Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера»

Виконав: студент II курсу
групи ___ЗПІ-22м____ спеціальності
121 – Інженерія програмного забезпечення
Токарчук Д. О.

Керівник: к.т.н., доц. каф. ПЗ Майданюк В. П.
«11» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Черняк О. І.
«11» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«11» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 19 » вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Токарчуку Дмитру Олександровичу

1. Тема роботи – моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера.

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 року № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: серверний модуль дистанційного тестування з використання технології Google Cloud Vision; Вхідні дані: мова програмування Java, водоспадна модель розробки, середовище розробки IntelliJ IDEA, прикладний програмний інтерфейс, REST API, протокол передачі даних HTTP, база даних MySQL..

4. Зміст текстової частини: вступ, аналіз та обґрунтування вибору методу розробки та постановка задачі дослідження, розробка архітектури та алгоритмів серверного модуля, розробка коду серверного модуля, тестування серверного модуля, економічна частина, висновки, список використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: порівняльний аналіз аналогів, блок-схема алгоритму роботи компонентів програми, архітектура серверного модуля, діаграма послідовності роботи додатку, структура прикладного програмного інтерфейсу, структура сховища фотоданих, тестування серверного модуля, апробація та публікації результатів роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Майданюк В. П. к. т. н., доцент кафедри ПЗ	20.09.2023	05.12.2023
5	Причепа І. В. к. е. н. доцент кафедри ЕПВМ	21.11.2023	25.11.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	20.09.2023 – 05.10.2023	вик
2	Розробка архітектури та алгоритмів роботи серверного модуля	06.10.2023 – 19.10.2023	вик
3	Аналіз і вибір мови програмування та середовища програмування	20.10.2023 – 23.10.2023	вик
4	Розробка серверного модуля	24.10.2023 – 16.11.2023	вик
5	Тестування роботи серверного модуля	17.11.2023 – 21.11.2023	вик
6	Економічна частина	22.11.2023 – 25.11.2023	вик
7	Оформлення матеріалів до захисту МКР	26.11.2023 – 01.12.2023	вик

Студент _____ (підпис) **Токарчук Д. О.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ (підпис) **Майданюк В. П.** (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.031.42

Токарчук Д. О. Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. – 104.

На укр. мові. Бібліогр.: 30 назв; рис.: 41; табл. 12.

У магістерській кваліфікаційній роботі було розроблено модуль сервера дистанційного тестування з використанням технології Google Cloud Vision. Цей програмний продукт створений для вдосконалення процесу створення та проходження онлайн-тестів з допомогою штучного інтелекту.

Програмний додаток створений на мові Java, з використанням реляційної бази даних і відзначається універсальністю застосування та захистом даних користувачів.

У роботі проведено глибокий аналіз стану програмного забезпечення для проведення тестування. Метою досліджень є підвищення ефективності та продуктивності тестування людей, забезпечення достовірності результатів тестів завдяки процесу тестування та використанню штучного інтелекту для розпізнавання зображень фотоконтролю.

Пропонується процес проведення онлайн-тестування користувача на основі різних типів текстово-візуальних тестових запитань та використання технології Google Cloud Vision у вигляді серверного модуля з прикладним програмним інтерфейсом.

Ключові слова: Дистанційне тестування, серверний модуль, прикладний програмний інтерфейс, Cloud Vision, REST.

ANNOTATION

Tokarchuk D. O. Models and software tools for remote testing using Google Cloud Vision technology. Master's thesis in the field of 121 - Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2023. – 104 p.

In Ukrainian language. Bibliographer: 30 titles; fig.: 41; tabl. 12.

In the master's research, a server module for a remote testing system using Google Cloud Vision technology was developed. This software product is designed to improve the process of creating and taking online tests with the help of artificial intelligence.

The software application is created in Java, using the Spring framework, and is characterized by its versatility and user data protection.

The work carries out an in-depth analysis of the state of software for testing purposes. The research objective is to increase the efficiency and productivity of people's testing, ensuring the reliability of test results through photo control of the testing process and the use of artificial intelligence for image recognition.

The proposed online testing process is based on various types of text-visual test questions, photo control, and the use of Google Cloud Vision technology in the form of a server module with a software interface.

Keywords: Remote testing, server module, application programming interface, Cloud Vision, REST.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження: В сучасному світі, вебтехнології відіграють важливу роль в житті більшості людей в усьому світі. Інформаційні та вебтехнології дозволяють автоматизувати й поліпшити різні процеси та задачі, такі як фінансові операції, обмін інформацією, пошук нової інформації, тощо.

Сфера освіти також активно інтегрує інформаційні технології [1]. Вебсайти навчальних закладів давно стали звичними й використовуються як потужний інструмент у процесі навчання та його організації. Вони публікують новини, оголошення, навчальні матеріали та інформацію для студентів та вступників. Додатково, вебінструменти дають змогу створювати онлайн журнали й системи контролю знань.

Одним зі способів онлайн контролю знань є тестування. Впровадження онлайн тестів дозволяє автоматизувати процес їх проходження та оцінювання, а також зняти більшість проблем, пов'язаних з людським фактором, які можуть виникнути під час проведення традиційних тестів. Однак, варто враховувати, що при проходженні онлайн тестів значно зростає ймовірність шахрайства і списування.

Академічне тестування є важливою складовою освітнього процесу, допомагаючи вимірювати рівень знань і розвивати критичне мислення учнів [2]. Це ефективний інструмент для оцінки навчальних досягнень, дозволяючи перевірити засвоєні матеріали і вміння. Тестування створює можливість порівняти результати та визначити обсяг знань, а також сприяє покращенню якості навчання через виявлення слабких місць у процесі вивчення матеріалу. Крім того, ця практика є основою для оцінки розвитку освітніх програм і вдосконалення методів викладання.

Відповідно до цього, вибір даної теми для дослідження є важливим та актуальним, оскільки він відображає потреби сучасного світу та сфери освіти в автоматизації процесів та підвищенні якості контролю знань. Саме тому темою

магістерської кваліфікаційної роботи було обрано «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера.».

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення .

Мета та завдання дослідження. Метою роботи є покращенні якості онлайн тестування та зниження кількості порушень академічної доброчесності за рахунок створення тестувальної онлайн-системи з використанням технологій штучного інтелекту Google Cloud Vision.

Основними задачами дослідження є:

– проектування багатошарової структури та архітектури серверного модуля;

– створення структури реляційної бази даних, яка зберігає дані серверного модуля;

– варіантний аналіз та обґрунтування вибору засобів для реалізації програмного за стосунку;

– інтеграція технології Google Cloud Vision у проект;

– створення сервісних класів та компонентів серверного модуля;

– розробка прикладного програмного інтерфейсу серверного модуля;

– розробка та виконання програмних модульних тестів;

– ручне тестування роботи розробленого програмного інтерфейсу;

– розробка документації до розробленого прикладного програмного інтерфейсу серверного модуля.

Об'єкт дослідження – процес здійснення онлайн тестування з використанням технології Google Cloud Vision.

Предмет дослідження – моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, методи побудови комп'ютерних мереж, методи планування та управління тестуванням, методи WEB-програмування для розробки моделей та

методів дистанційного тестування; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод дистанційного тестування, у якому на відміну від існуючих, використано технологію Google Cloud Vision для розпізнавання та аналізу зображень учасників тестування, що забезпечує підвищення рівня академічної доброчесності.

2. Подальшого розвитку отримав метод зберігання фотоданих у файловому сховищі, у якому на відміну від існуючих, використана ієрархічна структура, що підвищує швидкість пошуку фотоданих на 10 %.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано та розроблено серверний модуль тестувальної системи з використанням технології Google Cloud Vision

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: розробка серверної частини тестувальної системи з фотоконтролем [3]; аналіз засобів Google Cloud Vision [4]; використання Vision API [5]

Апробація матеріалів магістерської кваліфікаційної роботи Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: Науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.); Міжнародна науково-практична конференція «Інформаційні технології і автоматизація» (Одеса, 2023); Міжнародна науково-практична конференція «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях у матеріалах конференцій.

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

Тестування людей є ключовим елементом многих галузей та сфер діяльності людства, граючи важливу роль у сучасній ері інформації і знань. Вже тисячу років тому різні культури у всьому світі розпочали використовувати тести як форму перевірки знань та навичок людей. Такі ранні тестові методики були характерні за свою усну форму і відсутність стандартизації - не було чітких критеріїв оцінки або встановлених питань.

Очевидно, з часом методики та процедури тестування невпинно розвивалися. Лише за останнє століття, завдяки науково-технологічному прогресу та прояву педагогічних інновацій, у передових країнах світу були створені перші стандартизовані тести. Цей розвиток спричинив появу широкого спектра інструментів та методів для вимірювання та оцінки знань людини.

Сучасні тести можуть складатися з різноманітних видів тестових завдань, кожне з яких має свою власну складність та відповідає певним метрикам. Типово, тестові завдання можуть включати завдання з кількома варіантами відповіді, завдання-доповнення, завдання множинного вибору, завдання вільного викладу, завдання з альтернативними відповідями, завдання на встановлення відповідності або завдання на визначення правильної послідовності.

Всі ці типи завдань створюють шкалу оцінок, яка дає можливість міряти рівень знань та розуміння особою навчального матеріалу.

З появою інформаційних технологій тестування людей значно спрощується, а онлайн-тестування стає нормою [6]. Тепер можливо проводити не тільки педагогічне тестування, але й психологічне або соціологічне за допомогою цифрових платформ.

Безперечною перевагою цього є змога значно прискорити та спростити процес тестування. Використання онлайн-тестування стає особливо актуальним

у сучасних умовах всесвітньої пандемії і переходу до дистанційної освіти, що робить даний інструмент незамінним в руках викладачів та освітян.

1.2 Порівняльний аналіз аналогів

В останні роки, спостерігається збільшення кількості електронних інструментів та сервісів, які спрощують цей процес на проведення онлайн-тестування для викладачів та учнів. Тому стала актуальна задача дослідження ринку цих інструментів та проведення порівняльного аналізу деяких доступних сервісів, які є особливо популярними та корисними для широкого кола користувачів.

ClassMaker – онлайн-сервіс для створення тестів та опитувань, який можна використовувати як для бізнесу, так і для навчання (рис. 1.1) [7]. Сервіс дозволяє створювати тести та опитування з можливістю налаштування доступу до них, встановлення обмежень на кількість спроб, встановлення таймера, а також з можливістю автоматичної перевірки результатів. Сайд є спеціалізованим майданчиком для тестування, та надає цікаві можливості, наприклад обговорення наявних тестів.

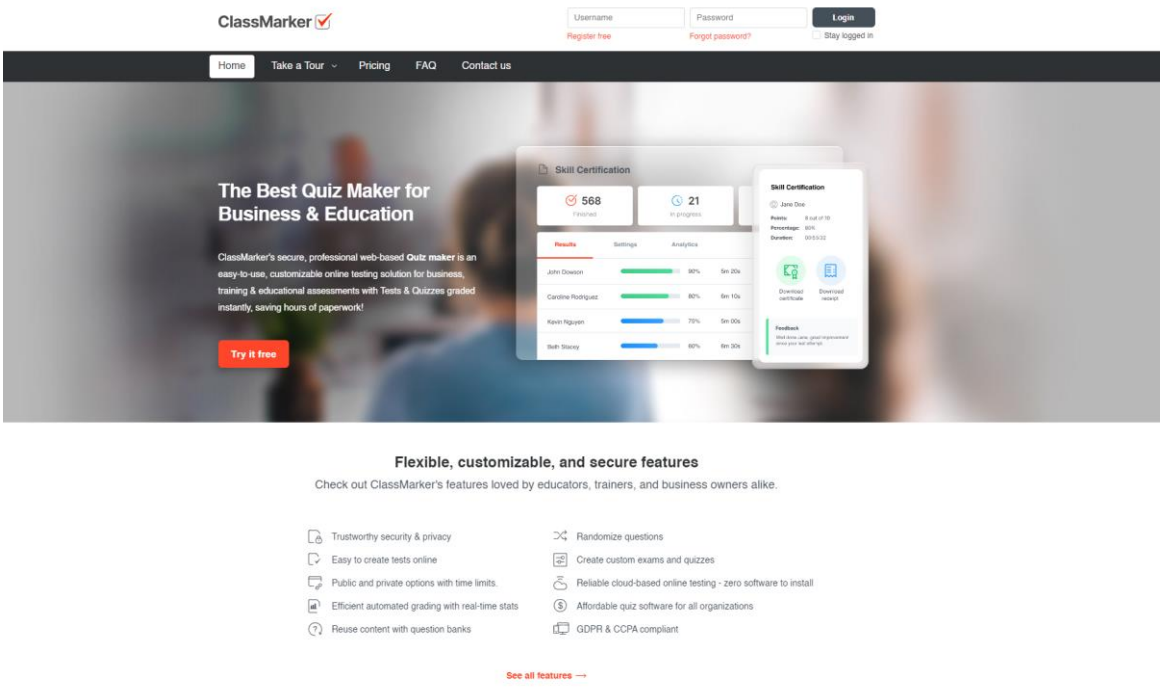

Рисунок 1.1 – Вебсайт ClassMaker

Проте цей сервіс має вагомий недолік – він не має функції, яка б дозволяла зробити фотографію користувача під час тестування. Така функція була б корисною для перевірки того, що користувач дійсно проходить тест самостійно, без допомоги або шахрайства. Функція фотофіксації також надасть можливість зберегтати історію тестування.

Розглянемо наступний аналог. Exam.com є сайтом, який допомагає вчителям та студентам проводити онлайн-іспити та оцінювання з високим рівнем безпеки [8]. Сайт надає безліч функцій, які дозволяють створювати складні іспити з різними типами питань, налаштовувати рівень безпеки, переглядати результати та отримувати зворотний зв'язок. Сайт також інтегрується з іншими системами навчання та дозволяє імпортувати та експортувати іспити у різних форматах (рис 1.2).

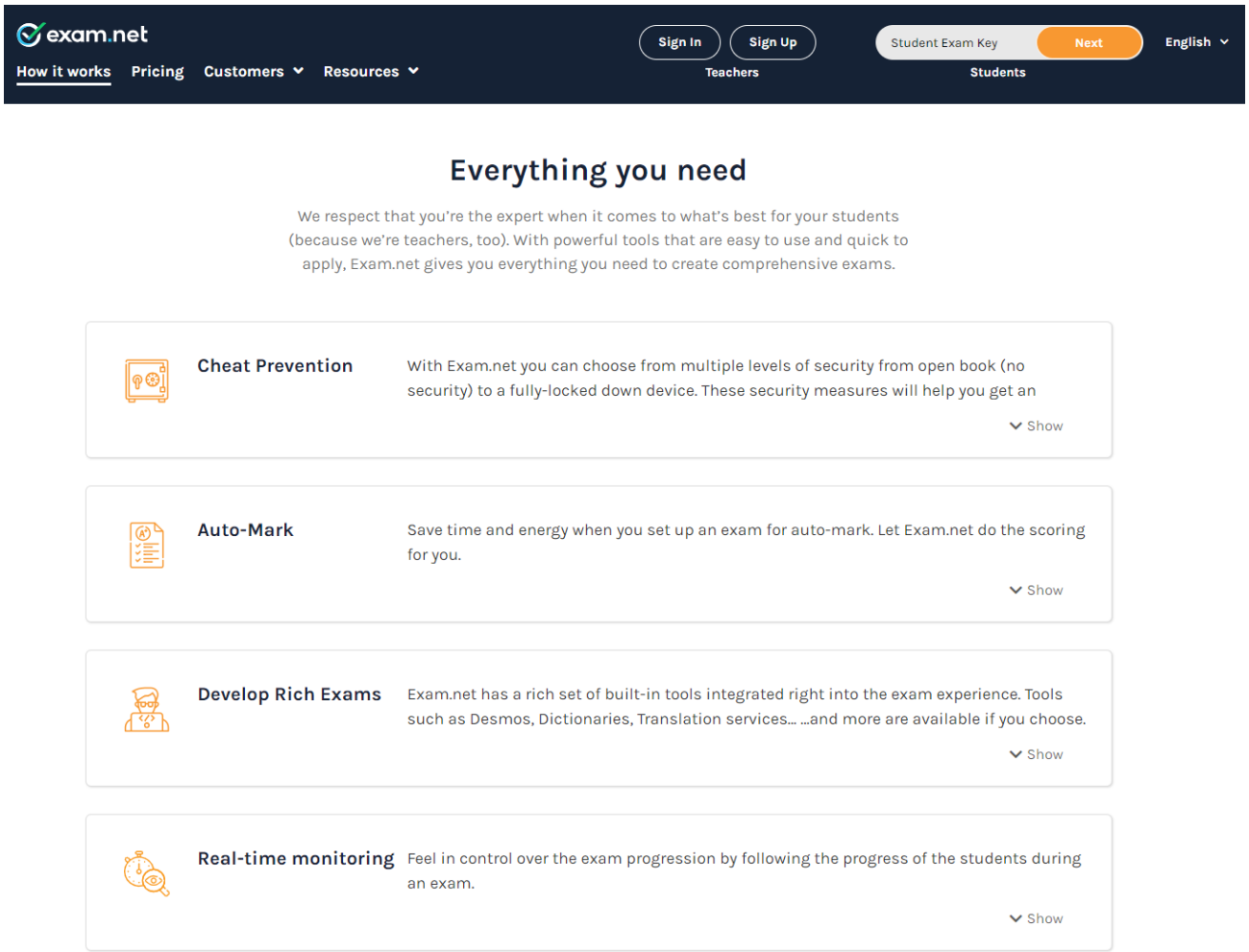

Рисунок 1.2 Вебсайт exam.net

Тим не менш, аналогісно попередньому аналогу, недоліком є відсутність функціональність фотофіксації процесу тестування та подальшо аналізу зображень.

Розглянемо ще один аналог. Testportal.net є онлайн-платформою для оцінки знань та навичок, яка дозволяє створювати власні тести, вікторини та іспити. За допомогою Testportal.net можна проводити дистанційну оцінку, отримувати автоматичну оцінку, зворотний зв’язок та аналітику результатів. Testportal.net також має функцію штучного інтелекту, яка генерує питання на основі наданих матеріалів або обраної теми (рис 1.3).

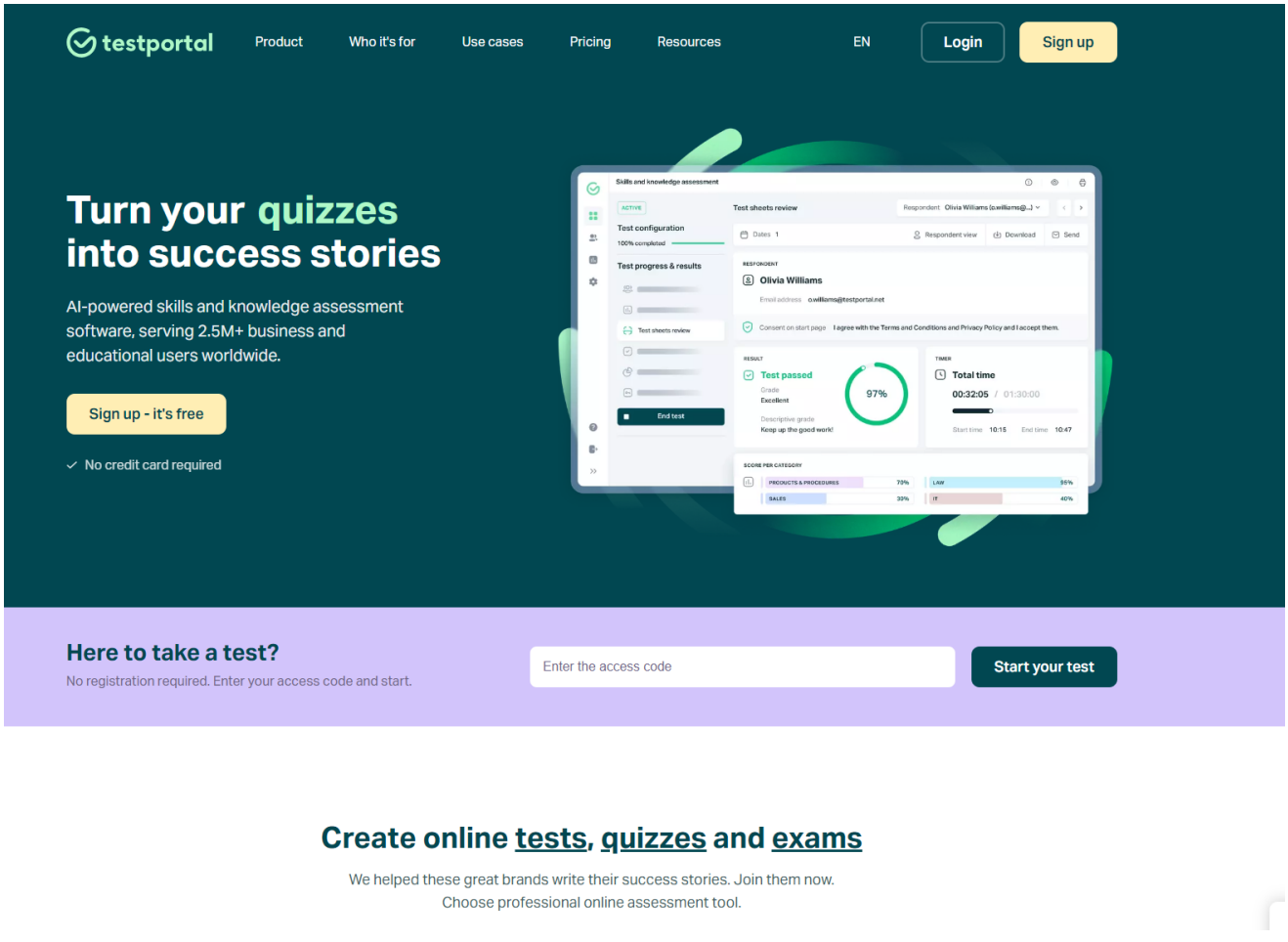

Рисунок 1.3 – Вебсайт Testportal

Однією з недоліків платформи є те, що вона не передбачає фотофіксації процесу тестування та не здійснює аналіз фотографічних матеріалів, що демонструють результати виконання завдань.

У результаті аналізу запропонованих аналогів, побудуємо порівняльну таблицю, в якій відобразимо переваги та недоліки кожного з них. Розглянемо таблицю 1.1.

Таблиця 1.1 – Порівняльна таблиця аналогів.

Критерій	Запропонований програмний засіб	ClassMaker	exam.net	testportal.net
Онлайн тестування	+	+	+	+
Конструктор тестів	+	+	+	+
Реєстрація корисутвачів	+	+	+	+
Фотофіксація процесу тестування	+	-	-	-
Аналіз фотоматеріалів результату тестування	+	-	-	-

За результатами аналізу даних, наведених у таблиці 1.1, можна зробити висновок, що додаток, розроблений в рамках магістерської кваліфікаційної роботи, має більшу універсальність для використання, оскільки дозволяє як проходити, так і створювати тести, а також надає додаткові можливості контролювати процес тестування та аналізувати його результати.

1.3 Вибір моделі життєвого циклу для розробки додатку

Вибір технології розробки програмного забезпечення є важливим фактором у процесі створення складних додатків. Для досягнення злагодженості, чіткості та послідовності у процесі розробці необхідно дотримуватися певної

технології [10]. В іншому випадку може знизитися продуктивність команди, що може призвести до серйозних проблем у якості та ефективності програмного продукту. З метою вирішення цієї проблеми було запропоновано безліч різноманітних методологій розробки програмного забезпечення. Кожна з методологій має свою специфіку та область застосування, але не може бути абсолютно універсальною для всіх видів проєктів.

Дослідимо найбільш поширені моделі життєвого циклу програмного забезпечення [11].

Водоспадна, каскадна або послідовна модель. Ця модель поділяє весь життєвий цикл розробки програмного забезпечення на шість чітко визначених та послідовно виконуваних етапів розробки:

– Формування вимог до програмного продукту;
– Проєктування архітектури та інтерфейсу програмного продукту;
– Реалізація програмного коду та інтеграція компонентів програмного продукту;
– Тестування функціональності та якості програмного продукту;
– Запровадження програмного продукту в експлуатацію;
– Супровід та підтримка програмного продукту.

Перехід до наступного етапу здійснюється тільки після повного завершення попереднього, тим самим забезпечується чітка послідовність та логічність процесів розробки. (рис. 1.4).

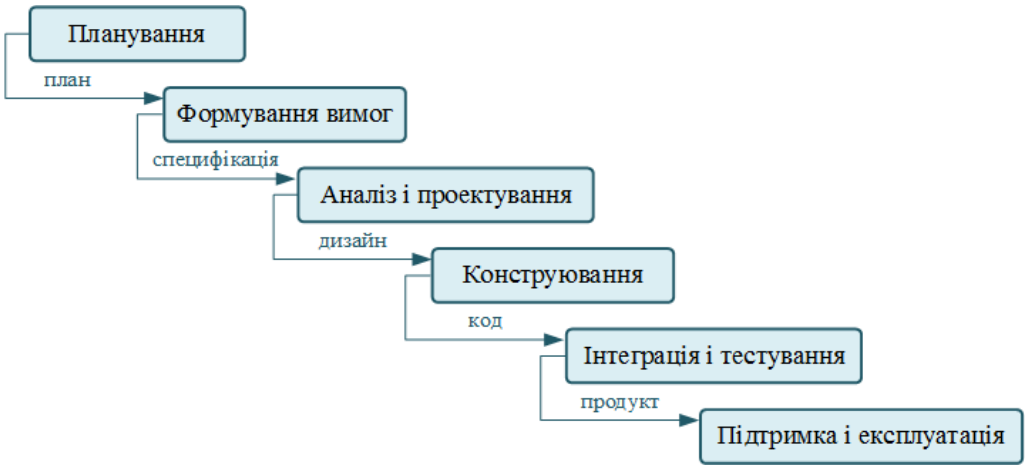

Рисунок 1.4 – Водоспадна модель

У процесі розробки програмного продукту за цією моделлю кожен етап супроводжується ретельною підготовкою технічної документації, яка відображає всі аспекти його виконання. До переваг цієї моделі належать наявність чітких критеріїв для оцінки стану проекту на будь-якій стадії розробки, а також докладна технічна документація, яка містить всю необхідну інформацію про продукт. Команда розробників має зрозуміле уявлення про поставлені перед ними завдання, а також можливість точно визначити терміни та вартість розробки.

Спіральна модель є однією з моделей процесу розробки програмного забезпечення, яка базується на принципі ітеративного підходу до створення продукту. Відмінністю цієї моделі від інших є те, що вона передбачає поступове удосконалення програмного забезпечення за допомогою методу прототипування, який дозволяє виявляти та виправляти помилки та недоліки на ранніх стадіях розробки. Кожна ітерація, або виток спіралі, є окремим циклом розробки, який складається з чотирьох фаз: визначення цілей та характеристик проекту, аналізу ризиків, розробки та тестування прототипу або версії програмного забезпечення, та оцінки якості отриманих результатів та планування робіт для наступної ітерації (рис. 1.5).

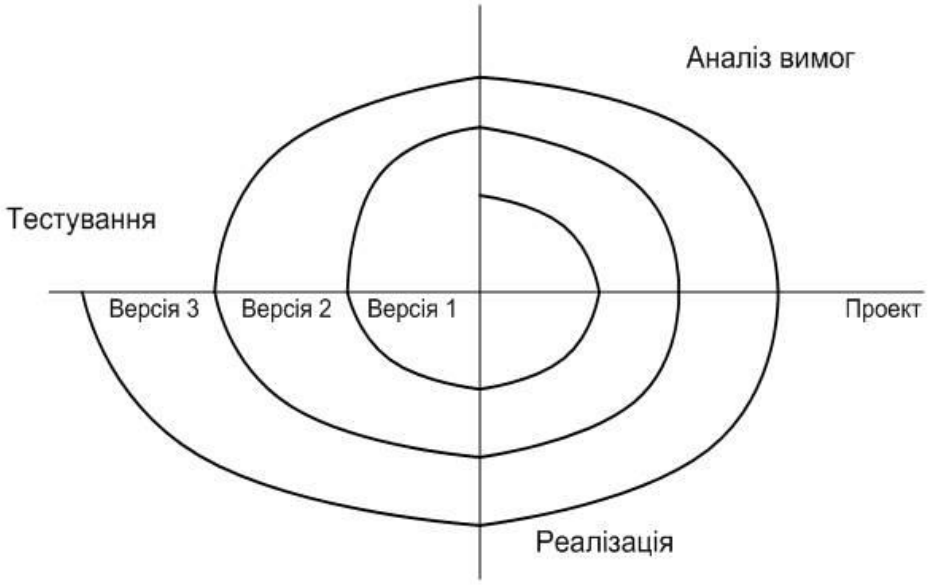

Рисунок 1.5 – Спіральна модель

В рамках кожного циклу спіральної моделі, проводиться детальна оцінка окремих ключових потенційних ризиків та питань. Наприклад, проаналізовуються наступні аспекти:

– Встановлюються можливі ризики, пов'язані з перевищенням виділених на проект термінів та ресурсів.

– Визначається чіткість поставленого технічного завдання для його ефективного виконання.

– Визначається потреба в подальшій ітерації розробки.

– Оцінюється цілісність проекту і виправданість його подальшого продовження.

Однією з важливих переваг спіральної моделі є її висока гнучкість та декомпозиція процесу розробки на менші, керовані ітерації. Кожна ітерація відображає створення окремого готового програмного продукту, що дозволяє легко вносити всілякі зміни до технічного завдання без серйозних наслідків. Отже, дана модель гарантує можливість адаптивної роботи з проектом.

Втім, водночас, концепція відсутності чітких меж виконання робіт може бути і недоліком, оскільки це ускладнює процес початку та закінчення окремих циклів спіралі. Таким чином вплив досвіду розробників стає більш критичним для ефективного виконання проєкту.

Незважаючи на переваги спіральної моделі, у контексті даної магістерської кваліфікаційної роботи було прийнято рішення скористатися водоспадною моделлю. Це обумовлено тим, що дана модель оптимально відповідає характеру виконуваних робіт, особливо у зв'язку з чіткою постановкою задач. Водоспадна модель впорядковує процес розробки, забезпечуючи послідовне виконання етапів проекту, та зменшує ймовірність відхилень від поточних завдань.

1.4 Постановка задачі

Цільовим завданням розробки модуля сервера дистанційного тестування із застосуванням технології Google Cloud Vision є створення сервісу, що містить серверний модуль та прикладний програмний інтерфейс. Головними вимогами

до програмного інтерфейсу є підтримка протоколу передачі даних HTTP та відповідність стандартам розробки прикладних програмних інтерфейсів REST API.

Основними завданнями серверного модуля є взаємодія з користувачами та обробка їх запитів, що полягає в наступному:

– Наявність реєстраційного процесу користувачів та можливість створення оновлення інформаційних профілів.

– Наявність процедур авторизації та автентифікації користувачів, захист їх даних та забезпечення конфіденційності.

– Забезпечення роботи системи із сталим сховищем для зберігання та обробки робочих даних.

– Відповідність системи різним видів сутностей тестів, наприклад, створення, редагування, та видалення. Реалізація гнучких параметрів тестів, які включають кількість тестових питань, типи питань, варіанти відповідей, кількість правильних відповідей та підтримку питань із зображеннями.

– Наявність можливості серверного модуля отримувати та зберігати результати тестування від користувачів, включаючи відповіді на питання, оцінки та зображення з вебкамери (за необхідності відповідно до вимог тесту).

– Аналіз та надання результатів зібраних зображень до відповідних користувачів, відображення попереджень про можливі порушення під час проходження тестування.

– Підтримка стандартних операцій читання, редагування та видалення основних робочих сутностей, таких як тести, тестові питання, варіанти відповідей, користувачі, зображення та інші.

1.5 Висновки

У даному розділі було проведено аналіз сучасного стану проблематики дистанційного тестування та обґрунтування необхідності розробки нового модуля сервера з використанням технології Google Cloud Vision. В ході аналізу

було розглянуто три перспективні аналогічні системи, які наразі представлені на ринку.

