КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ТЕСТУВАННЯ

Виконав студент 4-го курсу
Артем ПОЛТАВСЬКИЙ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Оксана ШКІЛЬНЯК _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р., протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 27 ілюстрацій, 18 джерел посилань.

АРХІТЕКТУРА, ВЕБСИСТЕМА, ЕФЕКТИСВНІСТЬ, НАВЧАННЯ, ПРОГРАМНИЙ ЗАСІБ, ПРОДУКТИВНІСТЬ, ТЕСТУВАННЯ.

Об’єктом роботи є аналіз та вдосконалення процесу тестування в навчальному середовищі шляхом розробки нового програмного засобу для тестування. Вебсистема спрямована на покращення якості та ефективності тестування.

Предметом роботи є вебсистема, яка призначена для проведення тестування в різних навчальних контекстах.

Метою роботи є створення вебзастосунку, який надає зручні та ефективні інструменти для проведення тестів, оцінки відповідей та генерації результатів. Вебсистема має на меті спростити та автоматизувати процес тестування, забезпечуючи точність та надійність результатів.

Методи розроблення: аналіз, проєктування архітектури, розроблення коду, тестування. Інструменти розроблення: мова програмування Java, інтегроване середовище розробки IntelliJ IDEA, засоби розробки фронтенду (HTML, CSS, JavaScript), засоби розроблення бекенду (Spring).

Результати роботи: розроблено та протестовано програмне забезпечення, проведено оцінку його якості, відповідності вимогам та специфікаціям, ефективності та продуктивності.

Розроблений програмний продукт є потужним інструментом для використання в навчальних контекстах, що може підвищити якість тестування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

HTTP (HyperText Transfer Protocol) - протокол передачі даних, що використовується в комп’ютерних мережах

CSS (Cascading Style Sheets) – спеціальна мова, яка використовується для запису оформлення сторінок, написаних мовами розмітки даних.

HTML (HyperText Markup Language) – стандартизована мова розмітки документів для перегляду вебсторінок у браузері.

MVC (Model-View-Controller) – архітектурних шаблон, який використовується під час проєктування та розробки програмного забезпечення.

JWT (JSON Web Token) – це стандарт токену доступу на основі JSON.

JSON (JavaScript Object Notation) – це текстовий формат обміну даними між комп’ютерами.

SQL (Structured Query Language) – декларована мова програмування ля взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними базами даних.

GUI (Graphical User Interface) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівники.

API (Application Programming Interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забеспечення.

REST (REpresentational State Transfer) – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному світі розвиток технологій та вебдодатків є надзвичайно актуальною та швидкозмінною сферою. Розробка вебзастосунків для тестування стала необхідністю у зв'язку зі зростанням популярності дистанційного навчання та онлайн-тестування. Створення зручного та ефективного інструменту для проведення тестування є важливим завданням, що дозволяє спростити та автоматизувати процес оцінки знань та навичок користувачів.

Актуальність роботи та підстави для її виконання. З урахуванням зростаючої потреби у проведенні тестування та оцінки знань, розробка вебзастосунку для тестування має великий практичний потенціал. Цей проєкт став можливим завдяки потужному розвитку технологій веброзробки, які надають широкі можливості для створення функціональних та зручних інструментів.

Мета й завдання роботи. Метою даної роботи є розробка вебзастосунку для тестування, який надає зручні та ефективні інструменти для проведення тестів, оцінки відповідей та генерації результатів. Основні завдання включають створення інтерфейсу для користувачів, реалізацію функціональності аутентифікації та авторизації, обробку запитів на створення тестів та відповідей, а також збереження та відображення результатів тестування.

Перед виконанням роботи були поставлені такі завдання:

а) Аналіз вимог до системи тестування та формулювання вимог.

б) Розробка архітектури вебзастосунку та вибір відповідних технологій розробки.

в) Розробка інтерфейсу користувача, який буде забезпечувати зручність та ефективність проведення тестів.

г) Розробка функціоналу авторизації та аутентифікації користувачів.

д) Реалізація функціоналу для створення, редагування та видалення тестів та питань.

е) Розробка алгоритмів оцінювання результатів тестування та підготовка звіту з результатами.

ж) Тестування, налагодження та оптимізація вебзастосунку для забезпечення його надійності та продуктивності.

з) Проведення порівняльного аналізу з існуючими рішеннями в галузі тестування.

Об'єкт, методи й засоби розроблення. Об’єктом дослідження є розробка вебзастосунку для тестування, який має на меті поліпшення процесу проведення тестів та оцінки знань. Для досягнення поставленої мети використовуються методи аналізу вимог, проєктування архітектури системи, розробки контролерів та сервісів, інтеграції з іншими компонентами системи.

Можливі сфери застосування. Розроблений вебзастосунок для тестування може знайти застосування в різних сферах, таких як освіта, підготовка до іспитів, перевірка знань працівників у компаніях тощо. Його можна використовувати як в навчальних закладах, так і у корпоративному середовищі, де потрібно провести ефективне тестування та оцінку знань користувачів.

Взаємозв’язок з іншими роботами. Розробка вебзастосунку для тестування базується на попередніх дослідженнях та розробках у сфері веброзробки, аутентифікації та авторизації, а також управління тестами.

Кваліфікаційна робота спрямована на розробку вебзастосунку, який спрощує процес тестування та поліпшує його ефективність. Результати цього дослідження можуть бути корисними для розробників програмного забезпечення, тестувальників та осіб, які займаються навчанням та оцінюванням знань.

РОЗДІЛ 1 ВПЛИВ ТЕСТУВАННЯ НА НАВЧАЛЬНИЙ ПРОЦЕСС

В сучасному світі тестування використовується як потужний інструмент для оцінювання знань, умінь, навичок. Тестування впливає на різні аспекти навчального процесу і визначає якість та ефективність навчання. Його роль полягає не тільки в оцінці знань або засвоєння матеріалу, але й у стимуляції активності, поглибленого вивчення та розвитку критичного та аналітичних мислень.

Одним з ключових аспектів тестування є забезпечення об'єктивності та надійності оцінювання. Тестування надає стандартизований підхід до оцінювання знань і навичок студентів, що дозволяє отримати об'єктивні результати. Це дуже важливо в ситуаціях, коли необхідно порівнювати досягнення студентів, оцінювати їх підготовку та визначати рівень знань.

Також тестування сприяє активізації навчального процесу. Воно стимулює студентів або учнів до поглибленого вивчення матеріалу, пошуку правильних відповідей і розв’язування завдань. Тестові завдання можуть бути розроблені таким чином, щоб активувати аналітичне, критичне мислення та творчий підхід до розв'язання проблеми. Це сприяє студентів покращувати їх здатність застосовувати знання в реальних ситуаціях.

Окрім своїх освітніх функцій, тестування також має позитивний вплив на мотивацію студентів або учнів. Чітко визначені цілі та оцінка успішності, яку надає тестування, стимулюють учнів до досягнення високих результатів. Відчуття досягнення і успіху під час тестування може підвищити мотивацію до навчання та сприяти розвитку самодисципліни.

РОЗДІЛ 2 ТЕОРЕТИЧНИЙ ОГЛЯД

2.1 Огляд літератури

В процесі підготовки до розробки вебзастосунку для тестування було проведено огляд літератури, щоб вивчити існуючі методи та підходи до розробки подібних систем. Основний акцент був зроблений на наступній літературі:

а) "Software Engineering: A Practitioner's Approach" by Roger S. Pressman [1]: книга зосереджується на процесі розробки програмного забезпечення та надає рекомендації щодо ефективного впровадження ітеративних та промислових практик у процес розробки.

