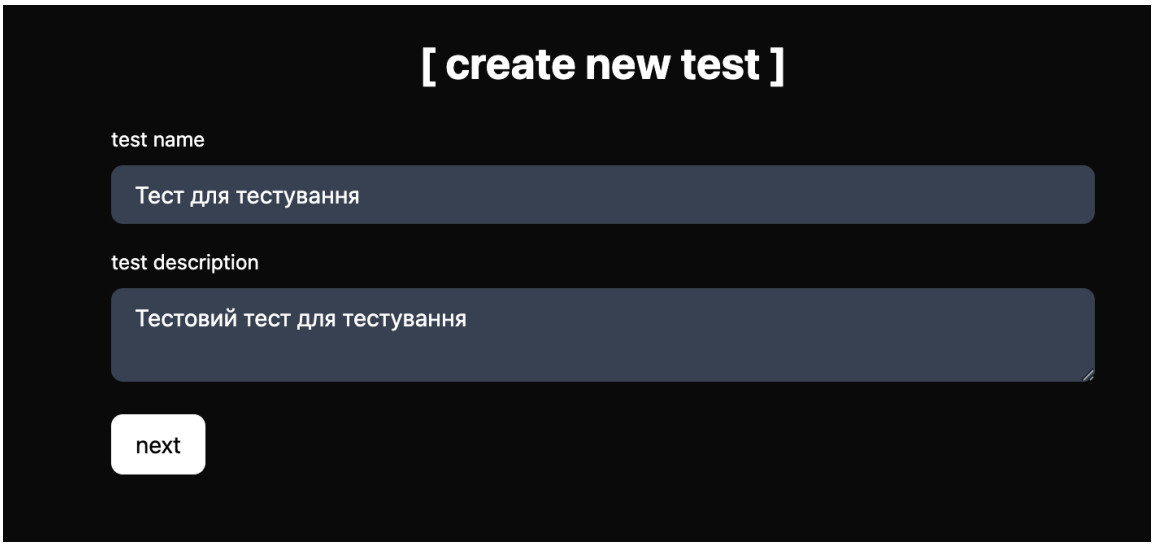

Рисунок В.13 – Вигляд інтерфейсу заповнення базової інформації про тест

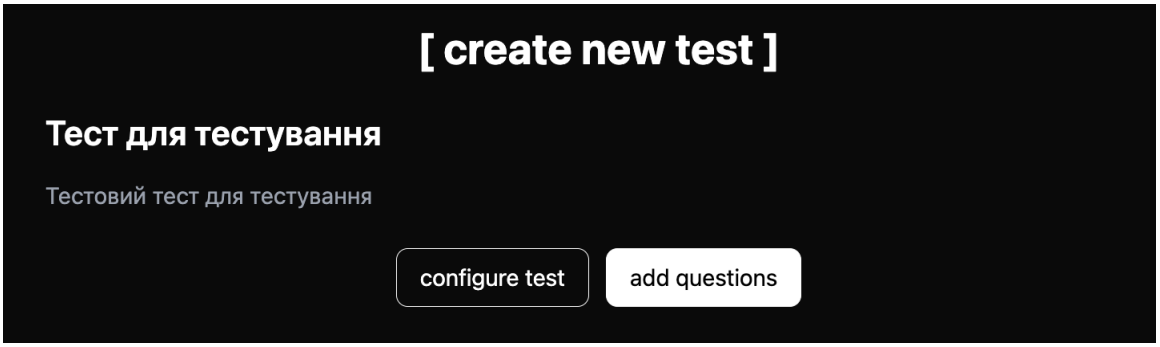

Рисунок В.14 – Вигляд інтерфейсу налаштування пулів тесту та додавання питань

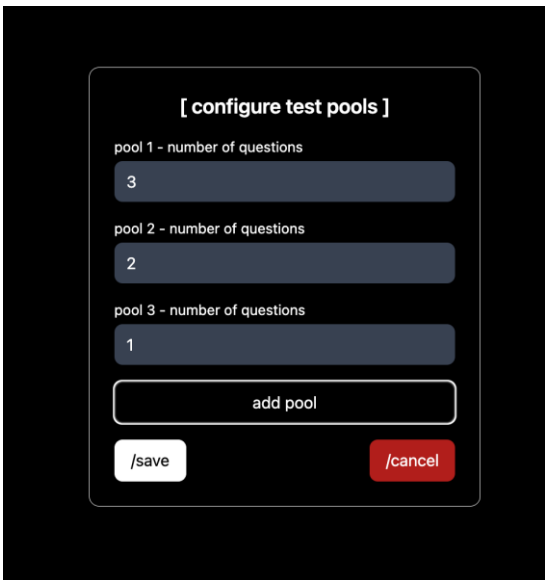

Рисунок В.15 – Вигляд інтерфейсу модального вікна для налаштування тесту

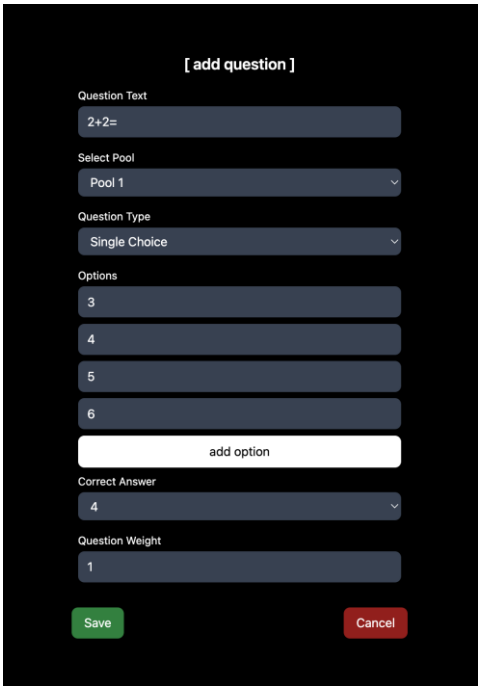

Рисунок В.16 – Вигляд інтерфейсу модального вікна для додавання питань з однією відповіддю