Здійснено порівняльний аналіз функціональних можливостей цих аналогів та проекту, що розробляється в межах даної магістерської кваліфікаційної роботи. Відповідно до результатів аналізу, розроблюваний додаток, що включатиме в себе всі переваги та можливості розглянутих систем, має призначення стати універсальним багатофункціональним рішенням у сфері дистанційного тестування.

Було розглянуто поширені моделі життєвого циклу програмного забезпечення та у зв'язку з цим, для розробки серверного модуля було вибрано водоспадну модель. Вибір цієї моделі було обґрунтовано і вона відповідає основним потребам проєкту.

На завершення, було сформовано перелік завдань, які визначають ключові особливості додатка що розробляється, та очікується, що вони будуть успішно реалізовані при завершенні проекту. Ці завдання являють собою основу для майбутньої роботи над розробкою продукту та стануть першочерговими кроками до досягнення його обраної мети.

2 **РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ СЕРВЕРНОГО МОДУЛЯ**

2.1 Розробка загальної архітектури продукту

Наданий момент існує багато рекомендацій та архітектурних патернів щодо побудови серверних модулів у зв’язку з їх масовою поширеністю. Це дозволяє скористатися найкращими практиками з досвіду інших проектів для побудови власного надійного серверного модуля [12].

Розглянемо поширені рекомендації для побудови серверних модулів з прикладним програмним інтерфейсом [13].

1. RESTful архітектура: Representational State Transfer (REST) – це широко використовуваний архітектурний стиль для проектування мережевих програм. Він покладається на зв’язок без збереження стану, де ресурси ідентифікуються за URL-адресами, а методи HTTP (GET, POST, PUT, DELETE) використовуються для виконання операцій CRUD (Створення, читання, оновлення, видалення).

2. Архітектура GraphQL: GraphQL є альтернативою REST, яка дозволяє клієнтам запитувати лише ті дані, які їм потрібні. Це дає змогу клієнтам визначати структуру відповіді, зменшуючи надмірність чи недостачу даних.

3. Мікросервісна архітектура: цей підхід розбиває програму на менші незалежні служби, які спілкуються одна з одною через прикладний програмний інтерфейс. Кожен мікросервіс зосереджений на певній власній функції і його можна розробляти, розгортати та масштабувати незалежно від інших.

4. Безсерверна архітектура: у безсерверній архітектурі хмарний провайдер динамічно керує інфраструктурою. Розробники пишуть і розгортають код як функції, а хмарний постачальник займається масштабуванням, розподілом ресурсів і керуванням серверами.

5. Архітектура керована подіями: ця архітектура базується на подіях і повідомленнях для запуску та обміну даними між різними частинами системи.

Компоненти реагують на події та можуть випромінювати події, на які реагують інші компоненти.

6. Багаторівнева архітектура: у цій архітектурі серверні системи поділяються на рівні чи шари (рівень презентації, рівень робочої логіки, рівень доступу до даних тощо), щоб розділити проблеми та покращити масштабованість, зручність обслуговування та гнучкість.

7. Архітектура SOAP: Хоча зараз менш поширений, простий протокол доступу до об’єктів (SOAP) визначає суворий протокол обміну повідомленнями та формат повідомлень на основі XML для обміну структурованою інформацією.

В контексті проектування серверного модуля, що розробляється, вигідним буде залучити до рішення задачі REST API та багаторівневу архітектури. Створення RESTful API за багаторівневою архітектурою приносить вагомі переваги:

– Масштабованість: RESTful API розроблено таким чином, щоб не мати стану та орієнтуватися на ресурси, що дозволяє легше масштабуватися горизонтально. Рівневі архітектури додатково підвищують масштабованість, розділяючи завдання на окремі рівні, що полегшує керування та масштабування кожного рівня незалежно.

– Модульність і придатність до обслуговування: багаторівневі архітектури сприяють модульності, розділяючи різні проблеми на окремі рівні. Це розділення полегшує розуміння, обслуговування та оновлення окремих частин системи, не впливаючи на інші.

– Гнучкість і багаторазове використання: RESTful API, з їх акцентом на ресурсах і стандартизованих методах HTTP, пропонують гнучкий і багаторазовий спосіб взаємодії з ресурсами сервера. Поєднання цього з багаторівневою архітектурою гарантує, що зміни на одному рівні часто можуть бути ізольовані, зменшуючи вплив на інші частини системи та полегшуючи повторне використання компонентів.

– Чіткий розподіл завдань: Багаторівневі архітектури забезпечують чіткий розподіл між різними функціями системи. Це поділ полегшує керування та тестування кожного рівня незалежно, покращуючи якість коду та зменшуючи складність.

– Безпека та продуктивність: RESTful API можуть отримати вигоду від багаторівневих реалізацій безпеки, де механізми безпеки застосовуються на різних рівнях архітектури (наприклад, автентифікація та авторизація на рівні презентації). Багаторівневі архітектури також дозволяють оптимізувати та підвищити продуктивність на окремих рівнях, не впливаючи на всю систему.

– Взаємодія та стандартизація: RESTful API, засновані на стандартах HTTP, сприяють взаємодії між різними системами та платформами. У поєднанні з багаторівневою архітектурою він гарантує, що протоколи зв’язку, формати даних та інтерфейси чітко визначені та стандартизовані між рівнями, полегшуючи інтеграцію зі сторонніми службами або клієнтами.

Поєднання принципів дизайну RESTful із багаторівневою архітектурою забезпечує структурований, гнучкий і масштабований підхід до побудови API, сприяючи поділу проблем, зручності обслуговування та легкості еволюції з часом. Щодо інших архітектурних патернів, для них притаманні деякі недоліки, які не зовсім поєднуються з особливостями серверного модуля, що розробляється, наприклад складнощі з кешуванням (архітектура GraphQL), підвищена складність проектування (мікросервісна архітектура), залежність від визначеного постачальника хмарних послуг (безсерверна архітектура), тісно обмежена веб сумісність (архітектура SOAP), тощо.

Розглянемо багаторівневу архітектуру в контексті серверного модуля, що розробляється. На рисунку 2.1 зображено схематичне зображення типової системи, яка побудована по багаторівневій архітектурі.

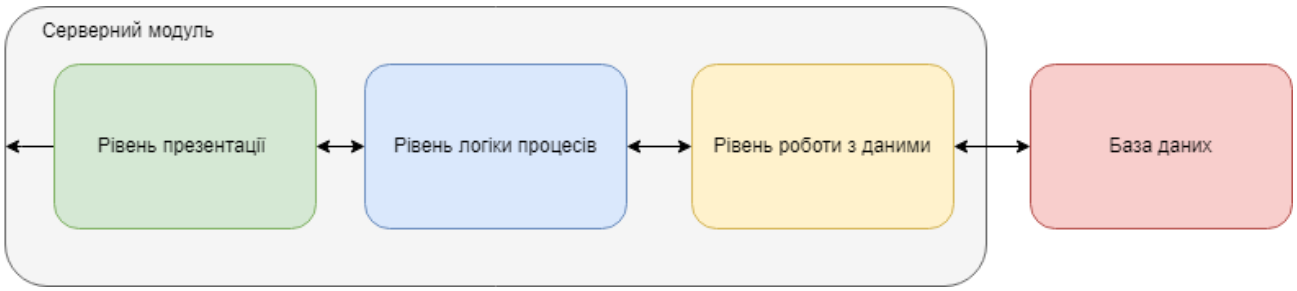

Рисунок 2.1 – Архітектура розробленого додатку

Аналогічний підхід до розподілення відповідальності між кодовими модулями серверного модуля буде застосовано для поточної розробки.

Рівень презентації буде представлений класами контролерами. Контролери є частиною архітектури програмного забезпечення, що відповідають за обробку вхідних запитів у вебзастосунках. Контролери відповідають за взаємодію з даними та представленням для обробки запитів користувачів. Вони містять методи, які обробляють вхідні HTTP-запити, аналізують отримані дані і викликають інші компоненти програми, такі як сервіси, для обробки цих запитів. Класи контролерів організують методи, які відповідають на різні типи запитів. Гарна практика полягає у тому, щоб кожен клас контролера відповідав за конкретну частину функціональності програми, допомагаючи у організованому збереженні коду.

Рівень логіки процесів буде представлений класами сервісами. Класи сервісів є складовою частиною програмного забезпечення, які відповідають за виконання певних операцій або функцій у системі. Вони містять код, який виконує конкретні завдання, такі як обробка даних, взаємодія з базою даних, виконання складних алгоритмів тощо. Ці класи допомагають відокремити операції та логіку від інших компонентів програми, таких як інтерфейс користувача або зберігання даних. Вони забезпечують модульність та зручну організацію коду, спрощуючи утримання системи та розвиток програми.

Рівень роботи з даними представлені класами репозиторіями. Класи репозиторії відповідають за взаємодію з базами даних чи іншими джерелами даних. Основна мета репозиторіїв – це абстрагувати логіку доступу до даних від

інших компонентів програми та надати зручний інтерфейс для роботи з ними. Класи репозиторіїв зазвичай містять методи для виконання операцій над даними, таких як отримання, збереження, оновлення та видалення. Вони виконують запити до баз даних або інших джерел даних, взаємодіючи з моделями даних та іншими компонентами програми.

На рисунку 2.2 зображено схематичне зображення класів серверного модуля відповідно до багаторівневої архітектури.

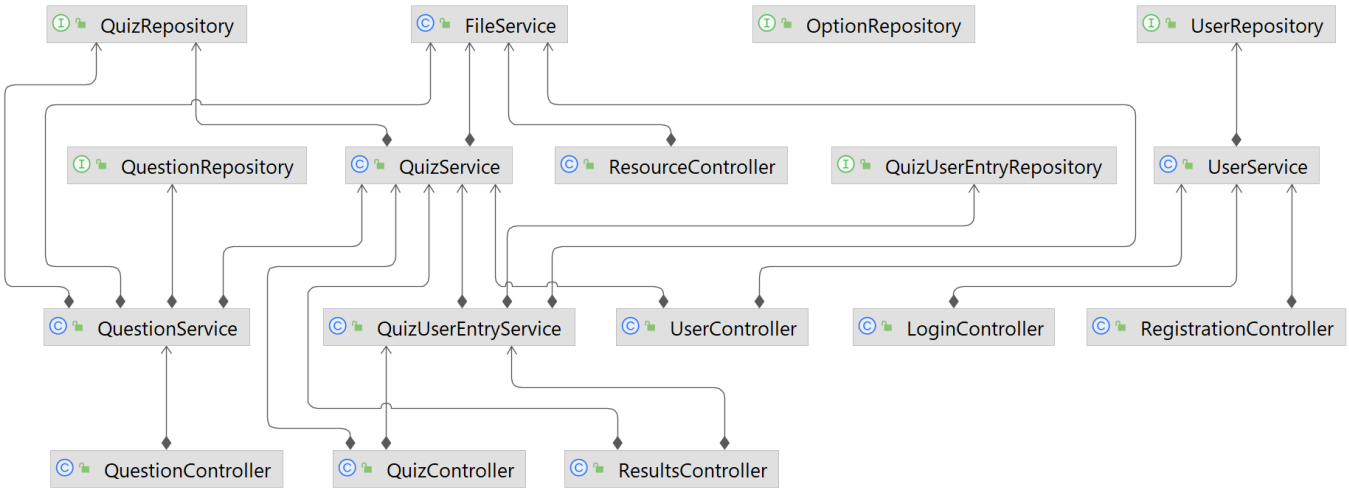

Рисунок 2.2 – Класи серверного модуля

Класи, які зображені на схемі, повинні містити основну логіку серверного модуля. Також буде існувати певний набір допоміжних чи утилітарних класів, які виконуватимуть загальні цілі та задачі і які не відносяться до жодного з архітектурних рівнів. Наприклад клас роботи з веб токенами, класи конфігурації серверного модуля тощо.

2.2 Проєктування бази даних та методу зберігання фотоданих

Згідно до поставлених вимог до серверного модуля що розробляється, розглянемо процес проєктування сталого сховища робочих даних системи. В якості робочих даних що будуть зберігатися в сталому сховищі даних визначаються такі як дані про користувачів, тести, тестові запитання, відповіді до запитань, різноманітні зображення.

Для забезпечення функцій постійного сховища існують два загальних типи рішень – реляційні бази даних та нереляційні бази даних, або SQL та NoSQL. SQL (Structured Query Language) і NoSQL – це дві широкі категорії систем керування базами даних, кожна з яких має свої особливості, переваги та варіанти використання [14] .

До особливостей баз даних SQL відносять наступні:

– Структуровані дані: Бази даних SQL базуються на структурованій схемі, часто використовують таблиці для організації даних у рядках і стовпцях. Вони дотримуються попередньо визначеної схеми, де структура є фіксованою.

– Властивості ACID: вони відповідають властивостям ACID (атомарність, консистенція, ізоляція, довговічність), що забезпечує надійність, послідовність і безпеку транзакцій даних.

– Реляційні бази даних: бази даних SQL, такі як MySQL, PostgreSQL, SQL Server і Oracle, є реляційними базами даних. Вони чудово обробляють структуровані дані з чітко визначеними зв’язками між об’єктами.

– Масштабованість: Традиційні бази даних SQL можуть зіткнутися з труднощами горизонтального масштабування, особливо з величезними обсягами даних або коли потрібно вносити часті зміни в схему.

До особливостей баз даних NoSQL відносять наступні:

– Гнучкість схеми: Бази даних NoSQL пропонують гнучкі конструкції схем, що дозволяє легко модифікувати та адаптувати до змінних структур даних. Вони можуть ефективно обробляти напівструктуровані та неструктуровані дані.

– Різноманітність моделей. Бази даних NoSQL представлені в різних моделях, наприклад на базі документів (MongoDB), сховищах ключів і значень (Redis), орієнтованих на стовпці (Cassandra) і графічних базах даних (Neo4j), кожна оптимізована для конкретних випадків використання.

– Масштабованість: Бази даних NoSQL часто мають більшу горизонтальну масштабованість, що полегшує обробку великих обсягів даних і розподіл між кількома серверами або вузлами.

– Теорема CAP: Бази даних NoSQL розроблено з іншими компромісами порівняно з базами даних SQL, зосереджуючись на гнучкості, масштабованості та доступності, часто за рахунок принесення в жертву суворої узгодженості (на основі теореми CAP).

Вибір між SQL і NoSQL часто залежить від характеру даних, вимог до масштабованості, складності зв'язків і конкретних потреб програми. Бази даних SQL добре підходять для структурованих даних і додатків, які потребують складних запитів і сумісності з ACID, тоді як бази даних NoSQL пропонують більшу гнучкість і масштабованість для обробки неструктурованих або напівструктурованих даних, особливо в розподілених або великомасштабних системах. Часто для оптимальної продуктивності та гнучкості застосовуються гібридні підходи або використання обох типів баз даних у різних частинах програми.

В якості рішення для постійного сховища даних в серверному модулю що розробляється, вибір був наданий реляційним базам даних у зв'язку з їх особливостями які явно поєднуються з поставленими задачами. Як було згадано в даному розділі, реляційні БД ефективно працюються зі фіксованими схемами даних, відповідають стандартам ACID, що, поміж інших особливостей, забезпечує можливість трансакційної роботи з даними та забезпечує полегшений контроль за консистентністю даних у сховищі. Також важливою перевагою є саме мова структурованих запитів, яка дозволяє здійснювати складні операції над даними сховища, тим самим полегшує розробку в цілому.

Щодо вибору конкретного постачальника системи керування баз даними, було обрано СКБД MySQL. MySQL це система керування базами даних із відкритим вихідним кодом, яка відома своєю надійністю, зручним інтерфейсом і повною інтеграцією з низку програм [15].

Працюючи в рамках SQL (Structured Query Language), MySQL дотримується моделі клієнт-сервер. Клієнтська програма ініціює запити до сервера MySQL, який належним чином обробляє ці запити та повертає результати клієнту. Дані в MySQL мають структурований формат,

розміщений у таблицях, упорядкованих рядками та стовпцями. Ці таблиці встановлюють зв’язки за допомогою первинних ключів, зовнішніх ключів та індексів, сприяючи ефективному пошуку та маніпулюванню даними. MySQL підтримує різні типи даних, включаючи цілі числа, рядки, дати та інші.

Варті уваги атрибути MySQL включають його суворе дотримання властивостей ACID, що забезпечує цілісність і послідовність транзакцій. Його масштабованість поширюється як вертикально, так і горизонтально, враховуючи розширення ресурсів і розподіл на кількох серверах.

Безпека є першорядною проблемою в MySQL, що проявляється через такі надійні функції, як автентифікація користувачів, контроль доступу, шифрування та аудит, ретельно розроблені для зміцнення цілісності даних. Природа MySQL з відкритим вихідним кодом сприяє активній спільноті, кульмінацією якої є широка підтримка та інтеграція з асортиментом мов програмування, фреймворків та інструментів.

Отже безпосередньо розглянемо процес проєктування схеми даних. Як було згадано в даному підпункті, виділяється декілька сутностей для зберігання в базі даних:

– користувач системи;
– роль користувача;
– тест;
– питання;
– варіант відповіді.

Користуючись поставленими задачами та підходами до нормалізації схеми бази даних, було розроблено наступну схему (рис. 2.3).

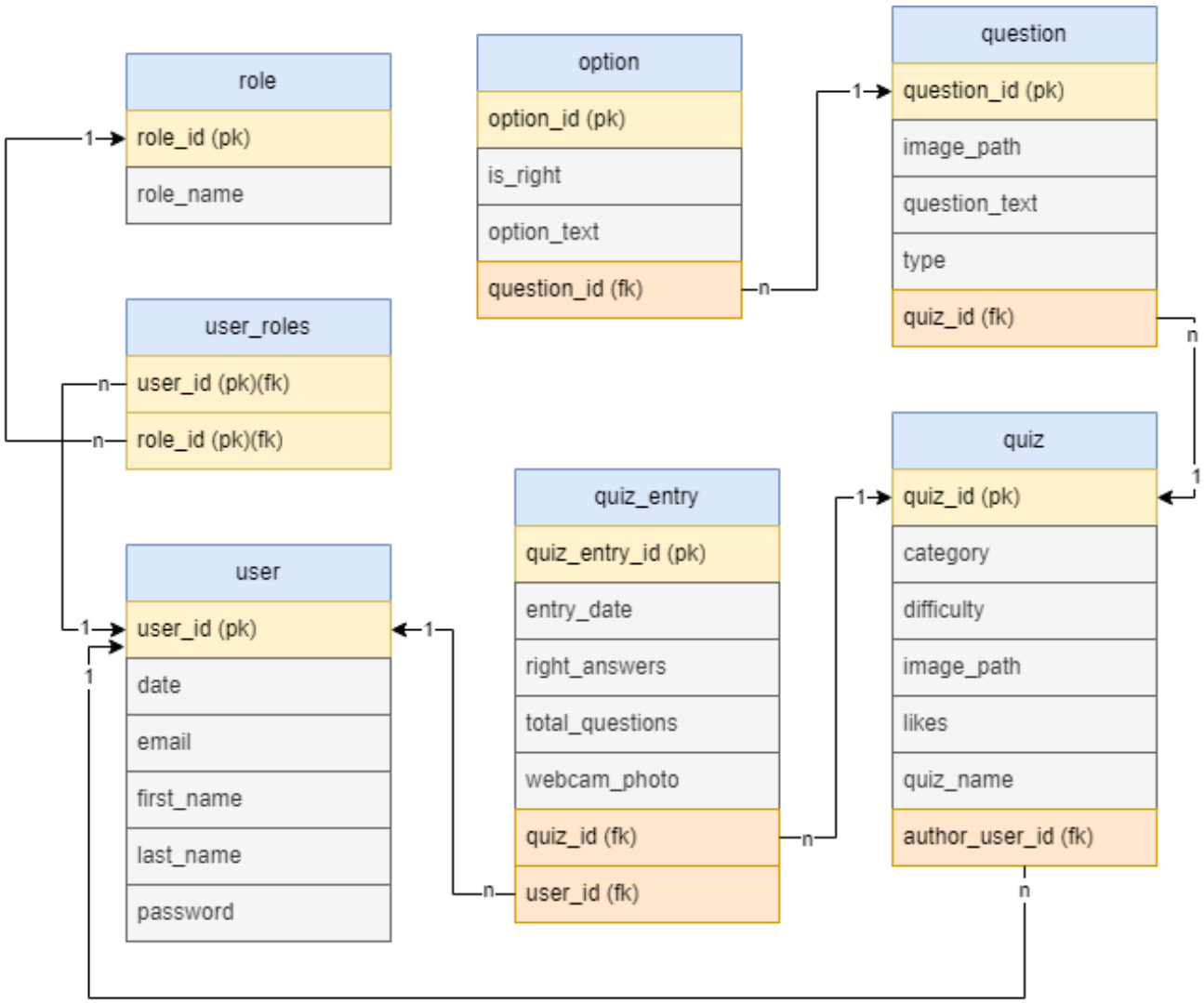

Рисунок 2.3 – Розроблена реляційна схема бази даних

Згідно створеної схеми, було створено 7 основних таблиць. Первинні ключі таблицю помічені відповідним маркуванням «(pk)», зовнішні ключі – «(fk)».

Розглянемо призначення кожної таблиці окремо. Таблиця «user» призначена для зберігання даних про активних користувачів системи. Має зв’язки «один до багатьох» з таблицями «user_role» та «quiz_entry». Має наступні колонки:

– «user_id» – сурогатний ідентифікатор користувача. Первинний ключ;
– «date» – дата реєстрації користувача;
– «email» - електронна адреса користувача;
– «first_name» - ім’я користувача;

– «last_name» - прізвище користувача;

– «password» - пароль користувача для доступу в систему.

Таблиця «role» призначена для визначення в системі існуючих ролей користувачів, а саме «учень» чи «вчитель». Має зв’язок «багато до одного» з таблицею «user_roles». Має наступні колонки:

– «role_id» - сурогатний ідентифікатор ролі. Первинний ключ;

– «role_name» - назва ролі.

Допоміжна таблиця «user_roles» виконує функцію зв’язування таблиць «user» та «role» для досягнення між ними зв’язку типу «багато до багатьох». Таким чином досягається можливість користувачеві мати одночасно декілька ролей у системі. Таблиця має наступні колонки:

– «user_id» - ідентифікатор користувач. Первинний ключ. Зовнішній ключ;

– «role_id» - ідентифікатор ролі. Первинний ключ. Зовнішній ключ.

Таблиця «quiz» використовується для зберігання сутностей тестів в системі. Має зв’язок «один до багатьох» з таблицями «quiz_entry» та «question». Має наступні колонки:

– «quiz_id» - сурогатний ключ тесту. Первинний ключ;

– «category» - предметна область, до якої відноситься тема тесту;

– «difficulty» - складність тесту;

– «image_path» - шлях до головного зображення тесту;

– «likes» - кількість позитивних оцінок користувачів тесту;

– «quiz_name» - головна назва тесту;

– «author_user_id» - ідентифікатор користувача, який створив тест. Зовнішній ключ.

Таблиця «quiz_entry» зберігатиме інформацію про звершені процеси тестування користувачів. Має зв’язок «багато до одного» з таблицями «quiz» та «user» таким чином встановлює між ними зв’язок багато до багатьох. Таблиця також має свій власний ідентифікатор, на відміну від «user_roles», таким чином,

за потреби, забезпечується можливість одному користувачу проходити один і той самий тест декілька разів. Таблиця має наступні колонки:

– «quiz_entry_id» - сурогатний ідентифікатор результату тестування. Первинний ключ

– «entry_date» - дата здійснення тестування користувачем;

– «right_answers» - правильні відповіді;

– «total_questions» - кількість відповідей;

– «webcam_photo» - шлях до зображень, отриманих в результаті тестування;

– «quiz_id» - ідентифікатор користувача;

– «user_id» - ідентифікатор тесту.

Таблиця «question» зберігатиме питання до окремих тестів. Має зв'язок «багато до одного» з таблицею «quiz» та «один до багатьох» з «option». Має наступні колонки:

– «question_id» - сурогатний ідентифікатор питання. Первинний ключ;

– «image_path» - шлях до зображення тестового питання;

– «question_text» - зміст тестового питання;

– «type» - різновид тестевого питання;

– «quiz_id» - ідентифікатор тесту. Зовнішній ключ.

Таблиця «option» зберігатиме варіанти відповідей до тестових питань. Таблиця має зв'язок «багато до одного» з таблицею «question». Має наступні колонки:

– «option_id» - сурогатний ідентифікатор варіанту відповіді. Первинний ключ;

– «is_right» - індикатор правильності варіанту відповіді;

– «option_text» - змісту варіанту відповіді;

– «question_id» - ідентифікатор тестового питання. Зовнішній ключ.

Таким чином, було описано вибір сталого сховища даних та описано процес проєктування основних таблиць і сутностей системи.

Розглянемо метод зберігання фотоданих серверного модуля. Для зберігання медіафайлів, а саме фотоданих, доцільно використовувати виділене файлове сховище. Зберігання фотоданих у файлових сховищах варто розглядати через їхню простоту управління та доступ до даних, що забезпечується за допомогою ієрархії папок [16]. Цей метод також забезпечує прямий доступ до файлів через файлову систему, що спрощує їхню обробку. Велика перевага полягає в швидкому та простому пошуку потрібних фотографій завдяки унікальним ідентифікаторам файлів. Зберігання в файлових сховищах дає можливість легко розширювати та масштабувати систему шляхом додавання нових файлів чи каталогів. Однак, при обробці великих обсягів даних, може виникнути проблема управління та пошуку необхідної інформації, а також безпека та резервне копіювання даних потребує додаткової уваги для запобігання втрати інформації.

Для зручної навігації та ідентифікації необхідних файлів сховища, було залучено ієрархічну файлову структуру. Для роботи зі сховищем файлів було спроєктовано спеціальний кодовий інтерфейс який визначає основні методи та операції при роботі з файловими даними. На рисунку 2.4 зображено схематичне зображення спроєктованого інтерфейсу.

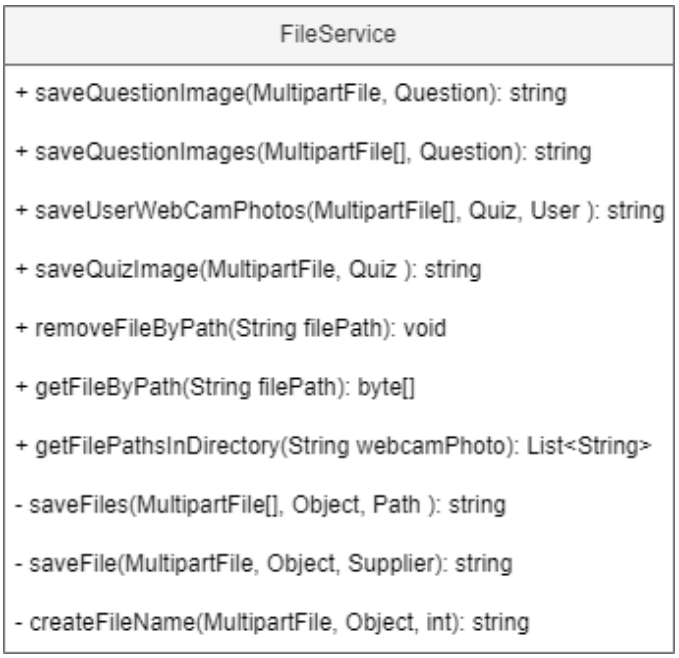

Рисунок 2.4 – Діаграма інтерфейсу FileService

Абстракція, яка надається кодовим інтерфейсом, відкриває можливості для реалізації різних файлових сховищ і, відповідно, потенційну заміну їх у майбутньому. На даному етапі розробки, передбачається використання файлової системи середовища серверного модуля як основного засобу для зберігання фотоданих. Це забезпечує швидкий доступ до інформації та легкість в імплементації. Проте, у майбутньому, з урахуванням архітектурних особливостей серверного модуля, конкретний вибір файлового сховища може бути переглянутий та змінений згідно нових потреб.

Отже, розглянемо безпосередньо спроєктовану структуру файлового сховища. У кореневому шляху до сховища розташовані високорівневі каталоги, які відповідають різним функціональним модулям сервера. На поточному етапі реалізовано єдиний високорівнева каталог – структура тестів, який містить інформацію про сутності тестів та їх компоненти. Таким чином, структура файлового сховища реалізує здатність до розширення завдяки розподіленням даних серверного модуля по різним високорівневим каталогам.

Розглянемо подальшу ієрархію високорівневого каталогу тестів. Даний каталог складається з каталогів окремих сутностей тестів, які ідентифікуються за допомогою унікальних сурогатних ідентифікаторів. Кожна така сутність може відображатися в файловому сховищі як каталог з назвою, яка відповідає ідентифікатору сутності. Таким чином, забезпечується можливість співставлення сутності тесту в базі даних та у файловому сховищі завдяки унікальності їх первинного ключа та назві каталогу. Кожен каталог окремо взятого тесту може містити другорядні каталоги, такі як каталог результатів фотофіксації тестування та каталог зображень тестових питань. Окрім цього безпосередньо в корені каталогу сутності тесту можуть міститися головне зображення тесту.

Будова другорядного каталогу результатів фотоконтролю тесту повторює ієрархію свого батьківського каталогу. Такий каталог містить в собі підкаталоги, які відображають сутність результатів тестування користувачів, або ж як згадувалося в даному підрозділі – таблиця «quiz_entry». Даний тип каталогів

містить в собі фотодані, які були отримані в результаті здійснення тестування тесту. Щодо другорядного каталогу тестових зображень, він відповідно буде містити зображення тестових запитань відповідної сутності тесту.

На рисунку 2.5 відображено схематичне зображення розробленої ієрархічної структури файлового сховища для збереження фотоданих серверного модуля, яке наповнене певними даними.

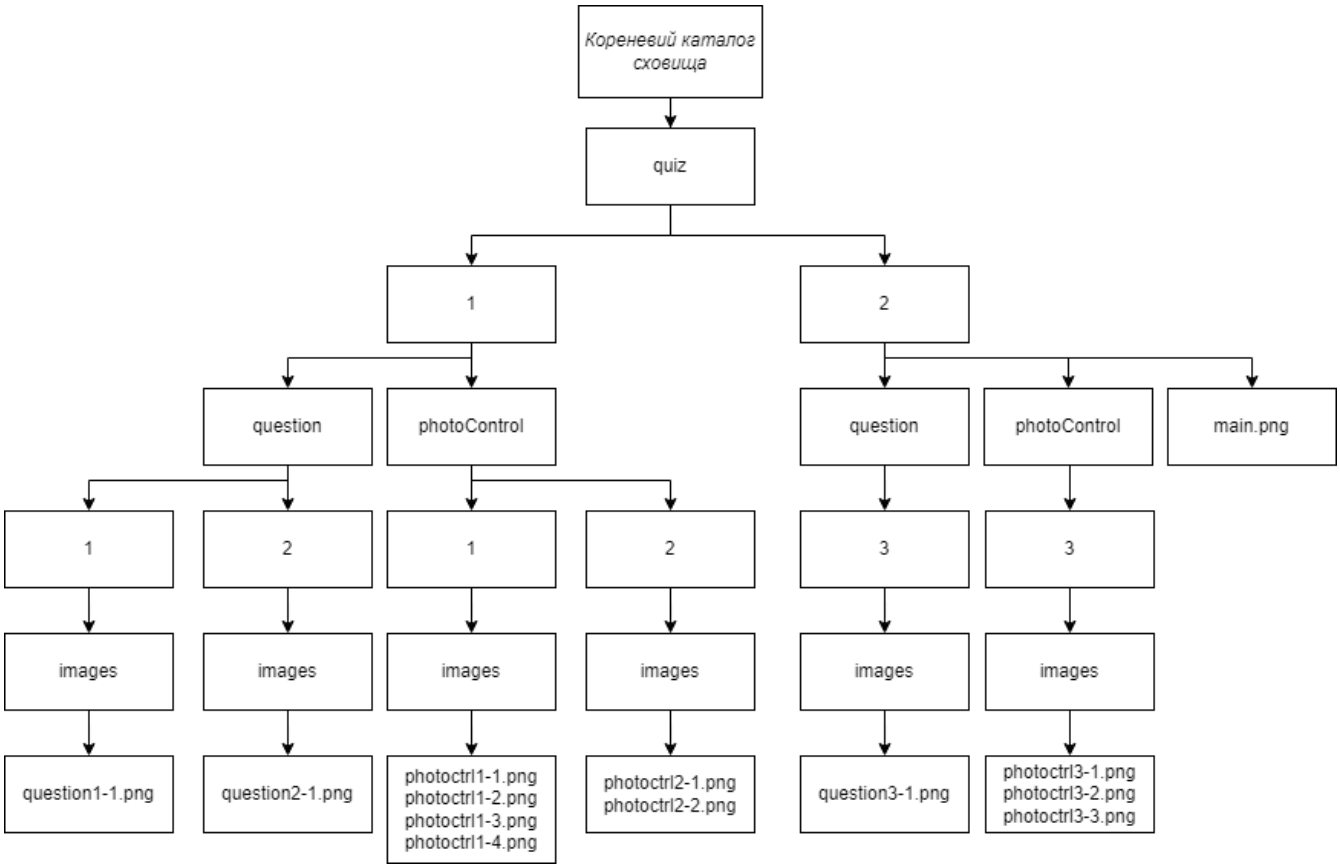

Рисунок 2.5 – Схематичне зображення ієрархічної структури сховища фотоданих

Отже було спроєктовано схему бази даних та ієрархічної структури файлового сховища фотоданих для серверного модуля, що є важливою складовою функціональності системи. Розроблена схема реляційної бази даних надає можливість серверному модулю ефективно зберігати та організовувати дані, необхідні для функціонування системи. Водночас спроектована структура

файлового сховища забезпечує структурований та швидкий доступ до фотоданих, що є критично важливим для роботи з даними серверного модуля.