б) "Web Development with Django Cookbook" by Aidas Bendoraitis [2]: книга зосереджується на веброзробці з використанням фреймворку Django. Вона надає практичні поради та приклади для розробки вебзастосунків з використанням Django.

в) "RESTful Web Services" by Leonard Richardson and Sam Ruby [3]: книга надає введення в архітектуру RESTful вебсервісів та надає рекомендації щодо розробки вебсервісів з використанням цієї архітектури.

г) "Secure Java: For Web Application Development" by Abhay Bhargav and B.V. Kumar [4]: книга зосереджується на безпеці вебзастосунків розроблених на мові Java. Вона надає практичні поради та приклади для забезпечення безпеки вебзастосунків.

Вивчення джерел дозволило отримати актуальну інформацію про існуючі методи та підходи до розробки вебзастосунків, а також виділити основні виклики та проблеми, які потрібно враховувати при розробці. За результатами огляду літератури було визначено найкращі практики та рекомендації, які будуть використовуватися при розробці вебзастосунку.

2.2 Огляд технологій веброзробки

Перелічимо ключові технології та інструменти, які використовуються для розробки вебзастосунків.

а) HTML (HyperText Markup Language): HTML є основою вебсторінок і використовується для створення структури та вмісту вебдокументів.

б) CSS (Cascading Style Sheets): CSS використовується для задання зовнішнього вигляду вебсторінок, включаючи кольори, шрифти, розташування елементів тощо.

в) JavaScript: JavaScript є мовою програмування, яка дозволяє створювати динамічні ефекти, взаємодіяти з користувачем та керувати поведінкою вебсторінок.

г) Frameworks: у сучасній веброзробці популярними є фреймворки, такі як React, Angular та Vue.js, які надають гнучкість та ефективність при розробці вебзастосунків.

д) Бази даних: для зберігання та управління даними вебзастосунків часто використовуються реляційні бази даних, такі як MySQL, PostgreSQL або NoSQL бази даних, такі як MongoDB.

2.3 Огляд ролі контролерів та сервісів у вебдодатках

У вебдодатках, особливо тих, що використовують архітектурний підхід MVC [11] (Model-View-Controller), контролери та сервіси відіграють важливу роль у розподілі функціональності та управлінні бізнес-логікою.

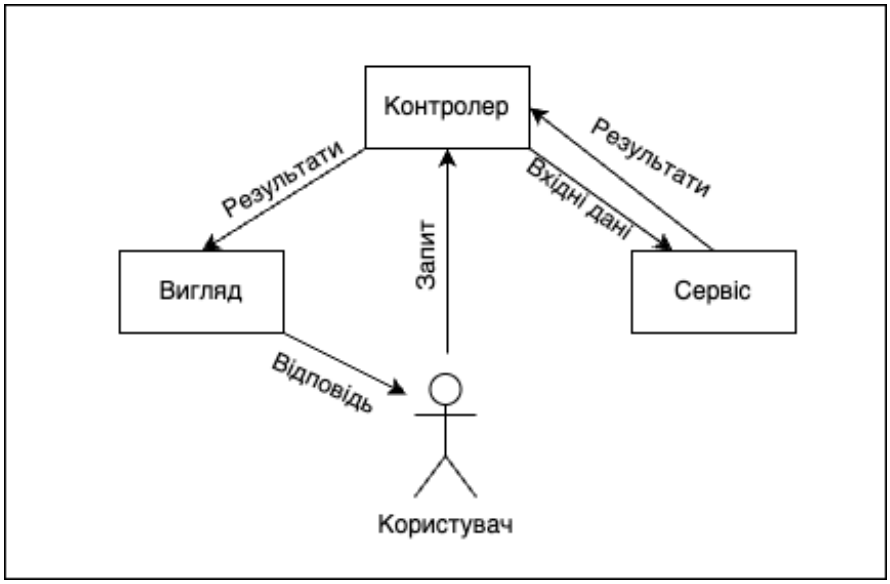

Рисунок 2.1 - Архітектурний підхід MVC.

Контролери є посередниками між клієнтом та сервером. Вони приймають HTTP-запити від клієнтів, обробляють їх та передають відповідь назад. Контролери відповідають за обробку маршрутів, параметрів запитів, а також валідацію даних, які надсилаються клієнтом. Вони викликають відповідні методи сервісів, які виконують бізнес-логіку та взаємодіють з базою даних (рисунок 2.1).

Сервіси, з іншого боку, відповідають за реалізацію бізнес-логіки додатку. Вони містять методи, які виконують специфічні операції та обчислення, пов'язані з функціональністю додатку. Сервіси можуть взаємодіяти з базою даних, іншими зовнішніми сервісами або компонентами додатку для виконання своїх завдань. Вони також можуть використовуватись для реалізації правил бізнес-логіки та управління транзакціями.

Розподіл функціональності між контролерами та сервісами дозволяє забезпечити розділення відповідальностей та полегшує тестування та обслуговування коду. Контролери відповідають за обробку HTTP-запитів та інтерфейсу з клієнтами, тоді як сервіси відповідають за реалізацію бізнес-логіки та обробку даних.

РОЗДІЛ 3 АУТЕНТИФІКАЦІЯ ТА АВТОРИЗАЦИЯ ВЕБДОДАТКІВ

3.1 Аутентифікація та авторизація

Аутентифікація та авторизація є важливими аспектами для забезпечення безпеки вебдодатків. Аутентифікація визначає ідентифікацію користувача, тобто перевірку, що він є тим, за кого себе видає. Авторизація визначає права доступу користувача до різних ресурсів або функціоналу системи.

У даний час існує багато підходів та методів для реалізації аутентифікації та авторизації вебдодатків. Деякі з них включають:

а) Сесії та кукі (Sessions and Cookies) [13]: Використовуючи сесії та кукі, сервер зберігає стан аутентифікації користувача та передає унікальний ідентифікатор сесії у куці користувачеві. Цей ідентифікатор використовується для перевірки аутентифікації при кожному запиті.

б) Токени доступу (Access Tokens) [14]: Використання токенів доступу є популярним підходом до аутентифікації та авторизації. При успішній аутентифікації сервер видає токен доступу, який передається у заголовку або запиті до сервера при кожному запиті. Сервер перевіряє цей токен для авторизації та визначення прав доступу.

в) OAuth і OpenID Connect [12]: OAuth та OpenID Connect є протоколами, які дозволяють користувачам надавати доступ до своїх облікових записів стороннім додаткам. Вони забезпечують безпеку та авторизацію шляхом обміну токенами між вебдодатком та провайдером ідентифікації.

Ці підходи до аутентифікації та авторизації мають свої переваги та недоліки, і вибір конкретного методу залежить від потреб вебдодатку та рівня безпеки. Тому в нашому випадку буде достатньо підходу з токеном доступу.

3.2 Використання JWT для аутентифікації

JWT (JSON Web Token) є стандартом для безпечної передачі інформації між сторонами у вигляді JSON-об'єктів. В контексті вебдодатків, JWT використовується для реалізації механізму аутентифікації, що дозволяє клієнтам отримувати токени для доступу до захищених ресурсів на сервері[10].

JWT складається з трьох частин (рисунок 3.1): заголовка, пейлоаду та підпису. Заголовок містить тип токена і алгоритм шифрування, пейлоад включає корисну інформацію, таку як ідентифікатор користувача та ролі, а підпис забезпечує перевірку цілісності та автентичності токена.