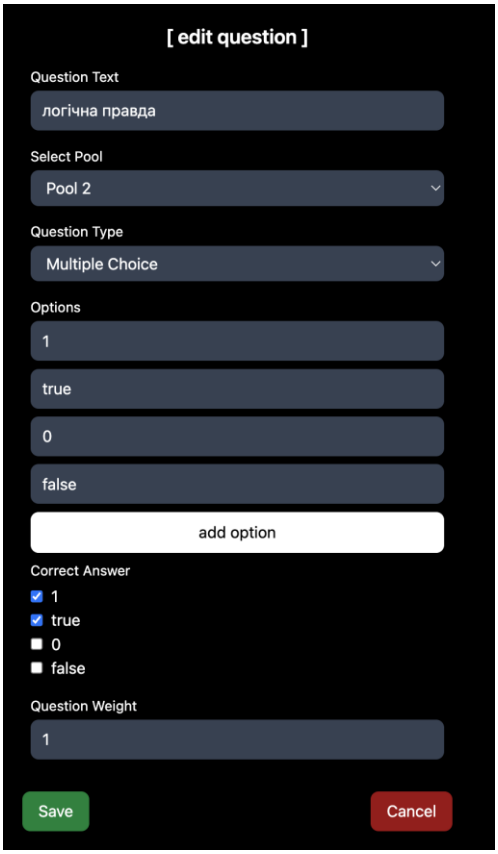

Рисунок В.17 – Вигляд інтерфейсу модального вікна для додавання питань з кількома відповідями

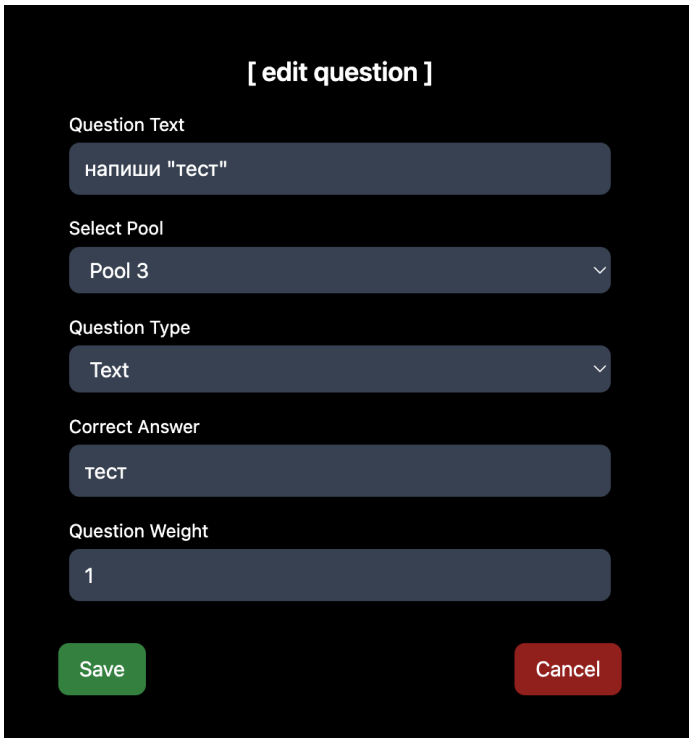

Рисунок В.18 – Вигляд інтерфейсу модального вікна для додавання питань з відповіддю у вигляді тексту

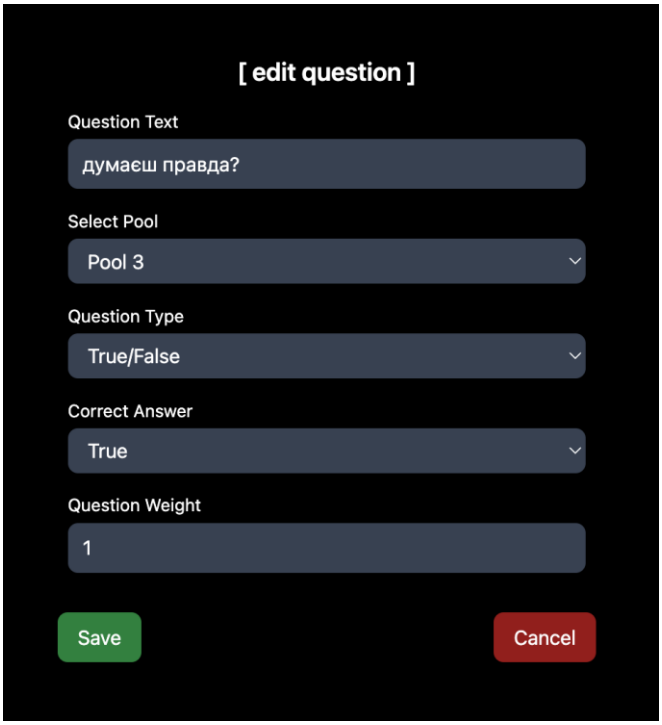

Рисунок В.19 – Вигляд інтерфейсу модального вікна для додавання питань з відповіддю правда/брехня