2.3 Розробка структури прикладного програмного інтерфейсу серверного модуля

Прикладний програмний інтерфейс (API) – це набір протоколів, інструментів і визначень, які дозволяють різним програмам спілкуватися та взаємодіяти один з одним [17]. Він служить посередником, який дозволяє різним програмним системам отримувати доступ і використовувати функції, дані або служби одна одної, не вимагаючи доступу до основного коду або деталей реалізації.

API визначають методи, правила та формати даних, які програми можуть використовувати для запиту та обміну інформацією, виконання певних дій або доступу до ресурсів з іншої програмної системи, служби чи платформи. Вони сприяють безперебійній інтеграції та взаємодії між різними системами, дозволяючи розробникам розвивати існуючі функції та створювати нові програми, використовуючи можливості, надані API.

По суті, API встановлює контракт між різними компонентами програмного забезпечення, описуючи, як вони можуть взаємодіяти та обмінюватися даними, що дозволяє створювати надійні та взаємопов’язані.

В контексті прикладного програмного інтерфейсу важливо також розглянути поняття «back end» та «front end» або ж серверна частина та клієнтська частина. Ці модулі взаємодіють через інтерфейс прикладного програмування (API), який полегшує зв’язок і обмін даними між цими окремими компонентами.

Серверна частина відноситься до серверної сторони програми, де відбувається обробка даних, зберігання та реалізація доменної логіки. Він надає API, які визначають кінцеві точки та операції, що дозволяють інтерфейсу або зовнішнім системам отримувати доступ, отримувати, маніпулювати або зберігати дані. Ці API забезпечують структурований інтерфейс для взаємодії

клієнтської частини з серверними службами, дозволяючи клієнтської частині запитувати певні функції, виконувати операції та отримувати відповідні дані чи відповіді.

І навпаки, клієнтська частина представляє інтерфейс користувача або клієнтську сторону програми, де користувачі взаємодіють із програмним забезпеченням. Цей модуль використовує API, який надається серверною частиною, для доступу та відображення інформації, отриманої з самого ж сервера. Інтерфейс використовує API для надсилання запитів, отримання відповідей і представлення даних користувачам у зручний та інтерактивний спосіб.

Загалом, серверна частина надає API, які визначають доступні функції та точки доступу до даних, тоді як зовнішня частина використовує ці API для зв’язку з серверною частиною, уможливлюючи представлення даних і взаємодію користувачів у програмі. API діє як посередник, який сприяє безперебійному зв’язку та обміну даними між серверними та зовнішніми компонентами програмної системи.

Таким чином, згідно поставлених задач у розділі 1.4 необхідно сформувати прикладний програмний інтерфейс серверного модуля тестувальної системи з використанням технології Google Cloud Vision. Розроблений прикладний програмний інтерфейс буде сформовано з врахуванням архітектурного стилю REST, згідно підрозділу 2.1. Розглянемо таблицю 2.1.

Таблиця 2.1 – Перелік ресурсів прикладного програмного інтерфейсу

HTTP Метод	Шлях до ресурсу	Опис	Можливі відповіді
POST	/login	Авторизація користувача у системі за допомогою облікових даних	200, 400, 401 5**

Продовження таблиці 2.1

HTTP Метод	Шлях до ресурсу	Опис	Можливі відповіді
GET	/quiz/{ІД тесту}/question	Отримати тестове питання по ІД тесту	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/question	Додати тестове питання до тесту по його ІД	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/questions	Додати декілька тестових питань до тесту по ІД тесту	200, 400, 401, 403, 404, 5**
GET	/quiz	Отримати сторінку тестів	200, 401, 403, 404, 5**
POST	/quiz	Створити новий тест	200, 400, 401, 403, 404, 5**
PUT	/quiz	Оновити сутність тесту	200, 400, 401, 403, 404, 5**
GET	/quiz/{ІД тесту}/entries	Отримати результати тестування за ІД тесту	200, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/image	Додати головне зображення до тесту по його ІД	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/submit	Подати результати тестування по ІД тесту	200, 400, 401, 403, 404, 5**
GET	/quiz/{ІД тесту}	Отримати тест по його ІД	200, 401, 403, 404, 5**
POST	/register	Зареєструвати нового користувача	200, 400, 5**

Продовження таблиці 2.1

HTTP Метод	Шлях до ресурсу	Опис	Можливі відповіді
GET	/resource/{ІД тесту}	Отримати ресурс по його шляху, наприклад зображення	200, 401, 403, 404, 5**
DELETE	/resource/{ІД тесту}	Видалити ресурс за його шляхом	200, 400, 401, 403, 404, 5**
POST	/results/{ІД тесту}/webcamPhotos	Зберегти зображення підчас здійснення тестування зі ІД результати тестування	200, 400, 401, 403, 404, 5**
GET	/results/entries	Отримати результати тестування користувача	200, 401, 5**
GET	/user	Отримати інформацію про користувача	200, 401, 5**
GET	/user/quizzes	Отримати тести створені користувачем	200, 401, 5**

Таким чином було сформовано так званий контракт для прикладного програмного інтерфейсу серверного модуля, який визначатиме поведінку ресурсів і їх призначення. Завдяки підходу REST розроблений прикладний програмний інтерфейс набуває більшої гнучкості. Наприклад, деякі ресурси можуть мати динамічний шлях, який в свою чергу може залежати від ідентифікатора сутності, як у випадку з ресурсом «GET /quiz/{ІД

тесту}/questions», який повинен повернути питання того тесту, ідентифікатор якого вказаний у шляху самого ресурсу.

Описано також різні HTTP коди відповіді для кожного ресурсу, що дозволяє клієнтам програмного інтерфейсу адаптувати свою поведінку в залежності від отриманих кодів. Це надає користувачам можливість попередньо передбачати реакцію ресурсів на різні ситуації та розробляти власні механізми обробки виняткових випадків. Такий підхід дозволяє оптимізувати взаємодію з серверними ресурсами та ефективно обробляти можливі помилки чи стан сервера, що виникають у процесі взаємодії між клієнтом і сервером.

В якості формату даних, який буде підтримувати прикладний програмний інтерфейс, було обрано формат JSON [18]. Загалом існує два поширених формати даних, які використовуються у сучасних API – XML та JSON. XML є мовою розмітки, яка використовується для структурування та зберігання даних у текстовому форматі. Він має відкриту структуру та може представляти складні дані з різних джерел у вигляді дерева. XML використовує теги для визначення різних елементів та атрибутів даних. Його часто використовують для обміну даними між системами, оскільки він дозволяє створювати власні теги та структури.

JSON, з іншого боку, є легким текстовим форматом обміну даними, який базується на синтаксисі мови програмування JavaScript. Він використовується для представлення об'єктів та масивів у вигляді пар "ключ-значення". JSON є більш легким у порівнянні з XML, його легше читати та розуміти, а також використовується менше зайвих символів в самому файлі, що робить його більш компактним.

У порівнянні між форматами JSON та XML у контексті використання в HTTP запитах, JSON виявляється більш вигідним, забезпечуючи ефективніший обмін даними. Його простота та компактність робить його більш популярним для використання в HTTP-запитах. На рисунку 2.6 приклад тіла HTTP запиту у форматі JSON, який буде вимагати ресурс «POST /register».

Рисунок 2.6 – Структура тіла запиту «POST /register».

JSON об’єкт, який зображений на рисунку 2.4, містить інформацію, яка використовується для процесу реєстрації користувача в системі. Його структура включає чотири поля: «email», «name», «password» та «surname». Кожне поле відповідатиме конкретному типу інформації, яка буде передаватися при реєстрації. Наприклад, «email» буде використовуватися для введення адреси електронної пошти, «name» для вказання імені користувача, «password» для задання пароля та «surname» для введення прізвища.

Таким чином було сформовано прикладний програмний інтерфейс серверного модуля дистанційного тестування з використанням технології Google Cloud Vision, описано його основні ресурси їх призначення та поведінку.

2.4 Розробка алгоритмів роботи програми

Розробка алгоритмів роботи для серверного модуля є критичним етапом у процесі його, оскільки вони визначають логіку та спосіб взаємодії сервера з модулями клієнтами та іншими компонентами системи. Розроблені алгоритми забезпечують ефективну та швидку роботу сервера, сприяючи оптимальному використанню ресурсів. Це важливо для забезпечення високої продуктивності та задоволення потреб користувачів. Також, алгоритми визначають способи обробки різноманітних запитів від клієнтів, управління даними та взаємодії з іншими системами. Правильна реалізація цих алгоритмів забезпечує надійність та коректність функціонування сервера. Крім того, алгоритми допомагають у вирішенні різноманітних завдань, таких як обробка великих обсягів даних,

забезпечення безпеки та конфіденційності, і взагалі вирішення завдань, які є унікальними для конкретної системи чи додатку.

Розглянемо створені блок-схеми алгоритмів для більшості процесів серверного модуля що розробляється. На рисунку 2.7 зображено блок-схему алгоритм роботи процесу створення нового тесту у системі.

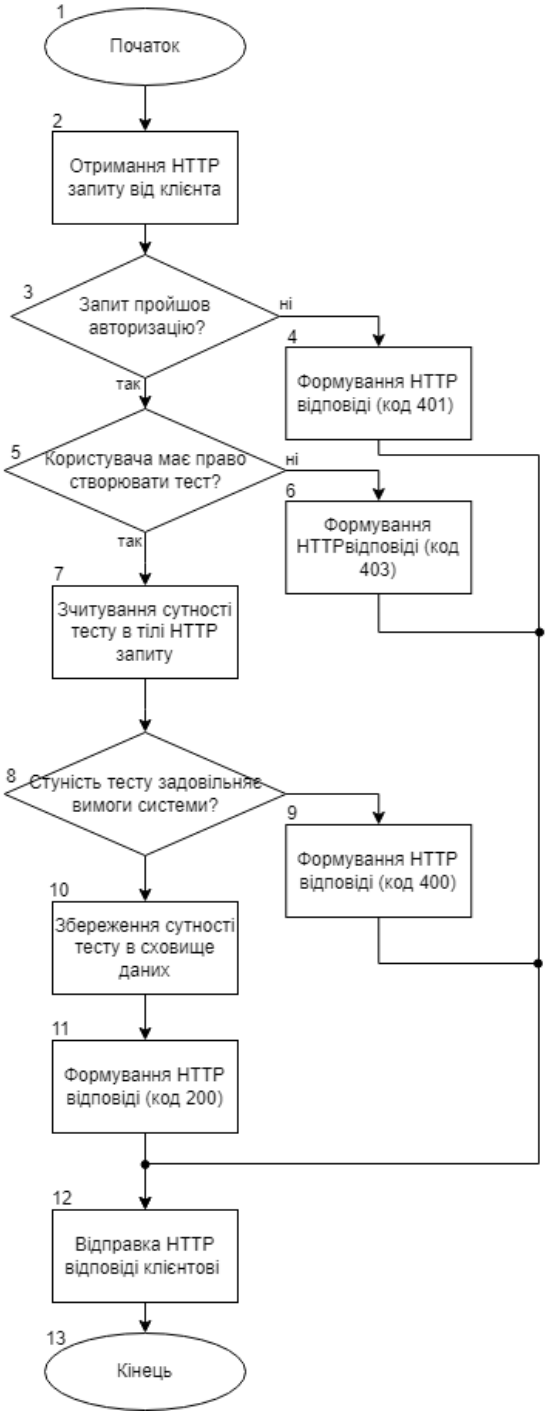

Рисунок 2.7 – Блок-схема алгоритму роботи процесу створення нового тесту

Операція збереження сутності тесту в сховищі даних серверного модуля передбачає кілька кроків. Коли серверний модуль отримує HTTP запит від клієнта, спершу проводиться процес авторизації для перевірки прав доступу користувача до відповідних даних. Після успішної авторизації, сервер аналізує отриманий запит, визначаючи тип операції: збереження нової сутності тесту чи оновлення існуючої.. Після валідації отриманих даних, сервер здійснює збереження сутності тесту в базі даних. Цей процес може включати створення нового запису або оновлення вже існуючого запису, залежно від характеру отриманого запиту. У разі виникнення помилок під час валідації або збереження, сервер генерує відповідну HTTP відповідь, що містить відповідний статус помилки або підтвердження успішності операції. Після цього, сформована відповідь відправляється клієнту, який ініціював запит, узгоджуючи спілкування між сторонами та підтверджуючи результат операції збереження тесту.

Невід’ємним процесом у роботів серверного модуля є процес автентифікації користувачів системою. Розглянемо цей процес детальніше. Коли сервер отримує HTTP запит від користувача, він починає процес автентифікації. Це включає перевірку наявності інформації для автентифікації в заголовку запиту, наприклад, ідентифікатора користувача та пароля або інші облікові дані. Після отримання цієї інформації сервер перевіряє її правильність та відповідність даним у системі. Це включає в собі перевірку коректності пароля та звернення до бази даних серверного модуля для перевірки існування та відповідності облікових даних. Якщо дані автентифікації визнані правильними та коректними, сервер продовжує процес, створюючи JWT токен. Цей токен містить інформацію про користувача та його права доступу. Після створення JWT токену, сервер включає його у тіло HTTP відповіді. Цей токен буде використовуватися користувачем для подальшої автентифікації та авторизації під час взаємодії з іншими ресурсами або послугами серверного модуля. Таким чином, після успішної автентифікації, сервер генерує відповідь з JWT токеном, який дозволяє користувачеві отримати доступ до ресурсів, які він має право використовувати, і

відправляє цей токен назад користувачеві як частину відповіді на його запит. На рисунку 2.8 зображено блок-схема алгоритм процесу автентифікації користувачів у системі.

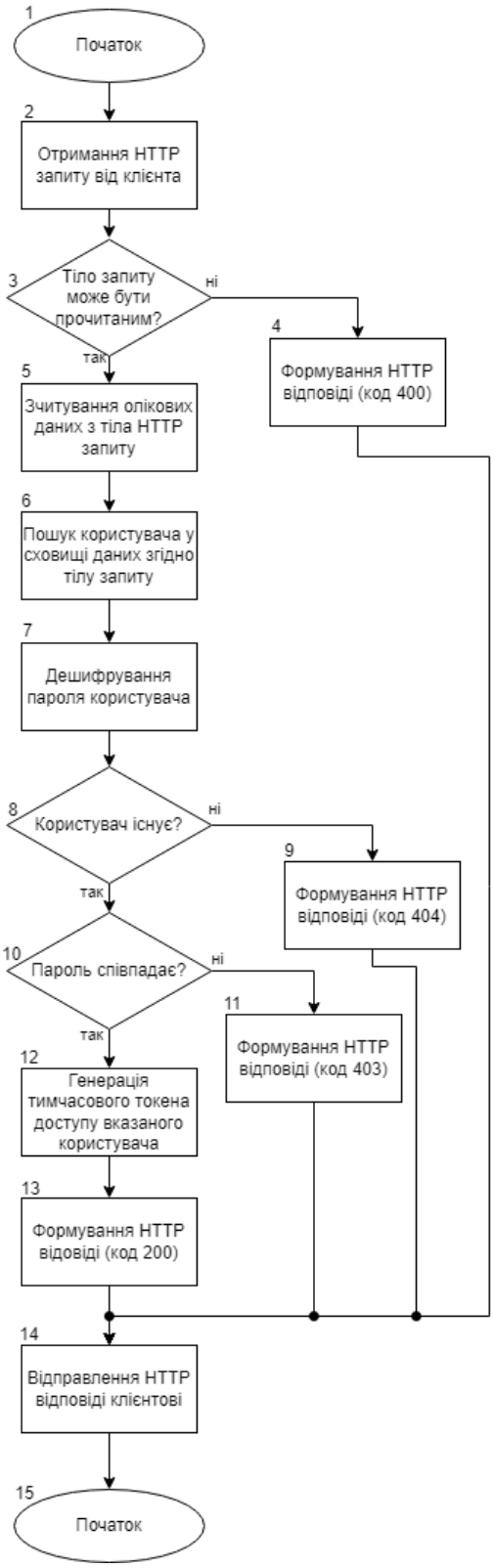

Рисунок 2.8 – Блок-схема алгоритм процесу автентифікації користувачів

Розглянемо роботу процесів зв’язаних з використанням технології Google Cloud Vision. На рисунку 2.9 зображено блок-схем алгоритм роботи процесу повернення результатів тестування.

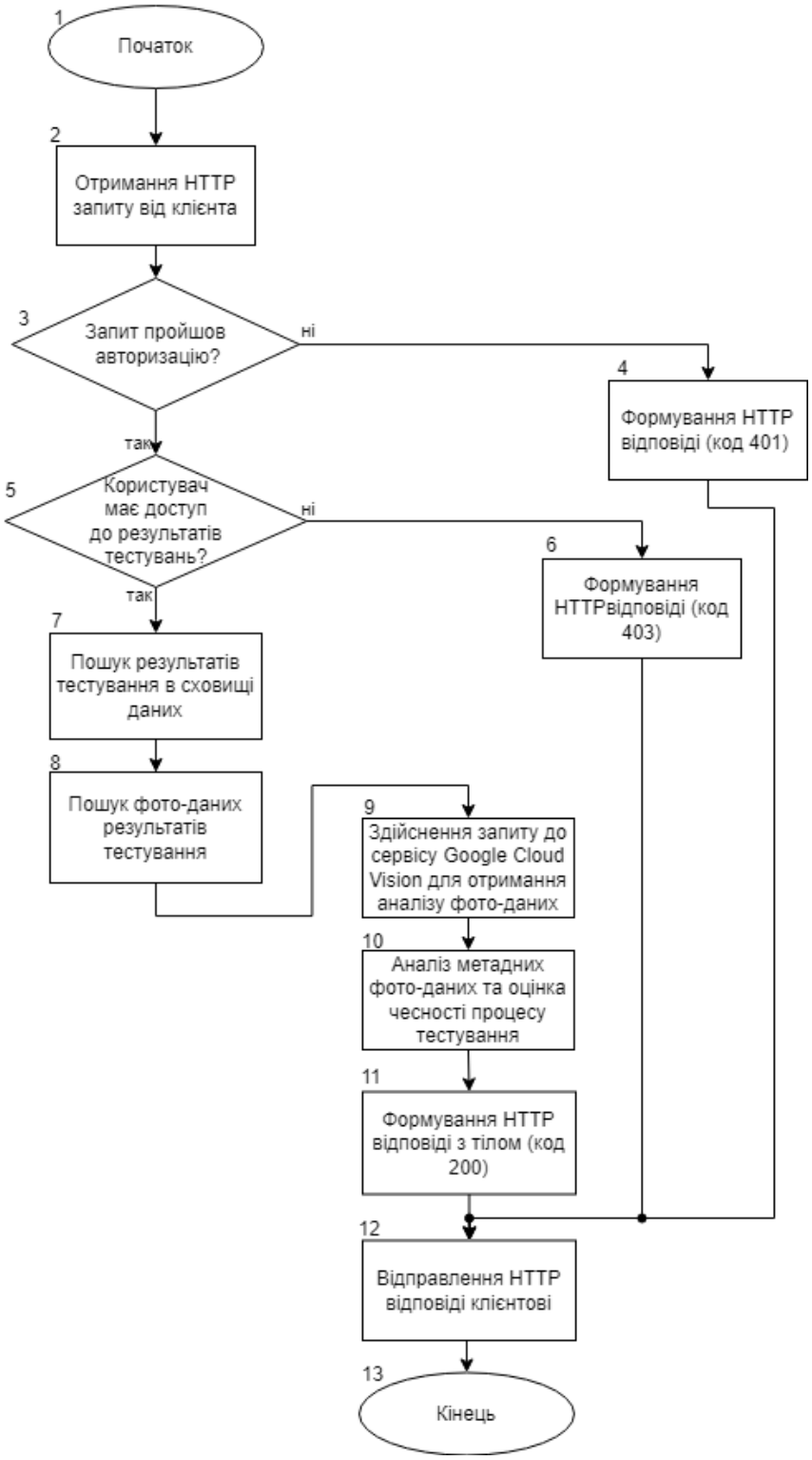

Рисунок 2.9 – Блок-схема алгоритм процесу отримання результатів тестування користувачів

Згідно блок-схемі на рисунку 2.9, серверний модуль отримує HTTP запит зі запитом на отримання результатів тестування користувачів. Спочатку сервер спробує знайти наявні сутності результатів тестування у сховищі даних, а наступним кроком, на основі отриманих результатів та фотоданих, проведе аналіз отриманих фотоданих, якщо такі. Сервер використовуватиме зовнішній сервіс Google Cloud Vision, для аналізу фотографічних даних. Сам аналіз буде включати виявлення обличчя на фотографіях, аналіз ключових особливостей зображень та спробу виявлення шахрайської дії, наприклад, використання фальшивих фотографій або спроб обману системи.

Після цього аналізу, сервер робить формує оцінку про відповідність отриманих даних очікуваним результатам. Якщо виявляються підозри на шахрайство або неправдивість даних, такі результати тестування помічаються як підозрілі та з вмістом потенційних порушень. В результаті сервер формує відповідь на HTTP запит, в тіло якого записує JSON масив всіх отриманих та оброблених результатів тестування.

2.5 Розробка методу дистанційного тестування з використанням Google Cloud Vison

Розглянемо розробку методу дистанційного тестування серверного модуля з використанням технології Google Cloud Vison.

В основі даного методу лежить використання фотоданих результатів фотофіксації процесу тестування та їх аналіз з використанням штучного інтелекту технології Cloud Vison на основі чого система може робити припущення про автентичність результатів тестування, що спонукатиме їх ретельної перевірки зі сторони користувачів системи.

Для належного застосування технологій розпізнавання та аналізу зображень розглянемо детально з технологію Google Cloud Vision в рамках харного сервісу Google Cloud Platfrom. Google Cloud Platform (GCP) – це набір хмарних послуг, інструментів та інфраструктури, які надаються компанією Google для створення, розгортання та управління програмами та послугами в

хмарному середовищі [19]. GCP пропонує широкий спектр сервісів, які включають обчислення, зберігання даних, аналітику, штучний інтелект, машинне навчання, Інтернет речей та інші послуги, що працюють у хмарному середовищі.

Google Cloud Vision в свою чергу є одним із видом послуг Google Cloud Platform. Google Cloud Vision – це хмарний сервіс для розпізнавання та аналізу зображень, розроблена компанією Google [20]. Цей сервіс використовує нейромережеві та комп'ютерні зорові технології для виконання різноманітних завдань обробки зображень. Основні можливості Google Cloud Vision включають:

– розпізнавання об'єктів: можливість ідентифікації та класифікації об'єктів на зображеннях;

– розпізнавання облич: аналіз зображень для визначення осіб та їх характеристик, таких як вік чи емоції;

– оптичне розпізнавання тексту (OCR): здатність виділяти та зчитувати текст зі зображень, що дозволяє перетворювати зображення з текстом на редаговані тексти;

– розпізнавання міток та логотипів: визначення логотипів та інших графічних елементів на зображеннях;

– аналіз відомостей про місце: вибірка інформації про об'єкти та місця на зображення;

– пошук відомостей про продукти: розпізнавання та ідентифікація товарів на зображеннях.

Google Cloud Vision може бути використана у різноманітних галузях, включаючи розробку додатків для розпізнавання зображень, автоматизацію обробки документів та впровадження функціоналу розпізнавання облич у додатках для безпеки. Вона надає розробникам потужний інструментарій для роботи з великим обсягом візуальних даних.

В рамках серверного модуля дистанційного тестування Google Cloud Vision буде інтегрована завдяки Cloud Vision API. Cloud Vision API – це

прикладний програмний інтерфейс (API), який надає доступ до функцій Google Cloud Vision.

На рисунку 2.10 наведене схематичне зображення загальної системи дистанційного тестування включно з інтеграцією Google Cloud Vision.

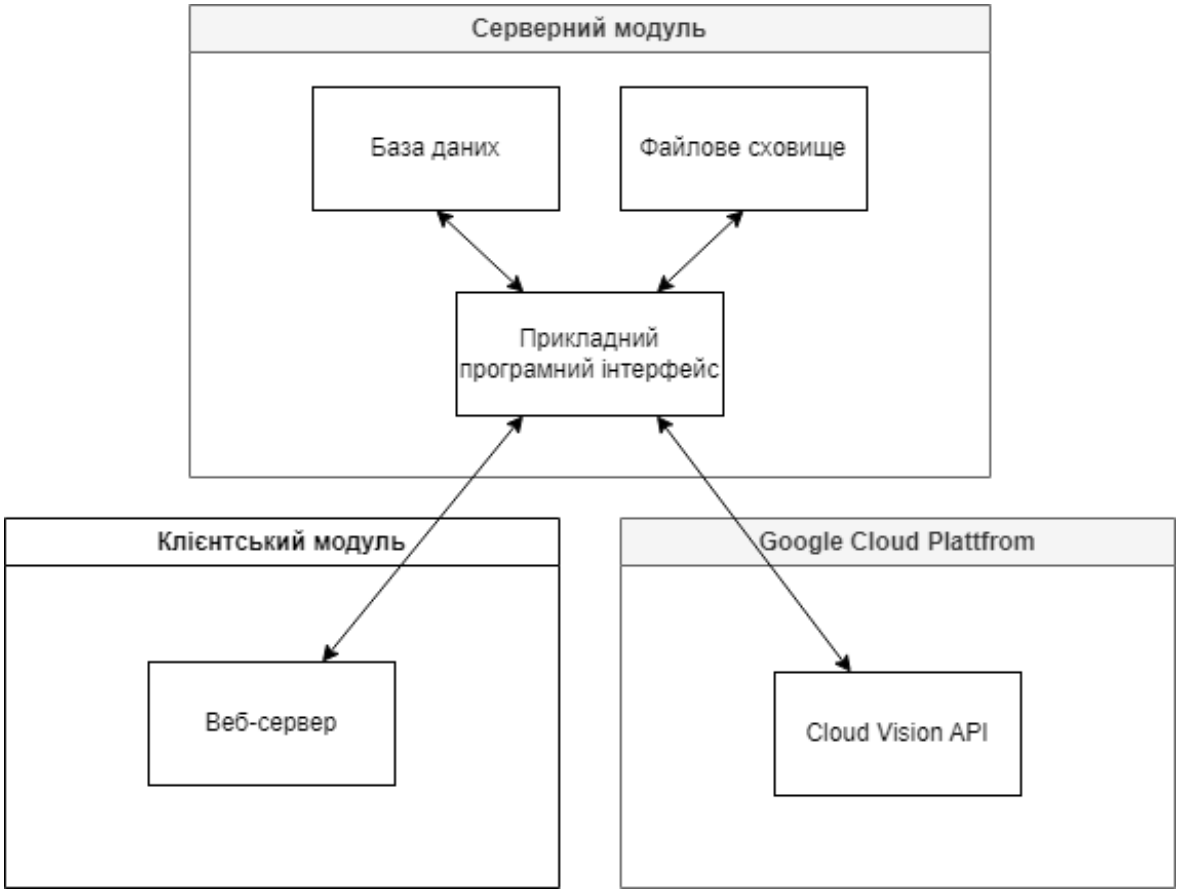

Рисунок 2.10 – Інтеграційна структура тестувальної платформи

Згідно наведеної схеми, серверний модуль використовуватиме Cloud Vision API для здійснення HTTP запитів відповідні ресурси з певною метою, наприклад аналіз фотоданих.

Розглянемо безпосередньо процес аналізу фотоданих серверним модулем з використанням технології Google Cloud Vision. На основі фотоданих, які були отримані в результаті здійснення тестування користувачем тестувальної платформи, серверним модулем формуватиметься аналітична оцінка достовірності результатів тестування. Таким чином, при оцінці результатів тестування відповідальною особі, буде забезпечена можливість одразу звернути

увагу на результати тестування, які система вважатиме з наявністю порушень. До порушень або підозрілих дій можуть відноситися наступні особливості об’єктів та атрибутів зображених у фотоданих:

– Наявність на зображені книг, зошитів, паперу або інших письмових записів

– Наявність на зображені мобільного телефону, смартфону, планшету та подібних пристроїв з доступом у мережу Інтернет;

– наявність більше ніж одного розпізнаного обличчя або відсутність жодного обличчя на зображені;

– наявність більше ніж одного розпізнаних особистості або відсутність жодної особистості на зображені;

– Наявність на обличчі сонцезахисних окулярів з темними лінзами;

– Розмите зображення обличчя чи погана якість зображення.

На зображенні 2.11 зображено демонстраційний аналіз зображення засобами Google Cloud Vision.