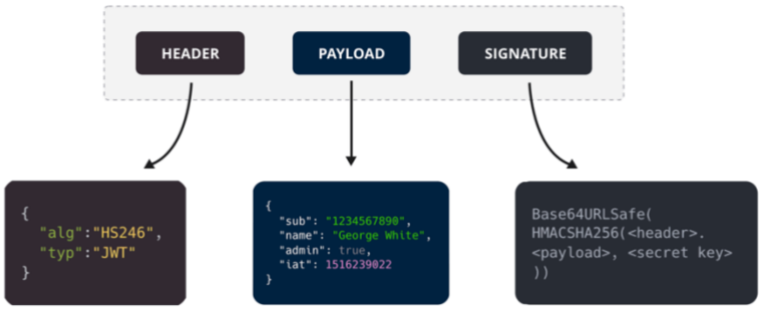

Рисунок 3.1 - Структура JWT.

При реалізації аутентифікації за допомогою JWT (рисунок 3.2), клієнт зазвичай надсилає свої облікові дані (наприклад, ім'я користувача та пароль) на сервер. Після успішної перевірки облікових даних, сервер генерує JWT токен, який надсилається назад до клієнта. Клієнт зберігає токен і надсилає його з кожним запитом до сервера. Сервер перевіряє підпис токена та інші перевірки, щоб підтвердити автентичність клієнта та надати доступ до захищених ресурсів.

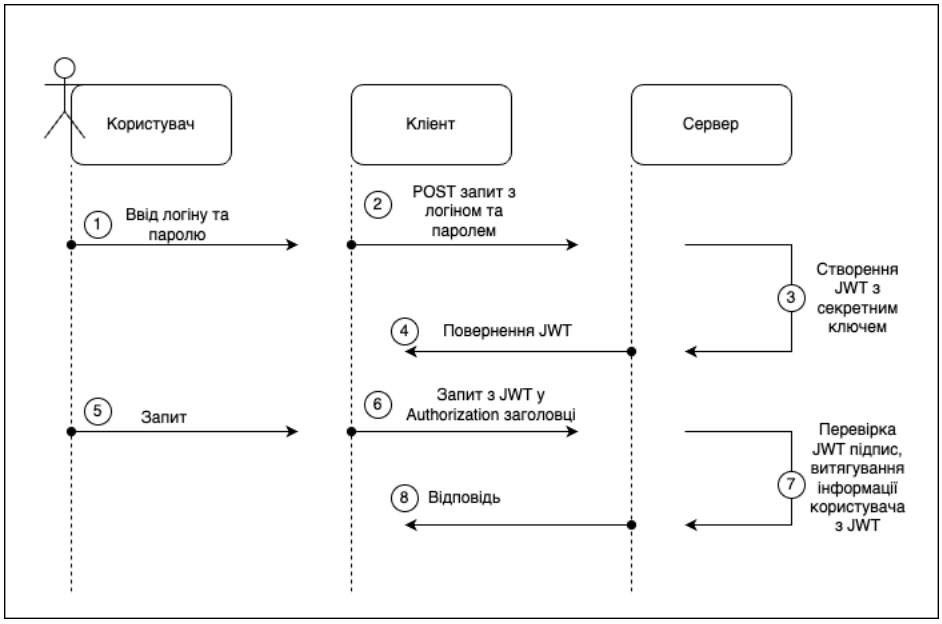

Рисунок 3.2 - Шлях запитів при праці з JWT.

Використання JWT для аутентифікації дозволяє створити безстанний (stateless) механізм аутентифікації, оскільки сервер не потребує збереження стану аутентифікації. Це робить його ефективним і масштабованим рішенням для вебдодатків.

РОЗДІЛ 4 БАЗА ДАНИХ

4.1 Вибір бази даних

Одним із важливих аспектів розробки вебзастосунків є вибір підходящої бази даних. У нашому випадку, під час розробки вебзастосунку для тестування в результаті вивчення різних баз даних було прийнято рішення обрати PostgreSQL [15].

PostgreSQL є потужною та надійною реляційною базою даних з відкритим вихідним кодом. Вона володіє широким спектром функціональності, включаючи підтримку SQL-запитів, транзакційності, реплікації та високої масштабованості. PostgreSQL також має активну спільноту розробників, що забезпечує стабільну підтримку та регулярні оновлення.

Базу даних PostgreSQL було обрано з декількох причин. По-перше, вона підтримує реляційну модель даних, яка добре підходить для зберігання структурованих даних, таких як інформація про користувачів, тести, питання та відповіді. По-друге, PostgreSQL має широкий набір функцій, таких як індексування, тригери та засоби оптимізації запитів, що дозволяють нам забезпечити ефективну роботу з базою даних навіть при великому обсязі даних. Крім того, PostgreSQL підтримує множинність засобів для забезпечення безпеки даних, таких як ролі та механізми аутентифікації.

4.2 Порівняння баз даних

Наведемо порівняння популярних баз даних MySQL та MongoDB з обраною PostgreSQL:

а) MySQL [16]: MySQL є ще однією популярною реляційною базою даних. Вона відома своєю швидкодією та простотою використання. Однак, порівняно з PostgreSQL, MySQL має меншу

кількість функціональних можливостей, особливо щодо розширення та гнучкості. Крім того, PostgreSQL надає більшу стійкість до відмов та кращу підтримку для складних операцій.

б) MongoDB[17]: MongoDB є документ-орієнтованою NoSQL базою даних, яка використовує JSON-подібні документи для зберігання даних. Вона відома своєю гнучкістю та масштабованістю. Однак, MongoDB має деякі обмеження щодо виконання складних операцій та транзакційності, що можуть бути важливими для нашого вебзастосунку.

Загалом, враховуючи вимоги нашого вебзастосунку, включаючи потребу в реляційній моделі даних, багатофункціональності та надійності, ми прийняли рішення вибрати PostgreSQL як базу даних для нашого проєкту. Вона забезпечує нам потрібний функціонал та ефективну роботу з даними, що допоможе забезпечити успішну реалізацію нашого вебзастосунку для тестування.

Вибір PostgreSQL як бази даних для вашого вебзастосунку для тестування є логічним рішенням. Основні причини цього вибору можуть бути такі [15]:

а) Реляційна модель даних: PostgreSQL підтримує реляційну модель даних, що добре підходить для зберігання структурованих даних, таких як інформація про користувачів, тести, питання та відповіді. Реляційні бази даних забезпечують зв'язки між таблицями, що дозволяє ефективно працювати зі складними структурами даних.

б) Функціональність: PostgreSQL має широкий набір функціональності, яка може бути важливою для вашого вебзастосунку. Це включає підтримку SQL-запитів, транзакційності, реплікації та високої масштабованості. PostgreSQL також надає інструменти для оптимізації запитів, що дозволяють покращити продуктивність застосунку.

в) Надійність: PostgreSQL відома своєю надійністю і стійкістю до відмов. Вона може гарантувати цілісність даних і забезпечити доступ

до них навіть у разі непередбачуваних ситуацій, таких як відключення електропостачання або системні збої. Це особливо важливо для вебзастосунків, які потребують надійного зберігання даних.

г) Спільнота розробників: PostgreSQL має активну спільноту розробників, що забезпечує стабільну підтримку та регулярні оновлення. Якщо у вас виникнуть питання або проблеми, можна звернутися до цієї спільноти для отримання допомоги та порад.

Порівняно з альтернативами, такими як MySQL і MongoDB, PostgreSQL має свої переваги. MySQL може бути простим у використанні, але має меншу кількість функцій і можливостей порівняно з PostgreSQL. MongoDB, хоча є гнучкою і масштабованою NoSQL базою даних, може мати обмеження щодо виконання складних операцій та транзакційності.