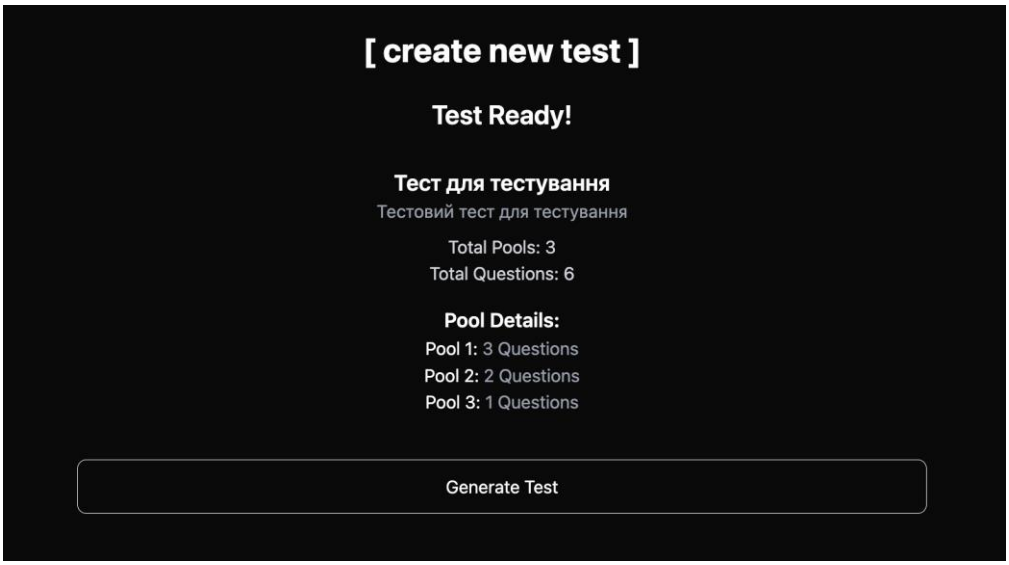

Рисунок В.20 – Вигляд інтерфейсу зведеної інформація про тест

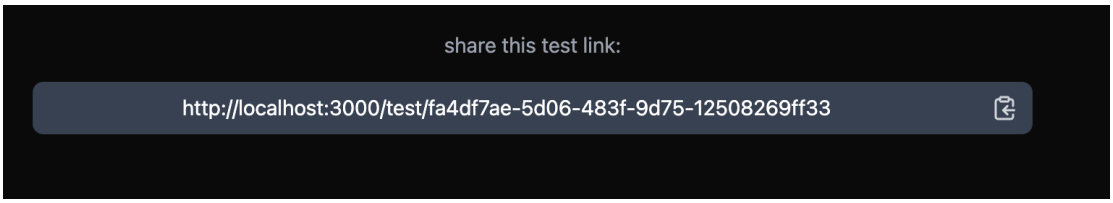

Рисунок В.21 – Вигляд інтерфейсу відповіді системи з посилання для поширення

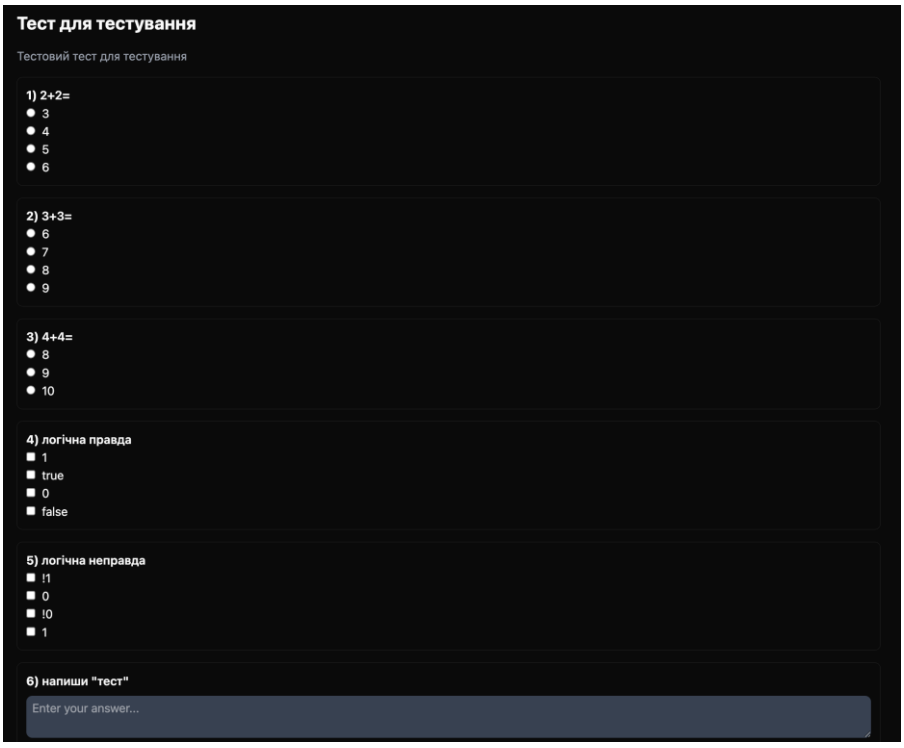

Рисунок В.22 – Загальний вигляд інтерфейсу проходження тесту

Рисунок В.23 – Вигляд інтерфейсу оцінки за пройдений тест

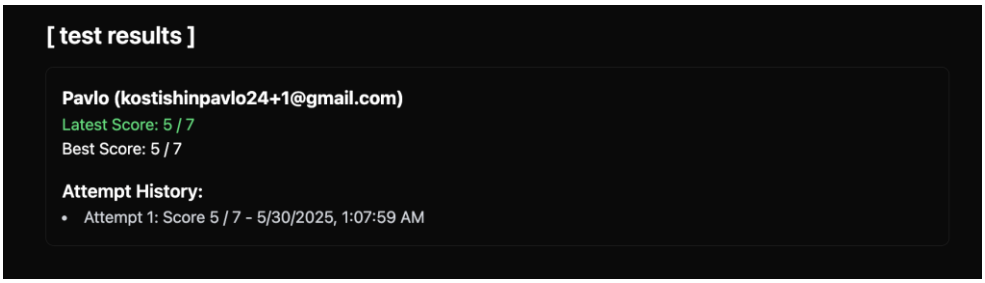

Рисунок В.24 – Вигляд інтерфейсу результату проходження тесту

ДОДАТОК Г (довідниковий)
Інструкція користувача

Для початку роботи з WEB-ресурсом потрібно встановити Node.JS версії 18. Після, встановити необхідні пакети командою (npm install) і запустити проєкт командою (npm run build). Після цього, WEB-ресурс відкриється у новій вкладці браузеру.

