Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до бакалаврської дипломної роботи
бакалавр
(освітньо–кваліфікаційний рівень)

на тему: «Розробка адаптивної тестувальної системи з фотоконтролем. Частина 1. Модуль сервера з використанням технологій Java і фреймворку Spring»

Виконав: студент IV курсу
групи 2ПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Токарчук Д.О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. КН Колесницький О.К.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н. _____
«____» _____ 2022 р.

ВНТУ– 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Ступінь вищої освіти – бакалавр
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
Романюк О. Н._____
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Токарчуку Дмитру Олександровичу

1. Тема роботи – Розробка адаптивної тестувальної системи з фотоконтролем. Частина 1. Модуль сервера з використанням технологій Java і фреймворку Spring.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року №66.

2. Строк подання студентом роботи хх червня 2022.

3. Вихідні дані до роботи: серверний модуль здійснення онлайн тестування з фотоконтролем; Вхідні дані: мова програмування Java, фреймворк Spring, водоспадна модель розробки, середовище розробки IntelliJ IDEA Ultimate Version, прикладний програмний інтерфейс REST, протокол передачі даних HTTP, база даних MySQL.

4. Зміст розрахунково–пояснювальної записки: вступ, аналіз та обґрунтування вибору методу розробки та постановка задачі дослідження, розробка структури та алгоритмів роботи програмного продукту, тестування програми, висновки, список використаних джерел, додатки.

5. Перелік графічного матеріалу: порівняльний аналіз аналогів, блок-схема алгоритму роботи компонентів програми. діаграма залежностей класів, діаграма послідовності роботи додатку, структура прикладного програмного інтерфейсу, програмна реалізація компонентів серверного модуля, тестування програми, апробація та публікації результатів роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д.І., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 10.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи серверного модуля	11.04.2022 – 17.04.2022	Вик.
4	Аналіз і вибір середовища програмування	18.04.2022 – 21.04.2022	Вик.
5	Розробка серверного модуля	22.04.2022 – 15.05.2022	Вик.
6	Тестування роботи розробленого програмного забезпечення	16.05.2022 – 23.05.2022	Вик.
7	Оформлення матеріалів до захисту БДР	24.05.2022 – 10.06.2022	Вик.

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

Керівник бакалаврської дипломної роботи _____ **Катєльніков Д.І.**
(підпис) (прізвище та ініціали)

Рецензент бакалаврської дипломної роботи _____ **Колесницький О.К.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В бакалаврській дипломній роботі розроблено модуль сервера адаптивної тестувальної системи з фотоконтролем. Розроблений програмний продукт призначений для покращення процесу створення та проходження онлайн тестів.

Даний програмний додаток написаний на мові Java з використанням фреймворку Spring, та характеризується універсальністю застосування та захищеністю даних користувачів.

Бакалаврська дипломна робота складається з 85 сторінок формату А4, на яких є 34 рисунків, 3 таблиці, список використаних джерел містить 15 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз стану програмного забезпечення для проведення тестування. Сформульовано мету досліджень – підвищення ефективності та продуктивності процесу тестування людей, забезпечення автентичності результату тесту з допомогою фотоконтролю процесу тестування.

Запропоновано процес здійснення онлайн тестування користувача, на основі різнотипових тектово-візуальних тестових питань й фотоконтролем процесу тестування користувача у формі прикладного програмного інтерфейсу серверного модуля.

Розроблено алгоритми для реалізації процесу онлайн тестування.

ANNOTATION

In the bachelor's thesis the module of the server of adaptive testing system with photocontrol is developed. The developed software product is designed to improve the process of creating and passing online tests.

This application is written in Java using the Spring framework, and is characterized by versatility and security of user data.

The bachelor's thesis consists of 85 pages of A4 format, which have 34 figures, 3 tables, the list of sources used contains 15 names.

In the bachelor's thesis a detailed analysis of the software for testing was conducted. The purpose of research is formulated - to increase the efficiency and productivity of the testing process of people, to ensure the authenticity of the test result through photocontrol of the testing process.

The process of online user testing is proposed, based on various types of text-visual test questions and photo-control of the user testing process in the form of an application software interface of the server module.

Algorithms for the implementation of the online testing process have been developed.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. В сучасному світі, веб технології стали невід'ємною частиною життя переважної частини людей, у більшості країн світу. Інформаційні та веб технології надають змогу автоматизувати та поліпшити різноманітні процеси та задачі, які ще декілька десятиліть тому, вимагали значних зусиль для їх здійснення, наприклад фінансові операції, обмін інформацією, знаходження нової інформації, тощо.

Інформаційна технології не обійшли сферу освіти. Існування власного веб сайту у навальних закладів вже давно стало звичністю. Такі веб ресурси слугують потужним засобом у процесі навчання та його організації. Це для прикладу публікація новин, створення оголошень, збереження пам'яток чи корисної інформації для студентів чи абітурієнтів. Також такі інструменти безпосередньо залучають в процес навчання, це для прикладу створення онлайн журналів, чи засобів для контролю знань.

Одним із видів онлайн контролю знань це тести. Перенесення тестів з паперу у веб середовище надає змогу автоматизувати їх проходження та оцінювання й виключає більшість людських факторів, які можуть виникнути при проведення звичайного тестування в навчальній аудиторії. Слід зауважити, що у форматі онлайн тесту різко підвищується можливість обману та списування при проходження тестування. Але ця проблема стає менш вагомою, так як поширене використання відео чи фото фіксації учасника тестування. Тільки завдяки цим перевагам, проходження тестувань онлайн може відгравати важливу роль у процесі навчання.

Цікавим є те, що онлайн тести стають у пригоді не тільки для учасників навчально процесу, а й у пересічних користувачів веб мережі, у яких є бажання чи потреба перевірити свої знання у певній галузі. Для прикладу дізнатися свій рівень знання певної іноземної мови. Саме тому, темою бакалаврської роботи

було обрано розробку модулю сервера адаптивної тестувальної системи з фотоконтролем.

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 розробка серверного модулю адаптивної тестувальної системи з фотоконтролом для поліпшення та процесу здійснення онлайн тестування.

Основними задачами дослідження є:

– визначення багатошарової структуру та архітектуру серверного модуля;

– моделювання структури реляційної бази даних;

– створення сервісних класів та компонентів серверного модулю;

– створення приладного програмного інтерфейсу серверного модулю з використанням мови програмування Java та інструментів фреймворку Spring;

– розробка програмних модульних тестів;

– ручне тестування роботи розробленого прикладного програмного інтерфейсу;

– створення документації для розробленого прикладного програмного інтерфейсу серверного модуля.

Об’єкт дослідження – процес здійснення онлайн тестування з фотоконтролем.

Предмет дослідження – методи та засоби розробки адаптивної тестувальної системи з фотоконтролем.

Методи дослідження. У процесі дослідження використовувались: аналіз існуючих технічних рішень задачі, з врахування переваг та недоліків для формування технічного завдання та вимог; засоби програмного проектування для розробки програмного додатку адаптивної тестувальної системи з

фотоконтролем; засоби та методи тестування програмного забезпечення для перевірки роботи програмного продукту.

Наукова новизна отриманих результатів.

1. Запропоновано реалізацію серверного модулю для здійснення онлайн тестування з підтвердження автентичності результатів з допомогою фотоконтролю та адаптивністю з можливістю створення різнотипових тестів,.

2. Запропоновано реалізацію прикладного програмного інтерфейсу серверного модулю з використання протоколу передачі даних HTTP з використанням протоколу обміну даних HTTP та дотриманням методології методології REST, з обміном даних у форматі JSON.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено серверний модуль адаптивної тестувальної системи.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: Розробка серверної частини адаптивної тестувальної системи з фотоконтролем з використанням технологій Java та фреймворку Spring [1].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися на LI Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022 р., м. Вінниця);

Публікації. За тематикою дослідження опубліковано 1 наукову працю праці у збірниках матеріалів конференцій.

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

Здійснення тестування людини, це важлива складова різних галузей та сфер сучасної діяльності людства. Ще тисячу років назад в людей в різних країнах виникла потреба проведення іспиту для людей. Характерною особливістю таких ранніх способів перевірок знань була їх усна форма, та відсутність якоїсь певної стандартизації, чітких критерій та встановлених питань.

Природнім є те, що з плином часу методики та способи проведення розвивалися. Тільки в останньому століття в передових країнах світу стали з’являтись перші стандартизовані іспити. В результаті, на даний час, існують різні інструменти та способи оцінки знань людини.

Сучасний тест може складатися з тестових завдань [2], складових тесту, які мають означену складність та відповідають певній встановленій метриці. Поширені тестові завдання таких типів:

- завдання з множиною варіантів;
- задання-доповнення;
- завдання множинного вибору;
- завдання вільного викладу;
- завдання альтернативних відповідей;
- завдання на встановлення відповідності;
- завдання на встановлення правильної послідовності.

Набір цих тестових завдань створюють оцінну шкалу, за допомогою якої можна оцінити рівень знань особи.

З розвитком інформаційних технологій, проведення онлайн тестуваннь для людей, загалом, стало звичною практикою. Причому, мова може йти не

тільки про педагогічне тестування, а й про тестування психологічне чи соціологічне.

Така форма проведення тестування дає змогу екзаменатору значно прискорити та спростити процес його проведення. В умовах всесвітньої пандемії та переходу навчання на віддалений онлайн формат, онлайн тестування набуває ще більшої актуальності, в якості інструмента викладачів начальних заходів [3].

1.2 Порівняльний аналіз аналогів

В наслідок попиту на здійснення онлайн тестування, існує певний перелік популярних електронних інструментів для проведення тестування. Тому було проведено порівняльний аналіз деяких наявних сервісів аналогів.

Cambridge English Language Assessment – веб сайт відомого Кембриджського університету в Англії, який надає можливість проведення тестування рівня англійської мови [4] (рис. 1.1). На сайті наявні чотири види тестування залежно від бажаної галузі застосування мови. Після проходження іспиту, сайт надає доволі розгорнутий результат проходження тесту.

Однак може зазначити не функціональність ресурсу, так як не використовується окремий модуль серверу та те, що тестування стосується тільки однієї теми – тестів з англійської мови.

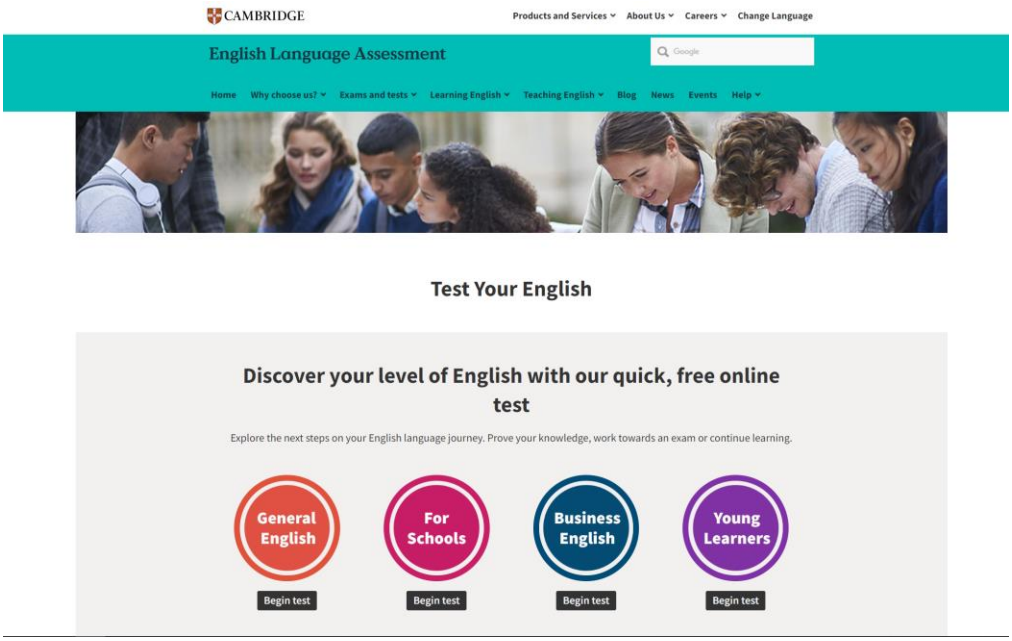


Рисунок 1.1 – Веб сайт Cambridge English

Online Test Pad – веб ресурс для проходження та створення тестів, з різних тем та галузей [5] (рис. 1.2). Сайд є спеціалізованим майданчиком для тестування, та надає цікаві можливості, наприклад обговорення наявних тестів.

Недоліком ж виступає відсутність окремого модулю сервера, який міг би стати єдиним програмним інтерфейсом для різних клієнтів, та розділити весь додаток на два окремі модулі.

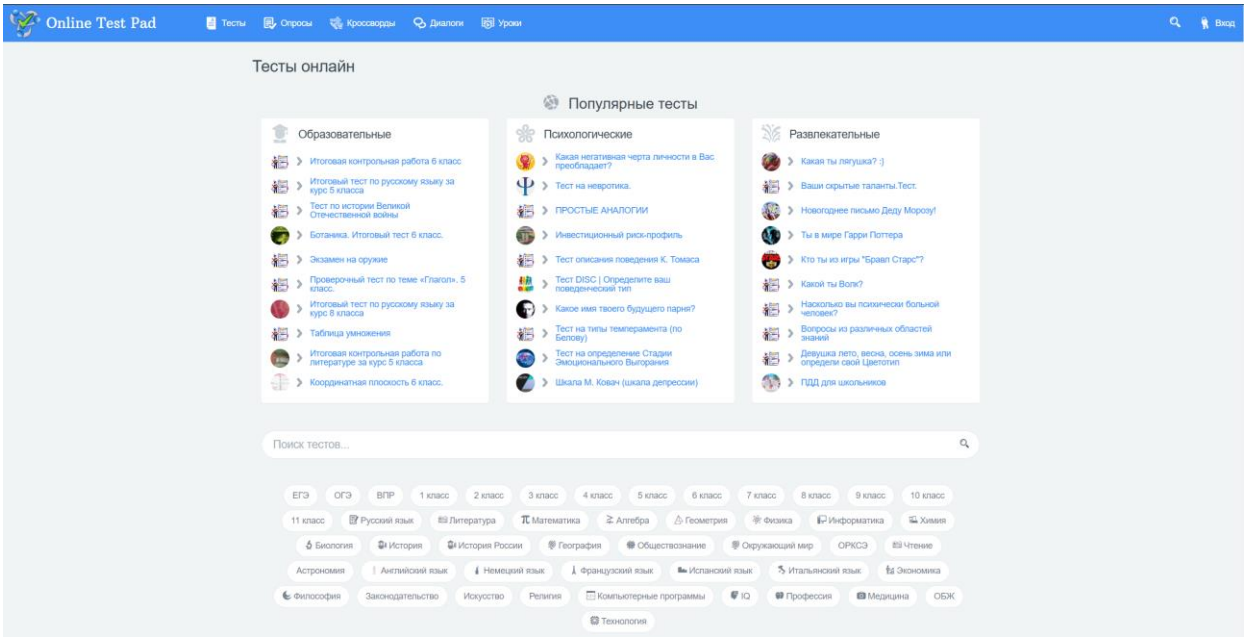


Рисунок 1.2 – Веб сайт Online Test Pad

Ще один аналог - веб ресурс test-english [6] (рис. 1.3). Як і один з попередніх аналогів, даний сайт надає широкий перелік тестів на тему англійської мови. На вибір користувачу надається перелік тем, такі як граматика, слухання, читання, писання, використання слів, та складність. Під час проходження з’являються тестові питання в різних формах. Також є можливість дізнатися певні правила, які стосуюсь обраної теми. Після завершення тестування надається результат тестування.