Загалом, вибір PostgreSQL відповідає вашим потребам у вебзастосунку для тестування. Вона надасть нам необхідні засоби для ефективної роботи з даними, забезпечує надійність і має підтримку спільноти розробників.

4.3 Структура бази даних

Для початку треба було вирішити проблему зберігання користувачів. Для цього було створено таблицю users, яка містить найголовнішу інформацію – username та password (логін та пароль) – для ідентифікації користувача. Якщо ми захочемо розширити та зберігати більше даних про користувача, то можна розширити цю таблицю, PostgreSQL надає таку можливість.

Також були додані ролі для користувачів. Щоб не обмежуватися на ADMIN або USER, було вирішено створити окрему таблицю roles для зберігання цих ролей та подальшого масштабування. Для поєднання

двух таблиць users та roles було створену проміжну таблицю user_ roles (рисунок 4.1).

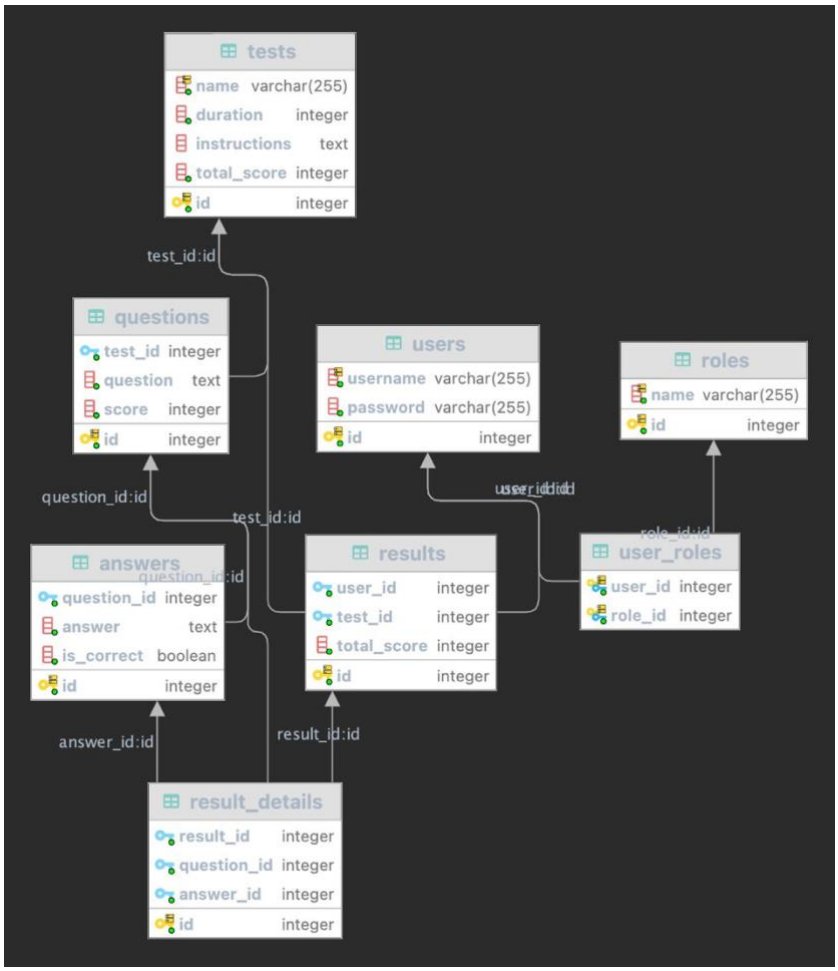

Рисунок 4.1 – Модель бази даних.

Для зберігання самих тестів після аналізу можливих варіантів було обрано наступну схему. Була створена таблиця tests для збереження загальної інформації про тест, а саме ім’я, тривалість, інструкція, загальний бал та id для цього тесту, щоб сервіси могли розуміти, з яким тестом вони зараз працюють. Далі йде таблиця questions, яка створена для зберігання самих питань та кількості балів за нього, для підрахунку загальної оцінки. Для розділення питань між тестами була створена колонка test_id, яка вказувала на id тесту, до якого відносилась. Так як варіантів відповідей може бути необмежена кількість або відповідь розгорнута, було вирішено відокремити окрему таблицю answers для

збереження відповідей на той тест. Відповідь може бути правильною або неправильною, тому було додане відповідне поле isCorrect (рисунок 4.1).

Для надання більших можливостей для нашого застосунку в майбутньому, такі як вивантаження результатів, підрахунок деякої статистики, було вирішено додати збереження результатів тестів. Було створено таблиці results та result_details. Перша таблиця створена для зберігання загальної, короткої інформації про закінчення тесту, а саме який користувач на скільки балів пройшов той чи інший тест. Друга таблиця створена для збереження результатів по кожному питанню окремо, це надає можливість ознайомитись зі своїми відповідями та опрацювати їх (рисунок 4.1).

При розробці структури було виявлено та вирішено ряд проблем, ось деякі з них.

а) Для збереження результатів було створено саме дві таблиці для оптимізації роботи з базою даних. Якби ми все зберігали в одній, а при кожному звернені фронтсервісу до загальних результатів ми б рахували їх, то бексервіс та сама база даних не витримали би цього навантаження в майбутньому.

б) Таблиця result_details розростається дуже швидко: якщо взяти 5 тестів по 20 питань та один клас з 30 учнів, то це мінімум 3000 записів. Тому було вирішено обрати в якості бази даних саме PostreSQL, яка в таких умовах може спокійно працювати, може горизонтально масштабуватись та не втрачати через це час на обробку запитів.

РОЗДІЛ 5 РОЗРОБКА БЕКЕНДУ

5.1 Вибір технологій розробки

Першим етапом при розробці будь-якого сервісу є вибір технологій. На початку було вирішено використовувати JWT [10] (JSON Web Token) та мову Java.

Використання мови програмування Java має багато причин, серед яких можна виділити наступні.

а) Широке поширення та підтримка: Java є однією з найпопулярніших мов програмування. Це означає, що є велика кількість ресурсів, документації, бібліотек та інструментів, які допомагають у розробці та підтримки проєктів.

б) Кросплатформність: програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції [5].

в) Велика екосистема: Java включає до себе чисельні бібліотеки, які розробляються сторонніми розробниками ще з часів народження мови Java – 1996 рік. До них належать, зокрема, відомі фреймворки Spring, Hibernate для серверного програмування, JavaFX для розробки GUI (Graphical User Interface), і більш спеціалізовані бібліотеки, наприклад бібліотека комп’ютерного зору OpenCV або машинного навчання TensorFlow [6].

г) Безпека: Робота програм знаходиться під повним контролем віртуальної машини, тому будь-який несанкціонований доступ до даних або підключення до іншої машини відразу ж переривається [7].

д) Масштабованість: Java дозволяє масштабувати продукт, додаючи в нього новий функціонал і грамотно розподіляючи при цьому навантаження [6].

Також були використані такі бібліотеки як:

а) Spring boot: дозволяє дуже легко інтегрувати архітектурний підхід MVC (Model-View-Controller);

б) Spring Security: надає легке API для користуванням JWT (JSON Web Token).

5.2 Реалізація контролерів та сервісів для аутентифікації та авторизації

Одним із важливих аспектів розробки бекенду було забезпечення безпеки та конфіденційності даних користувачів. Для цього було реалізовано систему аутентифікації та авторизації.