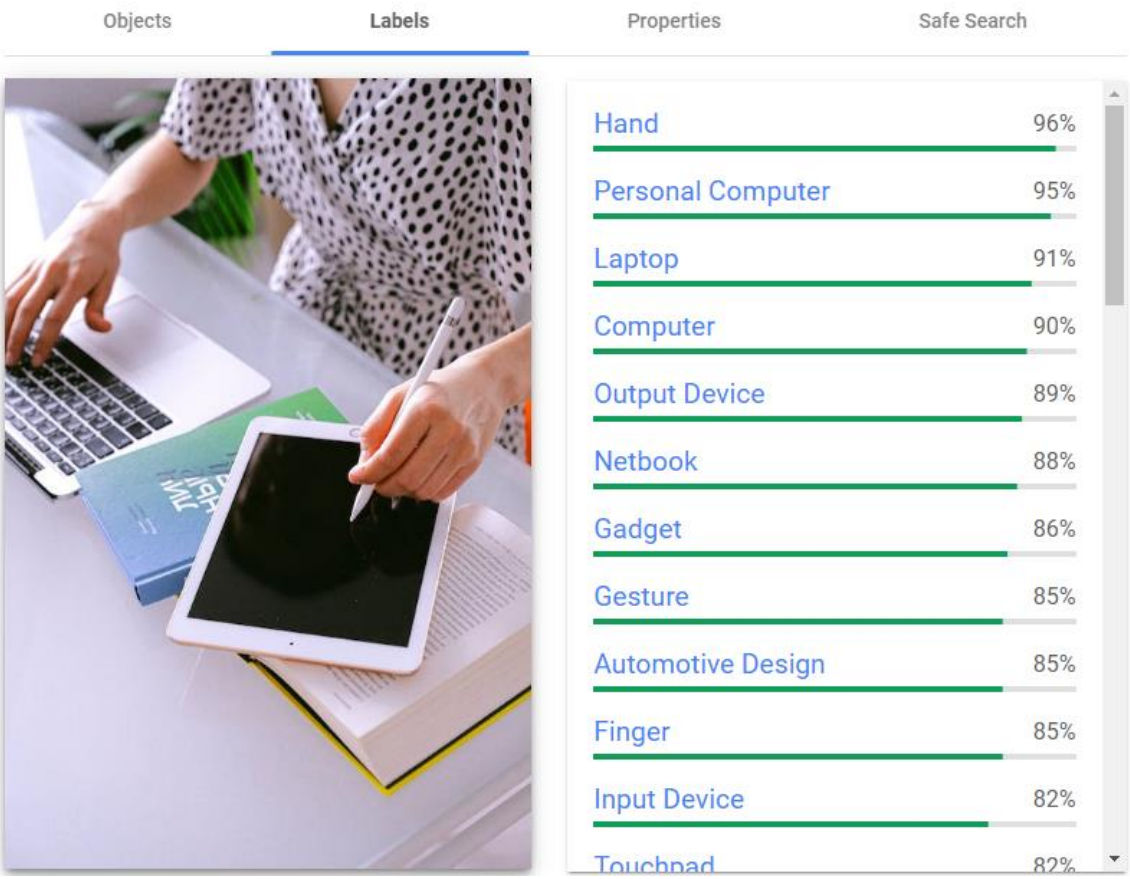

Рисунок 2.11 – Аналіз зображення від Google Cloud Vision

Аналіз зображення показав присутність на зображені таких тегів як «нетбук», «гаджет», «персональний комп’ютер» «комп’ютер», присутність об’єкту «планшетний комп’ютер» та зовсім відсутність жодного розпізнаного обличчя. Розглянемо наступний приклад на рисунку 2.12.

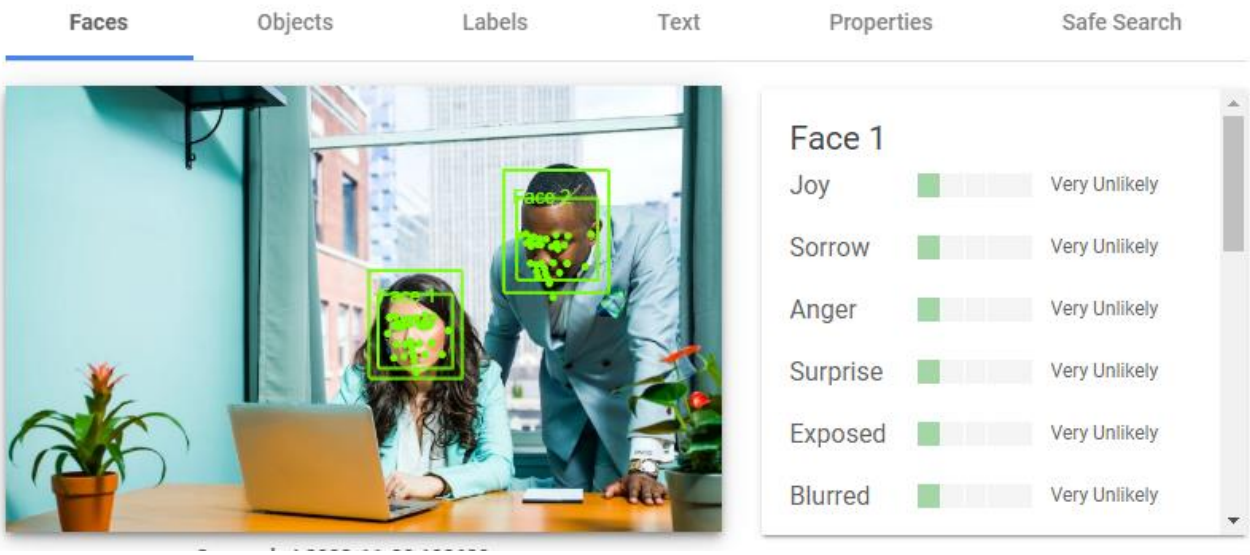

Рисунок 2.12 – Аналіз зображення від Google Cloud Vision

В даному прикладі, засобами Google Cloud Vision було розпізнано два обличчя на зображенні.

За допомогою функцій розпізнавання зображень серверний модуль дистанційного тестування отримує можливість здійснювати аналіз всіх фотоданих, що пов'язані з конкретними результатами тестування і користувачами. Оскільки кожен результат тесту може містити різну кількість зображень з фотофіксацією, оцінка наявності порушень здійснюється у відсотковому співвідношенні. Якщо більше ніж 20% усіх зображень містять певні ознаки порушень, аналітичний метод визначає, що тестування було проведено з порушеннями. В результаті, аналітична оцінка буде відображена кінцевому користувачу дистанційної тестувальної платформи, який може взяти її до розгляду. Даний метод дозволить кінцевим користувачам системи швидше та ефективніше знаходити можливі порушення академічної доброчесності від учасників тестування.

2.6 Висновки

У цьому розділі було розглянуто загальну архітектуру серверного модуля дистанційного тестування з використанням технології Google Cloud Vision. Були досліджені поширені сучасні архітектурні патерни, обрані та використані ті, які відповідають особливостям завдань, спрощуючи та прискорюючи розробку.

Була обрано тип сталого сховища робочих даних та спроєктована структура бази даних з урахуванням потреб системи. Спроєктовано сім основних таблиць, призначення яких було детально описано. Було спроєктовано та описано структуру файлового сховище фотоданих та наведено переваги його застосування.

Був детально описаний прикладний програмний інтерфейс (API) серверного модуля, у якому чітко визначено призначення та поведінку ключових ресурсів системи.

Були розроблені блок-схеми алгоритмів програми для полегшення розуміння роботи серверного модуля. Це надає уявлення про логіку роботи модулю при обробці запитів та виконанні операцій.

Також було описано розроблений метод дистанційного тестування з використанням Google Cloud Vison. Описано приклад застосування Cloud Vison API та роботу системи при аналітичному оцінювані фотоданих результатів тестування.

3 РОЗРОБКА КОДУ СЕРВЕРНОГО МОДУЛЯ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Вибір мови програмування грає важливу роль у розробці програмного забезпечення. Це може вплинути на ефективність, швидкість розробки, підтримку, а також загальну продуктивність команди розробників. Кожна мова програмування має свої особливості, сильні та слабкі сторони, і важливо аналізувати ці фактори перед вибором. В залежності від конкретної задачі, одна мова може бути більш пристосованою для роботи з великим обсягом даних, тоді як інша - для створення вебдодатків з інтерактивним інтерфейсом. Крім того, важливо враховувати екосистему і інструменти, що супроводжують мову програмування, оскільки вони можуть значно вплинути на швидкість розробки та підтримку коду.

Вибір мови програмування - це компроміс між потребами проєкту, вміннями команди розробників та технічними можливостями мови. Однак правильний варіантний аналіз дозволяє зробити обґрунтований вибір, що підвищує ефективність розробки програмного забезпечення та сприяє успішному впровадженню проєкту.

Розглянемо сучасні поширені варіанти мов програмування для розробки серверних рішень.

Java – це об'єктно-орієнтована мова програмування, що відома своєю незалежністю від платформи, простотою використання та великою спільнотою розробників [21]. Вона була розроблена компанією Sun Microsystems (згодом придбаною компанією Oracle) і має широке застосування у сферах розробки програмного забезпечення. Java відома своєю мультиплатформністю, оскільки програми, написані на Java, можуть запускатися на будь-якому пристрої або платформі, яка підтримує Java Virtual Machine (JVM). Це робить її популярним вибором для розробки кросплатформних додатків.

Мова має вбудовану підтримку багатопотоковості, що дозволяє розробляти програми, які виконують кілька завдань одночасно. Велика

стандартна бібліотека Java SE (Java Standard Edition) містить багато готових інструментів та функцій, що спрощують розробку різноманітних програм.

Також Java знаходить широке застосування у веброзробці, зокрема у створенні серверних додатків за допомогою фреймворків, таких як Spring або Hibernate. Вона також використовується у великих системах, іграх та мобільних додатках, зокрема на платформі Android.

Java відома своєю надійністю та безпекою. Одним з її переваг є автоматичне управління пам'яттю, що дозволяє уникнути багатьох помилок, пов'язаних з витоками пам'яті або неправильним використанням пам'яті.

Усі ці особливості роблять Java потужним інструментом для розробки різноманітних програм та систем, особливо там, де важлива переносимість, надійність та висока продуктивність розробки.

Розглянемо наступний варіант – мову програмування Python [22]. Python – це високорівнева мова програмування, яка відома своєю простотою, читабельністю та гнучкістю. Вона була розроблена з метою зробити код більш зрозумілим для програмістів, що сприяє швидкому вирішенню завдань. Python відкритий, має велику та активну спільноту розробників, яка постійно розвиває мову та надає величезний вибір бібліотек та модулів. Це робить Python дуже універсальним для різних областей, включаючи веброзробку, наукові дослідження, штучний інтелект, обробку даних, аналітику, автоматизацію та інші.

Однією з ключових переваг Python є його простота вивчення. Синтаксис мови досить лаконічний, що полегшує початок роботи новачкам у програмуванні. Також він володіє динамічною типізацією, що дозволяє зосередитися на розв'язанні завдань, а не на деталях типізації.

Python підтримує об'єктно-орієнтований, процедурний та функціональний підходи до програмування, що робить його досить універсальним і дозволяє використовувати різні стилі програмування в залежності від завдання.

Ще однією перевагою Python є його розширюваність та велика кількість сторонніх модулів. Наявність багатьох бібліотек, таких як NumPy для обробки

масивів даних, Pandas для роботи з таблицями даних, або Django для веброзробки, сприяє швидкому розвитку програм та спрощує роботу розробників.

Усі ці характеристики роблять Python популярним вибором для широкого спектру завдань у сфері програмування та розробки програмного забезпечення.

Розглянемо ще один варіант – мову програмування JavaScript. JavaScript - це високорівнева, інтерпретована мова програмування, яка використовується для розробки динамічних вебсторінок [23]. Вона є однією з ключових технологій для веброзробки, забезпечуючи можливість створення взаємодіючих елементів на вебсторінках. JavaScript широко використовується для створення функціональних ефектів на сторінках, таких як анімація, обробка подій користувача, зміна вмісту сторінки без перезавантаження (асинхронні запити) та багато іншого. Вона є ключовою складовою для реалізації багатьох сучасних вебдодатків та платформ.

JavaScript є мовою програмування з високим рівнем гнучкості. Вона підтримує об'єктно-орієнтований, функціональний та процедурний стилі програмування. Також вона має динамічну типізацію, що означає, що типи змінних визначаються автоматично під час виконання програми.

Однією з важливих особливостей JavaScript є її широке застосування не лише на стороні клієнта (браузер), а й на стороні сервера завдяки платформі Node.js. Це відкриває можливості для написання повноцінних серверних додатків на JavaScript.

JavaScript є основною мовою для веброзробки, і його невелика вартість входження та широкі можливості роблять його популярним інструментом для програмістів, що працюють у сфері вебтехнологій.

Після завершення аналізу сучасних мов програмування та їхніх інструментів для створення серверних додатків, наша наступна задача полягає в складанні порівняльної таблиці, що відображатиме переваги кожної мови (табл 3.1).

Таблиця 3.1 – Порівняльний аналіз мов програмування

Характеристика	Java	Python	JavaScript
Платформена незалежність	+ Відмінна	± Менш платформено-незалежна	- Залежна від браузера
Ефективність роботи	+ Висока	± Зазвичай висока	± Зазвичай висока
Обробка багатопотоковості	+ Добра	± Зазвичай відмінна	- Менш ефективна
Швидкодія	+ Швидка	± Зазвичай швидка	± Зазвичай швидка
Спільнота	+ Велика	+ Велика	+ Велика
Застосування	+ Застосовується в багатьох сферах, включаючи корпоративну розробку та мобільні додатки	+ Використовується в широкому спектрі, особливо у сферах науки, штучного інтелекту та аналізу даних	- Використовується головним чином для веброзробки та фронтенду
Підтримка технологій	+ Широка	+ Широка	+ Широка

Отже, після уважного порівняльного аналізу трьох мов програмування для розробки серверного модуля, вибір був зроблений на користь Java з кількох ключових причин. Java відома своєю великою стійкістю та надійністю у вирішенні завдань. Програмний код написаний на Java може запускатися на різних операційних системах без значних змін, що спрощує процес розгортання та утримання системи. Крім того, Java має велику спільноту розробників та широкий екосистему інструментів, включаючи потужні фреймворки, такі як

Spring та Jakarta EE, які полегшують розробку та підтримку серверних додатків. Ці фреймворки надають гнучкість та можливості для розширення функціональності модулю API.

3.2 Вибір середовища розробки

Вибір середовища розробки є важливим етапом у створенні програмного забезпечення, оскільки він визначає продуктивність, ефективність та зручність роботи розробника. Правильно обране середовище розробки допомагає зосередитися на створенні функцій, забезпечуючи необхідні інструменти для вирішення завдань швидко та ефективно. Від властивостей середовища залежить зручність написання коду, можливості візуального налагодження програми та взаємодії з іншими розробниками. Активна спільнота користувачів та наявність оновлень та підтримки також є ключовими факторами, які впливають на вибір середовища розробки. У правильно обраному середовищі розробник може працювати ефективніше, швидше вирішувати проблеми та розвивати програмний продукт відповідно до вимог.

Розглянемо сучасні поширені середовища розробки для мови програмування Java. IntelliJ IDEA – це потужне інтегроване середовище розробки, яке використовується для розробки програмного забезпечення на різних мовах програмування, зокрема Java, Kotlin, JavaScript, Python та інших (. Це інструмент, який надає широкий спектр можливостей для підвищення продуктивності розробника.

IntelliJ IDEA пропонує розширений набір функцій автодоповнення коду, підказок та рефакторингу, що сприяє швидкій та точній роботі з кодом. Вона має інтелектуальні інструменти, такі як аналіз коду, автоматичне виправлення помилок та оптимізацію шляхом підказок щодо вдосконалення коду.

Однією з важливих переваг є можливість легкої інтеграції з різними фреймворками та інструментами. Також, її відмінна підтримка віртуальних середовищ розробки дозволяє легко працювати з різними версіями мов та бібліотек.

IntelliJ IDEA надає великий спектр різноманітних плагінів та розширень, які допомагають розширити функціональність та адаптувати інструмент до потреб розробника. Інтеграція з системами контролю версій та зручний інтерфейс також роблять IntelliJ IDEA вибором багатьма розробниками для роботи над проектами різної складності (рис 3.1).

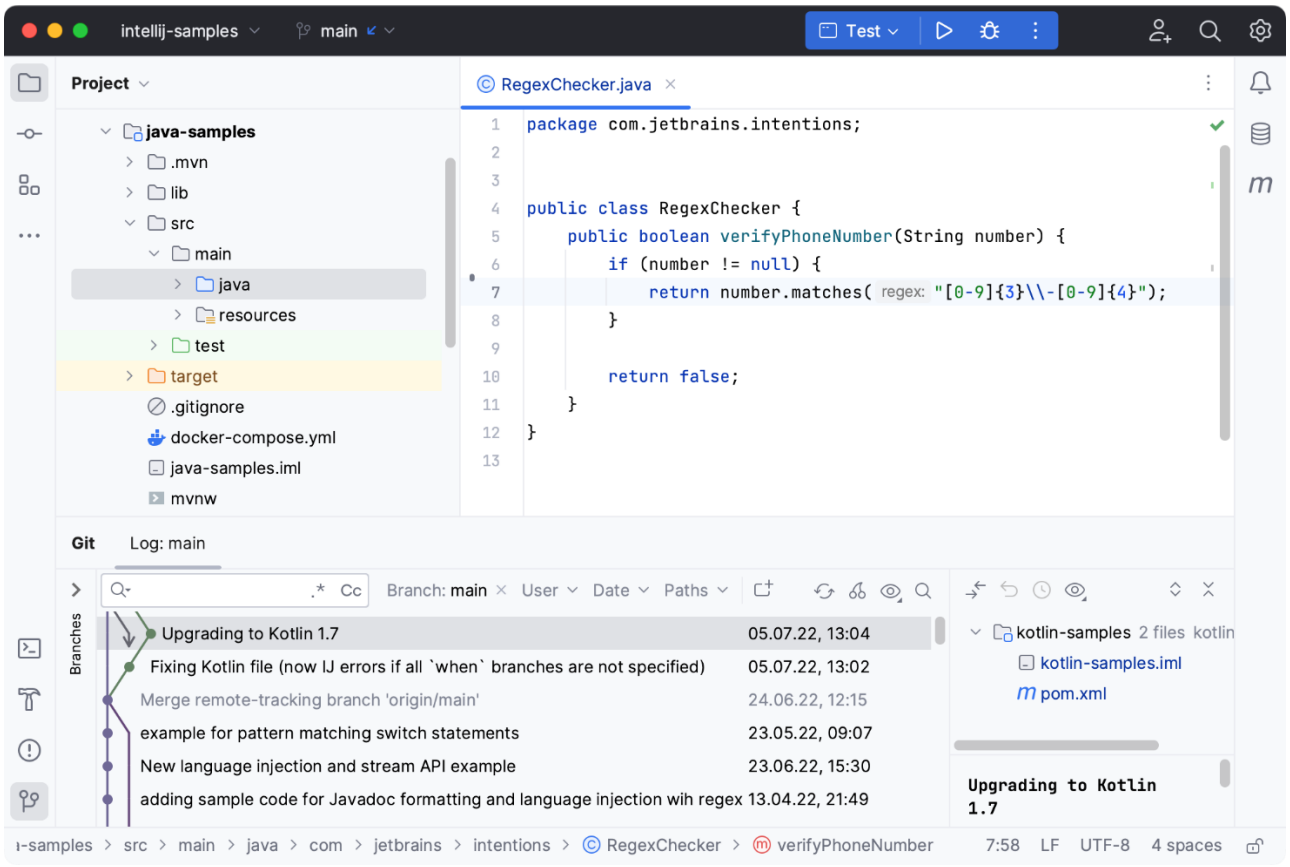

Рисунок 3.1 – Користувацький інтерфейс Intellij IDEA

Eclipse – це відоме інтегроване середовище розробки (IDE), що використовується для розробки програмного забезпечення на різних мовах програмування, включаючи Java, C/C++, Python та багато інших. Це вільно розповсюджувана платформа, яка надає багатий функціонал для розробки, тестування та налагодження програмного забезпечення. Однією з ключових особливостей Eclipse є його гнучкість та розширюваність завдяки великій кількості плагінів та розширень, що дозволяє адаптувати середовище до власних потреб розробки. Це включає в себе різноманітні інструменти для керування

проектами, інтеграцію з системами контролю версій, роботу з різними фреймворками та багато іншого.

Eclipse має потужні інструменти для аналізу коду, автодоповнення та візуального налагодження, що допомагає розробникам писати та оптимізувати код. Воно також забезпечує зручний інтерфейс для роботи з багаторівневими проектами та спрощує взаємодію між різними складовими програмного забезпечення. Eclipse відомий своєю відкритістю та активною спільнотою користувачів, що підтримує його розвиток та постійно вдосконалює його функціонал. Це робить його популярним інструментом для розробки програмного забезпечення різного рівня складності (рис. 3.2).

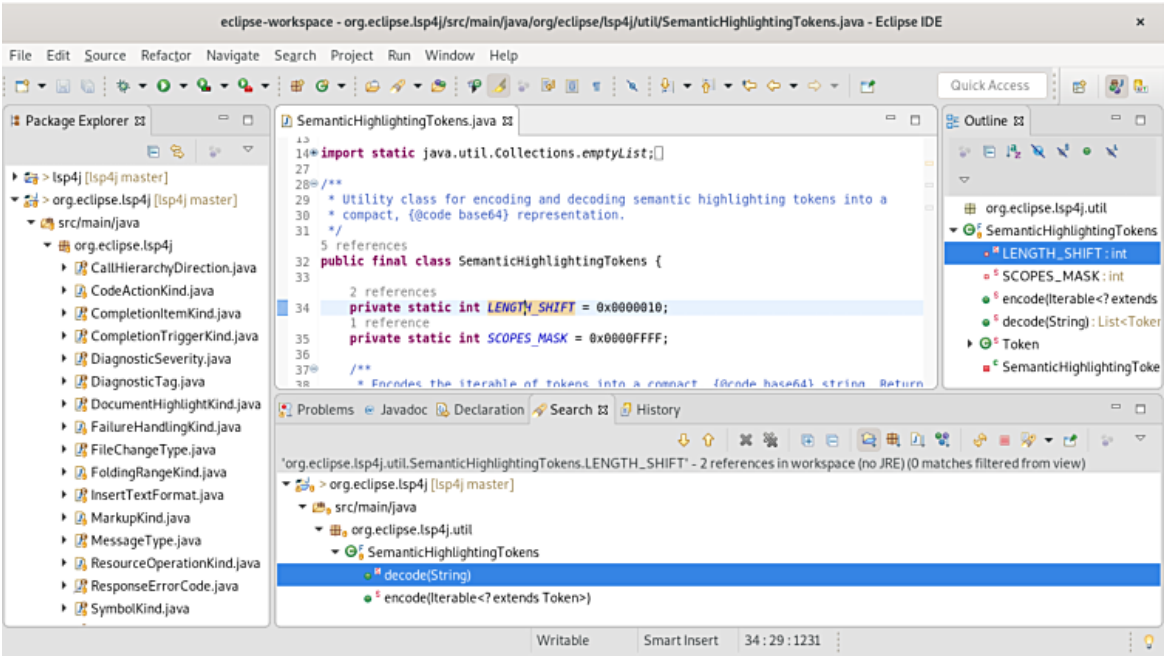

Рисунок 3.2 – Користувацький інтерфейс Eclipse

NetBeans – це інтегроване середовище розробки (IDE), яке використовується для розробки програмного забезпечення на різних мовах програмування, зокрема Java, JavaScript, PHP, Python та інших. Це вільно розповсюджувана платформа, що надає розгалужені можливості для розробки та тестування програмного забезпечення різної складності.

Однією з ключових переваг NetBeans є його потужний та зручний інтерфейс, який надає широкі можливості для роботи з різними проектами. Він

має ряд інструментів для автодоповнення коду, візуального редагування та налагодження програм, що сприяє швидкій та зручній розробці.

NetBeans підтримує широкий спектр мов програмування та фреймворків, а також має багато плагінів та розширень, що робить його гнучким та адаптованим до потреб розробника. Воно також інтегроване з системами контролю версій та надає інструменти для спільної роботи над проектами. Більшість функцій NetBeans є безкоштовними та відкритими для розробників, що дозволяє використовувати його для створення програмного забезпечення різних типів та рівня складності. Активна спільнота користувачів також забезпечує підтримку та розвиток цієї платформи (рис. 3.3).

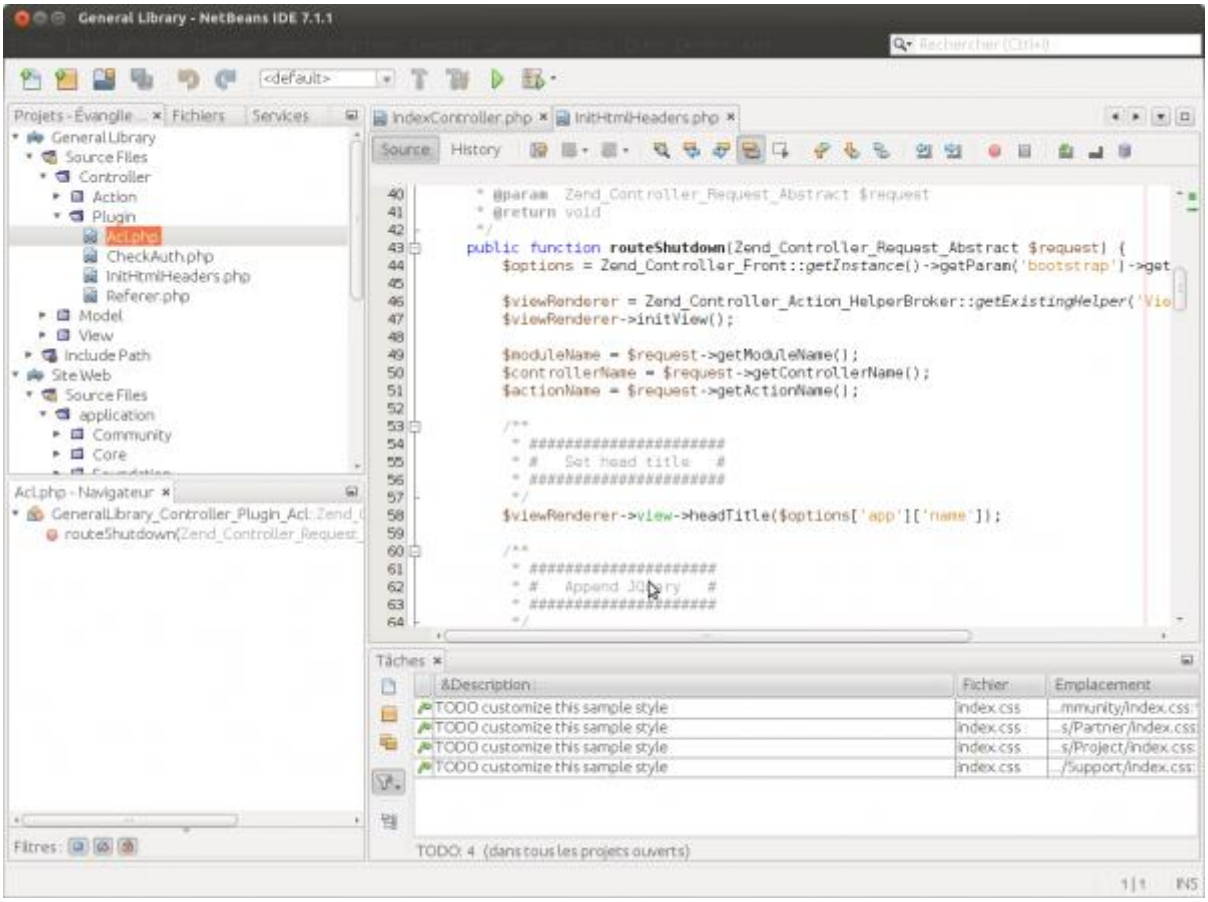

Рисунок 3.3 – Користувацький інтерфейс NetBeans

Оцінюючи різні інтегровані середовища розробки для мови програмування Java, було зроблено вибір на користь IntelliJ IDEA серед інших варіантів. Цей вибір базується на ряді ключових факторів, серед яких потужний набір

інструментів для створення та відлагодження програм, які допомагають не лише в розробці на Java, але й у використанні Spring Framework та його внутрішніх компонентів. IntelliJ IDEA відзначається своєю здатністю до ефективної роботи з Java-проєктами будь-якої складності. Воно надає інтуїтивно зрозумілий та потужний інтерфейс, який сприяє швидкій розробці та оптимізації коду. Крім того, його можливості у взаємодії з фреймворками, зокрема Spring Framework, роблять його перевагою для проектів, де використання цих технологій є ключовим аспектом. Такий вибір базується на потужних можливостях налагодження, широкому наборі функцій під час написання коду та візуального налагодження, що робить його привабливим для команд, які працюють над серверними модулями та API-інтерфейсами, де важлива якість та швидкість розробки.

3.3 Програмна реалізація

Фреймворки відіграють ключову роль у розробці програмного забезпечення, забезпечуючи структурованість, швидкість розробки та стандартизацію процесу. Вони пропонують готові набори інструментів, бібліотек та рішень, які спрощують вирішення типових задач, що дозволяє розробникам зосередитися на доменній логіки. Використання фреймворків сприяє відокремленню деталей реалізації від основного функціоналу, забезпечуючи стабільність, безпеку та масштабованість проектів. Це сприяє прискоренню розробки, полегшує підтримку та забезпечує високу якість програмного продукту завдяки використанню передових практик та стандартів, вбудованих у фреймворки.

Таким чином, згідно обраної мові програмування Java, в якості фундаменту для розробки серверного модуля було використано фреймворк Spring. Spring – це універсальний фреймворк для розробки програмного забезпечення на Java [24]. Він надає гнучкість, стандартизацію та велику кількість модулів для різних завдань у розробці. Основна перевага Spring полягає у створенні розширюваних інтерфейсів для взаємодії між компонентами

програми та забезпеченні ефективного управління конфігурацією. Він підтримує інверсію управління та аспектно-орієнтоване програмування, що сприяє полегшенню розробки, тестуванню та підтримці коду.

Фреймворк Spring пропонує модульну структуру, що дозволяє використовувати тільки ті компоненти, які необхідні для проекту, забезпечуючи таким чином високу гнучкість та розширюваність програмного забезпечення. Це відкриває можливості для використання різноманітних архітектурних підходів, таких як MVC (Model-View-Controller) або IoC (Inversion of Control).

Spring пропонує широкий спектр функціоналу, включаючи підтримку транзакцій, безпеки, роботу з базами даних, роботу з вебзастосунками та багато іншого. Цей фреймворк став стандартом у світі розробки на Java завдяки своїй простоті використання та потужному функціоналу.

За результатами програмної реалізації, розглянемо пакетну структуру розробленого серверного модуля (рис 3.4)

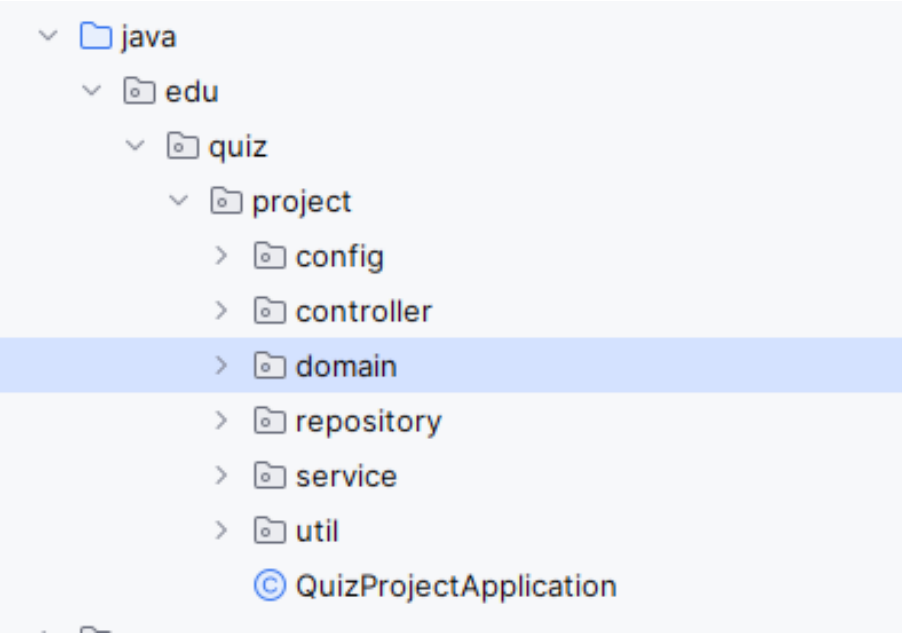

Рисунок 3.4 – Пакетна структура серверного модуля

Згідно багаторівневого архітектурного патерну, який був описаний у першому розділі, Java класи були відсортовані по різним пакетам, згідно їх призначенню. По стандартам мови Java, кореневим пакетом є однойменний

пакет. Далі по ієрархі можна зустріти такі пакети як «config», «controller», «repository» та інші. Розглянемо їх призначення детальніше:

– «config» вміщає класи-конфігуратори Spring Framework, які використуються самим фреймворком для його гнучкого налаштування.

– «controller» вміщає класи-вебконтролери серверного модуля для роботи з вебзапитами.

– «domain» вміщає класи, які представляють основні сутності та моделі додатку. Ці класи відображають структуру даних додатку, представляючи об'єкти, з якими вони взаємодіють.