Знову ж таки недоліком є те, що сайт спеціалізується на іспитам по темі англійської мови, та не має окремого серверного модуля.

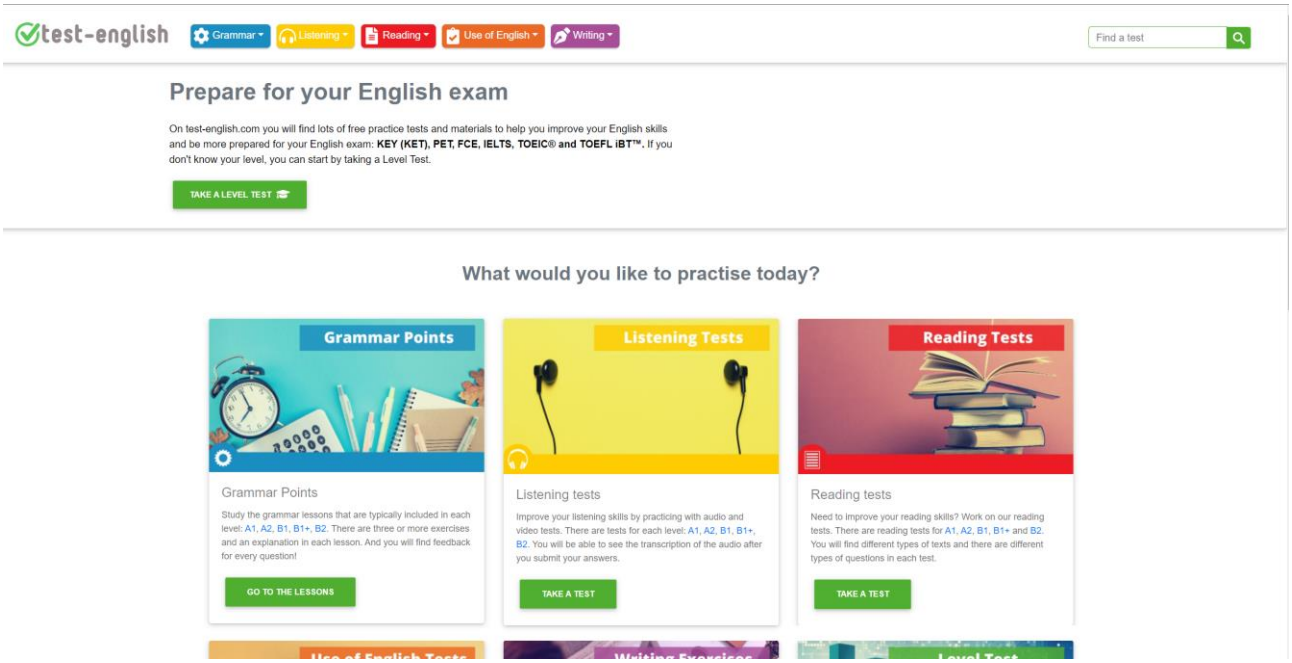


Рисунок 1.3 – Веб сайт test-english

В результаті, розглянувши наведені аналоги, складемо порівняльну таблиці аналогів, їх переваг та недоліків. Розглянемо таблицю 1.1.

Таблиця 1.1 – Порівняльна таблиця аналогів та розроблюваного додатку.

Критерій	Розроблюваний додаток	Cambridge English	Online Test Pad	test-english
Проходження тестів	+	+	+	+
Наявність окремого серверного модулю	+	-	-	-
Розподіл тестів по різним темам	+	-	+	-
Створення нових тестів	+	-	+	-

Продовження таблиці 1.1

Створення профілю користувача	+	-	+	+
Різні види тестових питань	+	+	+	-
Наявність додаткового захисту (фотофіксація)	+	-	-	-

Отже, після аналізу таблиці 1.1, можна зробити висновком, що додаток який розробляється в рамках бакалаврської роботи, є більш універсальним для використання і для проходження та для створення тестів, та надає додаткові можливості контролю проходження (фотофіксація).

1.3 Вибір моделі життєвого циклу для розробки додатку

У процесі розробки програмного забезпечення важливу роль відіграє обрана технологія розробки програмного забезпечення [7]. Розробка складних додатків вимагає злагодженості, чіткості та визначеної послідовності у процесі розробці. Якщо ж команда не притримується певної технологій, втрачається продуктивність, що в результаті може бути серйозним недоліком при розробці всього програмного продукту. В результаті виникло безліч різних методологій розробки програмного забезпечення. Будь яка з методологія має свою певну сферу використання, та не являється стовідсотково універсальною у застосуванні для будь яких проєктів.

Розглянемо найпоширеніші моделі життєвого циклу програмного забезпечення.

Водоспадна, каскадна чи послідовна модель. Дана модель розбиває весь життєвий цикл розробки програмного забезпечення на шість чітко визначених етапів розробки:

- Формування вимог;
- Проєктування;

– Реалізація;
– Тестування;
– Запровадження;
– Супровід.

Кожен з цих етапів починається тільки після закінчення попереднього, таким чином формується чіткий ланцюг послідовних процесів (рис. 1.4).

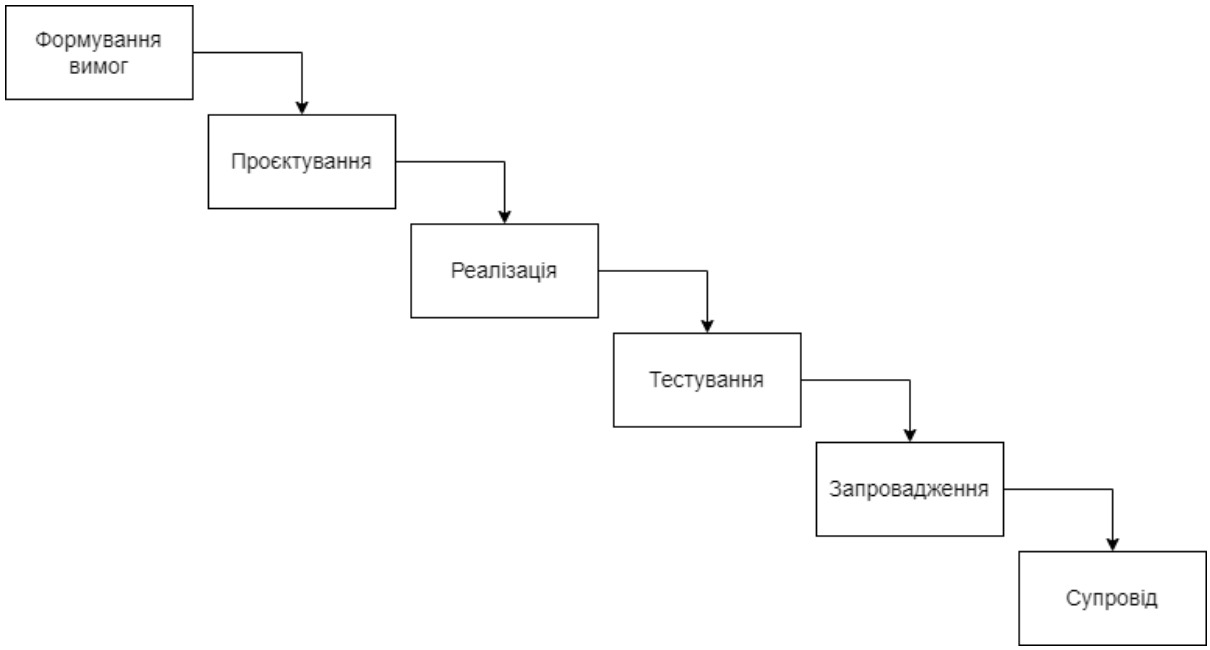


Рисунок 1.4 – Водоспадна модель

Для чіткого проведення таких етапів формується чітка технічна документація та фіксується весь процес їх проходження. Перевагами такої моделі є чітка визначеність на будь-якому етапі розробки, та детальна технічна документація. Команда може чітко розуміти поставленні задачі. Також доволі легко оцінювати чіткі терміни та витрати на розробку.

Спіральна модель – модель процесу розробки програмного забезпечення. Основною особливістю цієї моделі є те, що при використанні цієї моделі програмне забезпечення створюється в кілька ітерацій (витків так званої спіралі) методом прототипування. Кожна ітерація, чи виток спіралі, відповідає за створення прототипу або версії програмного забезпечення. Під час ітерації

встановлюються цілі та характеристики проекту, оцінюється якість отриманих результатів та плануються роботи для наступної ітерації (рис 1.5).

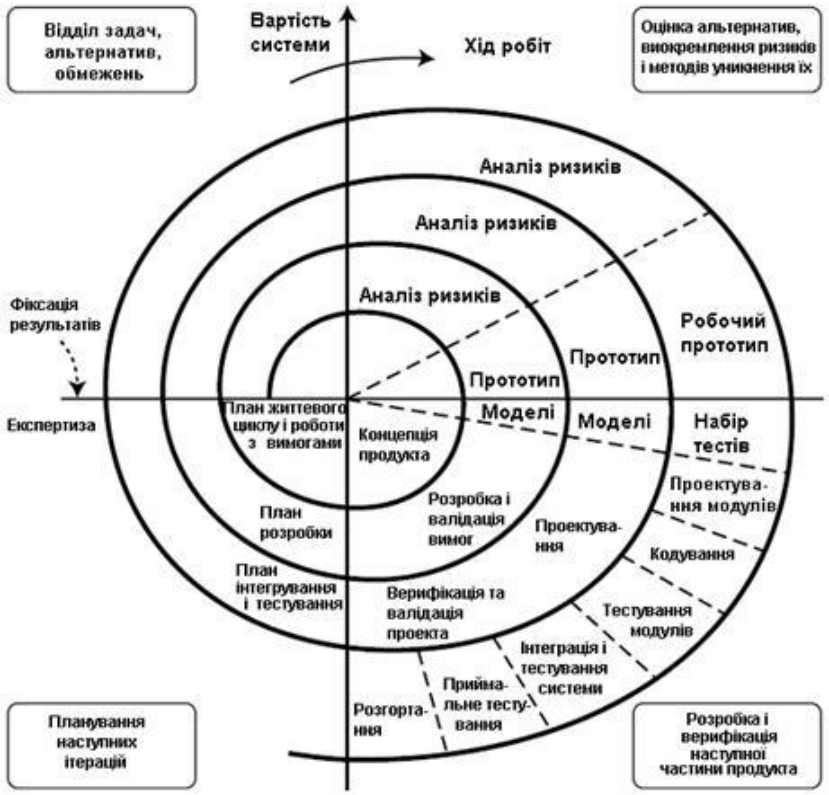


Рисунок 1.5 – Спіральна модель

Для кожного витка спіралі оцінюються певний перелік питань, наприклад:

– ризик перевищення термінів та вартості проєкту;
– чіткість поставленого технічного завдання для його реалізації;
– потреба у наступній інтерації;
– доцільність завершення роботи над проєктом.

Перевагою такої моделі являється її відносна гнучкість, та розділення всього процесу розробки на безліч ітерацій, кожна з яких є результатом певного готового програмного продукту. Через таку гнучкість технічне завдання має змогу змінюватися без значних наслідків. Одночасно це є й недоліком, так як немає чітких меж виконаних робіт, тому не завжди вдається чітко починати та

закінчувати нові витки спіралі, та часто необхідно спиратися на досвід досвідчених розробників.

В результаті, в якості моделі життєвого циклу у дані бакалаврські роботі, було обрано водоспадну модель. Дана модель цілком підходить під характер роботи яка виконується, адже наявна чітка постановка задач, та відхилення від нього існуючих завдань можуть бути незначні.

1.4 Постановка задачі

Модуль сервера адаптивної тестувальної системи з фотоконтролем передбачає у собі серверний модуль, який надає функціональність з створення тестів, їх проходження та збереження всієї інформації користувачів та пов’язаної з тестуванням, у базі даних. Модуль повинена передбачати з себе програмний інтерфейс, побудований з застосуванням протоколу передачі даних HTTP, для комунікації з іншими клієнтськими частинами через мережу Інтернет.

Повинно підтримуватися створення та збереження нових тестів за різними встановленими категоріями. Модуль повинен давати можливість створювати питання для тестів різних типів, таких як тести з однією відповіддю, тести з декількома відповідями, тести з відкритою відповіддю. Повинно бути реалізовано збереження медіа даних, а саме зображень, наприклад для тестових питань. Повинна бути реалізована можливість отримання зображення за його шляхом збереження в системі та назвою.

Має бути реалізований пошук тестів з фільтрацією за назвою й категорією, та наявною пагінацією знайдених результатів, тобто розділення результатів на умовні порції чи сторінки, для запобігання передачі над великих масивів даних.

Модуль повинен опрацьовувати проходження тесту користувачем. При отриманні запиту на зарахування результату тесту, програмний інтерфейс повинен правильно обрахувати результат відповідей, враховуючи типи

тестових запитань, які наявні в пройденому тесті, та зберегти їх у базі даних. Програмні клієнти повинні мати змогу отримувати інформацію по результатам проходження відповідного тесту. Також по запиту та за наявності, повинні зберігатися зафіксовані зображення особи яка проходила тестування.