У рамках реалізації були створені контролери та сервіси, які відповідають за обробку запитів користувачів, пов'язаних з аутентифікацією та авторизацією. Ми використовували JWT [10] (JSON Web Token) для забезпечення безпеки та ідентифікації користувачів. Під час аутентифікації користувача, було перевірено його облікові дані та згенеровано та підписано JWT, який був відправлений назад до клієнта. Після цього, при кожному наступному запиті, клієнт повинен надати JWT, який перевіряється на сервері для перевірки автентичності та визначення прав доступу користувача.

Реалізація контролерів та сервісів для аутентифікації та авторизації дозволяє нам ефективно керувати доступом користувачів до ресурсів системи та забезпечувати безпеку даних.

5.2.1 Клас AuthService та його методи

У класі AuthService реалізовано методи, які відповідають за логіку аутентифікації та авторизації користувачів.

Метод registerUser. Цей метод приймає об'єкт, який містить дані нового користувача, такі як ім'я та пароль. Метод перевіряє введені дані на валідність та перевіряє, чи існує вже користувач з таким самим ім’ям.

Якщо всі перевірки успішні, метод створює нового користувача в системі та зберігає його дані в базі даних (рисунок 5.2.1).

```
public ApiResponse registerUser(UserDto registrationRequestDto) {
  String username = registrationRequestDto.getUsername();
  if (userRepository.existsByUsername(username)) {
    return new ApiResponse( success: false,  message: "Username is already taken!");
  }

  User user = new User();
  user.setUsername(username);
  user.setPassword(passwordEncoder.encode(registrationRequestDto.getPassword()));

  Set<String> strRoles = registrationRequestDto.getRoles();
  Set<Role> roles = new HashSet<>();
```

Рисунок 5.2.1 – Реалізація методу registerUser.

Метод authenticateUser. Цей метод приймає об'єкт, який містить дані входу користувача, такі як електронна пошта та пароль. Метод перевіряє введені дані та перевіряє, чи існує користувач з такими ж даними в системі. Якщо перевірка успішна, метод аутентифікує користувача та повертає токен доступу, який може використовуватися для авторизації при наступних запитах (рисунок 5.2.2).

```
public String authenticateUser(UserDto loginRequest) {
  Authentication authentication = authenticationManager.authenticate(
      new UsernamePasswordAuthenticationToken(
          loginRequest.getUsername(),
          loginRequest.getPassword()
      )
  );

  SecurityContextHolder.getContext().setAuthentication(authentication);
  return tokenProvider.generateToken(authentication);
}
```

Рисунок 5.2.2 – Реалізація методу authenticateUser.

Метод changePassword. Цей метод приймає об'єкт, який містить дані для зміни пароля користувача. Метод перевіряє, чи введений поточний

пароль користувача вірний, і якщо так, то змінює пароль на новий, переданий в запиті (рисунок 5.2.3).

```
public void changePassword(ChangePasswordRequestDto changePasswordRequest, UserPrincipal userPrincipal) {
  User user = userRepository.findById(userPrincipal.getId())
      .orElseThrow(() -> new ResourceNotFoundException("User", "id", userPrincipal.getId()));

  if (!passwordEncoder.matches(changePasswordRequest.getOldPassword(), user.getPassword())) {
    throw new BadRequestException("Invalid old password");
  }

  user.setPassword(passwordEncoder.encode(changePasswordRequest.getNewPassword()));
  userRepository.save(user);
}
```

Рисунок 5.2.3 – Реалізація методу changePassword.

Метод logoutUser. Цей метод відповідає за вихід користувача з системи. Він прибирає JWT-токен, який був виданий користувачу, забезпечуючи таким чином вихід з системи (рисунок 5.2.4).

```
public void logoutUser(HttpServletRequest request, HttpServletResponse response) {
  String token = TokenExtractor.extractToken(request);

  if (token != null && tokenProvider.validateToken(token)) {
    CookieUtil.deleteCookie(response, name: "jwtToken");
  } else {
    throw new UnauthorizedException("Invalid or expired token.");
  }
}
```

Рисунок 5.2.4 – Реалізація методу logoutUser.

5.2.3 Клас JwtTokenProvider для генерації та перевірки токенів

Клас JwtTokenProvider містить методи, які відповідають за генерацію та перевірку JWT-токенів.

Метод generateToken. Цей метод приймає об'єкт UserDetails користувача і створює JWT-токен, використовуючи його інформацію. Токен містить дані про користувача та його права доступу

(рисунок 5.2.5).

```
public String generateToken(Authentication authentication) {
  UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();
  Date now = new Date();
  Date expiryDate = new Date(now.getTime() + jwtExpirationInMs);

  return Jwts.builder()
      .setSubject(Long.toString(userPrincipal.getId()))
      .setIssuedAt(new Date())
      .setExpiration(expiryDate)
      .signWith(SignatureAlgorithm.HS512, jwtSecret)
      .compact();
}
```

Рисунок 5.2.5 – Реалізація методу generateToken.

Метод validateToke. Цей метод приймає JWT-токен і перевіряє його автентичність та цілісність. Він перевіряє підпис токена, рядковий формат та часові обмеження (рисунок 5.2.6).

```
public boolean validateToken(String token) {
  try {
    Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);
    return true;
  } catch (MalformedJwtException ex) {
```

Рисунок 5.2.6 – Реалізація методу validateToken.

Ці класи та методи взаємодіють між собою для забезпечення безпеки та аутентифікації користувачів у системі.

5.3 Розробка контролерів та сервісів для управління тестами

У цьому розділі будуть розглянуті класи TestController та TestService, які відповідають за управління тестами в системі.

5.3.1 Клас TestController та його методи

Клас TestController містить методи, які обробляють HTTP-запити, пов'язані з управлінням тестами.

Метод createTest. Цей метод обробляє POST-запит для створення нового тесту. На вхід отримаємо дані про тест з HTTP-запиту, передаємо їх до TestService для створення тесту та повертаємо статус відповіді (рисунок 5.3.1).

```
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public TestDto createTest(@RequestBody TestDto testDto) {
    return testService.createTest(testDto);
}
```

Рисунок 5.3.1 – Реалізація методу createTest.

Метод getTestById. Цей метод обробляє GET-запит для отримання інформації про конкретний тест за його ідентифікатором. На вхід отримаємо ідентифікатор тесту з HTTP-запиту, передаємо його до TestService для отримання інформації про тест та повертаємо його відповідь (рисунок 5.3.2).

```
@GetMapping("/{id}")
public TestDto getTestById(@PathVariable Long id) {
    return testService.getTestById(id);
}
```

Рисунок 5.3.2 – Реалізація методу getTestById.

Метод updateTest. Цей метод обробляє PUT-запит для оновлення інформації про тест. На вхід отримаємо ідентифікатор тесту та нові дані з HTTP-запиту, передаємо їх до TestService для оновлення тесту та повертаємо статус відповіді (рисунок 5.3.3).

```
@PutMapping("/{id}")
public TestDto updateTest(@PathVariable Long id, @RequestBody TestDto testDto) {
    return testService.updateTest(id, testDto);
}
```

Рисунок 5.3.3 – Реалізація методу updateTest.

Метод deleteTest. Цей метод обробляє DELETE-запит для видалення тесту за його ідентифікатором. На вхід отримаємо ідентифікатор тесту з HTTP-запиту, передаємо його до TestService для видалення тесту та повертаємо статус відповіді(рисунок 5.3.4).

```
@DeleteMapping("/{id}")
public void deleteTest(@PathVariable Long id) {
    testService.deleteTest(id);
}
```

Рисунок 5.3.4 – Реалізація методу deleteTest.

5.3.2 Клас TestService та його методи

Клас TestService містить логіку, пов'язану з управлінням тестами.