Для початку роботи з основним функціоналом WEB-ресурсу, потрібно зареєструватися або зайти в раніше створений акаунт (рис. Г1, Г2).

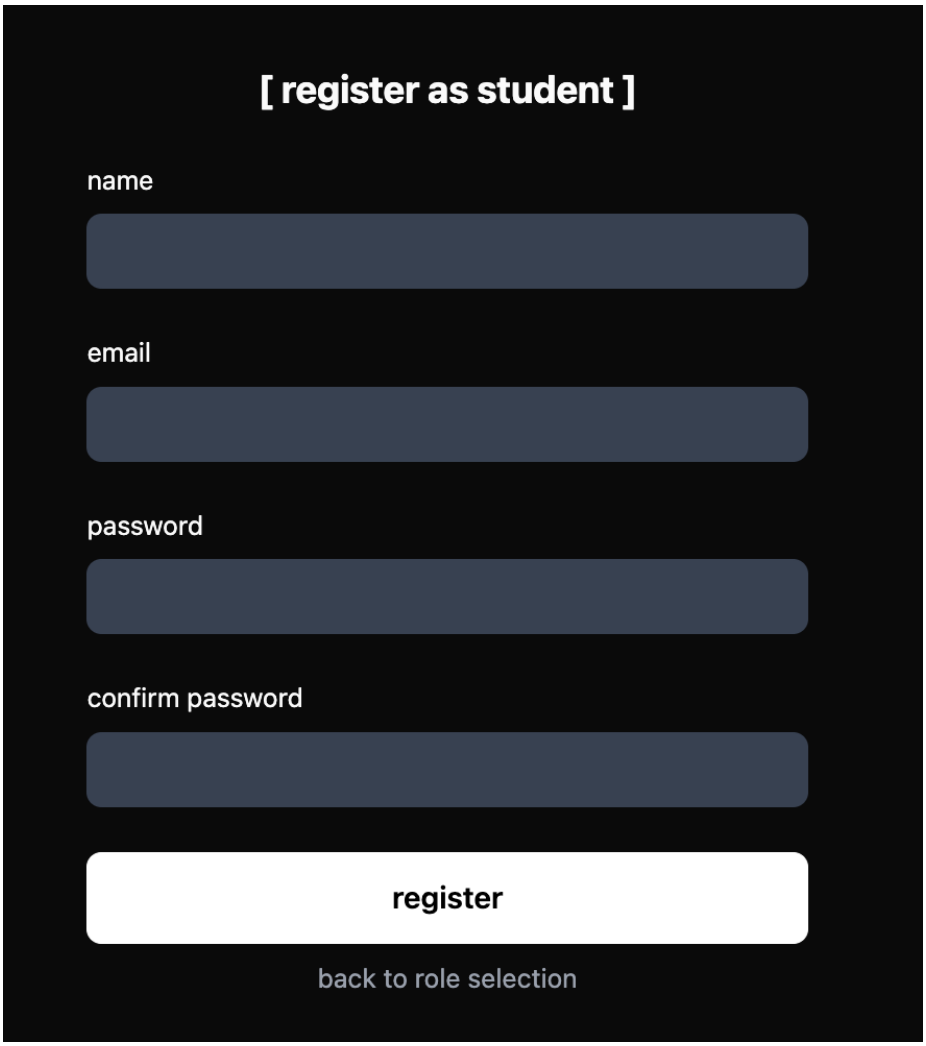

Рисунок Г.1 – Вигляд інтерфейсу сторінки реєстрації користувача

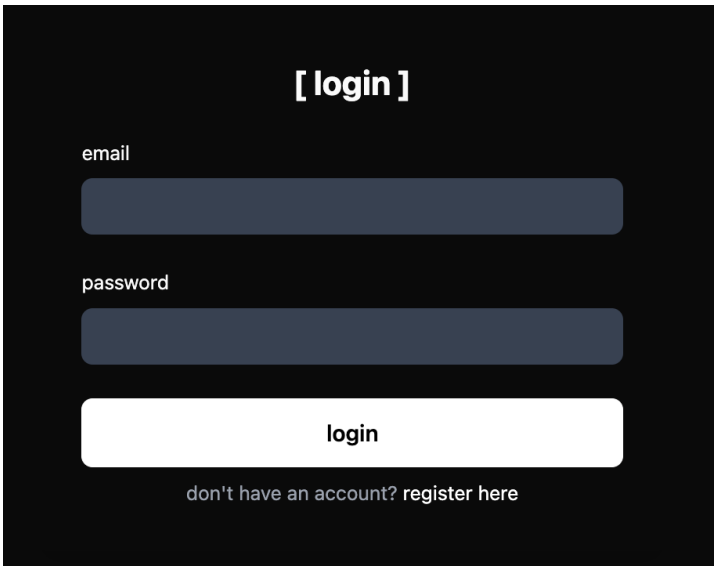

Рисунок Г.2 – Вигляд інтерфейсу сторінки авторизації користувача

Після авторизації, користувач потрапляє на сторінку профілю (рис. Г3).