Повинна бути наявна можливість реєстрації користувача в системі. Зареєстрованим користувачам, по запиту, сервер повинен генерувати веб токен, спеціальну текстову стрічку, яка є засобом для авторизації в системі. Додаток повинен використовувати авторизацію та автентифікацію клієнтських модулів за допомогою веб токенів, які посвідчують право отримання доступу до певного ресурсу сервера. Доступ до ресурсів не повинен бути публічним, тільки за наявності токена з відповідними правами, для запобігання витікання вразливої інформації.

Отже було встановлено основні цілі та задачі, які повинен виконувати модуль сервера адаптивної тестувальної системи з фотоконтролем.

1.5 Висновки

У даному розділі було проведено аналіз сучасного стану проблеми та обґрунтовано потребу у розробці модуля сервера адаптивної тестувальної системи з фотоконтролем

Було розглянути декілька основних аналогів системі, яка розробляється, а саме веб сайти: Cambridege English, Online Test Pad, test-english. Було здійснено порівняльний аналіз функціональних особливостей додатку що розробляється та аналогів. В результаті було зроблено висновок, що додаток який розробляється у дані бакалаврській роботі, матиме в собі усі особливості розглянутих аналогів, та виступатиме універсальний багатофункціональним рішенням.

Було розглянуто сучасні моделі життєвого циклу для розробки програмного забезпечення. В якості моделі для розробки було обрано водоспадну модель, та обґрунтовано сам виріб.

Наприкінці, була здійснена постановка задач, яка визначає основні функціональні особливості додатку що розробляється, які повинні бути виконанні в результаті роботи.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка загальної архітектури продукту

При розробці серверних застосунків з використанням мови Java та фреймворку Spring, існує типова архітекта таких додатків [7]. Загальна архітектура додатку, що розробляється, базується на розділенні всіх компонентів та класів серверу на три архітектурних шари:

- Шар контролерів (веб шар);
- Шар сервісів;
- Шар репозиторіїв.

Робота додатку відбувається послідовно від верхнього шару – шару контролерів, до нижнього – шару репозиторіїв (рис. 2.1).

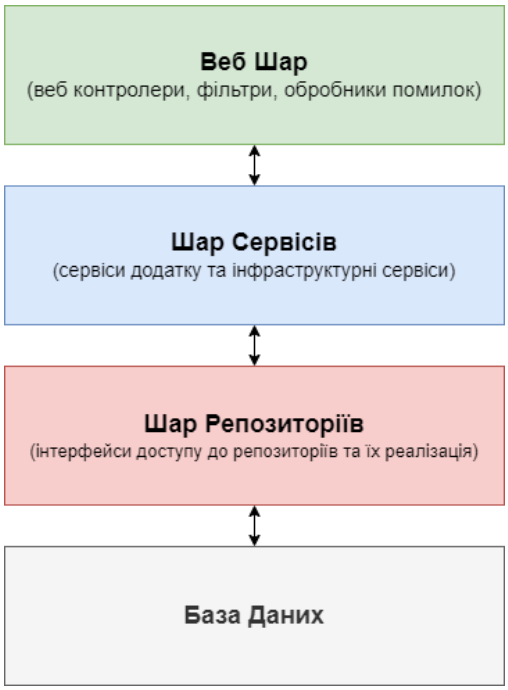


Рисунок 2.1 – Архітектура розробленого додатку

При такої структурній архітектури додатку, вдається чітко розподілити модулі додатку за їх призначення та функціями.

Шар контролерів представляє з себе набір спеціальних класів контролерів та їх методів вихідних точок, які виступають обробниками HTTP запитів програм-клієнтів розробленого серверного модуля та його фасадом [8]. Розроблені контролери отримують в якості аргументу об’єкт HTTP запиту та його параметри й тіло, для подальшого виклику модулі зі сервісного шару. Слід зауважити, що досягти такої поведінки контролерів дозволяє існування диспетчера контролерів, який відповідальний за делегування виконання запитів відповідним контролерам, який виконаний за шаблоном проєктування єдиної точки входу. Також важливим елементом життєвого циклу є фільтри запитів, які налаштовані обробляти вхідні запити, та зупиняти їх виконання у разі невдалої аутентифікації. Схематичне зображення роботи розроблених контролері можна розглянути на рисунку 2.2.

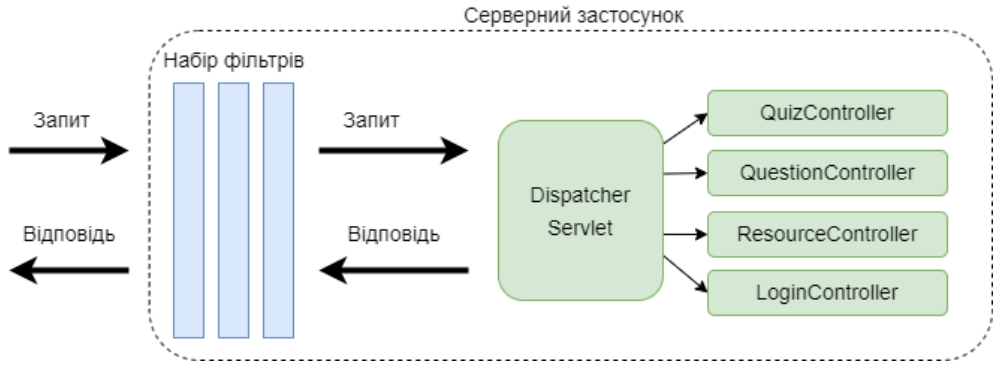


Рисунок 2.2 – Схематичне зображення роботи контролерів

Модулі шару сервісів відповідають за виконання безпосередньо основної логіки розробленого серверного модуля. Дані модулі здійснюють обробку таких процесів як реєстрація користувача, авторизація користувача, створення нового тесту, оновлення тесту, обробка результатів проходження тесту, пошук історії проходження тесту знаходження медіа даних на робочому середовищі сервера, тощо.

Для доступу до даних, які зберігаються в базі даних, сервісні модулі опираються на роботу модулів репозиторіїв. Класи репозиторіїв містять в собі

програмний код, який описує основні операції по доступу до бази даних серверного застосунку. Основними операціями виступає збереження, отримання, оновлення та знищення записів у базі даних.

На рисунку 2.3 можем розглянути типову поведінку серверного модуля та його клієнта у вигляді діаграми послідовності.

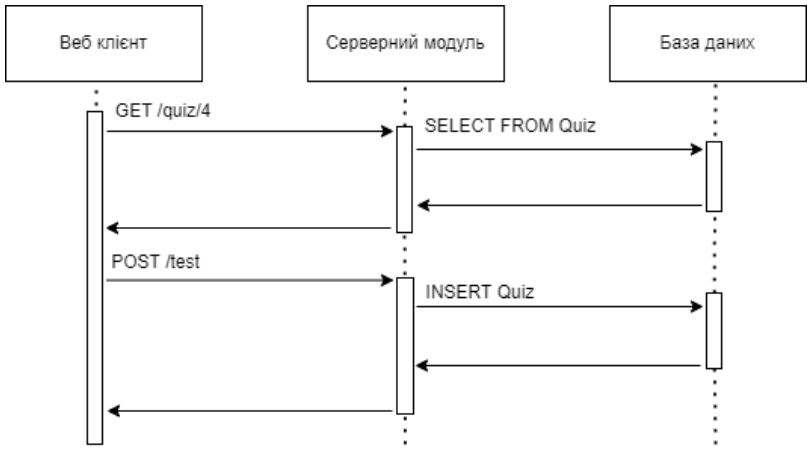


Рисунок 2.3 – Діаграм послідовності виклику ресурсів

На даній діаграмі відображено процес отримання інформації про тест з ідентифікатором 4, та створення зовсім нового тесту.

Також, на рисунку 2.4 можна побачити схематичне відображення залежносте між компонентами архітектурних шарів у вигляді діаграми класів.

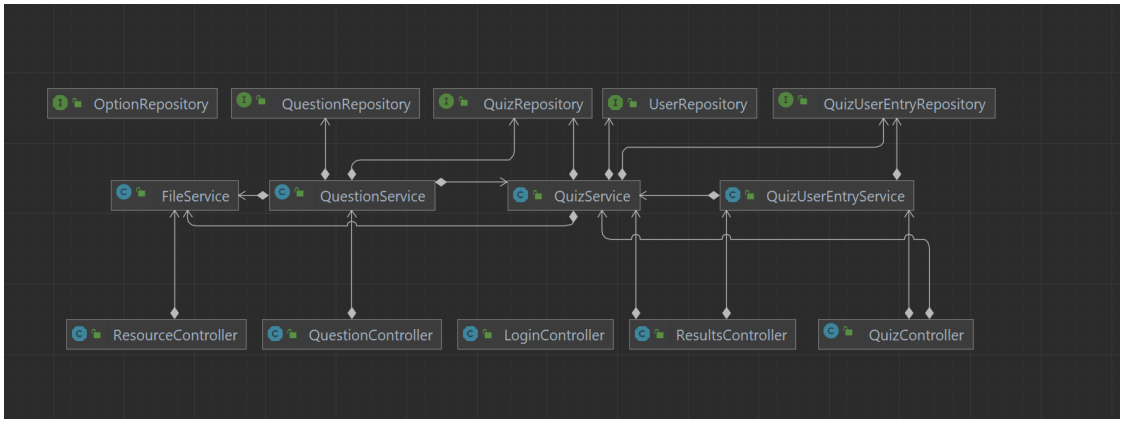


Рисунок 2.4 – Схематичне зображення структури архітектури

2.2 Проєктування бази даних

В якості сховища даних для розробленого серверного додатку було обрану SQL базу даних з використанням СУБД MySQL. MySQL – реляційна система управління базами даних. Розробкою бази даних займається корпорація Oracle [9]. Продукт розповсюджується як під ліцензією GNU General Public License, так і під власною комерційною ліцензією. MySQL є рішенням для малих та середніх програм. Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць.

В результаті проєктування бази даних, було розроблено структуру базу даних з якою буде працювати розроблений серверний модуль. На рисунку 2.4 зображено схематичне зображення 6 створених таблиць.

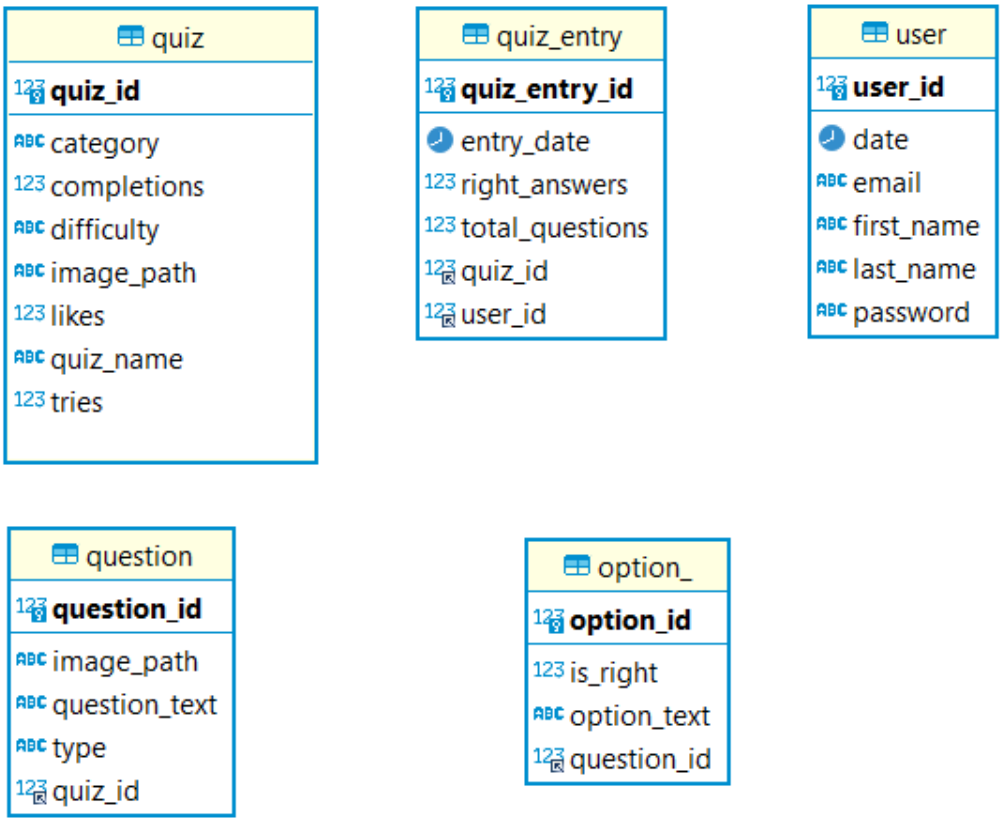


Рисунок 2.4 – Зображення розроблених таблиць

Розглянемо призначення кожної таблиці по окремості.

Таблиця User призначення для збереження даних кінцевих користувачів серверного застосунку та для використання в якості зовнішнього ключа для тестів. Поля таблиці:

– User_id – унікальний ідентифікатор користувача. Первинний ключ.

– Date – дата реєстрації користувача;

– Email – електронна адреса користувача, вказана під час використання;

– First_name – ім’я користувача, вказане під час реєстрації;

– Last_name – фамілія користувача, вказана під час реєстрації;

– Password – пароль користувача, вказаний під час реєстрації. Зберігається у зашифрованому виді.

Наступна таблиця Quiz, це таблиця яка використовується для збереження загальної інформації про створені тести. Поля таблиці:

– Quiz_id – унікальний ідентифікатор тесту. Первинний ключ.

– Category – категорія, до якої відноситься тест;

– Completitions – загальна кількість проходження тесту всіма користувачами;

– Difficulty – складність тесту;

– Image_path – шлях до зображення чи зображень;

– Likes – кількість вподобань тесту від користувачів;

– Quiz_name – назва тесту;

– Tries – максимальна кількість спроб проходження тесту для одного користувача;

Таблиця Question містить дані про тестові питання відповідного тесту. Поля таблиці:

– Question_id – унікальний ідентифікатор питання. Первинний ключ.

– Image_path – шлях до зображення чи зображень;

– Question_text – текстовий зміст питання;

– Type – різновид питання;

– Quiz_id – зовнішній ключ до таблиці Quiz. Вказує на асоціацію з відповідним тестом.

Наступна таблиця Option використовується для збереження варіантів вибору для тестових запитань. Поля таблиці:

– Option_id – унікальний ідентифікатор варіанту питання. Первинний ключ;

– Is_right – мітка чи є варіант вірним;

– Option_text – текстовий зміст. варіанту відповіді;

– Question_id – зовнішній ключ до таблиці Question. Вказує на асоціацію з відповідним тестовим питанням.