Метод createTest. Цей метод приймає дані про тест та реалізовує логіку для створення нового тесту. Він перевіряє правильність введених даних, створює об'єкт Test і зберігає його в базі даних (рисунок 5.3.5).

```
public TestDto createTest(TestDto testDto) {
    Test test = new Test();
    test.setName(testDto.getName());
    test.setDuration(testDto.getDuration());
    test.setInstructions(testDto.getInstructions());
    test.setTotalScore(testDto.getTotalScore());

    Test savedTest = testRepository.save(test);

    return mapTestToDto(savedTest);
}
```

Рисунок 5.3.5 – Реалізація методу createTest.

Метод getTestById. Цей метод приймає ідентифікатор тесту та реалізовує логіку для отримання інформації про тест за його ідентифікатором. Він виконує запит до бази даних для отримання відповідного запису тесту та повертає його (рисунок 5.3.6).

```
public TestDto getTestById(Long id) {
    Test test = testRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Test", "id", id));

    return mapTestToDto(test);
}
```

Рисунок 5.3.6 – Реалізація методу getTestById

Метод updateTest. Цей метод приймає ідентифікатор тесту та нові дані тесту та реалізовує логіку для оновлення інформації про тест. Він виконує запит до бази даних для оновлення відповідного запису тесту (рисунок 5.3.7).

```
public TestDto updateTest(Long id, TestDto testDto) {
    Test test = testRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Test", "id", id));

    test.setName(testDto.getName());
    test.setDuration(testDto.getDuration());
    test.setInstructions(testDto.getInstructions());
    test.setTotalScore(testDto.getTotalScore());

    Test updatedTest = testRepository.save(test);

    return mapTestToDto(updatedTest);
}
```

Рисунок 5.3.7 – Реалізація методу updateTest.

Метод deleteTest. Цей метод приймає ідентифікатор тесту та реалізовує логіку для видалення тесту за його ідентифікатором. Він виконує запит до бази даних для видалення відповідного запису тесту (рисунок 5.3.8).

```
public void deleteTest(Long id) {
    Test test = testRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Test", "id", id));

    testRepository.delete(test);
}
```

Рисунок 5.3.8 – Реалізація методу deleteTest.

Ці класи та методи дозволяють ефективно управляти тестами в системі, забезпечуючи їх створення, отримання, оновлення та видалення.

5.4 Розробка сервісів для управління питаннями

У цьому розділі буде розглянуто клас QuestionService, який відповідає за управління питаннями в системі.

5.4.1 Клас QuestionService та його методи

Клас QuestionService містить логіку, пов'язану з управлінням питаннями.

Метод createQuestion. Цей метод приймає дані про питання та реалізовує логіку для створення нового питання. Він перевіряє правильність введених даних, створює об'єкт Question і зберігає його в базі даних (рисунок 5.4.1).

```
public Question createQuestion(QuestionDto questionDto, Test test) {
    Question question = new Question();
    question.setQuestion(questionDto.getQuestion());
    question.setScore(questionDto.getScore());
    question.setTest(test);
    return questionRepository.save(question);
}
```

Рисунок 5.4.1 – Реалізація методу createQuestion.

```
public QuestionDto updateQuestion(TestDto testDto, Long questionId, QuestionDto questionDto) {
    Question existingQuestion = questionRepository.findById(questionId)
        .orElseThrow(() -> new ResourceNotFoundException("Question", "id", questionId));

    if (!testDto.getId().equals(existingQuestion.getTest().getId())) {
        throw new BadRequestException("Question with id: " + questionId + " does not belong to the test with id: " + testDto.getId());
    }

    existingQuestion.setQuestion(questionDto.getQuestion());
    existingQuestion.setScore(questionDto.getScore());

    List<Answer> existingAnswers = existingQuestion.getAnswers();
    List<AnswerDto> answerDtos = questionDto.getAnswers();

    existingAnswers.clear();

    for (AnswerDto answerDto : answerDtos) {
        Answer answer = new Answer();
        answer.setAnswer(answerDto.getAnswer());
        answer.setCorrect(answerDto.isCorrect());
        answer.setQuestion(existingQuestion);
        existingAnswers.add(answer);
    }

    Question updatedQuestion = questionRepository.save(existingQuestion);
    return toQuestionDto(updatedQuestion);
}
```

Рисунок 5.4.2 – Реалізація методу updateQuestion.

Метод updateQuestion. Цей метод приймає ідентифікатор питання та нові дані питання та реалізовує логіку для оновлення інформації про

питання. Він виконує запит до бази даних для оновлення відповідного запису питання та відповіді (рисунок 5.4.2).

5.5 Розробка методу для перевірки тесту

Найважливішим питанням полягало в тому, як обробляти результати. Цей метод має більше логіки, ніж усі перед цим розгляднуті.

Для початку перевіряємо наявність тесту та отримаємо дані по користувачу. Після чого створюємо результат для початку нульовий, щоб у випадку якоїсь несправності користувач міг бачити, що він пройшов тест. Далі для кожного питання ми беремо його відповіді з таблиці та перевіряємо на коректність з відповідями, які до нас прийшли, та оновлюємо загальний бал. Обов’язково не забуваємо зберегти ці дані у таблиці (рисунок 5.5.1).

```
public ResultSummaryDto submitTestResults(Long testId, TestResultsDto testResultsDto) {
  Test test = testRepository.findById(testId)
      .orElseThrow(() -> new ResourceNotFoundException("Test", "id", testId));

  User user = authService.getCurrentUser();

  Result result = new Result();
  result.setUser(user);
  result.setTest(test);
  result.setTotalScore(0);
  result.setSubmittedAnswers(new ArrayList<>());
  for (int i = 0; i < test.getQuestions().size(); i++) {
    Question question = test.getQuestions().get(i);
    AnswersDto answersDto = testResultsDto.getSubmittedAnswers().get(i);

    if (!questionService.isAnswerCorrect(question, answersDto)) {
      throw new BadRequestException("Answer is not correct for question with id: " + question.getId());
    }

    result.setTotalScore(result.getTotalScore() + question.getScore());

    final var answers = new ArrayList<Answer>();

    answersDto.getAnswers().forEach(answerDto -> {
      Answer answer = new Answer();
      answer.setQuestion(question);
      answer.setAnswer(answerDto);
      answer.setCorrect(true);
      answers.add(answer);
    });
    result.getSubmittedAnswers().addAll(answers);
  }

  result = resultRepository.save(result);

  return new ResultSummaryDto(result.getId(), test.getName(), result.getTotalScore(), result.getSubmittedAnswers().size());
}
```

Рисунок 5.5.1 – Реалізація методу submitTestResults.

РОЗДІЛ 6 РОЗРОБКА ФРОНТЕНДУ

6.1 Вибір технологій фронтенду

При розробці фронтенду для вебзастосунку було зроблено вибір певних технологій, що відповідають вимогам проєкту і забезпечують зручну та ефективну розробку і взаємодію з користувачем.

Основні технології, що були використані для розробки фронтенду, включають:

а) HTML/CSS: HTML використовується для створення структури сторінок, а CSS - для їх стилізації та вигляду.

б) JavaScript: JavaScript є основною мовою програмування для динамічної взаємодії з користувачем. Він використовується для розробки функціональності, валідації даних та взаємодії з бекендом.

в) Bootstrap: Bootstrap - це популярний фреймворк для розробки адаптивного та естетичного дизайну. Він забезпечує широкий набір готових компонентів, стилів та інструментів, що спрощують процес розробки і забезпечують єдність дизайну.