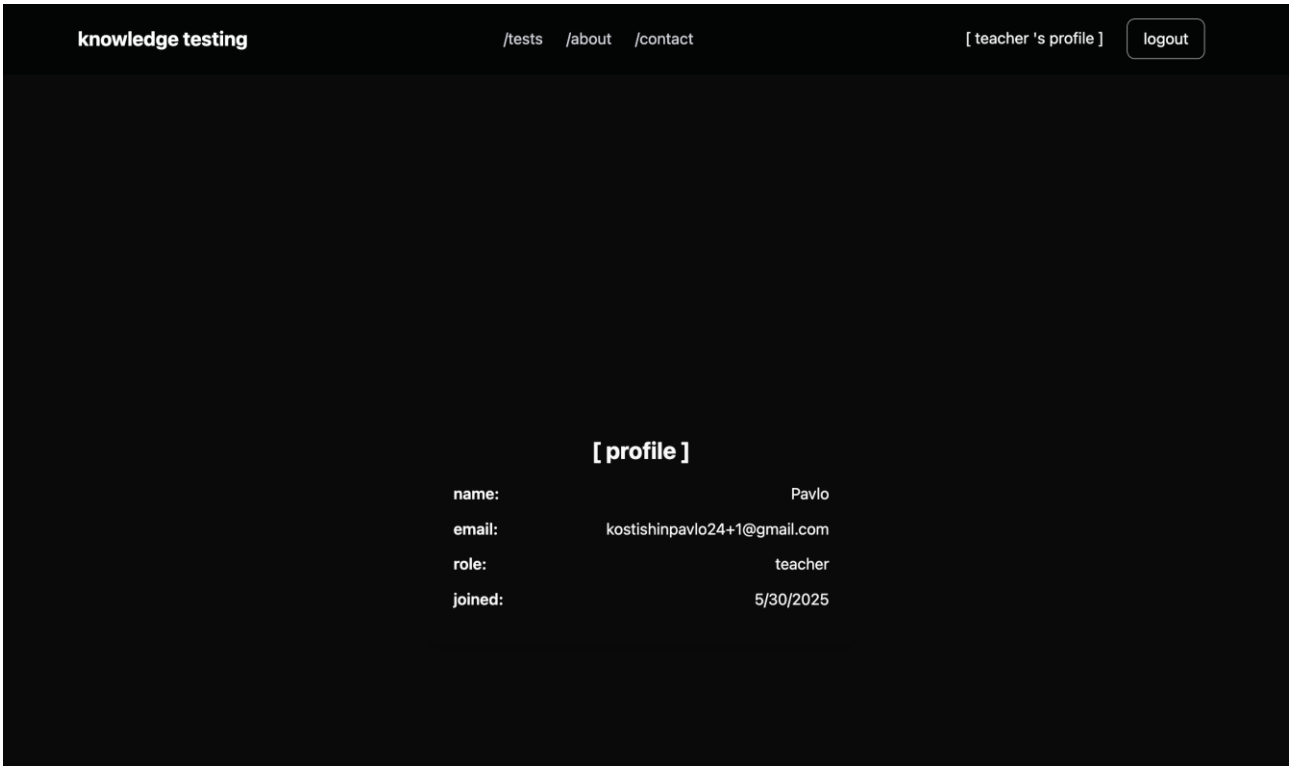

Рисунок Г.3 – Вигляд інтерфейсу сторінки профілю

Тут користувач може з навігаційної панелі перейти на різні сторінки, в тому числі на сторінку для створення та перегляду тестів (рис. Г4).

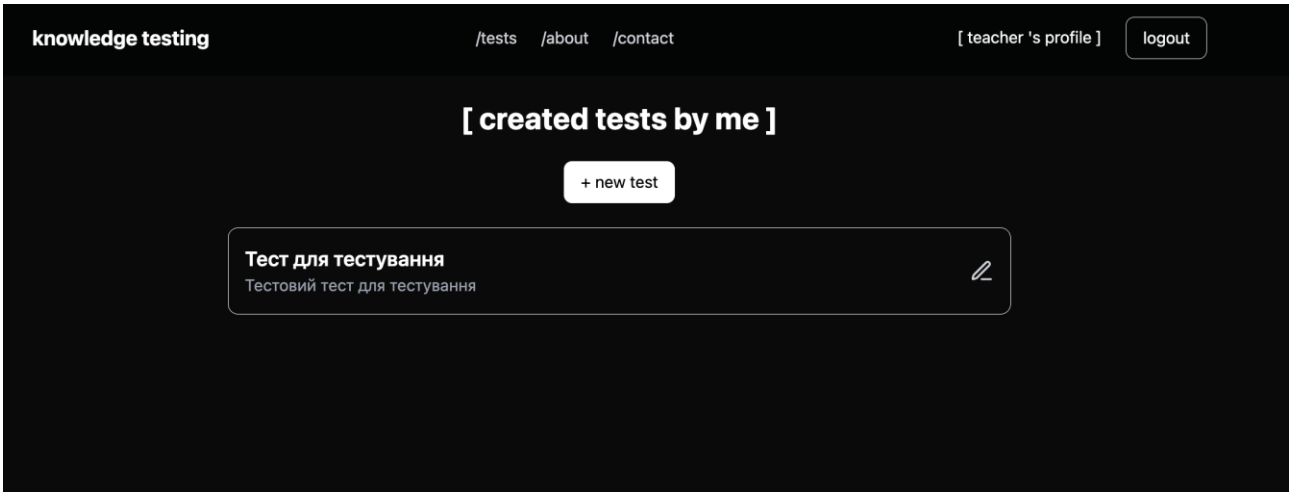

Рисунок Г.4 – Вигляд інтерфейсу сторінки тестів

Для того, щоб створити новий тест, потрібно натиснути кнопку «+ new test», після чого вказати основну інформацію про тест, а саме назву та опис (рис. Г5).