Остання таблиця Quiz_Entry створена для зберігання інформації про результат завершення тесту користувачем системи. Поля таблиці:

– Quiz_entry_id – унікальний ідентифікатор результату. Первинний ключ;

– Entry_date – дата завершення тесту користувачем;

– Right_answers – кількість правильних відповідей;

– Total_questions – загальна кількість питань в тесті на момент його завершення;

– WebcamPhotos – шлях у файловій системі серверного додатку, за якийм зберігаються фото отрмиані у результаді здійснення фотоконтроля;

– Quiz_id – зовнішній ключ до таблиці Quiz. Вказує на асоціацію з відповідним тестом;

– User_id – зовнішній ключ до таблиці User. Вказує на асоціацію з відповідним користувачем.

2.3 Розробка структури прикладного програмного інтерфейсу серверного модуля

Важливим етапом розробки є саме проєктування серверного модуля програмного інтерфейсу [10], який є ключовим компонентом при роботі з

програмами-клієнтами. При проєктуванні програмного інтерфейсу серверного модуля адаптивної тестувальної системи, було зроблено вибір сучасного та розповсюдженого архітектурного стиля побудови серверного додатку REST (Representational State Transfer).

REST – це набір правил того, як програмісту організувати написання коду серверної програми, щоб усі системи легко обмінювалися даними і програму можна було масштабувати [11]. Для веб-служб, побудованих з урахуванням REST, тобто які не порушують обмежень, що накладаються ним, застосовують термін RESTful. Незважаючи на те, що REST не є стандартом сам по собі, більшість RESTful реалізацій використовують такі звичайні стандарти, як HTTP, URL, JSON. Даний підхід оперує не тільки основними HTTP методами GET та POST, для отримання та передач інформації відповідно, алей розширює їх число додатковими методами PUT та DELETE, для оновлення та видалення даних відповідно. Перевагами підходу REST виділяють наступні пункти:

- Надійність, за рахунок відсутності необхідності зберігати інформацію про стан клієнта, яка може бути втрачена;
- Масштабованість;
- Прозорість системи взаємодії;
- Простота інтерфейсів;
- Портативність компонентів;
- Легкість внесення змін;
- Здатність еволюціонувати , пристосовуючись до нових вимог.

Отже спираючись на правила підходу проєктування серверних застосунків REST, було сформовано програмний інтерфейс серверного модуля тестувальної системи, який складається з набору вихідних точок. Наведемо приклад сформованих вихідних точок у таблиці 2.1.

Таблиця 2.1 – Вихідні точки прикладного програмного інтерфейсу

Метод запиту	Шлях до ресурсу серверу	Коментар
GET	/quiz/(ІД тесту)	Отримати тест за ідентифікатором
POST	/quiz	Створити новий тест
PUT	/quiz	Оновити дані про існуючий тест
POST	/quiz/(ІД тесту)/image	Додати зображення до тесту
POST	/question/(ІД питання)/image	Додати зображення до тестового питання
POST	/quiz/(ІД тесту)/submit	Подати результат проходження тесту
GET	/resource/(шлях до медіа файлу)	Отримати медіа файл за шляхом
GET	/result/entries	Отримати результати проходження тестів кінцевим користувачем
POST	/(ІД результату)/webcamPhotos	Зберегти фотографії отримані при фотоконтролі
POST	/login	Здійснити аутентифікацію кінцевого користувача та отримати токен доступу

В якості формату даних для обміну інформацією між серверним модулем , що розробляється, та програмою-клієнтом, було обрано формат JSON. Це текстовий формат обміну даними, заснований на JavaScript . Як і багато інших текстових форматів, JSON є легким для розуміння людьми. Формат вважається незалежним від конкретної мови і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існують готові інструменти для створення та обробки даних у форматі JSON, в тому числі й у мові програмування Java. Є декілька поширених реалізацій бібліотек, для зчитування JSON файлів у вигляді об’єктів та навпаки, генерування JSON файлів на основі об’єктів. Стандартна реалізація яка використовується Spring Framework – Jackson.

2.4 Розробка алгоритмів роботи програми

Важливим етапом є розробка алгоритмів програмного додатку та їх схематичне зображення у вигляді блок-схем. Наступним кроком розглянемо процес розробки алгоритму роботу програмного додатку, а саме основних функцій серверного застосунка.

Розглянемо блок-схему алгоритм роботи процес збереження нового тесту в базі даних, зображену на рисунку 2.5.

Рисунок 2.5 – Блок-схема алгоритму процесу збереження нового тесту

Таким чином можем спостерігати алгоритм збереження нової сутності тесту старений за запитом програми-клієнта. Початком алгоритму є отримання HTTP запиту в тілі якого повинна міститися інформація про нову сутність тесту. Дана інформація проходить етап валідації даних та зберігається у базі даних. В результаті відповіддю на отриманий HTTP запит є успішною. У разі виникнення помилок на визначених етапах алгоритму, формується та надсилається HTTP відповідь з кодом помилки. Варто зауважити, що даний алгоритм дуже схожий до інших спроектованих сутностей серверного застосунку.

Наступним кроком розглянемо блок-схему алгоритм процесу авторизації кінцевого користувача у системі, зображену на рисунку 2.6.

Рисунок 2.6 – Блок схема алгоритм процесу отримання

Даний алгоритм буде використовуватися при здійсненні запиту на аутентифікація кінцевого користувач програмою-клієнтом. Алгоритм перевіряє

наявність користувача та збіг паролів у базі даних та HTTP запитів. По результатам перевірок формується відповідна HTTP відповідь.

2.5 Висновки

Отже в даному розділі було розглянути загальну архітектуру серверного модуля адаптивної тестувальної системи з фотоконтролем. Спроєктовано розподіл додатку на різні програмні шари та роботу головного веб контролера сервера який делегує виконання веб запитів – диспетчера контролерів.

Спроєктовано будову бази даних проєкту, створено 7 основних таблиць, з врахуванням всіх описаних особливостей та потреб. Описано призначення кожного поля, та відносини між таблицями.

Описано прикладний програмний інтерфейс розробленого серверного додатку, та описано призначення та поведінку вихідних точок.

Розроблено блок-схеми алгоритмів програми, та наведено приклади типових алгоритмів для роботи з більшістю програмних сутностей.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Для розробки серверних застосунків існує різний перелік мов програмування та бібліотек. Розглянемо основні такі інструменти.

Java – це високорівнева, об'єктно-орієнтована мова програмування [12][13]. Ця мова програмування загального призначення, дозволяє програмістам писати програмний код один раз та запускати його де завгодно. Це означає, що скомпільований код Java може працювати на всіх платформах, які підтримують віртуальну машину Java (JVM), без необхідності перекомпіляції. Java програми, як правило, компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java незалежно від базової архітектура комп'ютера. Середовище виконання Java надає динамічні можливості, такі як відображення та модифікація коду під час виконання, які зазвичай недоступні в традиційних скомпільованих мовах.

Основним інструментом для розробки серверних додатків є Spring Framework. Це програмний фреймворк та контейнер інверсії контролю для платформи Java . Основні функції фреймворка можна використовувати будь-яким додатком Java, але є окремі модулі для створення веб-додатків поверх платформи Java EE (Enterprise Edition).

Розглянемо не менш популярну мову програмування Python. Python – це високорівнева інтерпретована мова програмування загального призначення. В цієї мови явно визначений підхід до читабельності коду [14][15]. Python використовує динамічну типізацію і може здійснювати очистку пам’яті за допомогою спеціального компонента – збирача сміття. Він підтримує декілька парадигм програмування, включаючи структуроване, зокрема процедурне, об’єктно-орієнтоване та функціональне програмування. Вперше був випущений в 1991 році як Python 0.9.0. Python 2.0 був випущений у 2000 році і представив нові функції, такі як спискові вирази, збирання сміття з визначенням циклу,

підрахунок посилань та підтримка Unicode . Python 3.0, випущений у 2008 році, був великою версією, яка не повністю сумісна з попередніми версіями. Python незмінно вважається однією з найпоширених мов програмування.

Популярним інструментом мови Python є фреймворк Django. Django – це безкоштовна веб-фреймворк на основі Python з відкритим вихідним кодом , який відповідає архітектурному шаблону model–template–views (MTV). Він підтримується Django Software Foundation (DSF), незалежною організацією, заснованою в США як неприбуткова організація. Основна мета Django – полегшити створення складних веб-сайтів, керованих базою даних. Python використовується повсюдно, навіть для налаштувань, файлів і моделей даних.

Розглянемо ще одну мову програмування – JavaScript. JavaScript є мовою програмування , яка є однією з основних технологій всесвітньої мережі , поряд з HTML і CSS [16][17]. Усі основні веб браузери мають спеціальний рушій JavaScript для виконання коду на пристроях користувачів. JavaScript – це мова високого рівня, яка відповідає стандарту ECMAScript, має динамічну типізацію, орієнтацію на прототип і функції першого класу. Мова підтримує подійно-орієнтоване програмування, функціональні та імперативний стилі програмування . Він має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами , стандартними структурами даних та об’єктною моделлю документа . (DOM). Механізми JavaScript спочатку використовувалися лише у веб-браузерах, але тепер вони є основними компонентами деяких серверів і різноманітних програм . Найпопулярнішою системою виконання для такого використання є Node.js.

Отже, по результатам здійсненого аналізу сучасних мов програмування та їх інструментів створення серверних додатків, складемо порівняльну таблицю переваг (табл. 1.2).

Таблиця 1.2 – Порівняння мов програмування

Особливість	Java	Python	JavaScript
Об’єктно орієнтована	+	+	+\-
Незалежність від платформи	+	-	+
Підтримка багатопоточності	+	+	-
Наявність веб фреймворку	+	+	+
Чистота коду	-	+	-
Підтримка складних застосунків	+	+\-	+\-

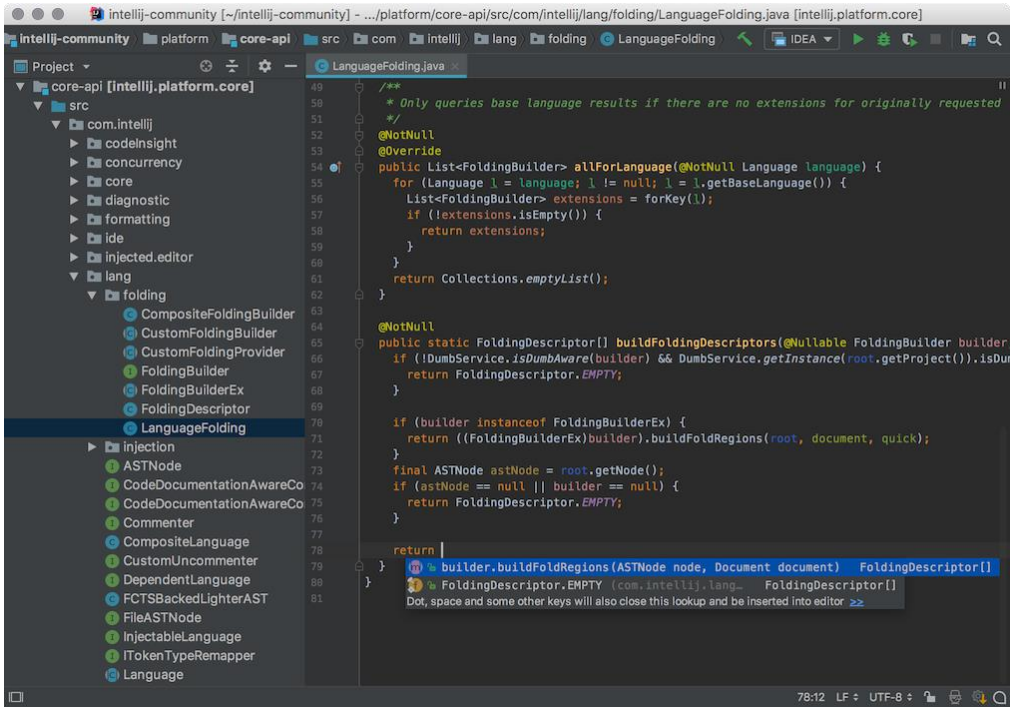
Отже, за результатами порівняльного аналізу переваг та недоліків поширених мов програмування, було обрано мову Java. Java дає можливості для створення стійких та масштабованих серверних застосунків, в за допоомги підтримки багатопоточності та наявності програмних фреймворків, наприклад Spring.

3.2 Вибір середовища розробки

Розробка сучасного програмного забезпечення потребує відповідних потужних сучасних інструментів розробки. Існують декілька поширених сучасними середовищам розробки за допомогою мови програмування Java, а

саме IntelliJ IDEA, Eclipse, NetBeans. Розглянемо кожне середовище детальніше.

IntelliJ IDEA – інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python, розроблене компанією JetBrains [18][19] (рис. 3.1).



Риснок 3.1 – Інтерфейс IntelliJ IDEA

Перша версія з'явилася в січні 2001 року і швидко набула популярності як перше середовище для Java з широким набором інтегрованих інструментів для рефакторингу. Дизайн середовища орієнтований продуктивність роботи програмістів, дозволяючи сконцентруватися на функціональних завданнях. Починаючи з версії 9.0, середовище доступне у двох редакціях: Community Edition і Ultimate Edition. У редакції Ultimate Edition, доступній під комерційною ліцензією, реалізована підтримка Java Enterprise Edition , UML - діаграм, підрахунок покриття коду , а також підтримка інших систем керування версіями, мовами та фреймворків, зокрема Spring. Слід зазначити, що Ultimate Edition безкоштовно розповсюджується у якості інструмента для вивчення мов

програмування, для учасників навчального процесу у освітніх закладах по всьому світі.

IntelliJ IDEA також реалізовує широку підтримку розробки Java додатків з використанням Spring Framework та його підпроєктів. Корисною функцією є розпізнавання створених користувачем програмних компонентів в контексті контейнера Spring, що дозволяє середовищу розуміти їх роль та давати підказки користувачу в процесі розробки.

Розглянемо наступне середовище розробки для мови Java. Eclipse – вільне інтегроване середовище розробки модульних кросплатформних додатків. Розвивається та підтримується Eclipse Foundation [20][21] (рис 3.2).