– «repository» вміщає інтерфейси репозиторіїв для роботи з сховищем даних серверного модуля.

– «service» вміщає класи-сервіси, які виконують основну доменну логіку серверного модуля.

– Каталог «util» містить допоміжні класи, які можуть використовуватися у різних ділянках коду проєкту.

Розглянемо основні важливі програмні реалізації серверного модуля. Важливою складовою серверних модулів є наявність постійного сховища даних, яке буде зберігати інформацію та підтримувати її консистенцію в продовж роботи самого модуля сервера. Згідно другого розділу, в якості постійного сховища даних було обрано реляційну базу даних MySQL.

Щодо реалізації конкретних класів та методі маніпуляції з даними, було застосовано фреймворк Hibernate. Hibernate – це фреймворк для роботи з базами даних у Java-програмах [25]. Його основна мета – спростити роботу з базами даних, надавши можливість розробникам працювати з об'єктами, а не з SQL-запитами безпосередньо. Використовуючи Hibernate, розробники можуть взаємодіяти з базою даних, використовуючи об'єктно-орієнтовані концепції, що значно спрощує розробку.

Цей фреймворк забезпечує різні можливості, такі як відображення об'єктів на таблиці бази даних, використання анотацій для налаштування, управління сесіями, транзакціями та кешуванням даних. Hibernate дозволяє використовувати

мову запитів HQL (Hibernate Query Language), яка абстрагує SQL-запити та дозволяє працювати на вищому рівні абстракції з об'єктами.

Однією з головних переваг Hibernate є його можливість роботи з різними базами даних без необхідності модифікацій у коді програми, що робить його досить гнучким та універсальним для використання в різних проектах. Також він полегшує розробку, оскільки спрощує завдання, пов'язані з доступом до баз даних та управлінням даними.

Інтеграція фреймворку Spring та Hibernate є популярним підходом у розробці програмних застосунків на Java. Обидва ці фреймворки забезпечують спрощений та потужний спосіб розробки, а їх поєднання дозволяє використовувати переваги обох. Розглянемо використання модулю Spring Data JPA в рамка інтеграції з Hibernate. Spring Data JPA є частиною Spring Framework і надає абстракцію для використання Java Persistence API (JPA). Hibernate, у свою чергу, є конкретною реалізацією JPA. Інтеграція цих двох технологій дозволяє спростити роботу з базою даних у Java-додатках.

Інтеграція Spring Data JPA з Hibernate полягає в тому, що Spring Data JPA використовує Hibernate як основний постачальник реалізації JPA для взаємодії з базою даних. Ця інтеграція дозволяє працювати на вищому рівні абстракції, використовуючи методи та анотації Spring Data JPA, тоді як реальна робота з базою даних (перетворення, операції CRUD і т.д.) виконується Hibernate неявно. Це спрощує розробку, зменшує кількість коду який повторюється та полегшує управління базою даних у Java-додатках.

Але, розробник все ще повинен заповнити файл конфігурацій програми відповідними значеннями, аби Hibernate зміг правильно піджинатися до бази даних. На рисунку 3.5 зображено вміст файлу конфігурації з необхідними значеннями.

```
spring.datasource.url=jdbc:mysql://localhost:3306/quizy?password=root&user=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=create
```

Рисунок 3.5 – Файл конфігурацію проєкту

Розглянемо конкретний приклад сутності-репозиторію, який використовується для маніпуляціями з даними сховища. На рисунку 3.6 зображено код інтерфейсу «UserRepository».

```
public interface UserRepository extends JpaRepository<User, Long> {

    @Query(value = "select u from User u " +
            "join fetch u.roles " +
            "where u.email = ?1 ")
    Optional<User> getUserByEmailEagerly(String email);

    @Query(value = "select u from User u " +
            "join fetch u.roles " +
            "where u.id = ?1 ")
    User getByIdEagerly(long userId);
}
```

Рисунок 3.6 – Код інтерфейсу «UserRepository»

Інтерфейс «UserRepository» наслідує інший інтерфейс «JpaRepository». Цього вимагає сам фреймворк, аби він мав можливість згенерувати конкретний екземпляр даного інтерфейсу в процесі запуску програми. Таким чином, існуватиме конкретна реалізація абстракції «UserRepository» з основними методами для роботи з даними типу «User», так звані CRUD (Create, Read, Update, Delete) операції та інші.

У випадку з репозиторієм «UserRepository», реалізація, яка надається модулем Sping Data JPA по замовчуванню, є недостатньою для вирішення

поставлених задач. Необхідно забезпечити пошук сутностей «User» за їх електронною адресою в базі даних. В такому випадку існує можливість використати допоміжні рішення: методи-запити JPA або спеціальна Java анотація «@Query». У випадку з «UserRepository», було застосовано спеціальну анотацію «@Query», яка дозволяє визначити HQL-скрипт який буде використано та застосовано для реалізації створеного методу при генерації екземпляру інтерфейсу «UserRepository». Щодо методів-запитів JPA, це механізм, що дозволяє створювати запити до бази даних за допомогою імен методів в інтерфейсах репозиторіїв в Spring Data JPA. Цей підхід дозволяє генерувати SQL-запити до бази даних на основі імен методів, зменшуючи кількість ручного написання SQL-запитів. Такий підхід також було використано у розробці програмної реалізації репозиторіїв серверного модуля (рис 3.7).

```
public interface QuizUserEntryRepository extends JpaRepository<QuizEntry, Long> {

    Page<QuizEntry> getByUserId(Long userId, Pageable pageRequest);
    Page<QuizEntry> getByQuizQuizId(Long quizId, Pageable pageRequest);

}
```

Рисунок 3.7 – Інтерфейс «QuizUserEntryRepository»

Розглянемо реалізацію наступної важливої складової серверного модуля – авторизація та автентифікація вебзапитів. Автентифікація та авторизація – це ключові аспекти забезпечення безпеки в серверних модулях. Автентифікація визначає особу користувача (підтвердження, що він той, за кого себе видає), тоді як авторизація визначає права користувача на доступ до певних ресурсів або функцій.

У серверних модулях це важливо для забезпечення захисту конфіденційності, запобігання несанкціонованому доступу до даних. Це дозволяє керувати правами доступу, обмежувати функціонал для певних груп користувачів та забезпечувати захист від несанкціонованого використання ресурсів.

Організація процесу автентифікації та авторизації у серверних модулях передбачає використання різних механізмів, таких як JWT. JWT, або JSON Web Token, це спосіб представлення інформації між двома сторонами у вигляді JSON об'єкта. Використовується для передачі даних про автентифікацію та авторизацію між клієнтом і сервером [26].

JWT складається з трьох частин: заголовка, корисної навантаження та підпису. Заголовок містить тип та алгоритм шифрування, корисне навантаження може містити будь-яку корисну інформацію, яка повинна бути передана, наприклад, дані про користувача або додаткові метадані, а підпис дозволяє перевірити цілісність даних та їх автентичність.

JWTs можуть використовуватися для автентифікації користувачів, коли вони увійшли в систему, і для обміну інформацією між сторонами, не зберігаючи стану на сервері. Це дозволяє підтверджувати ідентичність користувачів і дозволяти їм виконувати певні дії або мати доступ до певних ресурсів без необхідності постійного надсилання даних для авторизації на сервері.

Для реалізації генерації JWT було створено клас «JwtUtil», який відповідає за поширені операції над токенами, такі як генерація, перевірка, дешифрування, тощо. На рисунку 3.8 зображено реалізацію методі генерації JWT.

```
public String generateToken(UserDetails userDetails){
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String,Object> claims, String subject){
    return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 3_600_000))
            .signWith(SignatureAlgorithm.HS256, JWT_SECRET).compact();
}
```

Рисунок 3.8 – Методи генерації JWT

Для маніпуляцій з JWT використовується бібліотека «jsonwebtoken». Згідно прикладу реалізації, основним компонентом токену є суб’єкт, тобто користувач системи, для якого генерується токен. Метод генерації також визначає такі в тілі токена такі атрибути як час видачі та час до закінчення терміну дії. Дані атрибути дозволяють сприяти ротації JWT аби зробити його тимчасовим та зменшити вразливість системи. Генерація підпису JWT виконується з використання алгоритму шифрування HS256. Алгоритм HS256 використовується для підпису та перевірки цілісності JWT за допомогою хеш-функції HMAC (Hash-based Message Authentication Code) з алгоритмом хешування SHA-256. Він використовується для генерації підпису та перевірки автентичності даних, які передаються у JWT, забезпечуючи їх цілісність та недоступність для зміни.

Для забезпечення функціонування алгоритму HS256 використовується секретний ключ, який відомий тільки серверному модулю і не доступний для інших суб’єктів авторизації та автентифікації. Таким чином серверний модуль має можливість перевірити правильність підпису JWT та впевнитися, що він дійсно його згенерував та запобігти зловмисним маніпуляція з JWT.

Для забезпечення безпосередньо контролю над запитами до серверного модуля було використано спеціальний модуль Spring – Spring Security. Spring Security – це модуль безпеки для додатків на основі фреймворку Spring, який забезпечує автентифікацію, авторизацію, захист ресурсів та управління сеансами користувачів. Він дозволяє забезпечити безпеку веб-додатків шляхом встановлення прав доступу до функцій і ресурсів, обмеження доступу до захищених областей та управління ідентифікацією користувачів.

Spring Security має гнучку конфігурацію та можливість налаштування прав доступу для різних користувацьких ролей. Він надає різноманітні методи автентифікації, включаючи базову автентифікацію, автентифікацію з використанням JWT токенів, формування інтерфейсів для логіну та багато іншого. Він надає різноманітні засоби для захисту від атак, таких як CSRF, SQL-ін'єкції та інших видів атак на безпеку.

Основні можливості Spring Security включають у себе налаштування прав доступу, контроль безпеки HTTP-запитів, можливість реалізації різних видів автентифікації та інтеграцію з існуючими системами автентифікації. Він є потужним інструментом для створення безпечних додатків та забезпечення їх захисту від різних загроз. На рисунку 3.9 зображено метод для конфігурації безпекових налаштувань серверного модуля.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests() ExpressionInterceptUrlRegistry
            .anyRequest().authenticated() //restrict all others resources
            .and() HttpSecurity
                .csrf() CsrfConfigurer<HttpSecurity>
                .disable() HttpSecurity
            .headers() HeadersConfigurer<HttpSecurity>
            .frameOptions() FrameOptionsConfig
            .sameOrigin() HeadersConfigurer<HttpSecurity>
            .and() HttpSecurity
              .sessionManagement() SessionManagementConfigurer<HttpSecurity>
              .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
    http.headers() HeadersConfigurer<HttpSecurity>
            .frameOptions() FrameOptionsConfig
            .sameOrigin();
}
```

Рисунок 3.9 – Налаштування Spring Security

Згідно реалізованих налаштувань, було встановлено, що обслуговування клієнтів серверного модуля буде здійснюватися у безстановому виді, тобто сервер не буде зберігати сеанси користувачів у оперативні пам’яті середовища. Дану особливість забезпечує авторизація та автентифікація на основі JWT, адже сам токен безпосередньо буде містити всю необхідну інформацію та передаватися при кожному запиті до серверного модуля.

Також було реалізовано захист всіх вразливих ресурсів серверного модуля, що вимагатиме в клієнта наявності JWTв разі звернення до подібних ресурсів. Інструменти Spring Security дозволяють оголошувати та залучати власні так звані

вебфільтри. які обробляють HTTP-запити та відповіді. Вебфільтри дозволяють впроваджувати логіку, яка виконується до обробки HTTP запиту та після нього. Вебфільтри можуть модифікувати або перехоплювати дані запиту та відповіді, обробляти автентифікацію, авторизацію, логування, стискання даних, перекодування символів, фільтрацію вмісту та інші корисні операції. В контексті серверного модуля, було розроблено вебфільтр для авторизації та автентифікації користувачів (рис 3.10).

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
    String authorizationHeader = request.getHeader( s: "Authorization");

    String email = null;
    String jwt = null;

    if(authorizationHeader != null && authorizationHeader.startsWith("Bearer ")){
        jwt = authorizationHeader.substring( beginIndex: 7);
        email = jwtUtil.extractEmail(jwt);
    }

    if (email != null && SecurityContextHolder.getContext().getAuthentication() == null){
        UserDetails userDetails = this.userDetailsService.loadUserByUsername(email);
        if(jwtUtil.validateToken(jwt,userDetails)){
            UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =
                    new UsernamePasswordAuthenticationToken(userDetails, credentials: null,userDetails.getAuthorities());
            usernamePasswordAuthenticationToken
                    .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
        }
    }
    filterChain.doFilter(request,response);
}
```

Рисунок 3.10 – Вебфільтр авторизації та автентифікації

Даний фільтр є частиною обробки запитів у серверному модулі. Він використовується для перевірки та обробки токенів, що надходять у заголовку «Authorization» HTTP запиту. Фільтр перевіряє наявність та валідність JWT-токену у заголовку запиту. Якщо токен існує та має відповідний формат «Bearer», він зчитується для подальшої обробки. Фільтр перевіряє, чи не автентифікований вже користувач, використовуючи отриману адресу електронної пошти. Якщо користувач не автентифікований, фільтр завантажує деталі користувача через клас-сервіс користувачів та перевіряє валідність отриманого токену. Після

успішної перевірки, фільтр створює об'єкт з даними користувача, здійснює його автентифікацію та встановлює контекст автентифікації у «SecurityContextHolder». Усі наступні запити обробляються в звичайному порядку, пройшовши через інші фільтри або обробники, які можуть вимагати автентифікації для доступу до захищених ресурсів.

Розглянемо програмну реалізацію прикладного програмного інтерфейсу серверного модуля. Модуль Spring MVC надає спеціальний інструменти для створення та налаштування вебресурсів серверного модуля – контролери. Контролери в Spring – це класи, які обробляють HTTP запити. Вони відповідають за обробку запитів, виклик методів відповідно до отриманих URL та параметрів, і повертають відповідь клієнтам. У Spring контролери позначаються спеціальними анотаціями, такими як «@RestController», «@Controller» або «@RequestMapping». На рисунку 3.11 зображено реалізацію контролеру «LoginController» для операцій авторизації користувачів.

```
@RestController
@CrossOrigin(origins = "*")
@AllArgsConstructor
@Slf4j
public class LoginController {

    private final UserService userService;

    @PostMapping(value = "/login")
    public ResponseEntity<?> createAuthenticationToken(@RequestBody LoginRequest loginRequest) throws Exception {
        JwtDto body = userService.loginUser(loginRequest);
        return ResponseEntity.ok(body);
    }
}
```

Рисунок 3.11 – Контролер «LoginController»

Клас «LoginController» відмічений анотацією «@RestController», що повідомляє Spring про те, що він є контролером. Даний контролер містить єдиний метод, який помічений анотацією «@PostMapping» з параметром «/login». Цього є достатньо аби Spring створив ресурс серверного модуля зі шляхом до нього «/login». Для визначення критичних властивостей ресурсу Spring також буде використовувати сигнатуру методу. Наприклад параметр функції «loginRequest»

буде очікуватися фреймворком у тілі HTTP запиту. У випадку якщо тіло запиту буде дійсно заповнений JSON об’єктом який відповідає структурі Java об’єкту «LoginRequest», на основі такого тіла буде створено екземпляр об’єкту «LoginRequest» поміщено у аргументи методу і може бути використаний в тілі методу. Дана особливість дозволяє уникнути написання коду, який повторюється, пришвидшити розробку та покращити обробку помилок.

Також тип повернення методу, який вказаний в сигнатурі буде вказувати Spring яким повинна бути HTTP відповідь для клієнта, а саме значення заголовку «Content-Type». Таким чином відповідальність за формування деяких заголовків бере на себе сам фреймворк, що знову таки пришвидшує розробку і робить її більш декларативною та зручною. Виконання безпосередньо роботи по авторизації користувача здійснюється через виклик відповідного сервісу «UserService».

Розглянемо роботу методу для збереження та обчислення результатів тестування. На рисунку 3.12 зображено реалізацію методу для зарахування результатів тестування.

```
@Transactional
public QuizUserEntryDto submitTestEntryForUser(Long submittedTestId, List<SubmittedQuestionDto> submittedQuestions, User user) {
    Quiz originalQuiz = Optional.ofNullable(quizService.getQuizById(submittedTestId))
            .orElseThrow(() -> new RuntimeException("Submitted test not found!"));
    int score = compareResults(originalQuiz, new HashSet<>(submittedQuestions));
    QuizEntry quizEntry = QuizEntry.builder()
            .user(user)
            .quiz(originalQuiz)
            .rightAnswers(score)
            .totalQuestions(originalQuiz.getQuestions().size())
            .entryDate(new Date())
            .build();

    return testUserEntryDtoMapper.entityToDto(quizUserEntryRepository.save(quizEntry));
}
```

Рисунок 3.12 – Метод для збереження результатів тестування

Варто зауважити, що цей метод буде виконано транзакційно завдяки використанню однойменної анотації «@Transaclional». Анотація «@Transactional» у Spring вказує на те, що метод або клас, до якого застосована ця анотація, повинні виконувати свої операції всередині транзакції бази даних.

Ця анотація дозволяє управляти транзакціями у Spring-додатках без прямого втручання у код управління транзакціями. Коли метод або клас позначений анотацією «@Transactional», Spring буде створювати транзакцію перед викликом методу та завершувати її після його виконання. Якщо в методі виникає помилка, транзакція буде відкочена, тобто всі зміни, внесені в базу даних протягом цієї транзакції, будуть скасовані. Це дозволяє впевнитися, що операції бази даних відбуваються атомарно, тобто або вони виконуються повністю, або нічого не виконується. Це корисний підхід для уникнення некоректного стану бази даних та забезпечення цілісності даних.

Розглянемо роботу безпосередньо згаданого методу. Цей метод «submitTestEntryForUser» призначений для обробки та збереження результатів тестування користувача. Він перевіряє наявність поданого тесту у базі даних. Потім порівнюється подані відповіді з правильними відповідями для підрахунку оцінки тестування. Далі відбувається порівняння поданих відповідей із правильними відповідями для кожного питання тесту. Використовуються стратегії оцінювання для кожного типу питання. Враховуючи це, метод обчислює та повертає загальний бал користувача за тест. Далі результати тестування зберігаються у базі даних у вигляді запису «QuizEntry», який містить інформацію про користувача, тест та його результат.

Після збереження результатів тестування метод повертає об'єкт «QuizUserEntryDto», який містить збережений запис «QuizEntry» у вигляді об'єкта для використання в подальших операціях або для відображення результатів користувачеві.

З попереднього приклад було згадано два класи «QuizUserEntryDto», «QuizEntry». Хоч вони мають схожу назву, їх призначення різниться. Об’єкти-DTO (Data Transfer Object) – це об'єкти, які використовується для передачі даних між прошарками програми або між програмами в мережі. Вони є спрощеними моделями даних, які передаються для виконання конкретного завдання або обміну інформацією між різними частинами програми. DTO зазвичай використовується для забезпечення ізольованої та чіткої передачі даних між

компонентами системи без прямого доступу до основних об'єктів моделі. Вони можуть містити лише підмножину властивостей основної моделі або зовсім відрізнятися за структурою від оригінальних об'єктів.

Зазвичай об’єкти DTO створюються для певних операцій чи взаємодій, де потрібна чітка структура обміну даними. Вони можуть використовуватися для передачі даних через мережу, в обробці запитів або в зворотному зв'язку з клієнтами. Важливо, щоб DTO були легкими, містити необхідну інформацію та були простими у використанні. Для прикладу, клас «User» містить поле, яке зберігає значення паролю користувача у системі. В свою чергу клас «UserDto» не містить поля для паролю користувача і може безпечно використовуватися обміну даним з клієнтами серверного модуля.

Розглянемо програмну реалізацію модулю аналізу фотоданих. Для даної реалізації було залучено модуль Spring Cloud GCP Vision. Цей модуль надає інтеграцію сервісів розпізнавання зображень, зокрема Vision API, в рамках інфраструктури Spring Cloud GCP. Модуль дозволяє легко і зручно використовувати можливості Vision API для обробки зображень, використовуючи стандартні компоненти Spring та інтерфейси Vision API. Використання Spring Cloud GCP Vision дозволяє здійснювати виклики до Vision API, використовуючи різноманітні Spring-компоненти та інструменти, що надає Spring Cloud GCP. Це робить процес роботи з Vision API більш простим і зручним для інтеграції з існуючими додатками, побудованими на платформі Spring.

В цілому, модуль дозволяє виконувати різноманітні завдання розпізнавання зображень, такі як визначення об'єктів, розпізнавання тексту, аналіз емоцій та інше. Цей модуль допомагає легко інтегрувати функції Vision API у Spring-застосунку для обробки та аналізу зображень з використанням інфраструктури Google Cloud Platform.

На рисунку 3.13 зображено розроблений програмний код для виконання налізу зображення з допомогою Spring Cloud GCP Vision та Vision API.

```
public List<AnnotateImageResponse> analyzeImage(String imagePath) {
    try (ImageAnnotatorClient vision = ImageAnnotatorClient.create()) {
        ByteString imgBytes = ByteString.copyFrom(fileService.getFileByPath(imagePath));

        Image image = Image.newBuilder().setContent(imgBytes).build();
        List<AnnotateImageRequest> requests = new ArrayList<>();
        AnnotateImageRequest request = AnnotateImageRequest.newBuilder().setImage(image).build();
        requests.add(request);

        return vision.batchAnnotateImages(requests).getResponsesList();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Рисунок 3.13 – Реалізація аналізу зображень

Наведений метод «analyzeImage» у використанні Vision API для аналізу зображень створює клієнта Vision API за допомогою спеціального фабричного методу. Він отримує зображення за вказаним шляхом з аргументів функції, конвертуючи його у формат масиву байтів. Далі створюється об'єкт типу «Image» на основі отриманого масиву байтів та обгортається новостворенним об’єкт типу «AnnotateImageRequest», який відповідає за представлення запиту до API. Створений об’єкт запиту додається до списку «requests», що передається до методу «batchAnnotateImages» для пакетної обробки аналізу всіх зображень в списку. Результатом методу «batchAnnotateImages» є список, який містить результати аналізу кожного зображення у вхідному списку типу «AnnotateImageResponse». Об'єкти типу AnnotateImageResponse надають інформацію про результати аналізу зображення, яке було оброблене за допомогою Vision API. Цей об'єкт містить різні поля та методи, які дозволяють отримати різноманітні відомості про зображення, наприклад:

– інформацію про розпізнані об'єкти на зображенні;
– різні характеристики зображення, такі як кольори, розміри, аспекти, дозвіл та інше;
– інші розпізнані елементи, такі як текст або обличчя.

Під час виконання методу «analyzeImage», може виникнути виключення пов’язане з операціями з файлами, якщо відбувається помилка під час опрацювання файлів або при отриманні доступу до зображення. Це може виникнути, наприклад, у випадку, коли файл не знайдено або неможливо його прочитати.

Для роботи зі сховищем фотоданих було розроблено Java інтерфейс «FileService» який описує основні методи маніпуляції з фотоданими у сховищі (рис 3.14).

```
public interface FileService {

    byte[] getFileByPath(String filePath);

    List<String> getFilePathsInDirectory(String webcamPhoto);

    String saveQuestionImage(MultipartFile image, Question targetQuestion);

    String saveQuestionImages(MultipartFile[] photos, Question targetQuestion);

    String saveQuizImage(MultipartFile file, Quiz targetQuiz);

    String saveUserWebCamPhotos(MultipartFile[] photos, Quiz targetQuiz, User targetUser);

    void removeFileByPath(String filePath);

}
```

Рисунок 3.14 – Інтерфейс «FileStorage»

Такий підхід дозволяє абстрагувати роботу з фотоданими від конкретної реалізації, що спрощує його використання та забезпечує більшу гнучкість серверного модуля при роботі з сховищем фотоданих. В рамках розробки серверного модуля, було створено одну реалізацію наведеного інтерфейсу – «LocalFileService». Дана реалізація відповідає за збереження файлів у файлову систему середовища серверного модуля. На рисунку 3.15 зображено приклад конкретної кодової реалізації методу збереження зображення фотоконтролю.