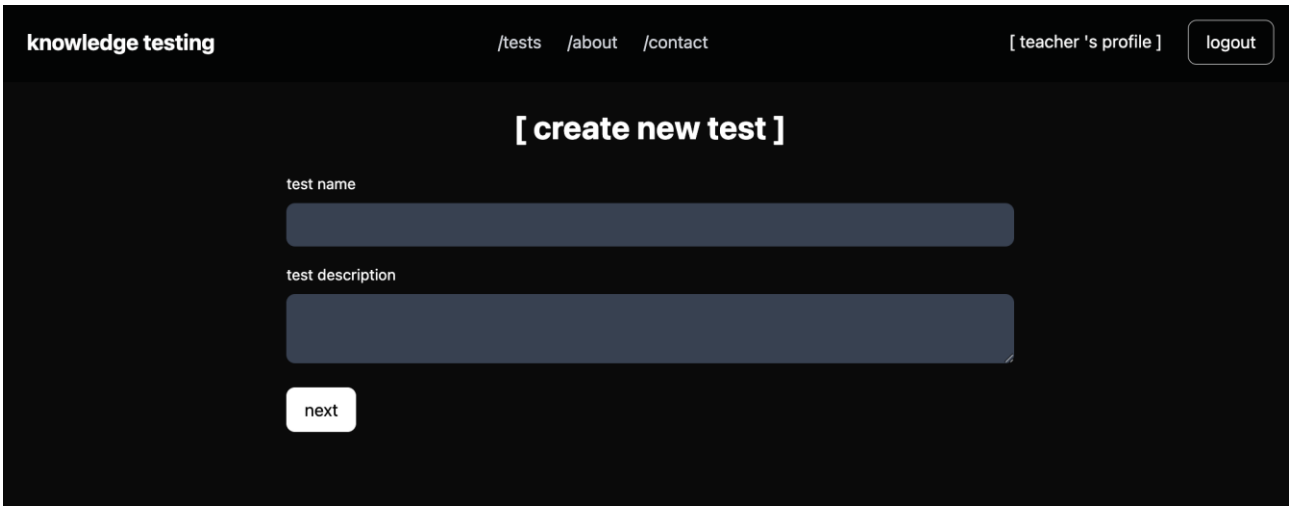

Рисунок Г.5 – Вигляд інтерфейсу сторінки наповнення основної інформації про тест

Після цього, користувачу доступно дві кнопки для редагування налаштувань тесту та для додавання питань до тесту. Налаштування тесту – це визначення кількості пулів питань і вказання скільки питань з кожного пулу має бути включено до кожного варіанту тесту (рис. Г6).

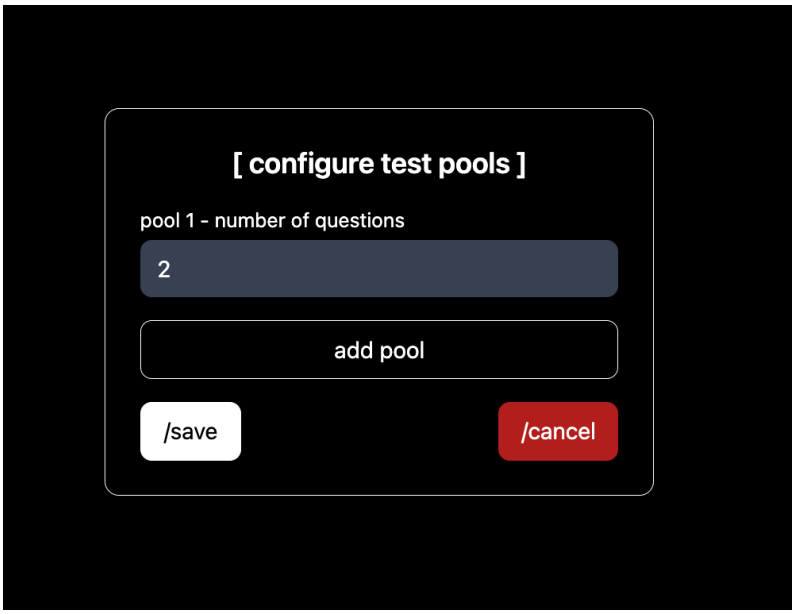

Рисунок Г.6 – Вигляд інтерфейсу налаштування тесту

Як тільки користувач налаштував необхідну кількість пулів, можна переходити до наступного етапу – додавання питань до тесту. Питання можуть бути чотирьох типів – з однією правильною відповіддю, з декількома правильними відповідями, правда/брехня та довільна текстова відповідь. При додаванні питань, користувач має обрати до якого пулу належить це питання. Питань в кожному пулі може бути більше ніж це було вказано на етапі налаштування тесту, це навпаки дасть можливість сформувати більшу кількість унікальних варіантів одного й того ж тесту. Також користувач може вказувати вагу питання (рис. Г7).