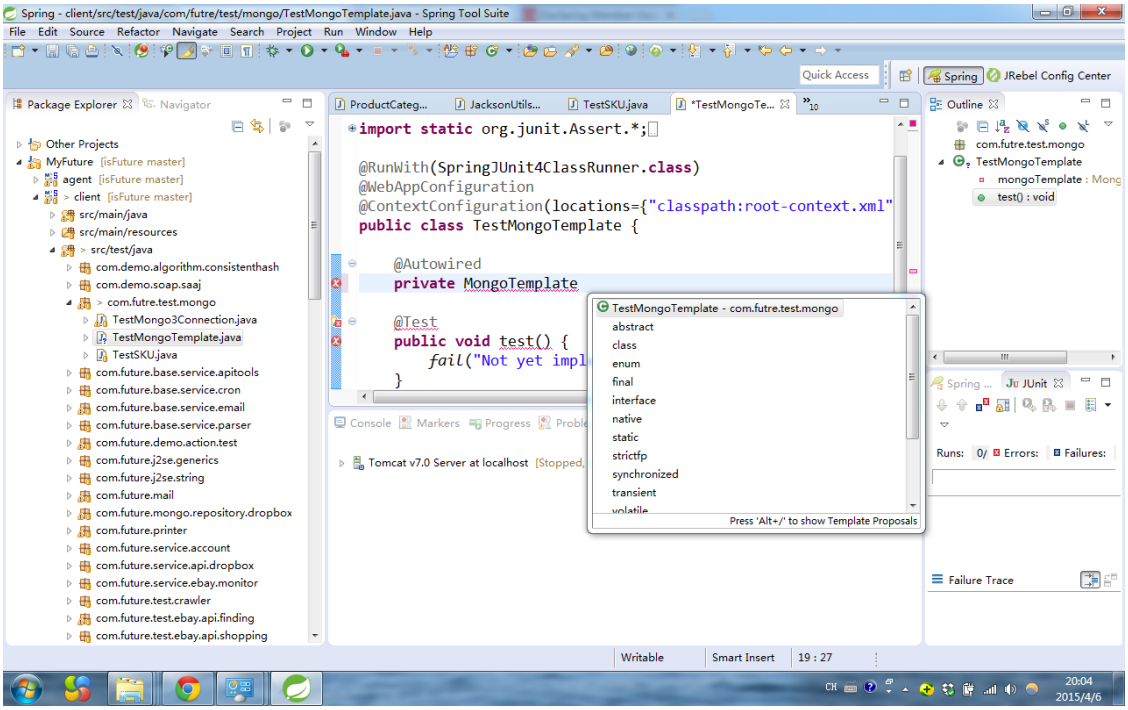


Рисунок 3.2 – Інтерфейс Eclipse

Eclipse служить насамперед платформою для розробки розширень, чим він і завоював популярність. Будь-який розробник може розширити Eclipse своїми модулями. Багато розширень доповнює середовище Eclipse диспетчерами для роботи з базами даних, серверами додатків та ін. В силу безкоштовності та високої якості, Eclipse у багатьох організаціях є

корпоративним стандартом для розробки програм. Eclipse написана на Java, тому є платформо-незалежним продуктом, крім бібліотеки SWT.

Слід зазначити, що існує спеціальне розширення для середовища Eclipse – Spring Tool Suite. Spring Tools для Eclipse добре підходить для початку роботи з Spring Boot і роботи з великими мікросервісними програмами, які базуються на Spring Boot.

Наприкінці розглянемо середовище NetBeans. NetBeans – це інтегроване середовище розробки для мови програмування Java [22]. Проект NetBeans IDE підтримується та спонсорується компанією Oracle, проте розробка NetBeans ведеться незалежною спільнотою розробників-ентузіастів та компанією NetBeans Org. Останні версії NetBeans IDE підтримують рефакторинг, профільування , виділення синтаксичних конструкцій кольором, автодоповнення конструкцій і безліч визначених шаблонів коду. Середовище розробки NetBeans за замовчуванням підтримувало розробку для платформ J2SE та J2EE. Хоча існують певні розширення для даного середовища розробки, все ж таки розробка програмними додатків з використанням Spring Framework вдається здійснювати не так інформативніше, як це можливо в попередньо розглянутих середовищах.

Отже в результаті розгляду актуальних інтегрованих середовищ розробки для мови програмування Java, серед трьох поширених варіантів було обрано саме IntelliJ IDEA. Вибір обумовлюються потужним інструментарієм розробки та налагоджування програм не тільки на мові програмування Java, але й з використанням Spring Framework та його внутрішнішніх підпроєктів.

3.3 Програмна реалізація

Серверний модуль адаптивної тестувальної системи з фотофіксацією було розроблено з допомогою типової структури для серверних додатків розроблених з використанням Spring. Розглянемо пакетну структуру розробленого серверного модулю (рис 3.3).

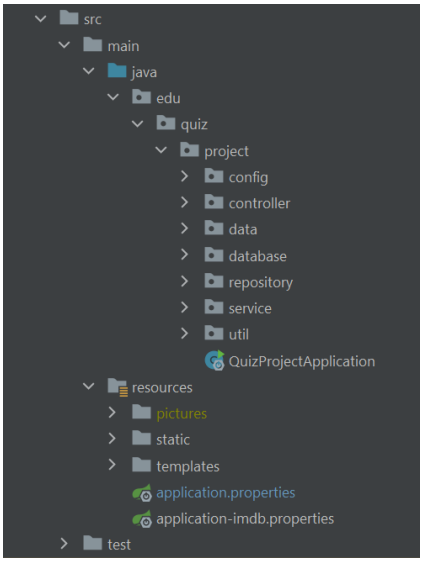


Рисунок 3.3 – Файлова структура серверного модулю

Основний Java код зберігається у однойменній директорії «java». Розглянемо призначення кожного з внутрішніх пакетів:

– «config» – зберігає в собі класи для налаштування різноманітних компонент Spring Framework, наприклад налаштування контролю по доступу до додатку – клас SecurityConfig;

– «controller» – містить в собі класи контролерів програми, наприклад QuestionController;

– «data» – містить в собі різноманітні сутності, з якими працює програма, та використовує як контейнери для робочих даних, наприклад класи Quiz, Question, Option чи QuestionType;

– «database» – використовується для збереження класів для завантаження тестових даних у при використанні додатків з тестовим профілем;

– «repository» – містить інтерфейси репозиторіїв, для роботи з базою даних проєкту, наприклад QuestionRepository;

– «service» – містить сервісні класи, які виконують основну робочу логіку серверного додатку, наприклад QuestionService;

– «util» – містить допоміжні класи, які можуть використовуватися у різних компонентах, наприклад JwtUtil.

На рисунку 3.4 можна побачити основні створені класи кожного з трьох архітектурних шарів додатку.

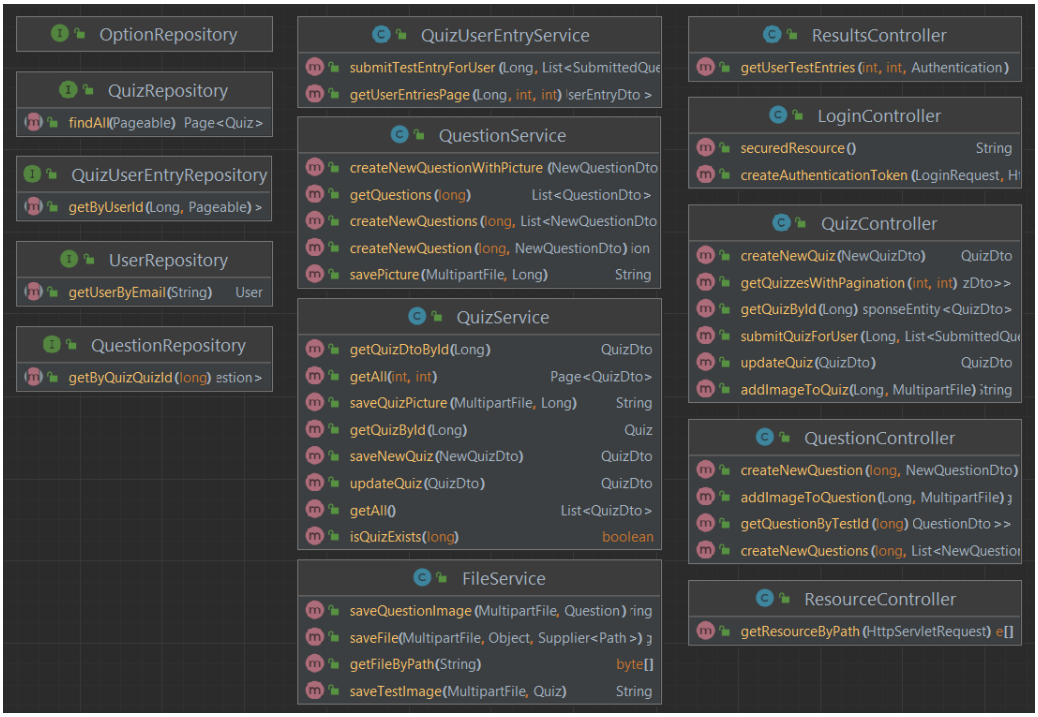


Рисунок 3.4 – Основні класи архітектурних структур

Далі розглянемо безпосередньо реалізацію окремих модулів серверного модулю. Для початку наведемо приклад реалізації роботи з сутностями в базі даних, а саме із сутностями тестів (рис 3.5).

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

import edu.quiz.project.data.entity.Quiz;
import edu.quiz.project.data.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);
}

```

Рисунок 3.5 – Реалізація модулю «QuizRepository»

В даному випадку QuizRepository – це інтерфейс. Всю реалізацію базових методів роботи з сутностями в базі даних, наприклад зберегти чи видалити, бере на себе Spring Framework, а саме Spring Data JPA. З допомогою унаслідування спеціального інтерфейсу, під час запуску застосунку, будуть сформовані всі основні методи роботи зі сутностями тестів. Щоб розширити поведінку даного репозиторію, було прямо оголошено сигнатуру нового абстрактного метода «findAll», який повинен отримувати визначену порцію від усіх наявних у базі даних тестів, інакше кажучи сторінку тестів. Дивлячись на назву метода, його параметри та тип, який він повертає, Spring знову ж таки згенерує весь програмний код метода. Така технічна особливість в більшості випадків звільняє від написання шаблонного коду при роботі з базою даних. Загалом, всі класи репозиторії в даному програмному модулі було розроблено схожим чином.

Розглянемо реалізацію методу сервісу QuizUserEntryService submitTestEntry, який зараховує результати тесту для користувача (рис 3.6).

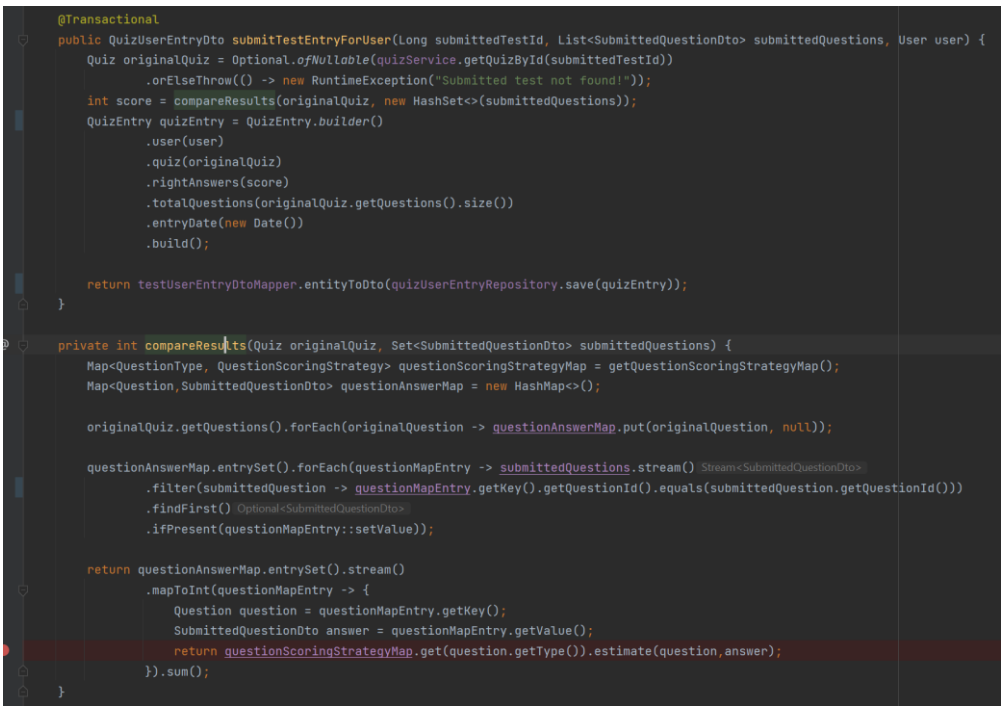
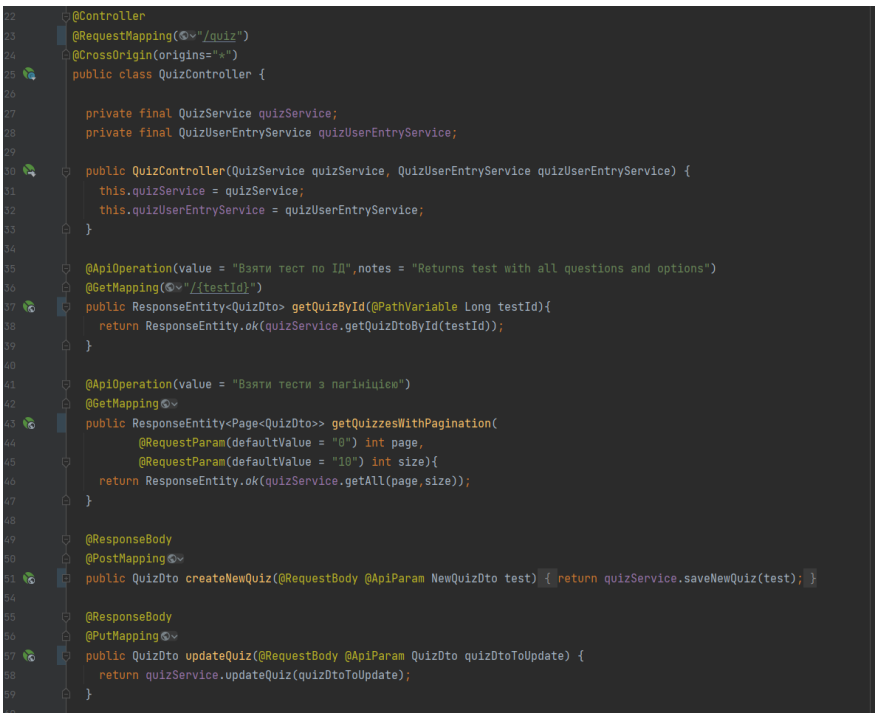


Рисунок 3.6 – Програмна реалізацію зарахування результатів

Метод позначений як транзакційний, тому менеджер транзакцій Spring й буде виконувати цей метод транзакційно. Таким чином, при виникненні виключної ситуації підчас виконання методу, всі зміни до бази даних, здійсненні в рамках даного методу, будуть відмінені. В якості аргументів, метод отримує ідентифікатор тесту, який було виконано, перелік виконаних тестових питань цього тесту, та користувача який здійснив тестування. Була реалізована валідація на наявність відповідного тесту в базі даних. Підрахунок результату здійснюється за допомогою патерну проєктування стратегія, тобто можуть існувати різні стратегії оцінки результатів тесту. Відповідна стратегія обирається в залежності від типу отриманого тестового питання. Таким чаном питання з однією та декількома відповідями будуть опрацьовуватись по різним алгоритмам. Результатом обчислення є загальна оцінка, яка зберігається в базі даних, разом з іншими необхідними даними, а саме дата тестування, кількість питань на момент тестування та ідентифікатор тесту та користувача.

Розглянемо реалізацію класу QuizController, який відповідає за обробку HTTP запитів на модифікацію чи отримання сутностей тестів(рис. 3.7).