```
public String saveUserWebCamPhotos(MultipartFile[] photos, Quiz targetQuiz, User targetUser) {
    return saveFiles(photos, targetQuiz,
            Paths.get(ROOT_FILE_STORAGE_PATH,
                    QUIZ_FILE_STORAGE_PATH,
                    targetQuiz.getQuizId().toString(),
                    USER_PHOTO_CONTROL_PATH,
                    targetUser.getId().toString(),
                    IMAGE_FOLDER_NAME
            ));
}
```

Рисунок 3.15 – Реалізація збереження зображення у сховище

Основна функція цього методу полягає в збереженні фотографій у певному місці файлової системи. Він отримує масив фотографій користувача, цільову об’єкт тесту та цільовий об'єкт користувача, що виконав це тестування. Метод викликає внутрішній приватний метод, який реалізує фактичну логіку збереження файлів. Створюється шлях для збереження фотографій у відповідності з ієрархією:

– коренева директорія для збереження файлів;
– директорія для збереження файлів тесту;
– директорія з ідентифікатором конкретного тесту;
– директорія для збереження фотографій фотоконтролю користувача;
– директорія з ідентифікатором результатів тестування користувача, який виконував тест;
– збережене зображення.

Отриманий шлях передається методу «saveFiles», який виконує збереження фотографій у вказаній директорії. Після успішного збереження метод повертає шлях до збережених фотографій, відносно кореневої директорії збереження файлів.

Таким чином було розглянути основні та важливі програмні реалізації модулів та компонентів серверного модуля дистанційного тестування з використанням технології Google Cloud Vision.

3.4 Висновки

В даному розділі було проведено порівняльний аналіз актуальних мов програмування. За результатом аналізу було обрано мову програмування Java, як таку що найкраще підходять для вирішення задачі розробки серверного модуля.

Було розглянуто сучасні інтегровані середовища розробки для мови програмування Java, їх описано користь їх застосування та їх особливості. В результаті аналізу, для розробки було обрано інтегроване середовище розробки Intellij IDEA.

Було детально проаналізовано програмну реалізацію основний модулів та компонентів серверного модуля дистанційної тестувальної системи. Наведено кодову реалізацію таких механізмів як робота з базою даних та даними системи, генерація JWT токенів та забезпечення авторизації та автентифікації HTTP запитів, створення прикладного програмного інтерфейсу серверного модуля, обробка та збереження результатів тестування, аналіз фотоданих та роботу зі сховищем фотоданих.

4 ТЕСТУВАННЯ СЕРВЕРНОГО МОДУЛЯ

4.1 Тестування серверного модуля

Тестування програмного забезпечення – це важливий процес, який використовується для оцінки функціональності програмного забезпечення, щоб переконатися, що воно відповідає заданим вимогам і працює належним чином. Це передбачає виконання програми або програми з метою пошуку помилок, прогалин або відсутніх вимог [27].

Тестування програмного забезпечення є важливим з наступних причин:

– виявлення дефектів: тестування допомагає виявити помилки, збої та дефекти програмного забезпечення. Пошук і усунення цих проблем на ранній стадії розробки заощаджує час і ресурси;

– гарантія якості: тестування гарантує, що програмне забезпечення відповідає очікуваним стандартам якості. Це допомагає надавати кінцевим споживачам надійний і високоякісний продукт;

– задоволеність клієнтів: добре перевірений програмний продукт працює краще, має менше проблем і забезпечує зручну роботу користувача. Це підвищує задоволеність клієнтів і довіру до продукту;

– економічна ефективність: виявлення та виправлення дефектів на ранніх стадіях розробки є набагато більш рентабельним, ніж їх усунення після розгортання програмного забезпечення. Тестування знижує загальну вартість розробки програмного забезпечення;

– зменшення ризиків: тестування допомагає виявити та зменшити ризики, пов’язані зі збоями програмного забезпечення, вразливими місцями безпеки або проблемами продуктивності, перш ніж програмне забезпечення буде запущено;

– відповідність і стандарти: у певних галузях існують спеціальні стандарти та правила, яким має відповідати програмне забезпечення. Тестування гарантує, що програмне забезпечення відповідає цим вимогам.

Існують різні типи тестування, зокрема:

– Функціональне тестування: перевіряє, чи кожна функція програми працює відповідно до вимог.

– Нефункціональне тестування: включає тести на продуктивність, безпеку, зручність використання, надійність та інші аспекти, не пов’язані з конкретними функціями.

– Автоматизоване тестування: передбачає використання інструментів і сценаріїв для автоматизації тестових випадків, що робить процес тестування швидшим і ефективнішим.

Застосовуючи різні методології та стратегії тестування, розробники можуть створювати програмне забезпечення, яке є надійнішим і краще відповідає очікуванням користувачів.

Тестування API (прикладного програмного інтерфейсу) передбачає перевірку його функціональності, надійності, продуктивності, безпеки та відповідності специфікаціям. Основні аспекти тестування API полягають у наступному:

– функціональне тестування: це гарантує, що API працює належним чином. Тестові випадки призначені для перевірки кінцевих точок, форматів запитів і відповідей, кодів статусу HTTP, обробки помилок і цілісності даних.

– перевірка вхідних і вихідних даних. Тестування передбачає надсилання різних вхідних даних до API та перевірку вихідних даних, щоб переконатися, що вони відповідають очікуваним результатам. Це включає тестування різних типів даних, граничних випадків і граничних значень.

– тестування продуктивності: оцінює роботу API за різних умов навантаження. Це включає тестування часу відповіді, пропускної здатності, затримки та використання ресурсів, щоб переконатися, що API може ефективно обробляти очікуваний обсяг запитів.

– тестування безпеки: гарантує, що API захищено від потенційних вразливостей, таких як SQL-ін’єкції, міжсайтовий скриптинг (XSS), проблеми з

автентифікацією та порушення даних. Він передбачає перевірку механізмів авторизації, шифрування та контролю доступу.

– тестування зручності використання: фокусується на простоті використання API, включаючи його документацію, повідомлення про помилки та загальну взаємодію з користувачем для розробників, які будуть з ним інтегруватися.

– тестування інтеграції: перевіряє, чи API правильно взаємодіє з іншими компонентами програмного забезпечення або службами сторонніх розробників, до яких він підключається. Це передбачає тестування різних сценаріїв взаємодії API з іншими системами.

– автоматизоване тестування: використовує автоматизовані інструменти та інфраструктури для виконання повторюваних і складних тестів, що допомагає скоротити час і зусилля на тестування.

Тестування API має вирішальне значення, оскільки вони діють як посередники між різними програмними системами, і будь-які проблеми чи вразливості в API можуть вплинути на загальну функціональність, безпеку та продуктивність систем, які покладаються на нього. Комплексна та добре виконана стратегія тестування API гарантує правильну та надійну роботу API, що забезпечує кращу взаємодію та плавну інтеграцію між різними програмними додатками.

Для проведення тестування було використано програму Postman. Це популярний інструмент для розробки, тестування та взаємодії з API. Він надає зручний та інтуїтивно зрозумілий інтерфейс для створення, надсилання та прийому HTTP-запитів та відповідей. Postman дозволяє розробникам, тестувальникам та іншим зацікавленим особам легко взаємодіяти з різними видами API: REST, SOAP, іншими веб-сервісами. Інтерфейс Postman складається з різних вкладок, таких як "Запити" для створення, редагування та надсилання запитів на сервер, "Колекції" для організації запитів у групи, "Змінні" для зберігання значень та інші. За допомогою Postman можна відправляти GET, POST, PUT, DELETE та інші типи запитів, налаштовувати заголовки, передавати

параметри, виконувати тести на валідність відповідей сервера. Також Postman дозволяє створювати та організовувати автоматизовані тестові набори, що спрощує тестування API та автоматизує процеси тестування.

Отже розглянемо процес тестування розробленого серверного модуля дистанційного тестування з використанням технології Google Cloud Vision. Для початку перевіримо роботу ресурсу для автентифікації користувачів системи. На рисунку 4.1 зображено результат виклику ресурсу «POST /login».

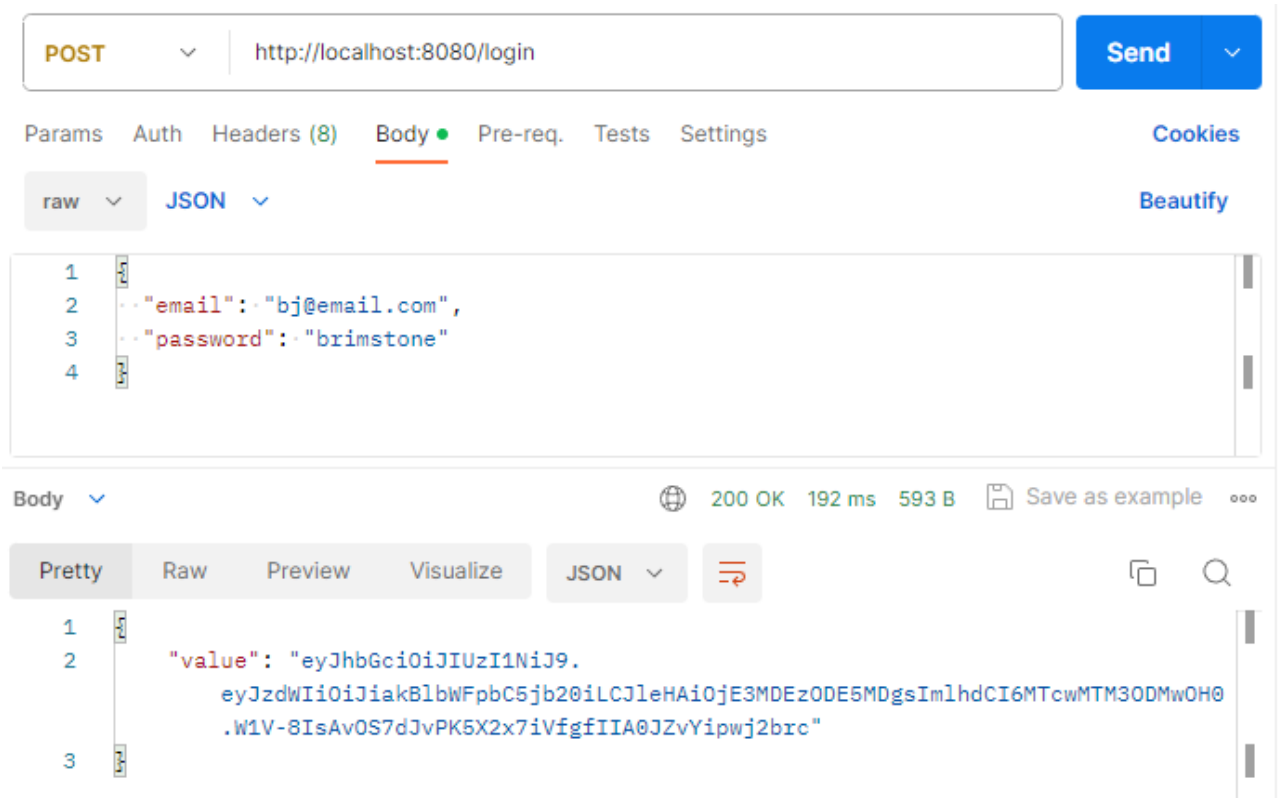

Рисунок 4.1 – результат виклику ресурсу «POST /login»

Для виконання HTTP запиту на згаданий ресурс необхідно було заповнити шлях до нього та сформувати тіло запиту в форматі JSON з відповідними обліковими даними, які на момент тестування вже існували в базі даних. За результатом тестування серверний модуль здійснив автентифікацію за наданими обліковими даними та сформував HTTP відповідь з кодом 200. Тіло відповіді містить JSON об’єкт з єдиним полем – згенерований JWT для користувача.

На рисунку 4.2 зображено результат декодування отриманого JWT.

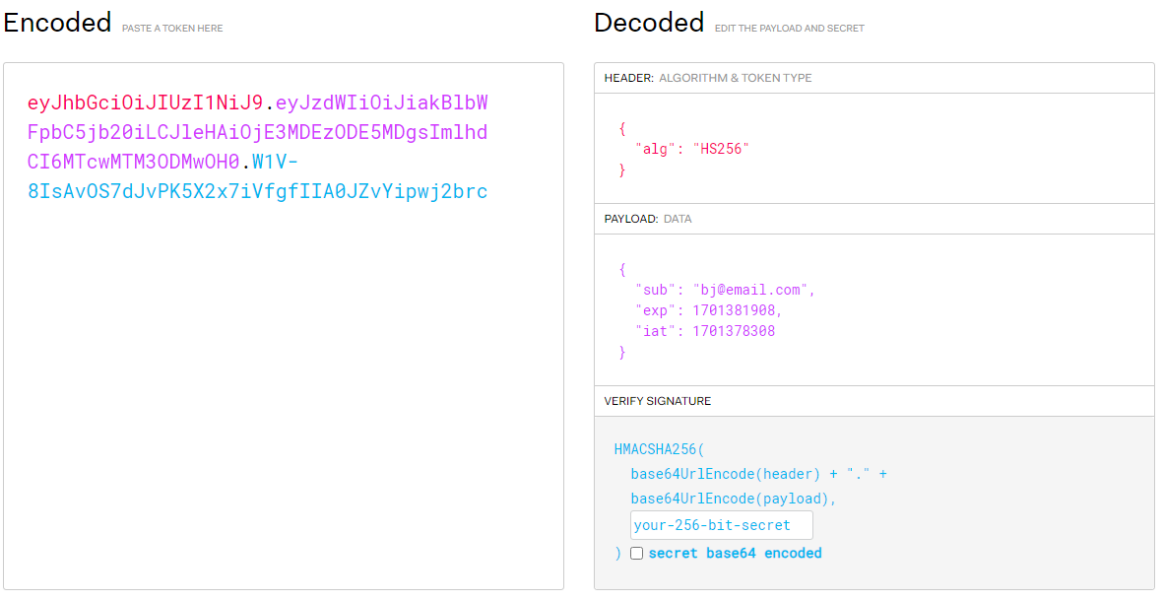

Рисунок 4.2 – Декодований JWT

Таким чином можна впевнитись, що JWT правильний та він був сформований для відповідного користувача. Отриманий JWT буде використовуватися для автентифікації при кожному наступному запиті до серверного модуля.

Розглянемо роботу ресурсу для створення нових сутностей тесту в системі (рис 4.3).

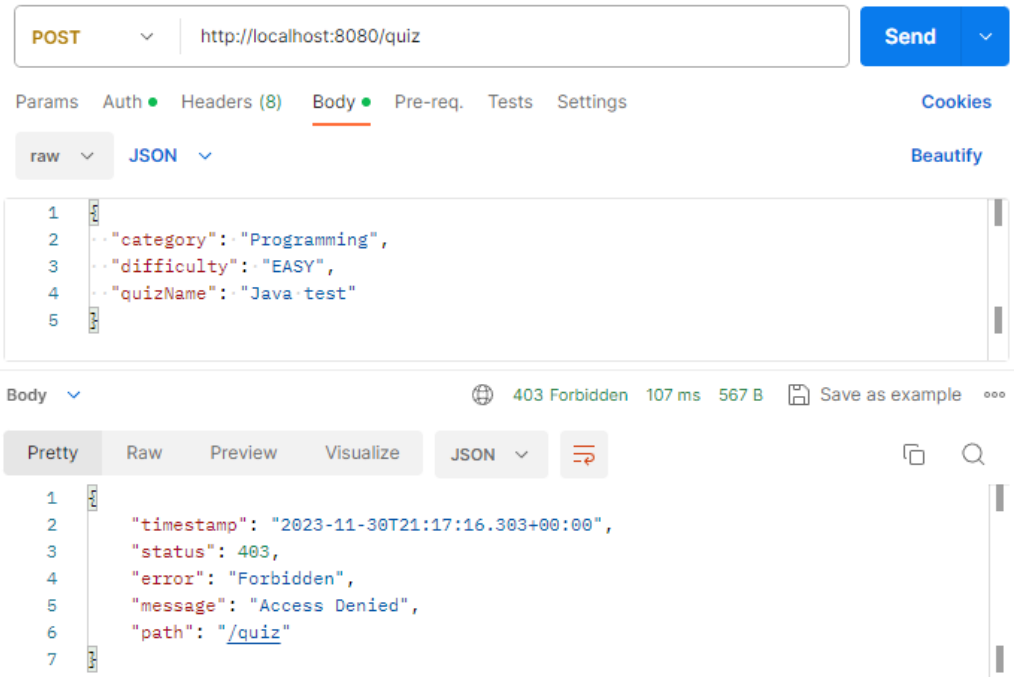

Рисунок 4.3 – Результат роботи ресурсу «POST /test» без JWT

Для початку було перевірено роботу при відсутності в HTTP запиту заголовку «Authentification», який відповідає за збереження JWT, відповідно за автентифікацію в системі. Згідно відповіді, було отримано код 403 та повідомлення «У доступі відказано», що є бажаною поведінкою при відсутності JWT. Здійснено запит на той самий ресурс вже з наявним JWT (рис 4.4).

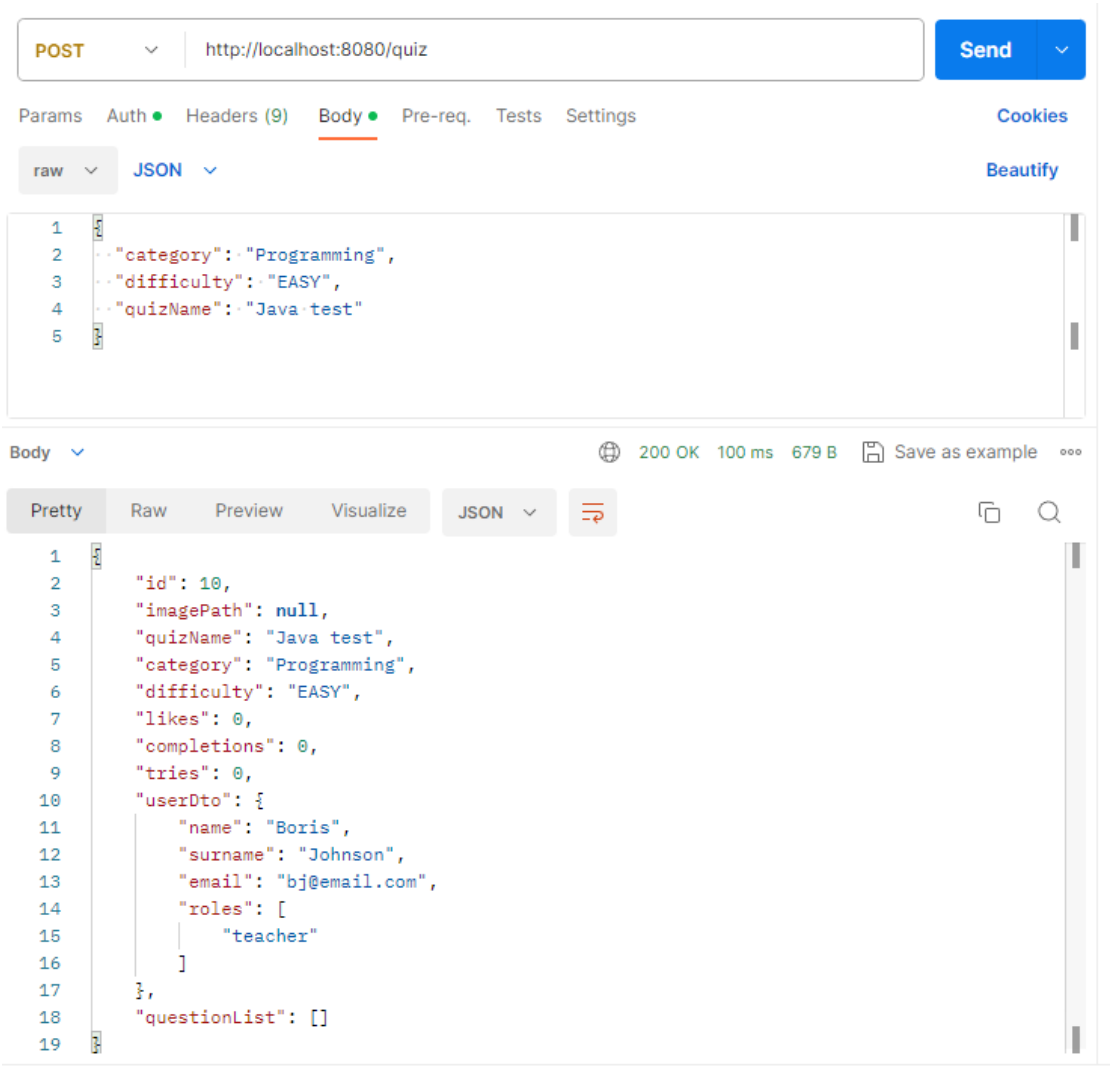

Рисунок 4.4 – Результат роботи ресурсу «POST /test» з JWT

В даному випадку, серверний модуль надав HTTP відповідь з кодом 200 та тілом, яке заповнене JSON об’єктом. Даний об’єкт відображає новий тест, який було створено безпосередньо в результаті тестування.

Наступним кроком перевіримо роботу ресурсу для отримання сутності тесту (рис 4.5).

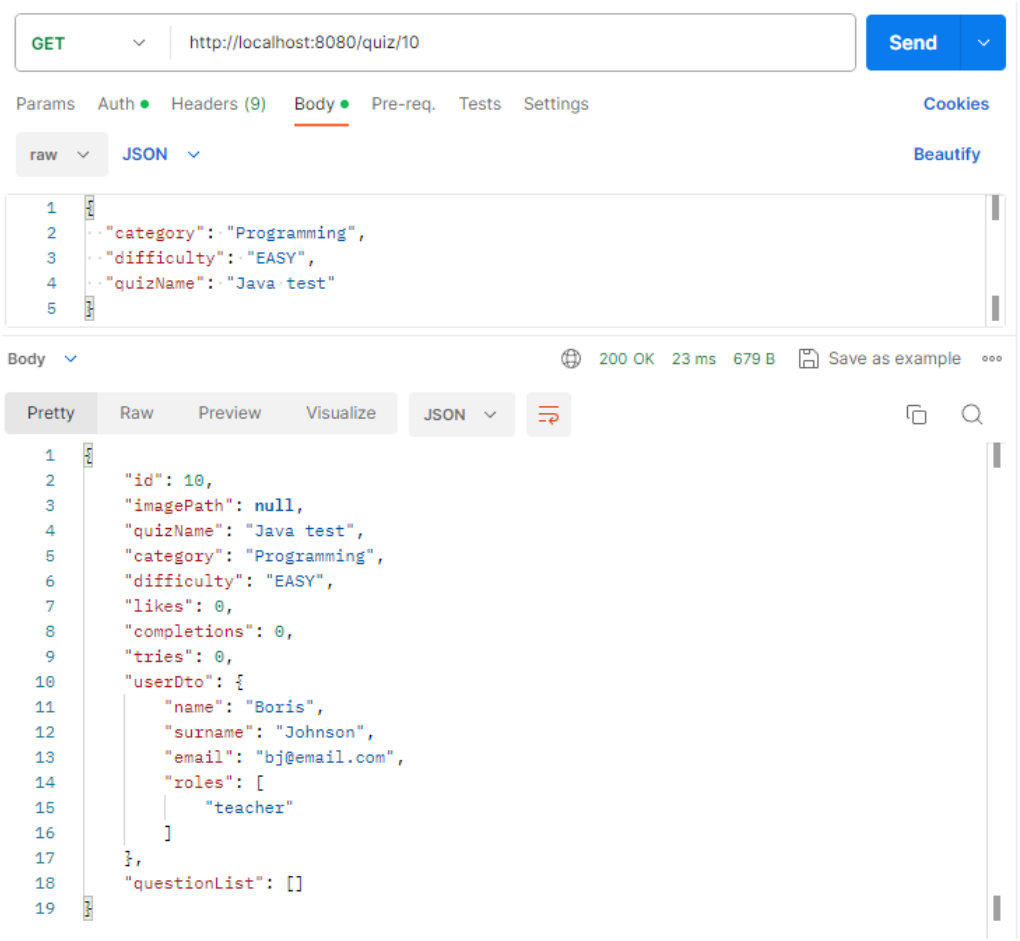

Рисунок 4.5 – Результат роботи ресурсу «GET /test/10»

Даний ресурс підтримує динамічний шлях в своєї структурі, а саме – число в кінці шляху. Дане число вказує на унікальний ідентифікатор сутності тесту, який повинен бути отриманий в результаті запиту. Завдяки тому, що на попередньому етапі тестування було створено нову сутність тесту та їй було присвоєно ідентифікатор «10», результатом виклику даного ресурсу стала та ж сама сутність тесту.

Розглянемо роботу ресурсу для отримання тестових питання певного тесту – «GET /quiz/2/question». Даний ресурс подібний до попереднього у випадку з динамічним шляхом. Подібним чином, в місце до потрібно визначити ідентифікатор тесту, вказуємо значення «2». Результатом здійснення HTTP запиту є відповідь з кодом 200 та тілом, яке заповнене JSON масивом об’єктів тестових питань, які належать згаданому тесту (рис 4.6).

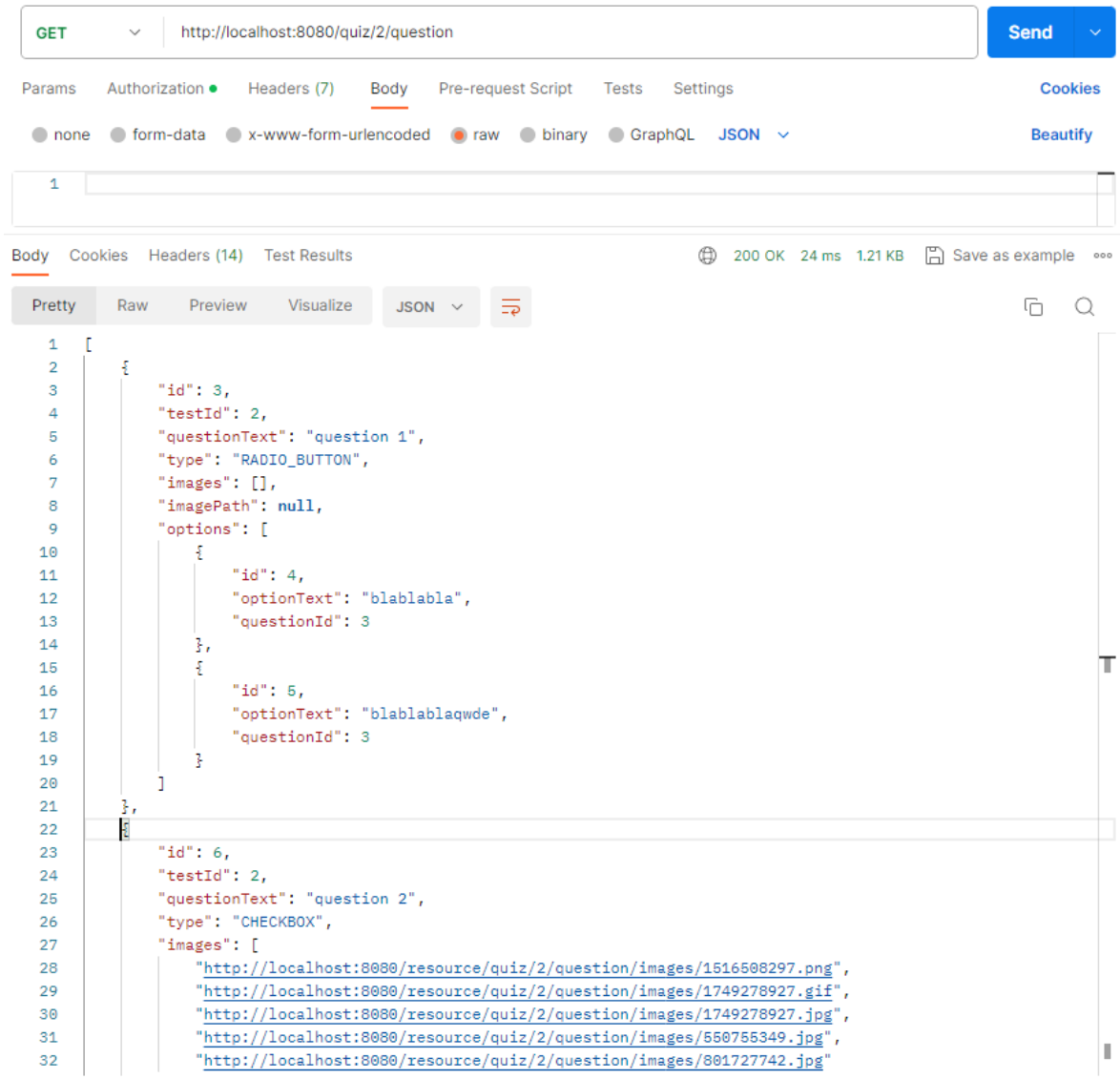

Рисунок 4.6 – Результат роботи ресурсу «GET /quiz/2/question»

Розглянемо роботу ресурсу для збереження зображень фотоконтролю. Даний ресурс виконується по шляху «POST /results/9/webcamPhotos». Аналогічно вже згаданим ресурсам, цифра «9» у шляху означає унікальний ідентифікатор результату тестування, який зв’язаний з певним користувачем та тестом. В даному запиті є відмінність адже, ресурс приймає HTTP запити з заголовком «Content-Type» значення якого становить «multipart/form-data». В такому випадку, для цього запиту тіло запиту повинно містити бінарні дані, а конкретно бінарне представлення файлу формату зображення. При використанні «multipart/form-data» в HTTP запиті, кожен елемент форми, наприклад, поле зображення або файл, обгортається у відповідний блок частини, що дозволяє

передавати бінарні дані без перекодування у текст. Результатом виклику ресурсу є HTTP відповідь з кодом 200. На рисунку 4.7 зображено результат роботи розглянутого ресурсу.

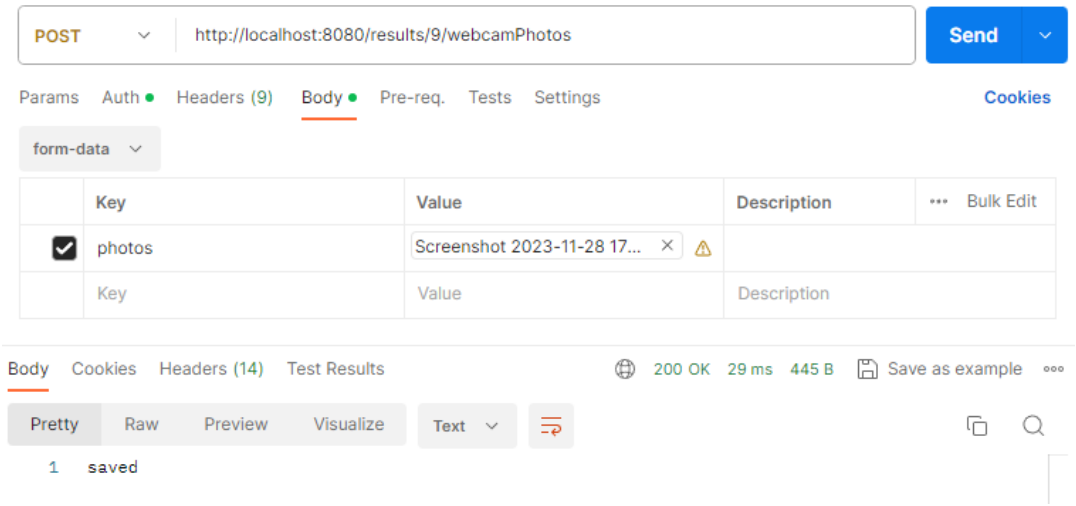

Рисунок 4.7 – Результат роботи ресурсу «POST /results/9/webcamPhotos»

Також розглянемо роботу ресурс для отримання ресурсів з сховища фотоданих серверного модуля (рис 4.8).

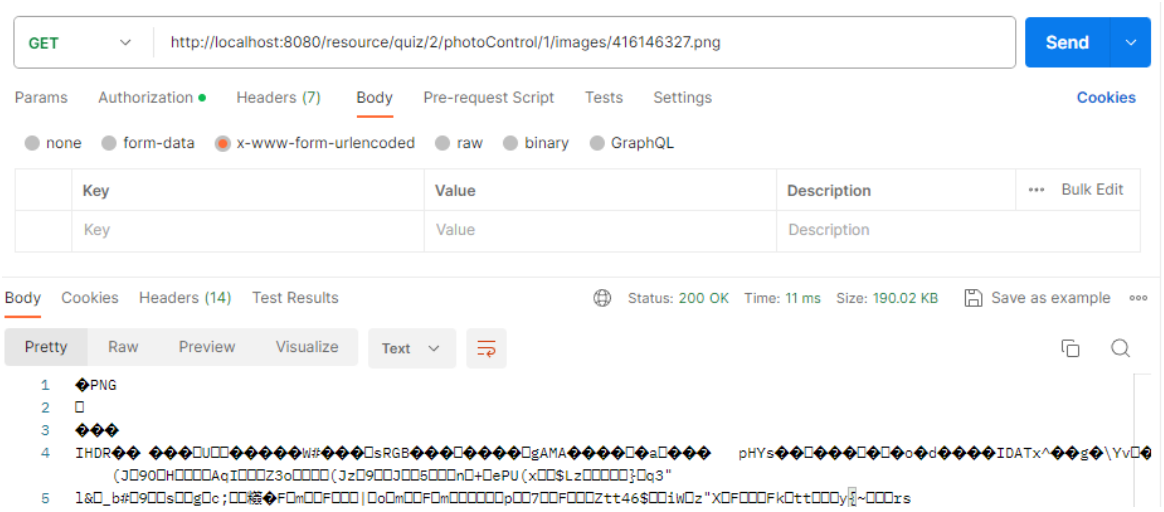

Рисунок 4.8 – Результат роботи ресурсу «GET /resource/quiz/2/photoControl/1/images/416146327.png»

Даний ресурс використовує динамічний шлях, тільки на відміну від попередній прикладів, динамічною частиною шляху вважаються всі символи, які

слідують після «resource/». Таким чином в шлях ресурс убуло визначено шлях зображення фотоконтролю результату тестування, згідно того як воно знаходиться в сховищі фотоданих. В результаті виконання запиту по вказаному ресурсу було отримано зображення в форматі «PNG», яке було закодоване в тілі HTTP відповіді. Це зображення є ідентичне тому, що було збережено на попередньому етапі тестування збереження фотоконтролю, згідно їх однаковому об’єму та змісту.

4.2 Документація прикладного програмного інтерфейсу

Для документування прикладного програмного інтерфейсу розробленого серверного модуля було використано інструмент Swagger. Це відкритий фреймворк для розробки, документування та використання RESTful вебзастосунків [28]. Він спрощує процес створення API, надаючи інструменти для документування, тестування та візуалізації API. Він надає динамічний і інтерактивний спосіб дослідження та тестування API прямо з браузера. Swagger добре поєднується з різними фреймворками та інструментами. Його можна легко використовувати з Spring Boot та іншими поширеними фреймворками для автоматичного створення документації API.

На рисунку 4.9 зображено вебсторінка, яка відображає всі ресурси та сутності прикладного програмного інтерфейсу серверного модуля.

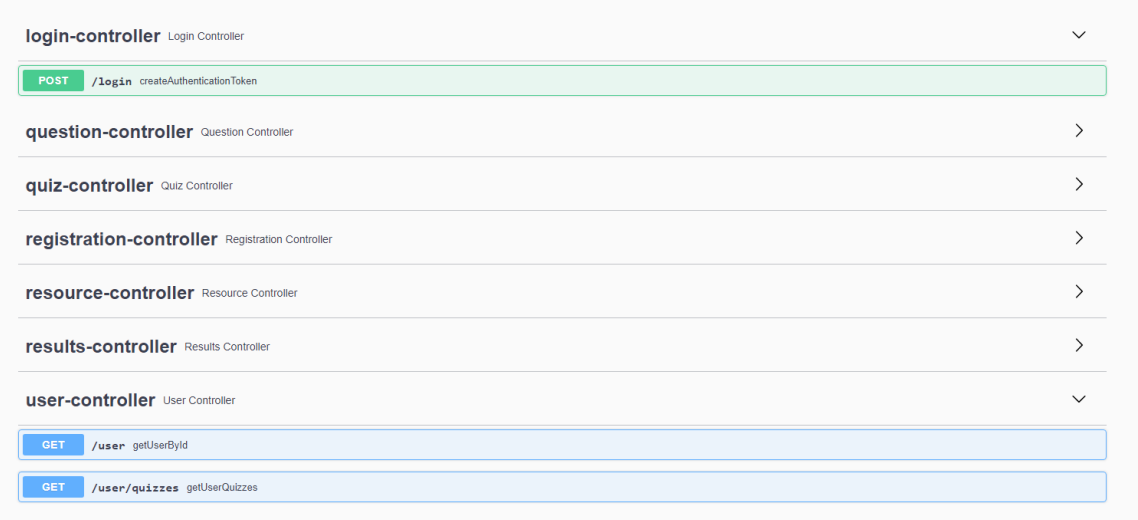

Рисунок 4.9 – Вебсторінка документації Swagger

При детально розгляді окремих ресурсів, наприклад «/quiz/{id}/question», можна отримати інформацію про назву, призначення, шлях, аргументи, тіло запиту та відповіді, можливі коди відповіді, тип змісту відповіді та запиту та інші важливі атрибути ресурсу.

Таким чином, програмний шляхом з використання інструменту Swagger було сформовано вебдокументацію прикладного програмного інтерфейсу сервреного модуля для більш ніж 30 ресурсів та 15 сутностей. Дана документація може бути поширене безпосередньо серверним модулем за ресурсом «/swagger-ui/».

4.3 Висновки

У даному розділі було здійснена перевірка роботоздатності серверного модуля дистанційного тестування з використанням технології Google Cloud Vision. Наведені результати тестування підтвердили, що розроблений прикладний програмний інтерфейс відповідає визначеним технічних вимогам та не має критичних дефектів.

Також була підготовлена документація для прикладного програмного інтерфейсу серверного модуля. Документація можу бути отримана безпосередньо при зверненні до розробленого серверного модуля.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку [29].

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

Метою проведення комерційного і технологічного аудиту дослідження за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» є оцінювання

науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [30].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
0	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
1	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
2	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	3	3	4
3. Ринкові переваги (ціна продукту)	4	4	3
4. Ринкові переваги (технічні властивості)	3	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	4
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	38	38	41
Середньоарифметична сума балів $СБ_c$	39		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [30].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» становить 39 бали, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [30]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 25000{,}00 \cdot 65 / 21 = 88636{,}36 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000	1190,48	65	88636,36
Інженер-розробник програмного забезпечення	22000	1047,62	65	73863,64
Консультант	15000	714,29	65	35454,55
Всього				197954,55

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [30];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 69{,}09$ грн.

$З_{р1} = 69{,}09 \cdot 11{,}00 = 760{,}03$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення обладнання	11	2	1,1	69,09	760,03
Підготовка робочого місця	5	2	1,1	69,09	345,47
Інсталяція програмного забезпечення	5	5	1,7	106,78	533,91
Всього					1639,41

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (197954,55+ 1639,41) · 12 / 100% = 21955,33 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (197954,55+ 1639,41+ 21955,33) · 22 / 100% = 48740,84 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 3 · 190,00 · 1,1 - 0,000 · 0,00 = 627 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний Zoom Stora Enso A4 80 г/м2	190	3	0	0	627
Набір офісного працівника JOBMAX з наповненням	150	2	0	0	330
Flesh-пам'ять Kingston 16 GB	130	1	0	0	143
Всього					1100

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення в даному дослідженні не використовувалось.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 6000 \cdot 1 \cdot 1,12 = 6720$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
IntelliJ IDEA Ultimate	1	2276	2549,12
ОС Windows 10	1	6000	6720
Прикладний пакет Microsoft Office 2019	1	5000	5600
Всього			14869,12

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \ , \qquad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (40000,00 \cdot 3) / (5 \cdot 12) = 2500грн.$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Acer Nitro 5 AN517-54-72WJ (NH.QF8EU.00S)	40 000	5	3	2500,00
Додатковий монітор Монітор ASUS 24” VY249HGE (90LM06A5-B02370)	6 000	5	3	2000,00
Маршрутизатор	2 500	2	3	300,00
Робоче місце дослідника	20000	5	3	312,50
Оргтехніка	7000	4	3	1000,00
Приміщення лабораторії	250000	20	3	437,50
ОС Windows 10	6720	3	3	3125,00
Прикладний пакет Microsoft Office 2019	5600	3	3	560,00
IntelliJ IDEA Ultimate	2276	3	3	466,67
				8391,33

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7{,}50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0{,}3 \cdot 520{,}0 \cdot 7{,}50 \cdot 0{,}95 / 0{,}97 = 1145{,}88$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Acer Nitro 5 AN517-54-72WJ (NH.QF8EU.00S)	0,3	520	1145,88
Додатковий монітор Монітор ASUS 24” VY249HGE (90LM06A5-B02370)	0,1	520	381,96
Маршрутизатор	0,05	400	572,94
Робоче місце дослідника	0,15	520	146,91
Оргтехніка	0,45	10	52,89
Всього			2300,57

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та

приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (191904,76+ 1639,41) · 20 / 100% = 39918,79 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$В_{сп}$ = (191904,76+ 1639,41) · 35 / 100% = 69857,88 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (191904,76+ 1639,41) · 50 / 100% = 99796,98 грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$B_{нзв} = (191904,76 + 1639,41) \cdot 120 / 100\% = 239\,512,74$ *грн.*

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.14)$$

$B_{заг} = 840637,54$*грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η =0,8.

$ЗВ = 840637,54 / 0,8 = 1050796,93$ *грн.*

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 2000 користувачів;

2-й рік – 3000 користувачів;

3-й рік – 1500 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 2500 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 105 000 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 30000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [30]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(30000·2500,00+135000·2000)·0,83·0,3·(1-0,18/100%)= 70696710 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(30000·2500,00+135000·(2000+3000))·0,83·0,3·(1-0,18/100%)= 153688500 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(30000·2500,00+135000·(2000+3000+1500))·0,83·0,3·(1-0,18/100%)= 195184395 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 70696710/(1+0,25)^1 + 153688500/(1+0,25)^2 + 195184395/(1+0,25)^3 = 254852418,24\text{грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.18}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=4;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1050796,93грн.

$PV = k_{інв} \cdot ЗВ = 4 \cdot 1050796,93 = 4203187,705\text{грн.}$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.19}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 254852418,24грн;

PV – теперішня вартість початкових інвестицій, 4203187,705грн.

$E_{абс} = ПП - PV = 254852418,24 - 4203187,705 = 250649230,53\text{грн.}$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 250649230,53грн;

PV – теперішня вартість початкових інвестицій, 4203187,705 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+250649230,53/\ 4203187,705)^{1/3} = 2,9.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,11+0,25 = 0,36 < 2,9 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (5.22)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 2,9 = 0,34 року.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера» становить 39 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу вище середнього).

Також термін окупності становить 0,34 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера».

ВИСНОВОК

У результаті виконання магістерської кваліфікаційної роботи було розроблено методи та програмні засоби серверного модулю дистанційного тестування з використанням технології Google Cloud Vision, що дозволило покращити якість онлайн тестування та зменшити кількість порушень академічної доброчесності.

Основні результати роботи такі:

1. Аналіз сучасного стану проблеми проведення дистанційного тестування показав, що у відомих програмних реалізаціях тестувальних систем недостатня увага приділяється питанням забезпечення достовірності тестування і зменшення кількості порушень академічної доброчесності. Зокрема, відсутній фотоконтроль осіб, що тестуються, з використанням засобів розпізнавання зображень. Тому актуальною є задача застосування засобів розпізнавання Google Cloud Vision в тестувальних системах.

2. Подальшого розвитку отримав метод дистанційного тестування, у якому на відміну від існуючих, використано технологію Google Cloud Vision для розпізнавання та аналізу зображень учасників тестування, що забезпечує підвищення рівня академічної доброчесності.

3. Подальшого розвитку отримав метод зберігання фотоданих у файловому сховищі, у якому на відміну від існуючих, використана ієрархічна структура, що підвищує швидкість пошуку фотоданих на 10 %.

4. Аналіз загальної архітектури серверного модуля системи дистанційного тестування дозволив залучити до проєктування сучасні та поширені архітектурні патерни, які значно пришвидшать процес розробки та підтримку програмного продукту, а саме такі патерни як RESTful архітектура та багаторівнева архітектура. Також спроєктовано структуру прикладного програмного інтерфейсу розробленого сервреного модуля, описано атрибути, призначення та поведінку спроєктованих ресурсів.

5. В результаті здійсненного порівняльного аналізу поширених мов програмування було обрано мову програмування Java, як таку, що найбільше пасує для вирішення поставлених задач.

6. Здійснено програмну реалізацію важливих компонентів та модулів, таких як маніпуляція даними у базі даних, авторизація та автентифікація, захист вразливих вебресурсів, створення прикладного програмного інтерфейсу, аналіз фотоданих та інших.

7. Тестування програмних модулів показало їх працездатність та відповідність завданню на роботу.

8. Сформовано та розглянуто веб-документацію прикладного програмного інтерфейсу серверного модуля, а саме ресурсів, їх опису, шляху, параметрів, тіла, заголовків, кодів відповіді та структуру об’єктів, які використовуються під час обміну даними.

9. Термін окупності становить 0,34 р., що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Педагогічне тестування. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Педагогічне_тестування

2. Кремень В. Г. Енциклопедія освіти / Кремень В. Г. – Київ: Юрінком Інтер, 2021. – 1054 с.

3. Токарчук Д. О. розробка серверної частини адаптивної тестувальної системи з фотоконтролем з використанням технологій Java та фреймворку Spring [Електронний ресурс] / Д.О. Токарчук, Д. І. Катєльніков // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. - 2022. - Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15529/13107

4. Токарчук Д. О. Використання розпізнавання зображень Google Cloud Vision в розробці програмного забезпечення [Електронний ресурс] / Д.О. Токарчук, В. П. Майданюк // XVI Міжнародна науково-практична конференція, Одеса: Одеський Національний Технологічний Університет. – 2023. – Режим доступу до ресурсу: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/38163/5aee68e5ad0c4282b89b8b3e049c987c.pdf?sequence=1&isAllowed=y

5. Токарчук Д. О. Міжнародна науково-практична Інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ». Вінниця: Вінницький національний технічний університет. – 2023. – Режим доступу до ресурс: https://drive.google.com/file/d/1eAb_QXS6CvA-e1ikKxerTqvw6P1IPRhc/view

6. Бикова Т. Б. Особливості реалізації міжкомпонентних взаємодій у процесі використання елементів дистанційного навчання / Т. Б. Бикова, М. В. Іващенко // Вісник Глухівського національного педагогічного університету імені Олександра Довженка. : Педагогічні науки. - 2018. - Вип. 1. - С. 163-169. - Режим доступу: http://nbuv.gov.ua/UJRN/vgnpu_2018_1_23.

7. ClassMaker. [Електронний ресурс] – URL: https://www.classmarker.com/

8. exam.net. [Електронний ресурс] – URL: https://exam.net/

9. Testportal. [Електронний ресурс] – URL: https://www.testportal.net/

10. Software development process – URL: https://en.wikipedia.org/wiki/Software_development_process

11. Популярні життєві цикли розробки ПЗ. [Електронний ресурс] – URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/

12. Тейлор Р. Software Architecture: Foundations, Theory, and Practice / Тейлор Р. John Wiley & Sons, 2009. – 736 с.

13. 14 software architecture design patterns to know. [Електронний ресурс] – URL: https://www.redhat.com/architect/14-software-architecture-patterns

14. SQL vs. NoSQL Databases: What's the Difference? [Електронний ресурс] – URL: https://www.ibm.com/blog/sql-vs-nosql/

15. Таxагогі С. Learning MySQL: Get a Handle on Your Data / Таxагогі С. O'Reilly Media, 2006. 618 с.

16. File system. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/File_system

17. API. [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/API

18. Working with JSON. . [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

19. Google Cloud Platform. [Електронний ресурс] – URL: https://cloud.google.com/?hl=en

20. Vision AI . [Електронний ресурс] – URL: https://cloud.google.com/vision?hl=en

21. Oracle Java . [Електронний ресурс] – URL: https://www.oracle.com/ua/java/

22. Python.org . [Електронний ресурс] – URL: https://www.python.org/

23. JavaScript . [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

24. Spring Framework. [Електронний ресурс] – URL: https://spring.io/projects/spring-framework

25. Hibernate ORM. [Електронний ресурс] – URL: https://hibernate.org/orm/

26. Introduction to JSON Web Tokens. [Електронний ресурс] – URL: https://jwt.io/introduction

27. Ачіне М. Effective Software Testing / Ачіне М. Manning, 2022. 328 с.

28. Swagger. [Електронний ресурс] – URL: https://swagger.io/

29. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

30. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

Додаток А

(обов'язковий)

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" 09 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к. т. н. Майданк В. П.

"19" 09 2023 р.

Виконав:

студент гр.3ПІ-22м Токарчук Д. О.

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера.».

Галузь застосування – онлайн тестування.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є покращенні якості онлайн тестування та зниження кількості порушень академічної доброчесності за рахунок створення тестувальної онлайн-системи з використанням технологій штучного інтелекту Google Cloud Vision.

Призначення роботи – здійснення онлайн тестування з використанням серверного модуля дистанційного тестування з прикладним програмним інтерфейсом.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Токарчук Д. О. розробка серверної частини адаптивної тестувальної системи з фотоконтролем з використанням технологій Java та фреймворку Spring [Електронний ресурс] / Д.О. Токарчук, Д. І. Катєльніков // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. - 2022. - Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15529/13107

2. Токарчук Д. О. Використання розпізнавання зображень Google Cloud Vision в розробці програмного забезпечення [Електронний ресурс] / Д.О. Токарчук, В. П. Майданюк // XVI Міжнародна науково-практична конференція, Одеса: Одеський Національний Технологічний Університет. – 2023. – Режим доступу до ресурсу: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/38163/5aee68e5ad0c4282b89b8b3e049c987c.pdf?sequence=1&isAllowed=y

3. Токарчук Д. О. Міжнародна науково-практична Інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ». Вінниця: Вінницький національний технічний університет. – 2023. – Режим доступу до ресурс: https://drive.google.com/file/d/1eAb_QXS6CvA-e1ikKxerTqvw6P1IPRhc/view

4. Тейлор Р. Software Architecture: Foundations, Theory, and Practice / Тейлор Р. John Wiley & Sons, 2009. – 736 с.

5. Тахагогі С. Learning MySQL: Get a Handle on Your Data / Тахагогі С. O'Reilly Media, 2006. 618 с.

6. Modern Java in Action. Рауль-Габріель Урма, Маріо Фаско, Manning, 2018. – 592 с

5. Технічні вимоги

Прикладний програмний інтерфейс, авторизація та автентифікація, фотофіксація, збереження даних системи, збереження та аналіз фотоданих, база даних, сховище фотоданих, інтеграція технології Google Cloud Vision.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	20.09.2023 – 05.10.2023
2	Розробка архітектури та алгоритмів роботи серверного модуля	06.10.2023 – 19.10.2023
3	Аналіз і вибір мови програмування та середовища програмування	20.10.2023 – 23.10.2023
4	Розробка серверного модуля	24.10.2023 – 16.11.2023
5	Тестування роботи серверного модуля	17.11.2023 – 21.11.2023
6	Економічна частина	22.11.2023 – 25.11.2023
7	Оформлення матеріалів до захисту МКР	26.11.2023 – 01.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б 112

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ МАГСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: **«Моделі та програмні засоби дистанційного тестування з використанням технології Google Cloud Vision. Частина 1. Модуль сервера»**

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ЗПІ – 22м

Науковий керівник: д.т.н., професор каф. ПЗ Майданюк В. П.

Unicheck	
Оригінальність	97.5 %
Схожість	2.5 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Токарчук Д. О.

Керівник роботи _____ Майданюк В. П.

Додаток В

(довідниковий)

Лістинг коду сервреного модуля

edu.quiz.project.QuizProjectApplication