6.2 Інтеграція фронтенду з бекендом

Для успішної роботи вебзастосунку необхідна інтеграція між фронтендом і бекендом. Ця інтеграція забезпечує обмін даними та взаємодію між клієнтською та серверною частинами додатку.

Основні принципи інтеграції фронтенду з бекендом включають наступне.

а) Використання HTTP протоколу. Клієнтська частина взаємодіє з серверною частиною за допомогою HTTP запитів і відповідей. Запити включають отримання даних, надсилання даних для збереження, оновлення або видалення, а також аутентифікацію та авторизацію.

б) Використання API. Бекенд повинен надавати API (інтерфейс програмування додатків), який описує доступні операції та формат даних для взаємодії з фронтендом. Це може бути RESTful API, GraphQL, або інший стандарт вебсервісів.

в) Використання бібліотек та фреймворків. У фронтенді можна використовувати спеціальні бібліотеки та фреймворки, які спрощують роботу з взаємодією з бекендом. Наприклад, для виконання HTTP запитів можна використовувати Axios або Fetch API.

г) Обробка даних. Фронтенд повинен вміти коректно обробляти та відображати дані, які отримує від бекенду. Це може включати парсинг JSON-даних, валідацію та форматування даних перед відображенням на сторінці.

Завдяки інтеграції фронтенду з бекендом користувачі отримують доступ до потужної та функціональної системи, яка забезпечує зручну роботу з тестами, аутентифікацію та авторизацію, а також обмін даними між користувачем та сервером.

6.3 Розробка форми для створення тестів

Для ефективного використання функціоналу нашого бекенд сервісу, потрібно було пропрацювати форму створення тестів (рисунок 6.3.1).

```
<form id="create-test-form">
    <label for="test-name">Назва тесту:</label>
    <input type="text" id="test-name" name="test-name" required>

    <label for="test-description">Опис тесту:</label>
    <textarea id="test-description" name="test-description"></textarea>

    <div id="question-container">
    </div>

    <button type="button" onclick="addQuestion()">Додати питання</button>
    <button type="submit">Створити тест</button>
</form>
```

Рисунок 6.3.1 – Реалізація форми для тестів

Було написана JavaScript функція, яка додавала динамічно поля до форми тестів, щоб користувачу було зручно задавати необхідну кількість питань та відповідей (рисунок 6.3.2).

```
function addQuestion() {
    const container = document.getElementById('question-container');
    const questionDiv = document.createElement('div');
    questionDiv.classList.add('question');

    const questionLabel = document.createElement('label');
    questionLabel.textContent = `Питання ${questionCount}:`;
    const questionInput = document.createElement('input');
    questionInput.type = 'text';
    questionInput.name = `question${questionCount}`;
    questionInput.required = true;

    const answerContainer = document.createElement('div');
    answerContainer.classList.add('answer-container');

    const answerLabel = document.createElement('label');
    answerLabel.textContent = 'Відповіді:';

    const addAnswerButton = document.createElement('button');
    addAnswerButton.textContent = 'Додати відповідь';
    addAnswerButton.addEventListener('click', () => {
        const answerInput = document.createElement('input');
        answerInput.type = 'text';
        answerInput.name = `question${questionCount}-answer`;
        answerInput.required = true;
        answerContainer.appendChild(answerInput);
    });

    questionDiv.appendChild(questionLabel);
    questionDiv.appendChild(questionInput);
    questionDiv.appendChild(answerLabel);
    questionDiv.appendChild(answerContainer);
    questionDiv.appendChild(addAnswerButton);

    container.appendChild(questionDiv);

    questionCount++;
}
```

Рисунок 6.3.2 – Реалізація динамічно поля до форми тестів.

Після заповнення всієї форми треба зібрати усі дані та відправити на бекенд сервіс, який їх обробить та надішле помилку, у разі якої користувач повинен щось змінити та спробувати ще раз (рисунки 6.3.3 та 6.3.4).

```
};
fetch('http://localhost:8080/tests', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(test)
})
    .then(response => response.json())
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error('Error:', error);
    });
```

Рисунок 6.3.3 – Реалізація інтеграції з бекенд сервісом.

```
document.getElementById('create-test-form').addEventListener('submit', (event) => {
    event.preventDefault();
    const formData = new FormData(event.target);
    const testInfo = {
        name: formData.get('test-name'),
        description: formData.get('test-description')
    };
    const questions = [];
    for (let i = 1; i <= questionCount; i++) {
        const questionText = formData.get(`question${i}`);
        const answers = [];

        let answerIndex = 1;
        while (formData.get(`question${i}-answer-${answerIndex}`)) {
            const answerText = formData.get(`question${i}-answer-${answerIndex}`);
            answers.push(answerText);
            answerIndex++;
        }

        const question = {
            text: questionText,
            answers: answers
        };

        questions.push(question);
    }
```

Рисунок 6.3.4 – Реалізація збору даних перед відправкою.

РОЗДІЛ 7 ВДОСКОНАЛЕННЯ СИСТЕМИ

Наведемо подальші можливі напрямки розвитку розробленого вебдодатку.

7.1 Використання Kubernetes

Зараз для покращення сервісу, легкого та незалежного масштабування, є безліч можливостей, зокрема Kubernetes. «Kubernetes» (K8s)— відкрита система автоматичного розгортання, масштабування та управління застосунками у контейнерах. Система підтримує ряд інструментарію з управління контейнерами, у тому числі Docker [8] (рисунок 7.1).

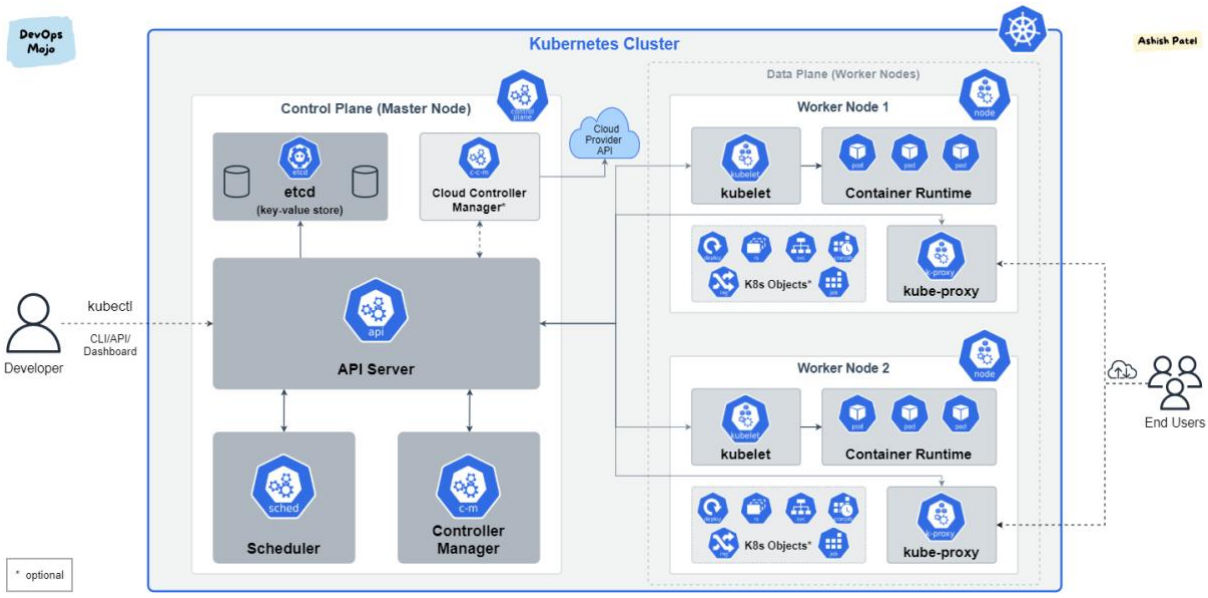

Рисунок 7.1 – Архітектура Kubernetes та його роботи[18].