```java
@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 пагінцією")
  @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) { return quizService.saveNewQuiz(test); }

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

Рисунок 3.7 – Реалізація класу QuizController

За допомогою Java анотацій, клас контролера помічений як контролер та задано адресу для обробки запитів «/quiz». Таким чином Spring збереже екземпляр цього класу в якості контролера, та присвоїть йому вказану адресу. Завдяки цьому, для обробки ресурсу за запитом по адресі «/quiz», диспетчер сервлетів Spring буде викликати екземпляр саме цього класу. Методи цього класу, являються так званими вихідними точками. Вони також мають власну адресу, за якою будуть викликатися вже в рамках самого контролеру. Шляхи контролера та його методів складаються. На рисунку 3.8 зображено приклад адреси для виклику метода «getQuizById».

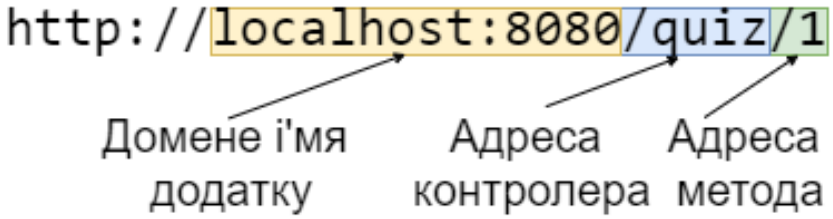


Рисунок 3.8 – Приклад виклику методу «getQuizById».

Розглянемо метод «getQuizzesWithPagination». Метод помічений як HTTP GET метод. Він не має свої спеціальної адреси, тому він використовує адресу контролера, а саме «/quiz». Параметрами метода виступають номер сторінки та кількість елементів на одній сторінці. Диспетчер сервлетів буде передавити визначені аргументи при виклику даного метода, попередньо знайшовши їх у параметрах HTTP запиту, а саме параметри «page» та «size». Інакше, для цих параметрів, буде встановлене значення по замовчуванню. Результатом роботи контролера є сформована HTTP відповідь, в тіло якої було попередньо колекцію сутностей тесту.

Важливим аспектом будь-якого серверного модуля є налаштування прав доступу до наявних його ресурсів. Spring Framework надає спеціальний інструмент Spring Security, який використовується для налаштування фільтрації HTTP запитів із застосуванням веб фільтрів. Налаштування загальний прав доступу до ресурсів додатку, було створено клас конфігурації безпекових налаштувань додатку «SecurityConfig» (рис 3.9).

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

Рисунок 3.9 – Безпекова конфігурація серверного модулю

Таким чином було обмежено доступ до наявних контролерів серверу для користувачів які не пройшли аутентифікацію та авторизацію. Хоча контролер за адресою «/login» залишився незахищеним, адже перед здійсненні аутентифікації користувач не можу бути вже аутентифікованим в системі.

Слід зазначити, що було вимкнуто створення сесій для користувачів та додано новий веб фільтр «JwtRequestFilter». Ці дії необхідні для реалізації правильної роботи JWT токенів. Реалізацію «JwtRequestFilter» зображено на рисунку 3.10.

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException,
    String authorizationHeader = request.getHeader( name: "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 – Програмна реалізація «JwtRequestFilter»

Даний фільтр отримує об’єкт HTTP запиту, та робить спробу знайти в ньому значення заголовка з ключем «Authorization». Значення повинне представляти з себе безпосередньо текстовий токен. Текстовий токен перевіряється на достовірність, за допомогою спеціального секретного ключа, який доступний тільки серверу, з використанням алгоритму шифрування SHA-256. При успіху, у базі даних знаходить відповідний користувач, та запит передається далі по ланцюгу фільтрів.

3.4 Розробка модуля збереження результатів фотоконтролю

Розглянемо реалізацію важливого модуля створеної тестувальної системи, а саме фотоконтроль за процесом здійснення тестування. Було створено контролер «ResultController», який спеціалізується на роботі з результатами тестування. Даний контролер дає можливість програмі-клієнту зберегти фотографії кінцевого користувача системи, який проходив певний тест. Метод контролера «addUserWebcamPhotos» створено так, щоб він отримував POST HTTP запит, з наявним в ньому масивом медіа файлів, а саме фотографій з веб камери користувача (рис 3.11).

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

Рисунок 3.11 – Реалізація методу «addUserWebcamPhotos»

Контролер налаштований сприймати медіа файли MIME групи «image/*». Це дозволяє отримувати зображення у будь-якому з поширених розширень, наприклад JPEG, JPG, JPE, PNG, GIF.

У класі «FileService» було реалізовано збереження отриманих фотографій безпосередньо у файловій системі серверного модулю. Таким чином, клас сервісу містить необхідну функціональність для збереження різноманітних медіа даних, в тому числі й зафіксовані фото користувачів, під час проходження тесту. На рисунку 3.12 можна побачити процес збереження колекції фотографій.

```
List<Path> filePaths = Arrays.stream(files) Stream<MultipartFile>
        .map(file -> {
            String fileName = createFileName(file, targetEntity);
            Path filePath = Paths.get(directory.toString(), fileName);
            if (Files.exists(filePath)) {
                int orderNumber = 2;
                do {
                    fileName = createFileName(file, targetEntity, orderNumber++);
                    filePath = Paths.get(directory.toString(), fileName);
                } while (Files.exists(filePath));
            }
            return filePath;
        }) Stream<Path>
        .collect(Collectors.toList());
```

Рисунок 3.12 – Реалізація збереження фотографій

В процесі збереження, серверний модуль виконує необхідні перевірки перед збереження файлів у спеціально визначеній папці файлової системи додатку, наприклад перевірка наявності відведеної папки. В результаті, у відповідну колонку «webcamPhoto» таблиці «Quiz_Entry» поміщається шлях у файлової системи до збережених фотографій На рисунку 3.13 можна побачити файлову ієрархію збережених ресурсів серверного модулю, а саме – збережені медіа файли фотоконтролю.

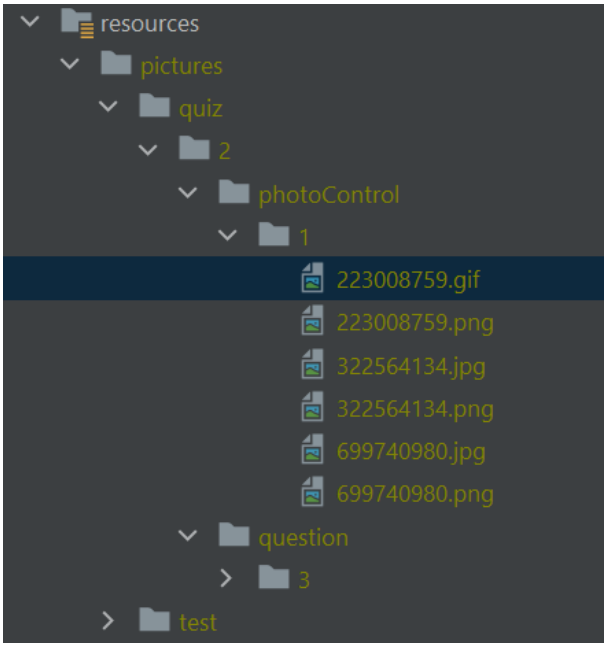


Рисунок 3.13 – Файлова ієрархія директорії ресурсів

Для подальшого користування збереженими ресурсами, було реалізовано клас «ResourceController». Даний універсальний контролер дозволяє отримати збережені медіа дані за шляхом, по якому вони зберігаються сервером. На рисунку 3.14 зображено реалізація методу контролера шукає та повертає знайдений медіа файл у файлові системі середовища.

```
@ResponseBody
@ApiOperation(value = "Get resource by internal filepath",
        notes = "Append file path to get resource. \n\rExample: '{baseURL}/resource/test/2/question/3/1101215343.gif'")
@GetMapping(path = "/**", produces = "image/*")
public byte[] getResourceByPath(HttpServletRequest request) {
    String filePath = request.getRequestURI().replace( target: "/resource/", replacement: "");
    return fileService.getFileByPath(filePath);
}
```

Рисунок 3.14 – Пошук даних у файлові системі

Метод звертається до файлового сервісу «FileService», та передає йому шлях, отриманий в HTTP запиті. Результатом виклику методу є HTTP відповідь, у тілі якої міститься масив байтів, що і являється знайденим ресурсом.

Таким чином була реалізований зручний доступ до ресурсів серверного модуля. Схематичне зображення отримання ресурсу можна побачити на рисунку 3.15.

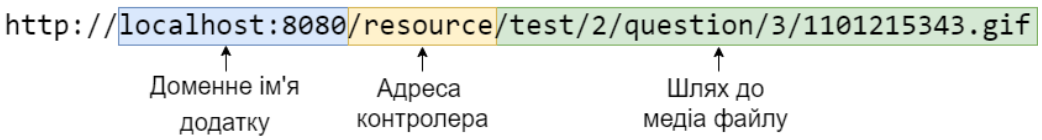


Рисунок 3.15 – Приклад запиту для отримання медіа файлу

3.5 Висновки

В даному розділі було обґрунтовано вибір середовища розробки IntelliJ IDEA. Було розглянуто три поширених рішення, проаналізовано їх переваги, та на основі цього сформульовано вибір.

Було розглянуто безпосередньо програмну реалізацію серверного модуля адаптивної тестувальної системи. Було наведено код реалізації основних функціональних особливостей серверу, таких як репозиторії, контролери, підрахунок результатів тестування, збереження медіа файлів фотофіксації, отримання медіа файлів за їх адресою, безпекова конфігурація та фільтр для веб токенів.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування програмного забезпечення

Тестування програмного забезпечення – одна з найважливіших стадій його розробки. Вона допомагає виявити помилки, що не були знайдені під час розробки програмного забезпечення, а також перевіряє на відповідність додатку вимогам специфікації. Хоча тестування прикладних програмних інтерфейсів відрізняється від додатків з користувацьким інтерфейсом, загальні залишаються такими ж самими [23]. Основними цілями тестування прикладного програмного інтерфейсу можуть бути наступні:

– переконатися, що реалізація прикладного програмного інтерфейсу працює правильно, як і очікувалося – без помилок;

– гарантувати, що реалізація прикладного програмного інтерфейсу працює відповідно до специфікації;

– запобігти регресії між написаним кодом і та новою версією.

Розглянемо основині етапи тестування прикладних програмний інтерфейсів.

– Перевірка правильність коду стану HTTP;

– Перевірка корисне навантаження HTTP відповіді;

– Перевірка заголовки HTTP відповіді;

– Перевірка правильність стану програми;

– Перевірка базову працездатність.

Отже протестуємо роботу програмного інтерфейсу загалом. Для проведення тестування скористуємося інструментом для створення специфікацій Swagger [24]. Swagger також надає набір інструментів розробника для налагоджування або тестування прикладних програмних інтерфейсів [25]. Сконфігурований користувацький інтерфейс Swagger UI зображено на рисунку 4.1.

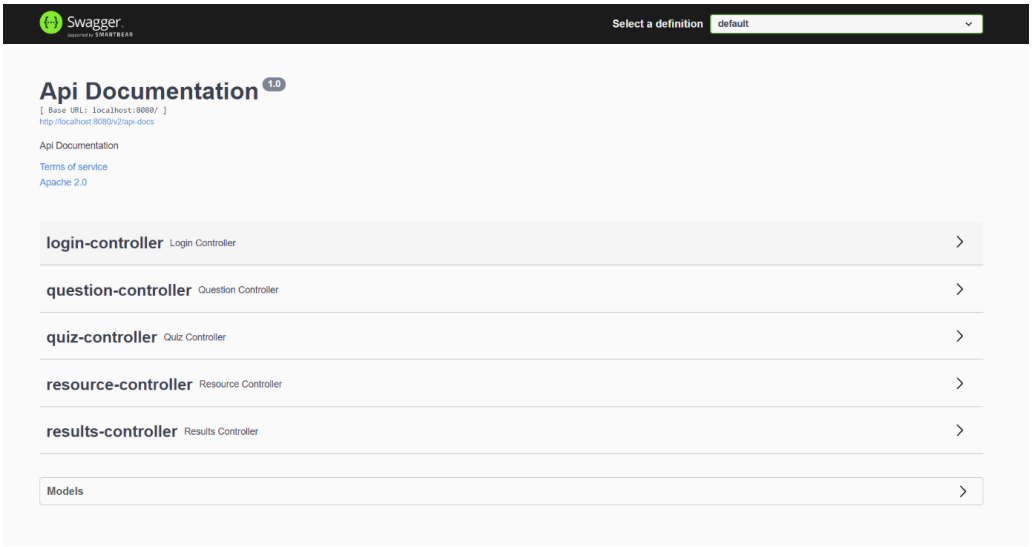


Рисунок 4.1 – Користувацький інтерфейс Swagger UI

Для початку, розглянемо роботу ресурсу за посиланням «POST /quiz». При здійснені HTTP запиту на дане посилання розробленого серверного модулю, він повинен отримати дані з тіла запиту, обробити та зберегти оброблену сутність у базі даних, і в результат повернути відповідь з кодом 200, з тілом відповіді у вигляді новоствореної сутності тесту разом зі присвоєним ідентифікатором. Результати тестування наведені на рисунку 4.2.