```
package edu.quiz.project;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class QuizProjectApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuizProjectApplication.class, args);
    }

}
```

edu.quiz.project.domain.entity.User

```
package edu.quiz.project.domain.entity;

import edu.quiz.project.domain.constant.UserRole;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
```

```
import java.util.Date;
import java.util.List;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@SequenceGenerator(name = "userSequence", sequenceName = "userSequence")
@Table(name = "User")
public class User implements UserDetails {

    @Id
    @Column(name = "user_id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    private String email;

    private String firstName;

    private String lastName;

    private String password;

    private Date date;

    @ElementCollection(targetClass = UserRole.class)
    @CollectionTable(
            name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id")
    )
    @Column(name = "roles")
    private List<UserRole> roles = new ArrayList<>();
```

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
   return roles;
}

@Override
public String getPassword() {
   return password;
}

@Override
public String getUsername() {
   return email;
}

@Override
public boolean isAccountNonExpired() {
   return true;
}

@Override
public boolean isAccountNonLocked() {
   return true;
}

@Override
public boolean isCredentialsNonExpired() {
   return true;
}

@Override
public boolean isEnabled() {
   return true;
}
```

```
}
```

edu.quiz.project.domain.entity.Quiz

```
package edu.quiz.project.domain.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import edu.quiz.project.domain.constant.Difficulty;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.Accessors;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Accessors(chain = true)
@SequenceGenerator(name = "quizSequence", sequenceName = "quizSequence")
@Table(name = "Quiz")
public class Quiz {

    @Id
    @Column(name = "quiz_id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long quizId;

    private String quizName;
```

```
private String category;

private String imagePath;

@Enumerated(value = EnumType.STRING)
private Difficulty difficulty;

@Builder.Default
private Integer likes = 0;

@Builder.Default
private Integer completions = 0;

@Builder.Default
private Integer tries = 0;

@JsonIgnore
@OneToMany(mappedBy = "quiz", cascade = CascadeType.ALL)
private List<Question> questions;

public void addQuestion(Question question) {
   if (questions == null) {
      questions = new ArrayList<>();
   }
   questions.add(question);
   question.setQuiz(this);
}

public void addQuestions(Collection<Question> questions) {
   questions.forEach(this::addQuestion);
}

@ManyToOne
private User author;
```

```
    public User getAuthor() {
        return author;
    }

    public void setAuthor(User manyToOne) {
        this.author = manyToOne;
    }
}
```

edu.quiz.project.controller.LoginController

```
package edu.quiz.project.controller;

import edu.quiz.project.domain.dto.JwtDto;
import edu.quiz.project.domain.dto.LoginRequest;
import edu.quiz.project.service.UserService;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "*")
@AllArgsConstructor
@Slf4j
public class LoginController {

    private final UserService userService;

    @PostMapping(value = "/login")
    public ResponseEntity<?> createAuthenticationToken(@RequestBody LoginRequest
loginRequest) throws Exception {
        JwtDto body = userService.loginUser(loginRequest);
        return ResponseEntity.ok(body);
```

```
    }
}
```

edu.quiz.project.controller.QuizController

```
package edu.quiz.project.controller;

import edu.quiz.project.domain.dto.question.SubmittedQuestionDto;
import edu.quiz.project.domain.dto.quiz.NewQuizDto;
import edu.quiz.project.domain.dto.quiz.QuizDto;
import edu.quiz.project.domain.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.domain.entity.User;
import edu.quiz.project.service.QuizService;
import edu.quiz.project.service.QuizUserEntryService;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import springfox.documentation.annotations.ApiIgnore;

import java.util.List;


@Controller
@RequestMapping("/quiz")
@CrossOrigin(origins="*")
public class QuizController {

  private final QuizService quizService;
  private final QuizUserEntryService quizUserEntryService;
```

```
    public QuizController(QuizService quizService, QuizUserEntryService
quizUserEntryService) {
      this.quizService = quizService;
      this.quizUserEntryService = quizUserEntryService;
    }

    @ApiOperation(value = "Взяти тест по ІД",notes = "Returns test with all questions and
options")
    @GetMapping("/{testId}")
    public ResponseEntity<QuizDto> getQuizById(@PathVariable Long testId){
      return ResponseEntity.ok(quizService.getQuizDtoById(testId));
    }

    @ApiOperation(value = "Взяти тести з пагіниціею")
    @GetMapping
    public ResponseEntity<Page<QuizDto>> getQuizzesWithPagination(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "10") int size){
      return ResponseEntity.ok(quizService.getAll(page,size));
    }

    @ResponseBody
    @PostMapping
    public QuizDto createNewQuiz(@RequestBody @ApiParam NewQuizDto test,
                    @ApiIgnore Authentication authentication){
      User user = (User) authentication.getPrincipal();
      return quizService.saveNewQuiz(test, user);
    }

    @ResponseBody
    @PutMapping
    public QuizDto updateQuiz(@RequestBody @ApiParam QuizDto quizDtoToUpdate) {
      return quizService.updateQuiz(quizDtoToUpdate);
    }
```

```
    @ResponseBody
    @PostMapping("/{quizId}/image")
    public String addImageToQuiz(@PathVariable Long quizId, @RequestPart MultipartFile
file) {
      return quizService.saveQuizPicture(file,quizId);
    }

    @ResponseBody
    @PostMapping("/{quizId}/submit")
    public QuizUserEntryDto submitQuizForUser(@PathVariable Long quizId,
                                              @RequestBody List<SubmittedQuestionDto>
submittedQuestions, Authentication authentication) {
      User user = (User) authentication.getPrincipal();
      return quizUserEntryService.submitTestEntryForUser(quizId, submittedQuestions, user);
    }

    @ApiOperation(value = "Отримати результати тестування всіх корисутвачів")
    @ResponseBody
    @GetMapping("/{quizId}/entries")
    public Page<QuizUserEntryDto> getQuizEntriesPage(@RequestParam(defaultValue = "0")
int page,
                                                     @RequestParam(defaultValue = "10") int size,
                                                     @ApiParam(hidden = true) Authentication authentication) {
      User user = (User) authentication.getPrincipal();
      return quizUserEntryService.getQuizEntriesPage(user.getId(), page, size);
    }

  }
```

edu.quiz.project.controller.RegistrationController

```
  package edu.quiz.project.controller;

  import edu.quiz.project.domain.dto.JwtDto;
  import edu.quiz.project.domain.dto.UserRegistrationDto;
```

```
import edu.quiz.project.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
public class RegistrationController {

    private final UserService userService;

    @PostMapping("/register")
    public ResponseEntity<JwtDto> registerUser( @RequestBody UserRegistrationDto registrationDto) {
        return ResponseEntity.ok(userService.registerNewUser(registrationDto));
    }

}
```

edu.quiz.project.controller.ResultsController

```
package edu.quiz.project.controller;

import edu.quiz.project.domain.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.domain.entity.User;
import edu.quiz.project.service.QuizService;
import edu.quiz.project.service.QuizUserEntryService;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

@Controller
@RequestMapping("/results")
@CrossOrigin(origins="*")
@RequiredArgsConstructor
public class ResultsController {

    private final QuizService quizService;
    private final QuizUserEntryService quizUserEntryService;

    private final CloudVisionTemplate cloudVisionTemplate;

    @ApiOperation(value = "Отримати результати тестування користувача")
    @ResponseBody
    @GetMapping("/entries")
    public Page<QuizUserEntryDto> getUserTestEntries(@RequestParam(defaultValue =
"0") int page,
                                                     @RequestParam(defaultValue = "10") int size,
                                                     @ApiParam(hidden = true) Authentication authentication) {
        User user = (User) authentication.getPrincipal();
        return quizUserEntryService.getUserEntriesPage(user.getId(), page, size);
    }

    @ApiOperation(value = "Зберегти зображення з веб камери")
    @ResponseBody
    @PostMapping(value = "/{quizEntryId}/webcamPhotos", consumes = "multipart/form-
data")
    public String addUserWebcamPhotos(@PathVariable("quizEntryId") Long quizEntryId,
                                      @RequestParam MultipartFile[] photos,
                                      Authentication authentication) {
        User user = (User) authentication.getPrincipal();
        quizUserEntryService.saveUsersWebcamPhotos(photos, quizEntryId, user);
        return "saved";
```

```
    }

}
```

edu.quiz.project.repository.QuizRepository

```
package edu.quiz.project.repository;

import edu.quiz.project.domain.entity.Quiz;
import edu.quiz.project.domain.projection.TestFullProjection;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(excerptProjection = TestFullProjection.class)
public interface QuizRepository extends JpaRepository<Quiz, Long> {
    Page<Quiz> findAll(Pageable pageable);

    Page<Quiz> getQuizByAuthorId(long authorId, Pageable pageable);
}
```

edu.quiz.project.repository.UserRepository

```
package edu.quiz.project.repository;

import edu.quiz.project.domain.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
```

```
    @Query(value = "select u from User u " +
            "join fetch u.roles " +
            "where u.email = ?1 ")
    Optional<User> getUserByEmailEagerly(String email);

    @Query(value = "select u from User u " +
            "join fetch u.roles " +
            "where u.id = ?1 ")
    User getByIdEagerly(long userId);
}
```

edu.quiz.project.service.QuizService

```
package edu.quiz.project.service;

import edu.quiz.project.domain.dto.quiz.NewQuizDto;
import edu.quiz.project.domain.dto.quiz.QuizDto;
import edu.quiz.project.domain.entity.Quiz;
import edu.quiz.project.domain.entity.User;
import edu.quiz.project.domain.mapper.Mapper;
import edu.quiz.project.domain.updater.QuizDtoUpdater;
import edu.quiz.project.repository.QuizRepository;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class QuizService {
```

```
@Qualifier(value = "TestMapper")
private final Mapper<Quiz, QuizDto> quizMapper;
@Qualifier(value = "NewTestMapper")
private final Mapper<Quiz, NewQuizDto> newTestMapper;

private final QuizDtoUpdater quizDtoUpdater;

private final QuizRepository quizRepository;
private final LocalFileService localFileService;

public QuizService(Mapper<Quiz, QuizDto> quizMapper,
                   Mapper<Quiz, NewQuizDto> newTestMapper,
                   QuizDtoUpdater quizDtoUpdater, QuizRepository quizRepository,
                   LocalFileService localFileService) {
   this.quizMapper = quizMapper;
   this.newTestMapper = newTestMapper;
   this.quizDtoUpdater = quizDtoUpdater;
   this.quizRepository = quizRepository;
   this.localFileService = localFileService;
}

public boolean isQuizExists(long testId) {
   return quizRepository.existsById(testId);
}

public QuizDto getQuizDtoById(Long testId) {
   return quizMapper.entityToDto(getQuizById(testId));
}

public Quiz getQuizById(Long testId) {
   return quizRepository.findById(testId)
         .orElseThrow(() -> new RuntimeException("Test not exists!"));
}
```

```
    public List<QuizDto> getAll() {
        return quizRepository.findAll().stream()
                .map(quizMapper::entityToDto)
                .collect(Collectors.toList());
    }

    public Page<QuizDto> getAll(int page, int size) {
        return quizRepository.findAll(PageRequest.of(page, size))
                .map(quizMapper::entityToDto);
    }

    @Transactional
    public String saveQuizPicture(MultipartFile file, Long testId) {
        Quiz targetQuiz = Optional.ofNullable(quizRepository.getById(testId))
                .orElseThrow(() -> new RuntimeException("Test not found"));
        targetQuiz.setImagePath(
                localFileService.saveQuizImage(file, targetQuiz));
        quizRepository.save(targetQuiz);
        return targetQuiz.getImagePath();
    }

    @Transactional
    public QuizDto saveNewQuiz(NewQuizDto test, User user) {
        Quiz quizEntity = newTestMapper.dtoToEntity(test);
        quizEntity.setAuthor(user);
        return quizMapper.entityToDto(quizRepository.save(quizEntity));
    }

    @Transactional
    public QuizDto updateQuiz(QuizDto quizDtoToUpdate) {
        return quizMapper.entityToDto(
                quizRepository.save(
                        quizDtoUpdater.applyUpdate(quizDtoToUpdate,
quizRepository.getById(quizDtoToUpdate.getId()))
                )
```

```
        );
    }


    public List<QuizDto> getQuizzesByUserId(Long userId, int page, int size) {
        return quizRepository.getQuizByAuthorId(userId, PageRequest.of(page, size)).stream()
                .map(quizMapper::entityToDto)
                .collect(Collectors.toList());
    }
}
```