Для розгортання сервісу у Kubernetes треба створити Dockerfile, який відповідає за згортання всього сервісу в один image, який потрібен Kubernetes для розгортання сервісу. Dockerfile заватнажує усі допоміжні та основні бібліотеки для сервісу, і вказує, як правильно його підняти однією командою. Dockerfile – це інструкція або алгоритм дій.

7.2 Представлення і обробка результатів тестів

Також можна більш оптимізувати чи зробити більш автономним підрахунок балів та результатів. PostgreSQL надає цю можливість. MATERIALIZED VIEW – функціонал, який надає PostgreSQL. Це ніби таблиця, яка створюється з інших таблиць, та при цьому на неї можна накласти тригери, які будуть її оновлювати (рисунок 7.2).

```
CREATE MATERIALIZED VIEW UserTotalScores AS
SELECT user_id, test_id, SUM(score) AS total_score
FROM ResultDetails
GROUP BY user_id, test_id;
```

Рисунок 7.2 – Приклад реалізації MATERIALIZED VIEW.

В перспективі є ідея додати маркери для тестів, щоб їх можна було розділяти на певні категорії. Це може стати підґрунтям для багатьох можливостей: персоналізації, оформлення на сайті, пошуку.

Можна також додати вибірку результатів по певним тестам, за вказаний період часу, або ще по деяким параметрам з виведенням отриманих результатів у csv файл тощо.

7.3 Використання технологій штучного інтелекту

Застосування технологій штучного інтелекту в сучасному світі знаходить все більше прикладів і має великий потенціал у різних галузях. З розвитком цих технологій відкриваються нові можливості для вдосконалення процесів у різних сферах життя.

Є багато галузей, де штучний інтелект вже почали використовувати для надання певних послуг. Наведемо деякі приклади [9].

а) Маркетинг: може допомогти з персоналізацією в режимі реального часу, а також з організацією кампанії, щоб розширити, впорядкувати й автоматизувати маркетингові процеси та завдання.

б) Фінанси: найкращими кандидатами для швидкого використання штучного інтелекту є динамічні процеси, які вимагають оцінки та включають неструктуровані, нестабільні та високошвидкісні дані.

в) Продаж: визначення нових потенційних клієнтів і можливостей на основі схожих існуючих клієнтів, налагодження відносин за допомогою інтелектуального відстеження активності та обміну повідомленнями.

Інтегрування штучного інтелекту в розроблену систему може надати багато переваг та покращити функціональність. За допомогою штучного інтелекту можливо підбирати персоналізований контент для користувача. Саме для цього і можна використати маркери для тестів. Штучний інтелект може дивитись, які тести проходить користувач, наскільки успішно, з якої теми, та підбирати для нього тести схожі або складніші, чи з метою покращення результатів з певної теми, в якій не дуже успішний

Одним з можливих напрямків розвитку системи є інтегрування алгоритмів машинного навчання для покращення аналізу результатів тестування та продуктивності системи. Застосування методів класифікації може допомогти виявити неточності та брак знань у студента. Це дасть розширення можливості даної системи та забезпечить більш точне відслідковування прогресу в навчальному процесі.

Застосування технологій штучного інтелекту у вебсистемі для тестування має великий потенціал у поліпшенні навчального процесу, підвищенні ефективності та точності оцінювання знань студентів. Дослідження даної можливості свідчать про переваги та значимість використання штучного інтелекту у сфері освіти. Дальші дослідження та

роботи в цьому напрямку можуть сприяти подальшій оптимізації та розвитку системи тестування з використанням штучного інтелекту.

ВИСНОВКИ

В ході проведення роботи було розроблено вебзастосунок для тестувань. Результати досліджень та практичної частини свідчать про потенціал для розвитку цієї системи для поліпшення користування, оцінки знань та навчального процесу.

Перед виконанням роботи були поставлені задачі, які були успішно виконані. А саме:

а) було проаналізовано вимоги до системи,

б) було розроблено архітектуру застосунку та обрано відповідні сучасні технології для розробки,

в) було досліджено існуючі функціонали авторизації та аутентифікації користувачів та відповідно інтегровано у проєкт,

г) було розроблено функціонал для створення, редагування та видалення тестів і питань та алгоритм оцінювання результатів тестування,

д) було проведено порівняльний аналіз з існуючими рішеннями в галузі тестування,

е) було досліджено можливості і перспективи покращення створеної вебсистеми.

Розроблений вебзастосунок може використовуватися не лише в учбових закладах, але і в різних галузях, де потрібно тестування знань або навичок. Гнучкість та можливості для розширення даної системи надають їй перспективи для подальшого розвитку та вдосконалення.

Створена система для тестування може стати ефективним інструментом, який покращує навчальний процес та забезпечує більш об’єктивне та незалежне оцінювання знань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pressman R. S. Software Engineering: A Practitioner's Approach / R. S. Pressman, B. Maxim., 2014. – 976 с. – (8th Edition).
2. Bendoraitis A. Web Development with Django Cookbook / Aidas Bendoraitis., 2014. – 294 с.
3. Richardson L. RESTful Web Services / L. Richardson, S. Ruby., 2007. – 454 с. – (First Edition).
4. Bhargav A. Secure Java: For Web Application Development / A. Bhargav, B. V. Kumar., 2010. – 308 с. – (First Edition).
5. The Java Language Specification / J.Gosling, B. Joy, G. Steele, G. Bracha., 2005. – 650 с. – (Third Edition).
6. Oracle. Java Client Roadmap Update / Oracle., 2018. – 8 с. – (1.01).
7. Вострецов О. Коли та чому Java використовується для розробки? [Електронний ресурс] / Олександр Вострецов. – 2021. – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/kogda-i-pochemu-java-ispolzuetsja-dlja-razrabotki-prilozhenij.
8. Офіційна документація Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io.
9. Семенюк В. Штучний інтелект: де використовують сьогодні та коли він замінить людину [Електронний ресурс] / Валентин Семенюк – Режим доступу до ресурсу: https://cybercalm.org/analytics/shtuchnij-intelekt-de-vikoristovuyut-sogodni-ta-koli-vin-zaminit-lyudinu/.
10. Mishra D. Generate JWT Token and Verify in Plain Java [Електронний ресурс] / Deepak Mishra. – 2021. – Режим доступу до ресурсу: https://metamug.com/article/security/jwt-java-tutorial-create-verify.html.
11. Застосування архітектурного шаблону MVC у розробці ВЕБ ужитків / В. Гнот, В. Якимович // Збірник наукових праць Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України. — К.: ІПМЕ ім. Г.Є. Пухова НАН України, 2011. — Вип. 58. — С. 185-187.

12. Офіційна документація Oauth [Електронний ресурс] – Режим доступу до ресурсу: https://www.oauth.com.
13. Using HTTP cookies [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.
14. Hardt D. The OAuth 2.0 Authorization Framework [Електронний ресурс] / Dick Hardt. – 2012. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6749.
15. Офіційна документація PostreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
16. Офіційна документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/.
17. Офіційна документація MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/.
18. Patel A. Kubernetes — Architecture Overview [Електронний ресурс] / Ashish Patel. – 2021. – Режим доступу до ресурсу: https://medium.com/devops-mojo/kubernetes-architecture-overview-introduction-to-k8s-architecture-and-understanding-k8s-cluster-components-90e11eb34ccd.