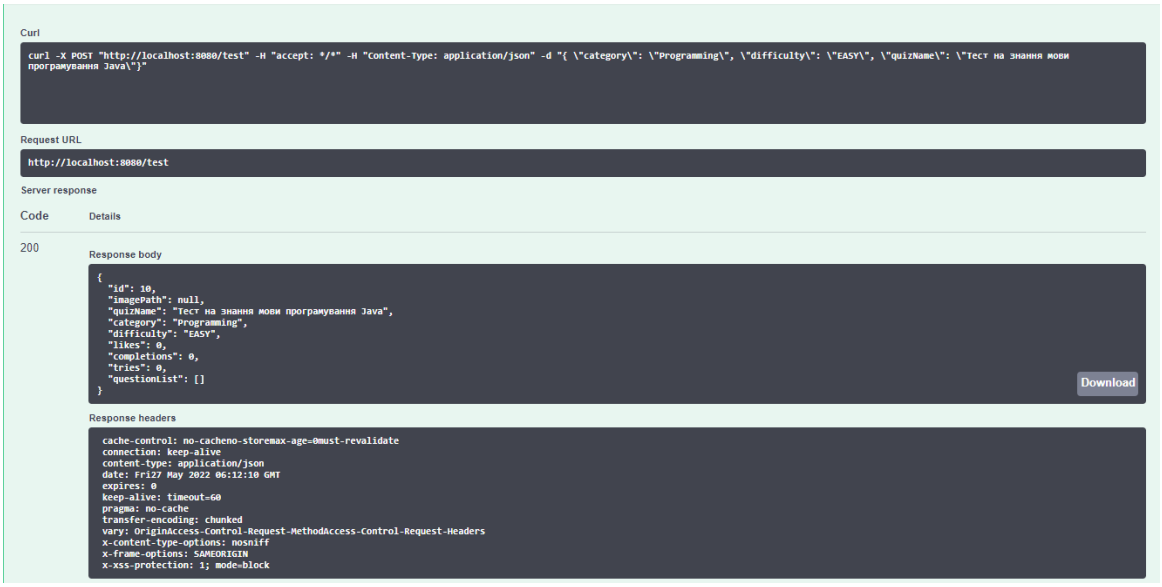


Рисунок 4.2 – Результат тестування ресурсу «POST /quiz».

У вікні результатів тестування можемо спостерігати, зверху до низу, посилання, заголовки та тіло здійсненого запиту до серверного модуля, з допомогою програмного засобу CURL. Далі йде код відповіді – 200, тіло функції з новоствореною сутністю тесту, та заголовки HTTP відповіді серверного модулю. Всі вихідні дані відповідають вимогам, та отримані успішно.

Проведемо тестування ресурсу за посиланням «GET quiz/10». Ресурс повинен знайти сутність тесту в залежності від його ідентифікатору, який вказаний прямо у стрічки запиту. У разі тесту у базі даних, зі статусом 200 повертається знайдена сутність у тілі відповіді. У разі невдачі – відповідь з кодом 404. Результати тестування наведені на рисунку 4.3.

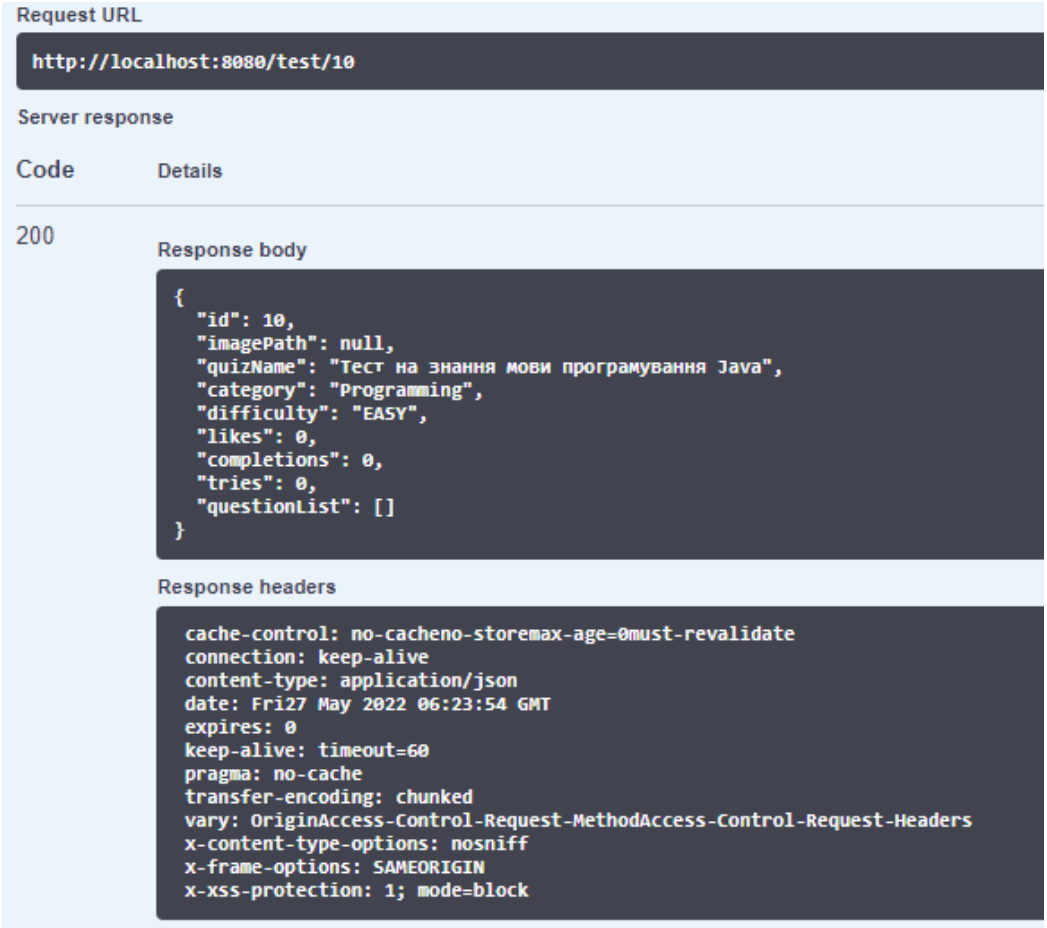


Рисунок 4.3 – Результати тестування ресурсу «GET quiz/10»

Також проведемо тестування того сценарію, коли в базі даних тесту з таким ідентифікатором не існує. Результати сценарію зображені на рисунку 4.4

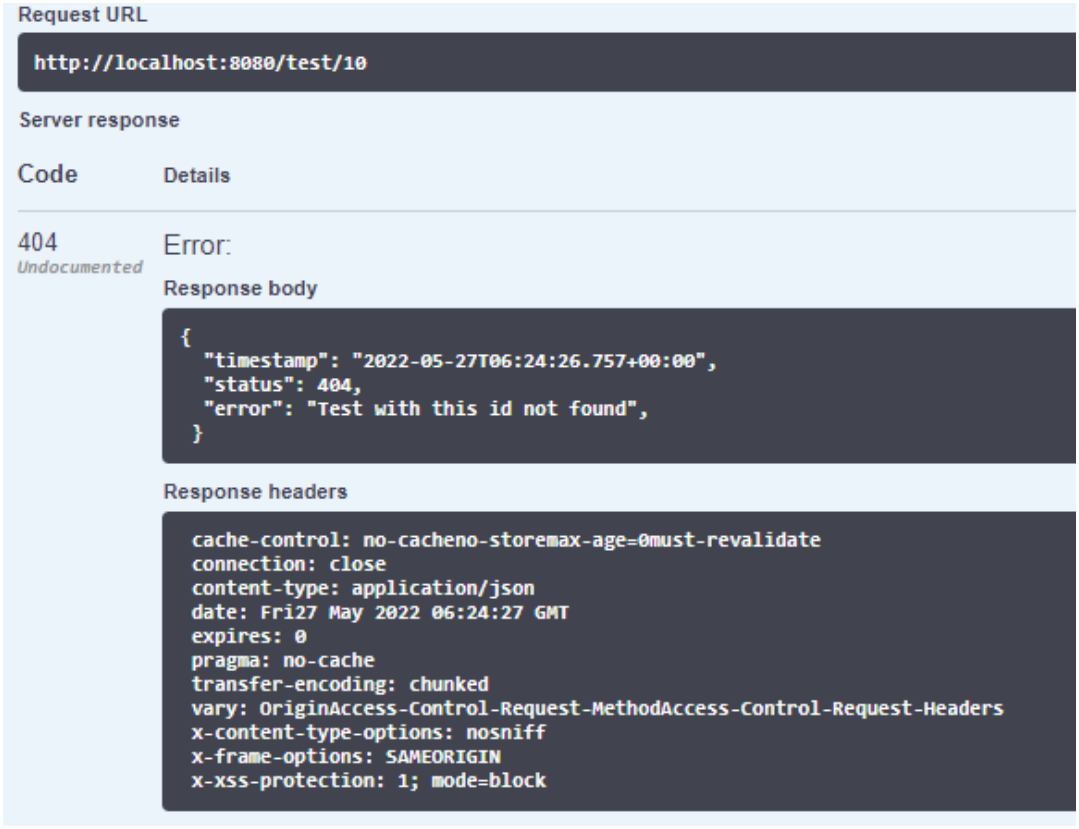


Рисунок 4.4 – Результати тестування ресурсу «GET quiz/10»

Всі вихідні дані даного ресурсу відповідають вимогам, та отримані успішно. Протестовано два різних сценарії взаємодії.

Наступним розглянемо роботу ресурсу «POST /quiz/10/questions». Даний ресурс повинен додати масив питанньь до вже існуючого тесту за його ідентифікатором, який вказаний прямо у посиланні до ресурсу. Розглянемо взідні дані при тестуванні (рис 4.5).

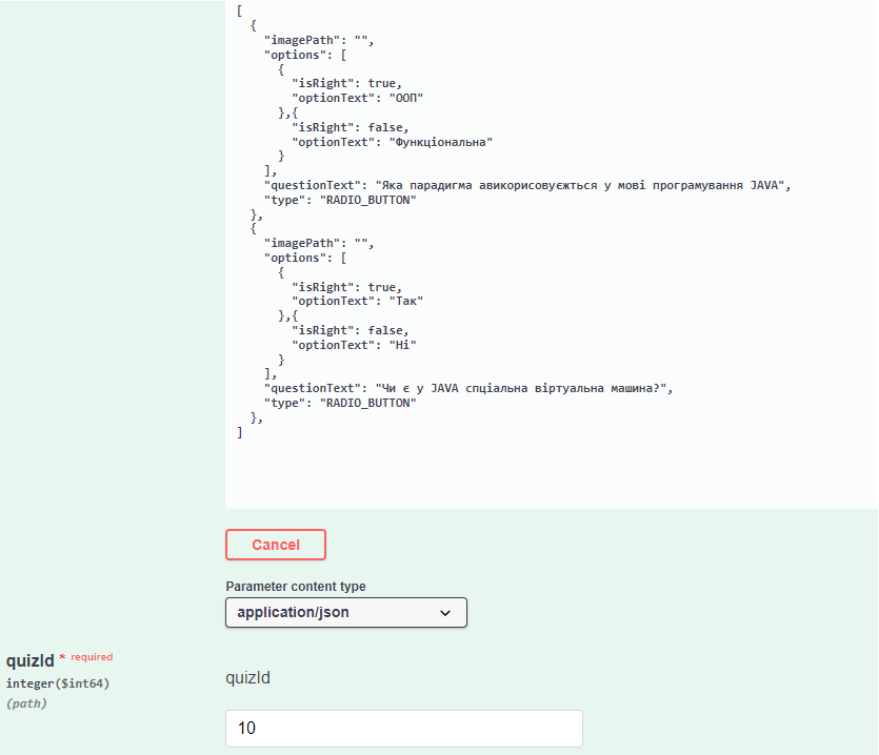


Рисунок 4.5 – Вхідні дані при тестуванні «POST /quiz/10/questions»

На вхід подається масив з двох об’єктів питання та ідентифікатор цільового тесту – 10. Розглянемо результат тестування ресурсу на рисунку 4.6.

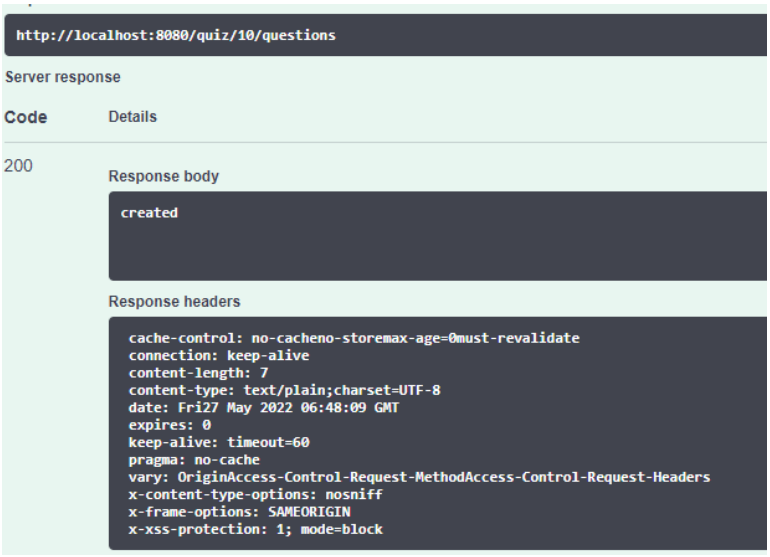


Рисунок 4.6 – Результат тестування ресурсу «POST /quiz/10/questions»

В результаті ми отримали статус відповіді 200 та тіло, яке сигналізує що запит дійсно пройшов успішно.

Також розглянемо роботу контролера за адресою «GET /resource/». При виклику даного контролера можна отримати медіа файли, які зберігаються у файлові системі середовища серверного модуля. Параметром для даного контролера є адреса за якої зберігається, і вказується як продовження адреси запиту. Протестуємо виклик ресурсу який знаходиться за вказаним посиланням (рис 4.7).