edu.quiz.project.util.JwtUtil

```
package edu.quiz.project.util;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Service
public class JwtUtil {

    private String JWT_SECRET = "super_secret";

    public String extractEmail(String token){
        return extractClaim(token, Claims::getSubject);
    }
```

```
public Date extractExpiration(String token){
   return extractClaim(token,Claims::getExpiration);
}

public <T> T extractClaim(String token, Function<Claims,T> claimsResolver) {
   Claims claims = extractAllClaim(token);
   return claimsResolver.apply(claims);
}

private Claims extractAllClaim(String token){
   return Jwts.parser().setSigningKey(JWT_SECRET).parseClaimsJws(token).getBody();
}

private boolean isTokenExpired(String token){
   return extractExpiration(token).before(new Date());
}

public String generateToken(UserDetails userDetails){
   Map<String, Object> claims = new HashMap<>();
   return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String,Object> claims, String subject){
   return Jwts.builder()
         .setClaims(claims)
         .setSubject(subject)
         .setIssuedAt(new Date(System.currentTimeMillis()))
         .setExpiration(new Date(System.currentTimeMillis() + 3_600_000))
         .signWith(SignatureAlgorithm.HS256, JWT_SECRET).compact();
}

public boolean validateToken(String token, UserDetails userDetails){
   String username = extractEmail(token);
   return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
}}
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МОДЕЛІ ТА ПРОГРАМНІ ЗАСОБИ ДИСТАНЦІЙНОГО ТЕСТУВАННЯ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ GOOGLE CLOUD VISION. ЧАСТИНА 1. МОДУЛЬ СЕРВЕРА.

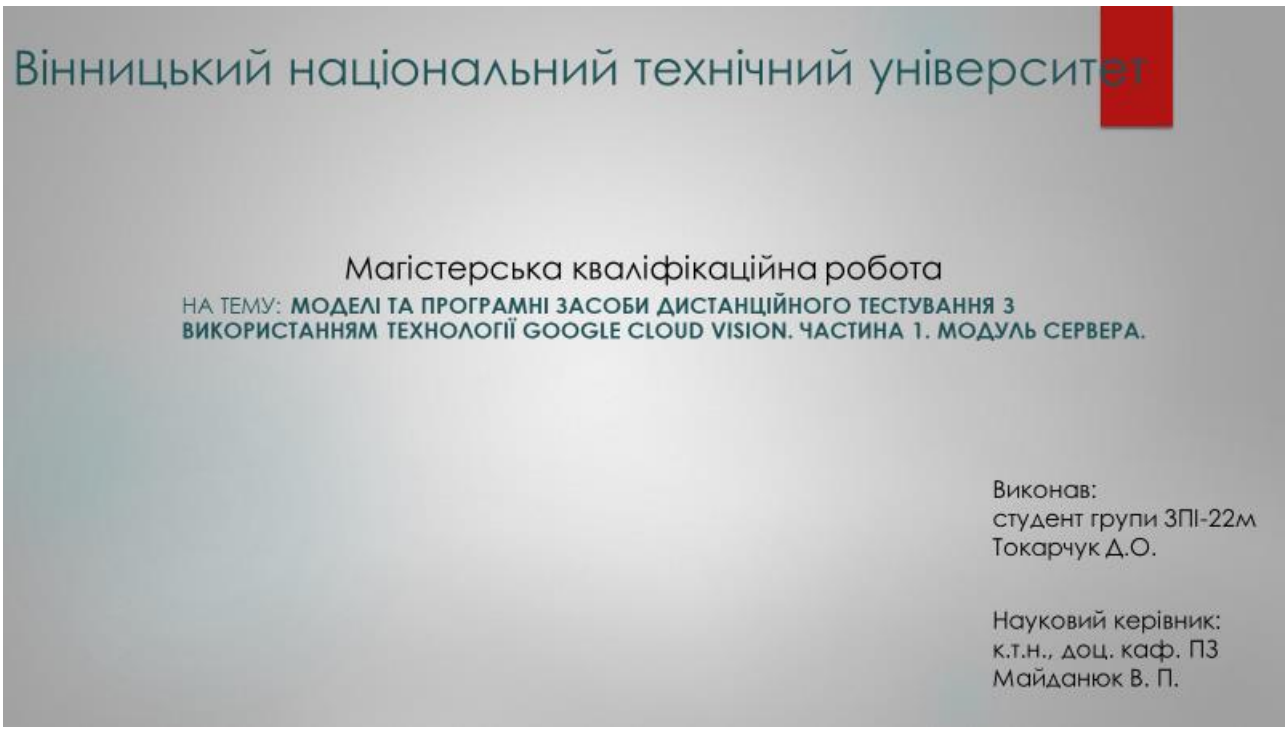

Рисунок Г.1 – Слайд №1

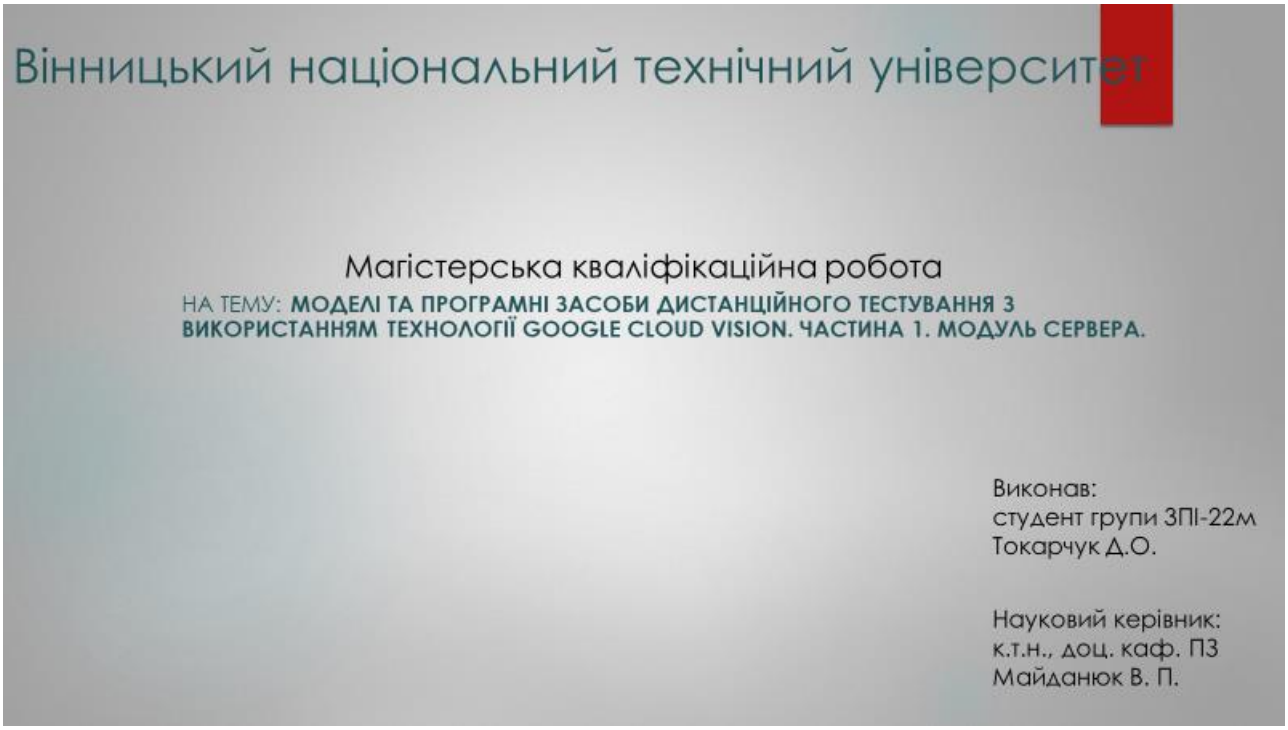

Рисунок Г.2 – Слайд №2

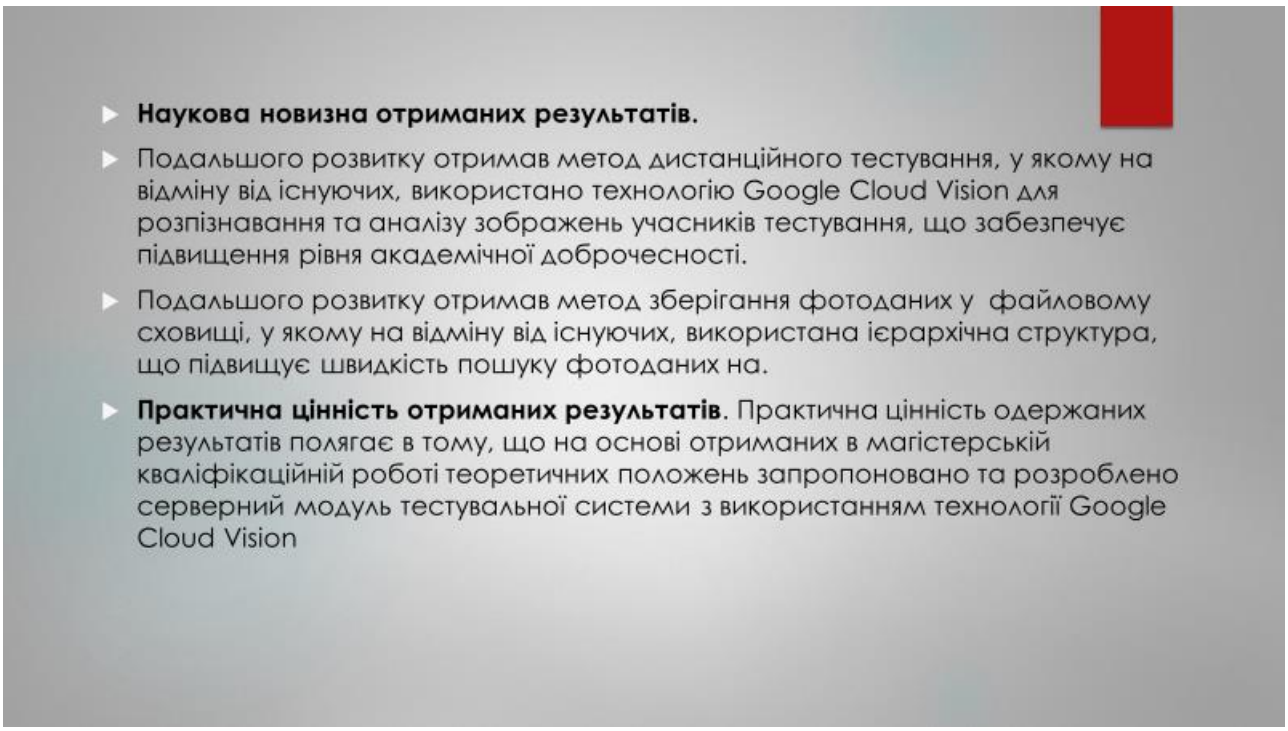

Рисунок Г.3 – Слайд №3

Рисунок Г.4 – Слайд №4

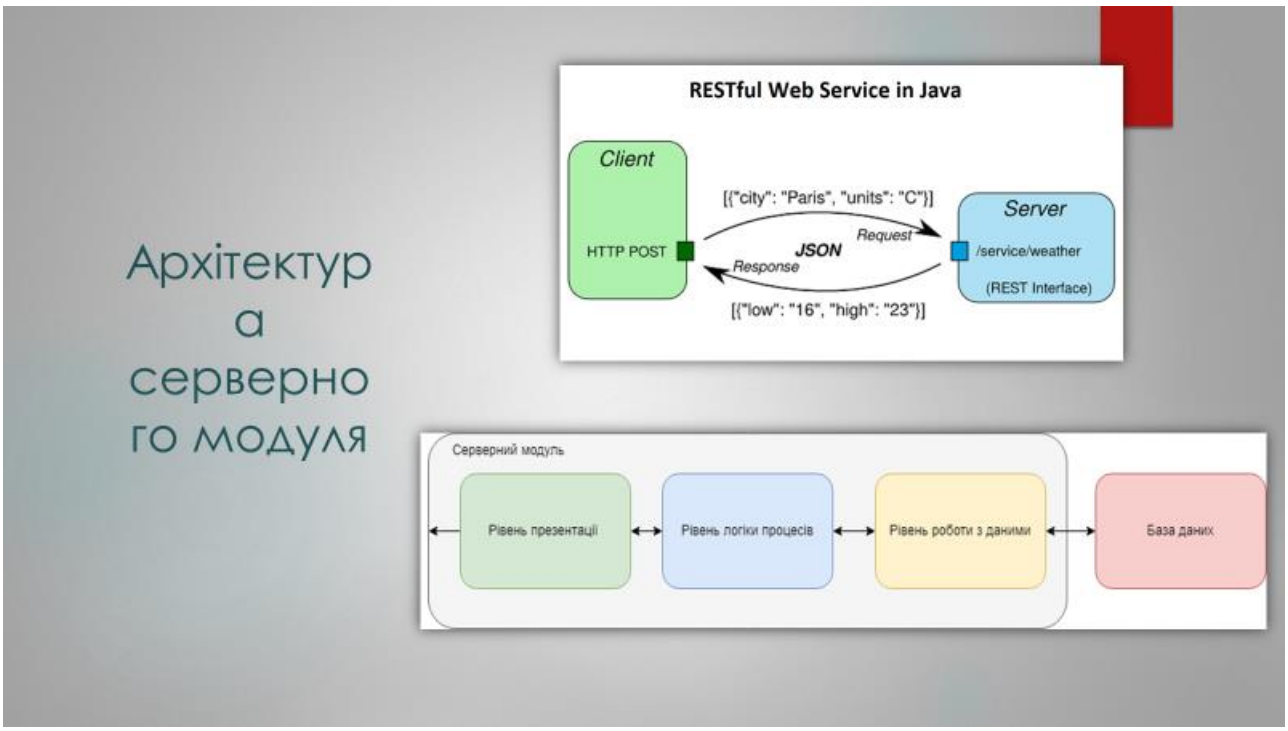

Рисунок Г.5 – Слайд №5

Рисунок Г.6 – Слайд №6

GET	/quiz/{ІД тесту}/question	Отримати тестове питання по ІД тесту	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/question	Додати тестове питання до тесту по його ІД	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/questions	Додати декілька тестових питань до тесту по ІД тесту	200, 400, 401, 403, 404, 5**
GET	/quiz	Отримати сторінку тестів	200, 401, 403, 404, 5**
POST	/quiz	Створити новий тест	200, 400, 401, 403, 404, 5**
PUT	/quiz	Оновити сутність тесту	200, 400, 401, 403, 404, 5**
GET	/quiz/{ІД тесту}/entries	Отримати результати тестування за ІД тесту	200, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/image	Додати головне зображення до тесту по його ІД	200, 400, 401, 403, 404, 5**
POST	/quiz/{ІД тесту}/submit	Подати результати тестування по ІД тесту	200, 400, 401, 403, 404, 5**
GET	/quiz/{ІД тесту}	Отримати тест по його ІД	200, 401, 403, 404, 5**
POST	/register	Зареєструвати нового користувача	200, 400, 5**

Рисунок Г.7 – Слайд №7

Рисунок Г.8 – Слайд №8

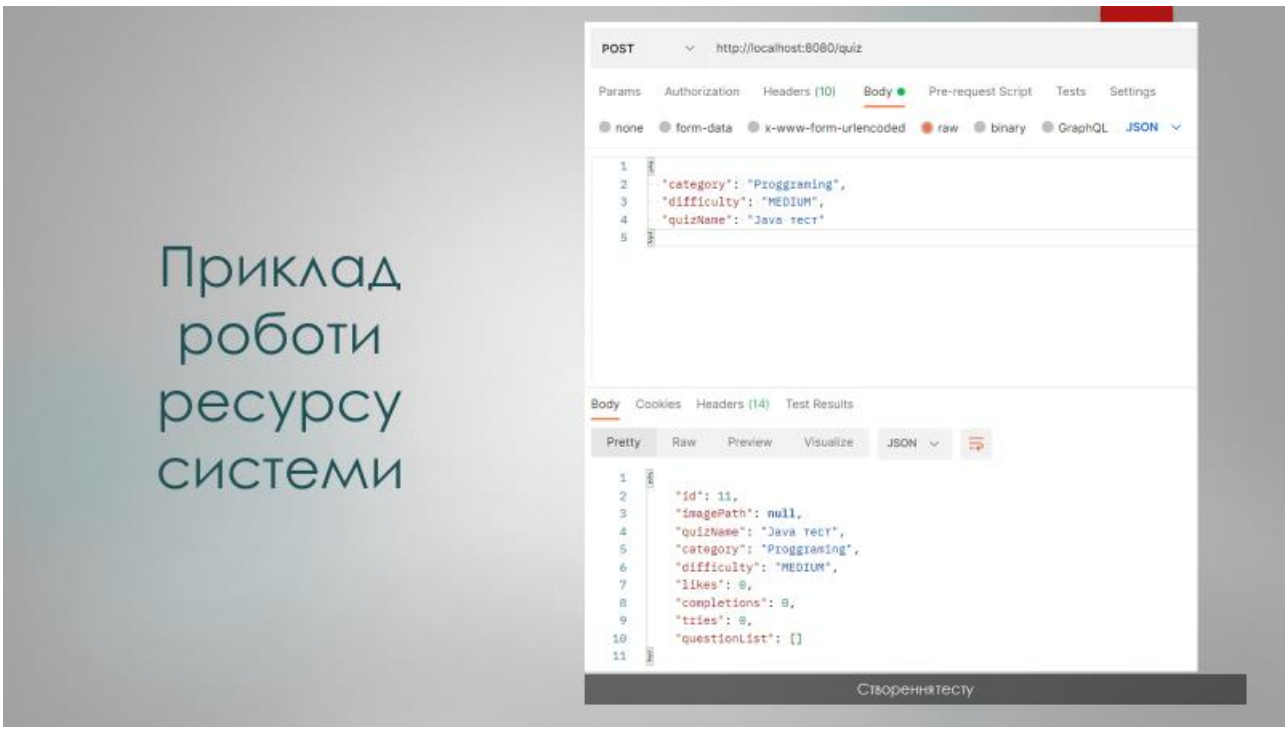

Рисунок Г.9 – Слайд №9

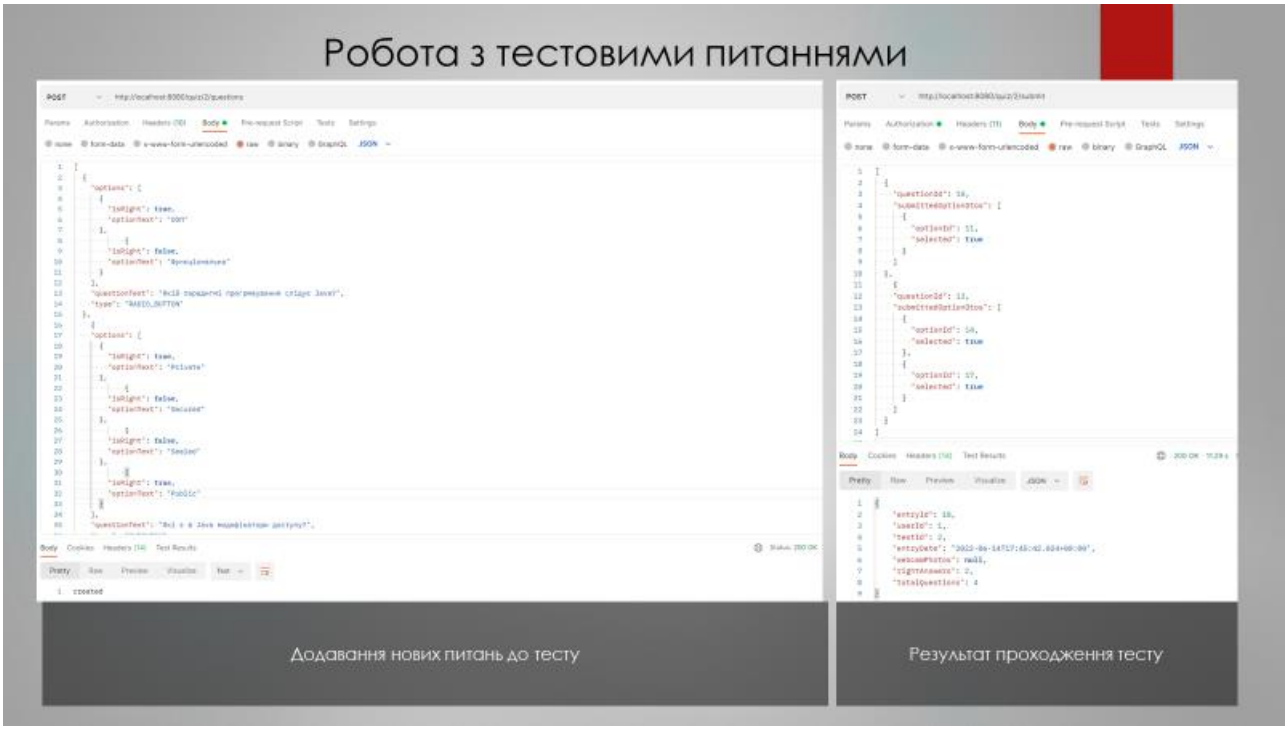

Рисунок Г.10 – Слайд №10

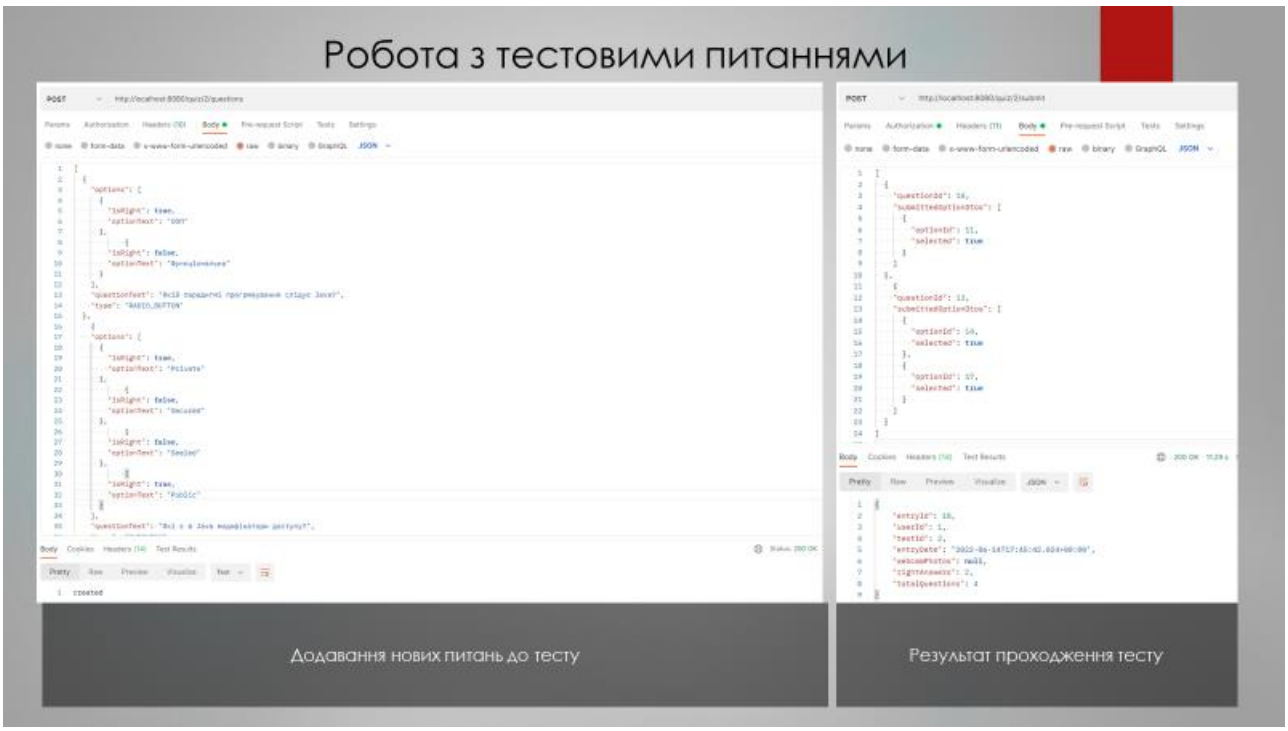

Рисунок Г.11 – Слайд №11

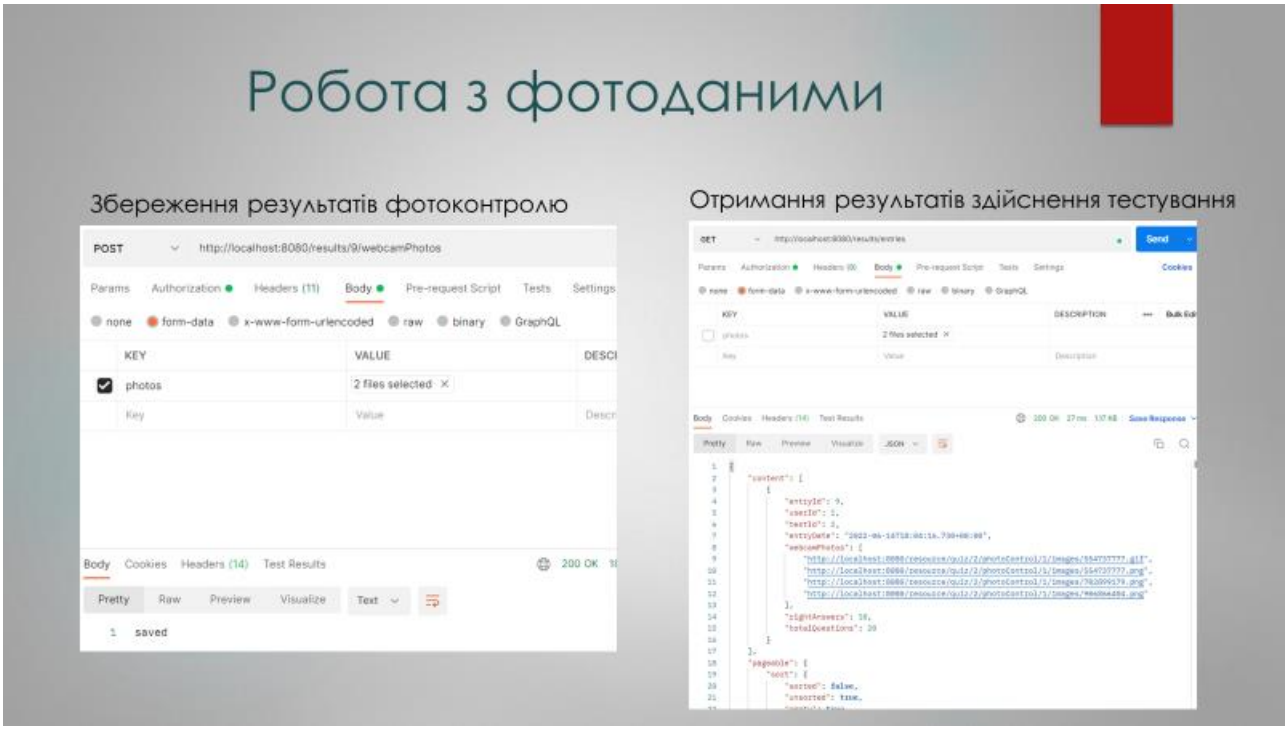

Рисунок Г.12 – Слайд №12

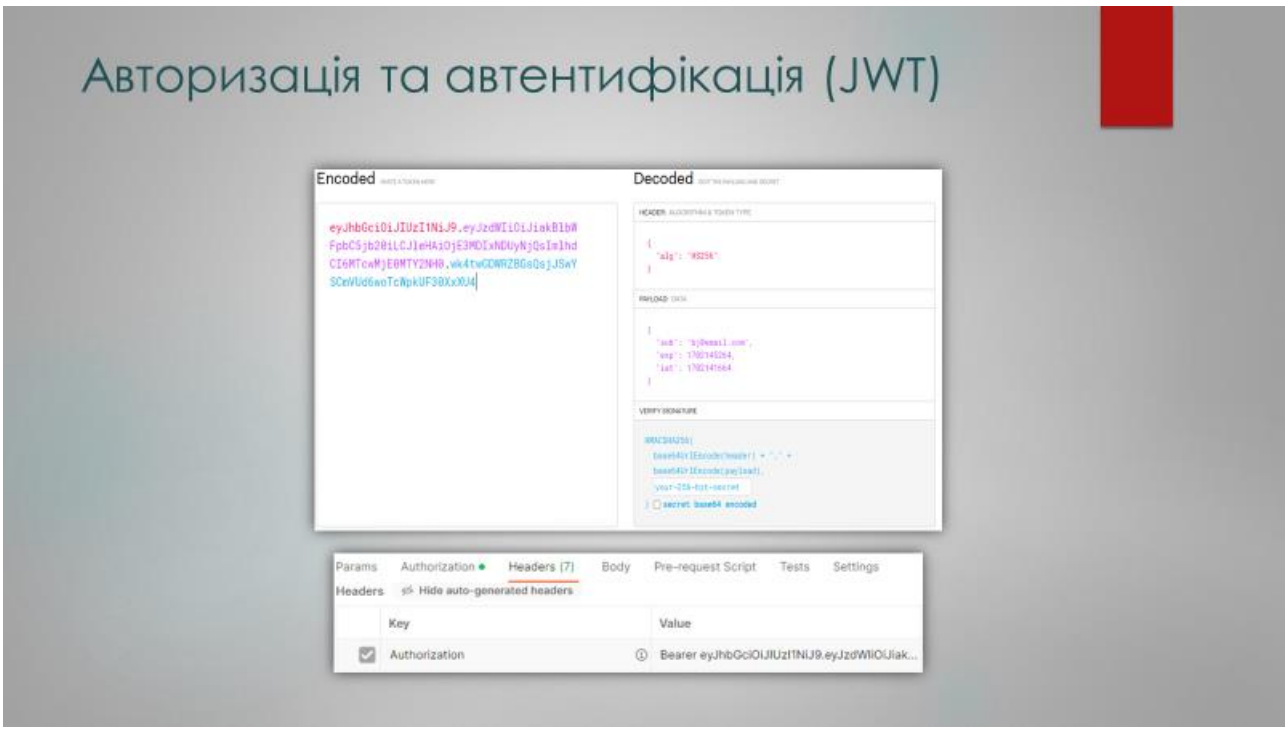

Рисунок Г.13 – Слайд №13

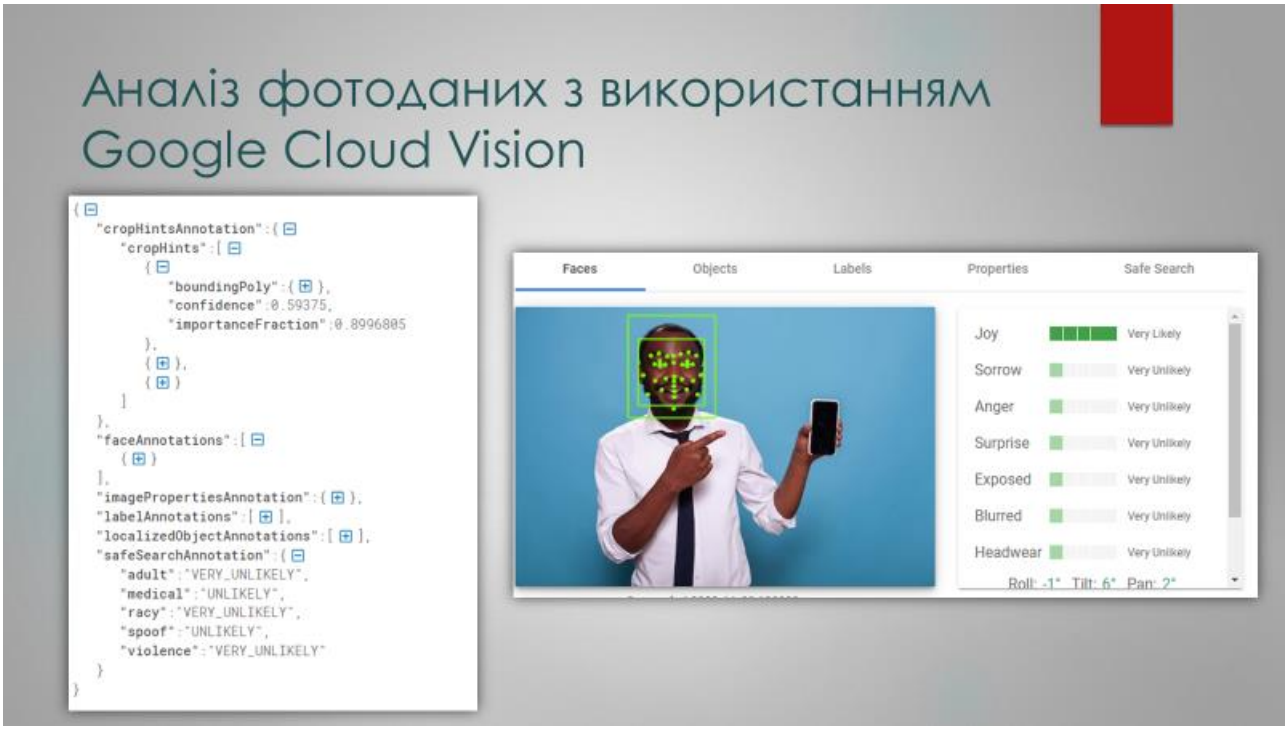

Рисунок Г.14 – Слайд №14

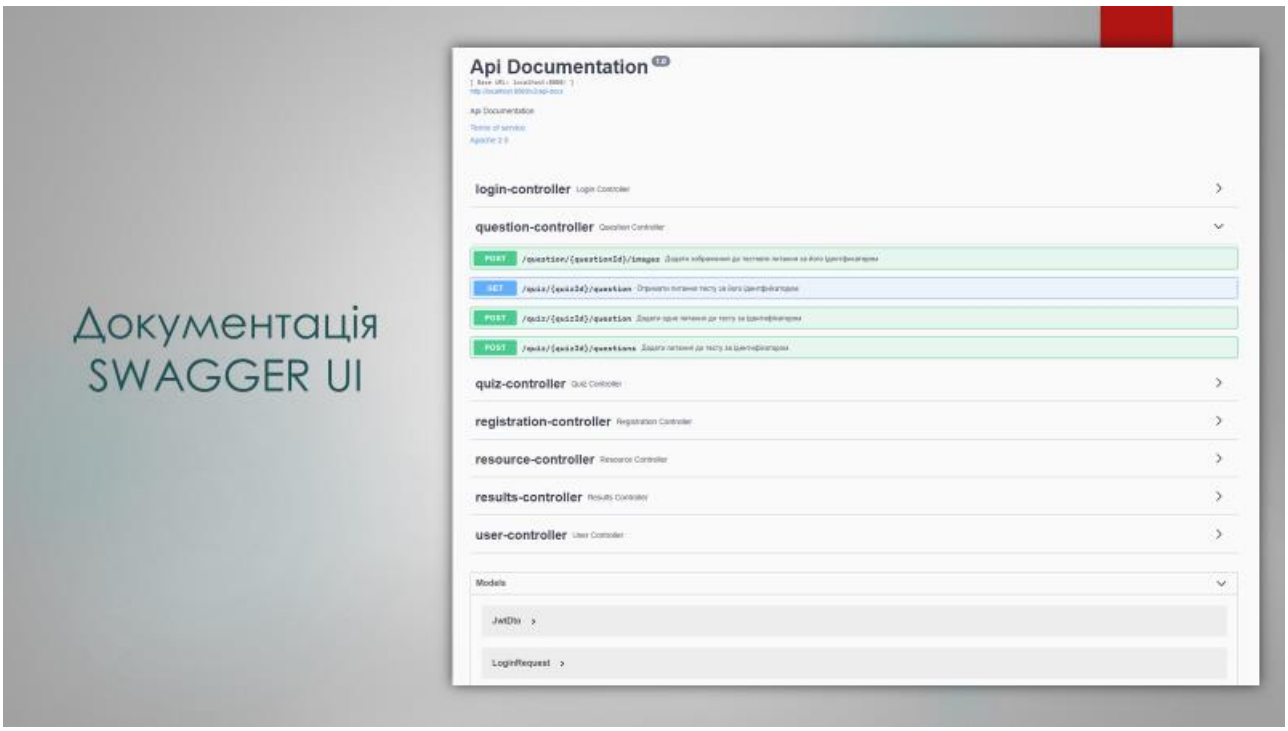

Рисунок Г.15 – Слайд №15

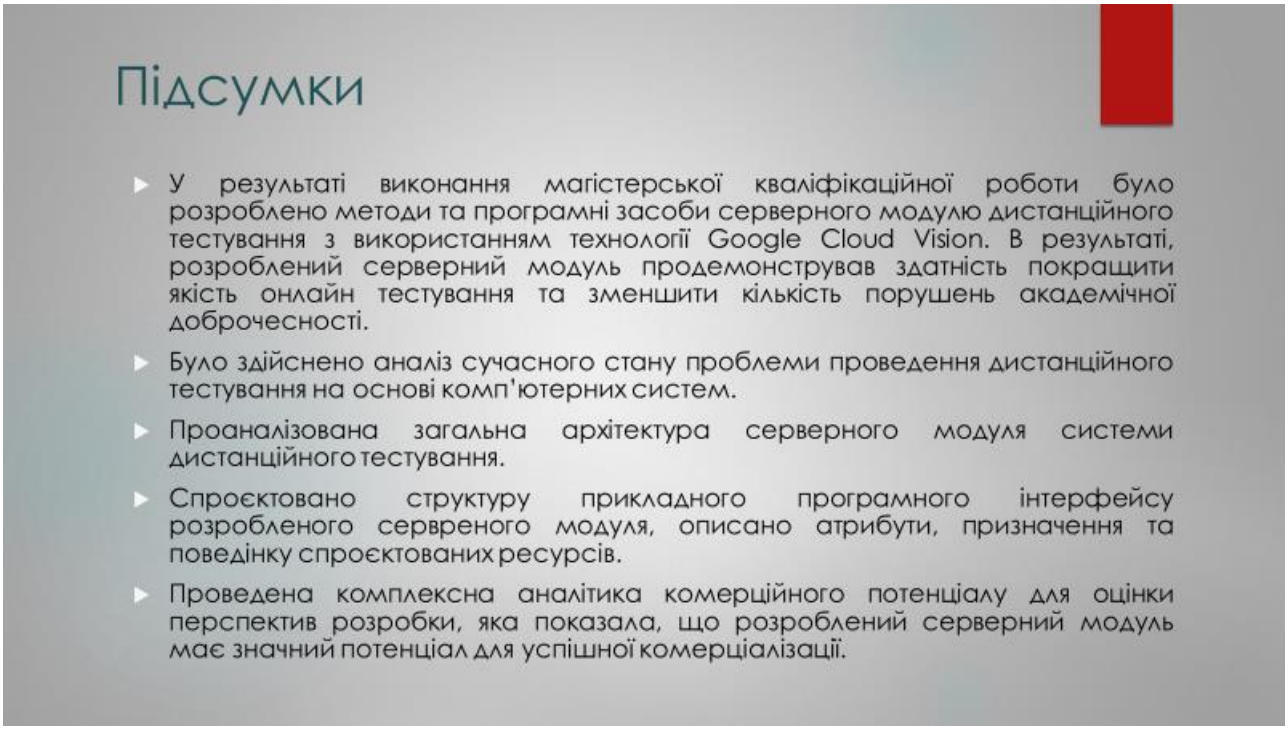

Рисунок Г.16 – Слайд №16