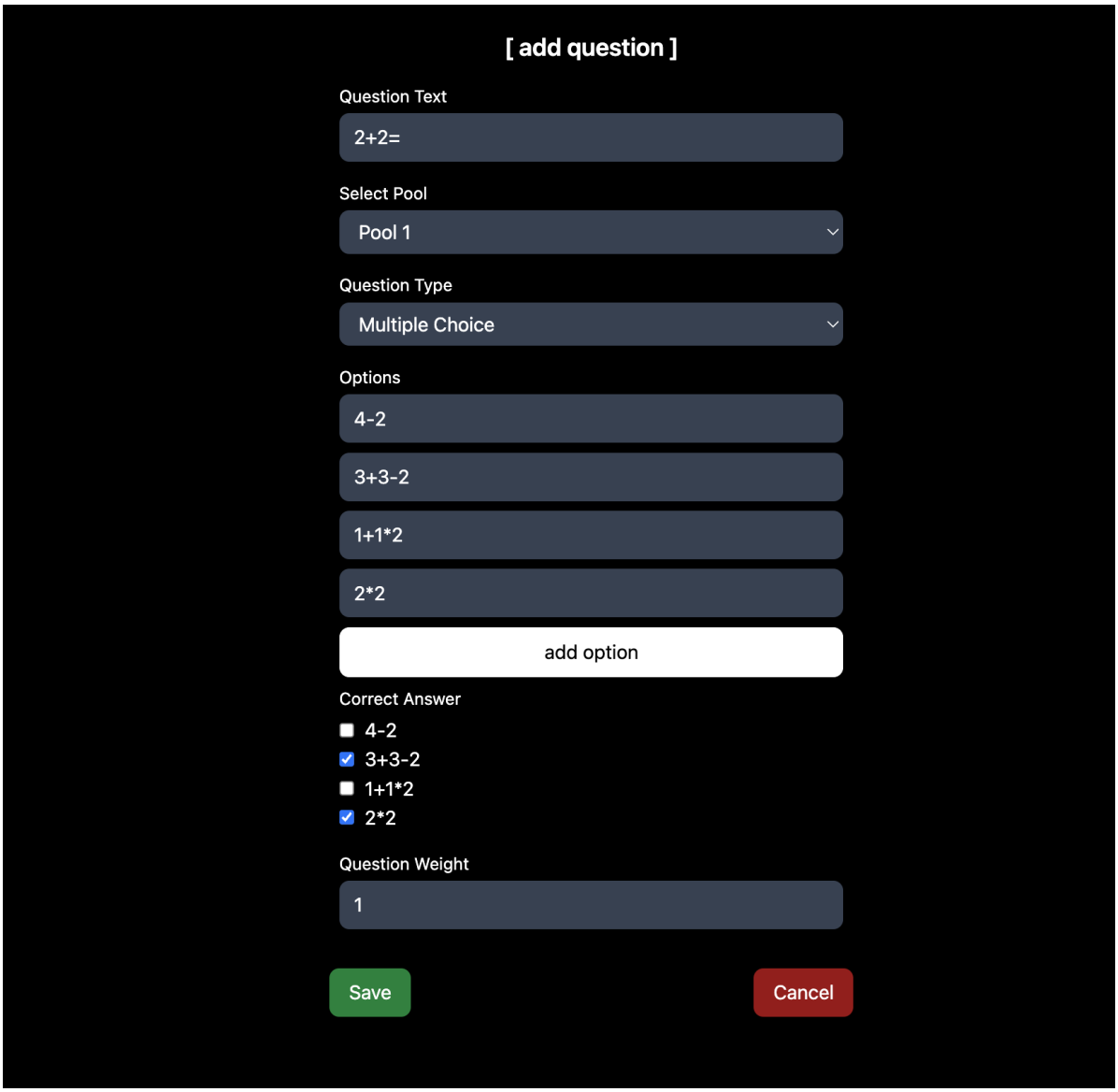

Рисунок Г.7 – Вигляд інтерфейсу додавання питання до тесту

Після того як користувач наповнив тест питаннями, він має можливість переглянути всі питання в загальному вигляді щоб переконатися, що вони сформовані належним чином і згенерувати тест (рис. Г8).

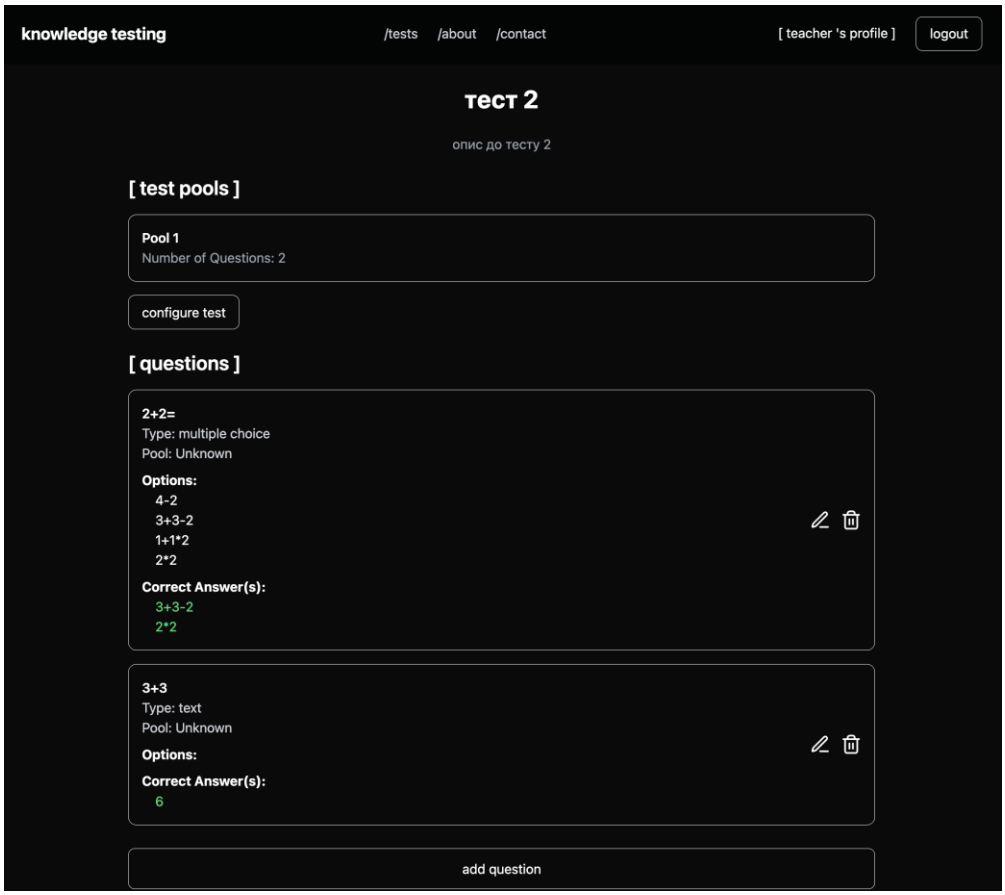

Рисунок Г.8 – Вигляд інтерфейсу сформованого тесту

Коли користувач згенерує тест, система дасть посилання, перейшовши за яким, можна пройти цей тест (рис. Г9).

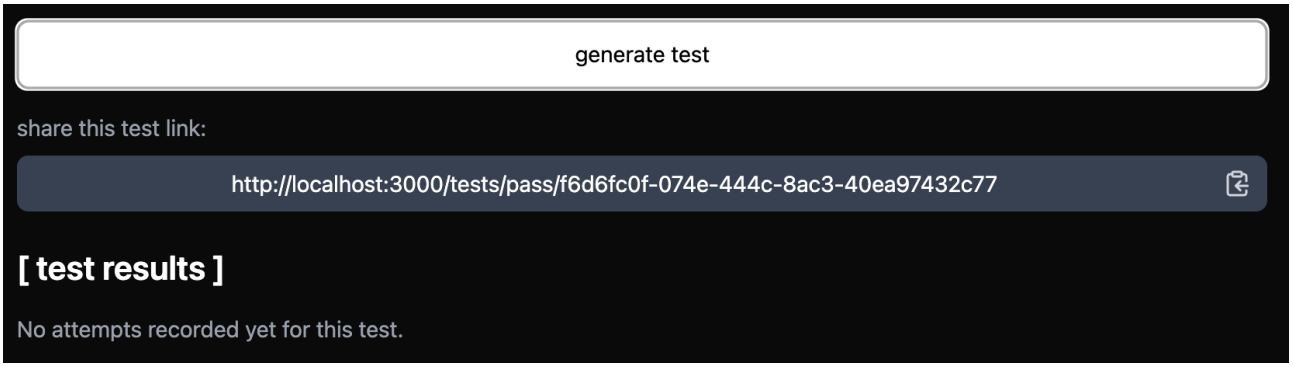

Рисунок Г.9 – Вигляд інтерфейсу згенерованого посилання для тесту

Перейшовши за посиланням, авторизований користувач побачить інтерфейс проходження тесту (рис. Г10).

Рисунок Г.10 – Вигляд інтерфейсу проходження тесту

Після проходження тесту, користувач, який створив цей тест може переглядати результати проходження цього тесту (рис. Г11).

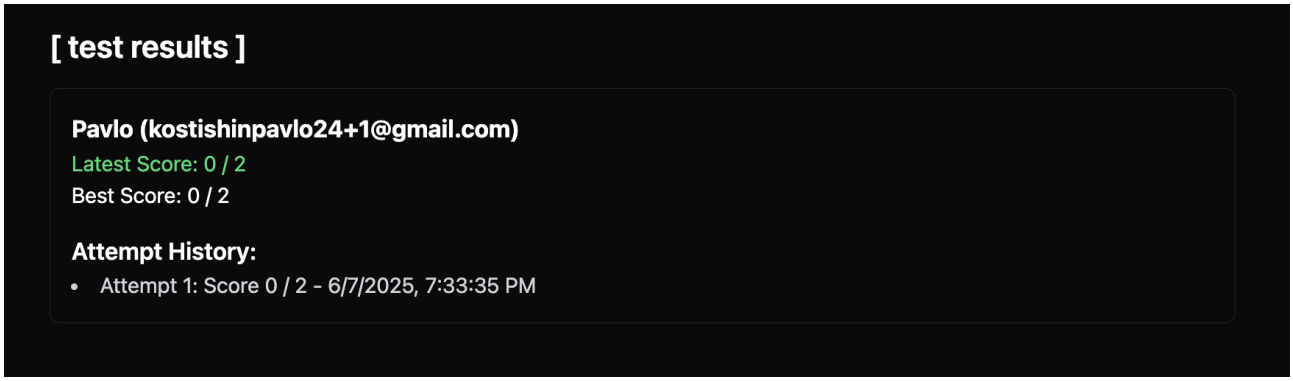

Рисунок Г.11 – Вигляд інтерфейсу перегляду результатів тесту

На мобільному пристрої послідовність кроків аналогічна, змінюється лише розташування головного меню.

ДОДАТОК Д

Свідоцтво про реєстрацію авторського права на твір

УКРАЇНА

СВІДОЦТВО

про реєстрацію авторського права на твір

№ 136024

Комп'ютерна програма «WEB-ресурс для тестування знань студентів»

(вид, назва твору)

Автор (співавтори) **Крилик Людмила Вікторівна, Костишин Павло Віталійович**

(прізвище, ім'я, по батькові (за наявності), псевдонім (за наявності))

Авторські майнові права належать спільно **Крилик Людмила Вікторівна, вул. Політехнічна, 3, кв. 314, м. Вінниця, 21021; Костишин Павло Віталійович, вул. Келецька, 98, м. Вінниця, 21021**

(прізвище, ім'я, по батькові (за наявності) фізичної особи / найменування юридичної особи, адреса)

Дата реєстрації 12 травня 2025 р.

Директор Державної організації «Український національний офіс інтелектуальної власності та інновацій» **Олена ОРЛЮК**

УКРНОІВІ