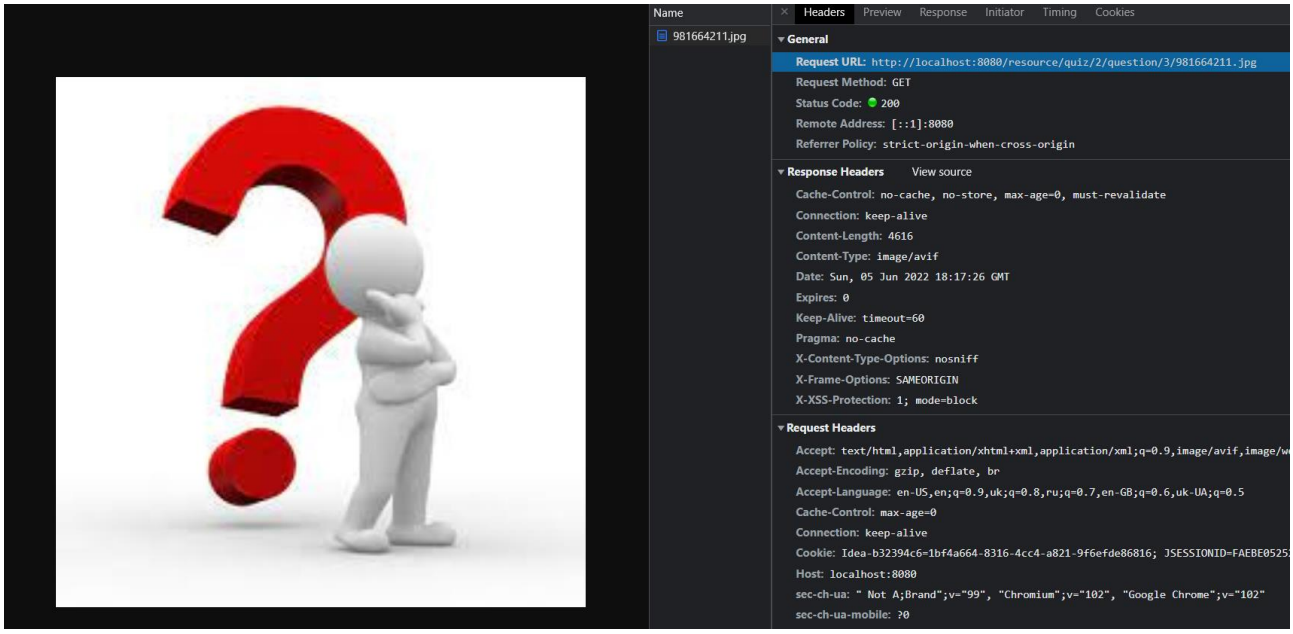


Рисунок 4.7 – Результат тестування ресурсу «GET /resource/ quiz/2/question/3/981664211.jpg

В результаті запиту до контролера з вказаною адресою ресурсу, було отримано відповідне зображення, яке за ним зберігалося.

4.2 Документація прикладного програмного інтерфейсу

За допомогою вище згаданого інструменту Swagger UI, було створено документацію прикладного програмного інтерфейсу розробленого серверного модуля адаптивної тестувальної системи. Документація містить опис

призначення всіх запитів до програмного модулю, їх параметрів можливих відповідей. На рисунку у 4.8 можна побачити приклад документації контролера «QuestionController».

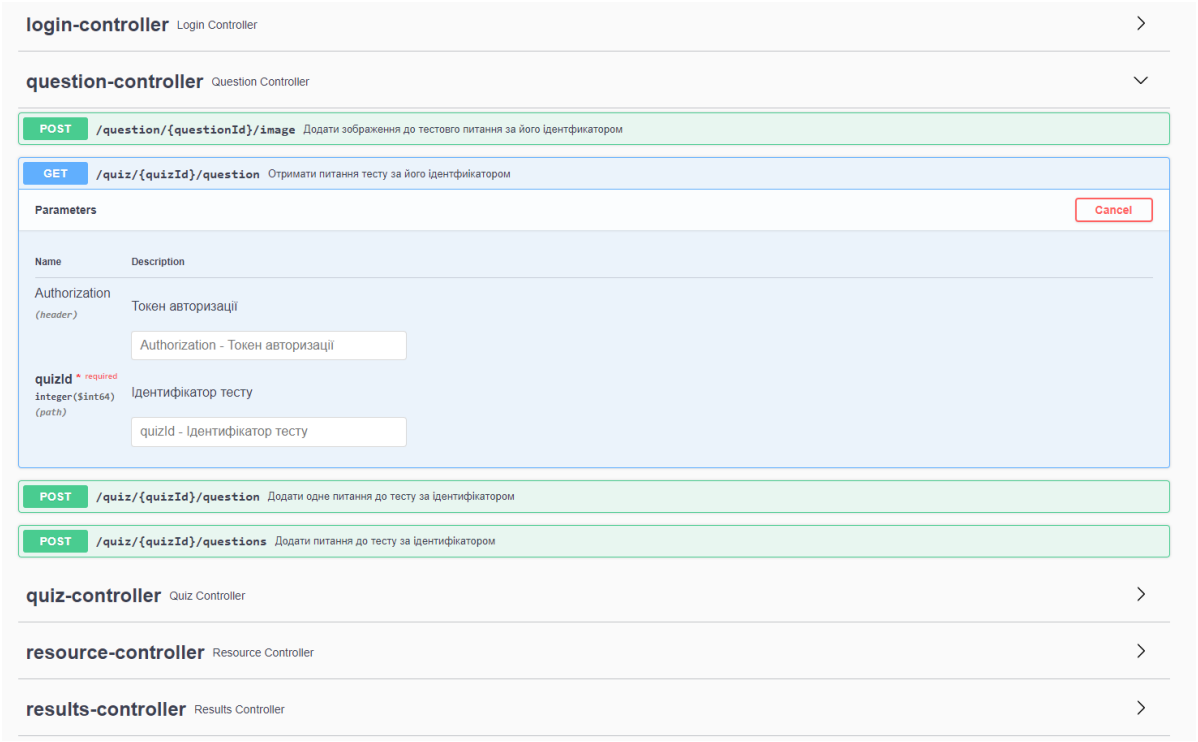


Рисунок 4.8 – Документація контролеру «QuestionController».

Також було додано документацію, яка описує всі сутності, за допомогою яких здійснюється обмін даними між серверним модулем та його клієнтом. Наприклад сутномті QuizDto, QuestionDto, OptionDto, NewQuizDto та інші.

4.3 Висновки

У даному розділі було проведено тестування серверного модулю адаптивної тестувальної системи з фотоконтролем. Тестування показало, що прикладний програмний інтерфейс розроблено відповідно до технічних вимог та немає критичних помилок. Було складено документацію для прикладного програмного інтерфейсу серверного модуля.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було здійснення проєктування та розробку серверного модуля адаптивної тестувальної системи з фотоконтролем.

Було розглянути декілька основних аналогів системі, яка розробляється. Було здійснено порівняльний аналіз функціональних особливостей додатку що розробляється та аналогів. В результаті було зроблено висновок, що додаток який розробляється у дані бакалаврській роботі, матиме в собі усі особливості розглянутих аналогів. Було розглянуто сучасні моделі життєвого циклу для розробки програмного забезпечення. В якості моделі для розробки було обрано водоспадну модель, та обґрунтовано сам виріб.

Було розглянуто загальну архітектуру серверного модуля адаптивної тестувальної системи з фотоконтролем. Спроєктовано розподіл додатку на різні програмні шари та роботу головного веб контролера сервера який делегує виконання веб запитів – диспетчера контролерів. Спроєктовано будову бази даних проєкту, створено 7 основних таблиць. Описано прикладний програмний інтерфейс розробленого серверного додатку, та описано призначення та поведінку вихідних точок.

Було розглянуто програмну реалізацію серверного модуля адаптивної тестувальної системи. Було наведено код реалізації основних функціональних особливостей серверу, таких як репозиторії, контролери, підрахунок результатів тестування, збереження медіа файлів фотофіксації, отримання медіа файлів за їх адресою, безпекова конфігурація та фільтр для веб токенів.

Наприкінці роботи, було здійснено тестування прикладного програмного інтерфейсу розробленого серверного модуля. Тестування показало, що прикладний програмний інтерфейс розроблено відповідно до технічних вимог та немає критичних помилок.

ПЕРЕЛІК ПОСИЛАНЬ

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

3. Робота в системі дистанційного навчання MOODLE. Інструкції для викладача – Тернопіль, ННІІОТ ТНЕУ, 2014. – 73 с

4. Cambridge English. [Електронний ресурс] – URL: https://www.cambridgeenglish.org/

5. Online Test Pad. [Електронний ресурс] – URL: https://onlinetestpad.com/

6. Test-english. [Електронний ресурс] – URL: https://test-english.com/

7. Книга Інженерія програмного забезпечення. Посібник для студентів вищих навчальних закладів – І. Л. Бородкіна, Г. О. Бородкін, Київ, 2018. – 204 с

8. Spring in Action, Fifth Edition. Крейг Уоллс, Manning, 2018. – 520 с

9. Introducting to Spring Web MVC framework. [Електронний ресурс] – URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

10. Learning MySQL. Вініцій М. Гріппа, O'Reilly Media. 2021. – 1102 с

11. Прикладний програмний інтерфейс. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Прикладний_програмний_інтерфейс

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

13. Java 8 in Action Lambdas, streams, and functional-style programming. Рауль-Габріель Урма, Маріо Фаско, Manning, 2014. – 424 с

14. The Quick Python Book, Third Edition. Наомі Седер, Manning, 2018. – 472 с

15. Python. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Python

16. JavaScript. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/JavaScript

17. Get Programming with JavaScript. Джон Ларсен, Manning, 2016. – 432 с

18. IntelliJ IDEA. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA

19. IntelliJ IDEA Essentials. Ярослав Крохмальський, 2014. – 276 с

20. Eclipse. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Eclipse

21. Eclipse IDE Pocket Guide: Using the Full-Featured IDE. Ед Бернетт, O'Reilly Media, 2005. – 136 с

22. NetBeans. [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/NetBeans

23. Design of Web APIs. Арно Лоре, Manning, 2019. – 392 с

24. Документація Swagger [Електронний ресурс] – URL: https://swagger.io/docs/

25. Коротко про API і його тестування. [Електронний ресурс] – URL: https://www.quality-assurance-group.com/korotko-pro-api-jogo-testuvannya/

Додаток А

(обов’зковий)

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
25 березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення для управління конфігураціями при розгорненні та масштабуванні електронних ресурсів» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент Д. І. Катєльніков
"__" __ 2022 р.
Виконав:
_____ студент гр. 2ПІ-18б Д. О. Токарчук
"__" __ 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка адаптивної тестувальної системи з фотоконтролем. Частина 1. Модуль сервера з використанням технологій Java і фреймворку Spring».

Галузь застосування – онлайн тестування.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від 25 березня 2022 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської роботи є покращення процесу здійснення онлайн тестування шляхом розробки власного серверного застосунку з прикладним програмним інтерфейсом, що базуються на аналізі недоліків та особливостей роботи існуючих аналогів.

Призначення роботи – розробка та програмна реалізація додатку серверного модуля адаптивної тестувальної системи з фотоконтролем.

4. Вихідні дані для проведення НДР

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

1. Spring in Action, Fifth Edition. Крейг Уоллс, Manning, 2018. – 520 с
2. Modern Java in Action. Рауль-Габріель Урма, Маріо Фаско, Manning, 2018. – 592 с
3. Learning MySQL. Вініцій М. Гріппа, O'Reilly Media. 2021. – 1102 с

5. Технічні вимоги

Модель розробки – водоспадна, мова програмування Java, фреймворк Spring, водоспадна модель розробки, середовище розробки IntelliJ IDEA Ultimate Version, прикладний програмний інтерфейс REST, протокол передачі даних HTTP, база даних MySQL.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. 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 бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 10.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи серверного модуля	11.04.2022 – 17.04.2022	Вик.
4	Аналіз і вибір середовища програмування	18.04.2022 – 21.04.2022	Вик.
5	Розробка серверного модуля	22.04.2022 – 15.05.2022	Вик.
6	Тестування роботи розробленого програмного забезпечення	16.05.2022 – 23.05.2022	Вик.
7	Оформлення матеріалів до захисту БДР	24.05.2022 – 10.06.2022	Вик.
№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б

(обов’язковий)

Протокол перевірки кваліфікаційної роботи

на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка адаптивної тестувальної системи з фотоконтролем. Частина 1. Модуль сервера з використанням технологій Java і фреймворку Spring

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Катєльніков Д.І.

Оригінальність	
Схожість	

Аналіз звіту подібності

- **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- Виявлені у роботі запозичення не мають о знак 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.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Токарчук Д.О.

Керівник роботи _____ Катєльніков Д.І.

Додаток В

(довідниковий)

Лістинг програми

edu/quiz/project/QuizProjectApplication.java

```
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/repository/QuestionRepository.java

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

import edu.quiz.project.data.entity.Question;
import edu.quiz.project.data.projection.QuestionFullProjection;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import java.util.List;

@RepositoryRestResource(excerptProjection = QuestionFullProjection.class)
public interface QuestionRepository extends JpaRepository<Question, Long> {
    List<Question> getByQuizQuizId(long quizId);
}
```

edu/quiz/project/repository/QuizRepository.java

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

import edu.quiz.project.data.entity.Quiz;
import edu.quiz.project.data.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);
}
```

edu/quiz/project/repository/QuizUserEntryRepository.java

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

import edu.quiz.project.data.entity.QuizEntry;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;

public interface QuizUserEntryRepository extends JpaRepository<QuizEntry, Long> {

   Page<QuizEntry> getByUserId(Long userId, Pageable pageRequest);
}
```

edu/quiz/project/repository/UserRepository.java

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

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

public interface UserRepository extends JpaRepository<User, Long> {
   User getUserByEmail(String email);
}
```

edu/quiz/project/config/JwtRequestFilter.java

```
package edu.quiz.project.config;

import edu.quiz.project.util.JwtUtil;
import lombok.AllArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@AllArgsConstructor
@Component
public class JwtRequestFilter extends OncePerRequestFilter {

    private DbUserDetailsService userDetailsService;
    private JwtUtil jwtUtil;



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

        String email = null;
        String jwt = null;

        if(authorizationHeader != null && authorizationHeader.startsWith("Bearer ")){
            jwt = authorizationHeader.substring(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,null,userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        filterChain.doFilter(request,response);
    }
}
```

edu/quiz/project/controller/LoginController.java

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

import edu.quiz.project.config.DbUserDetailsService;
import edu.quiz.project.data.dto.LoginRequest;
import edu.quiz.project.util.JwtUtil;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;

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

    private AuthenticationManager authenticationManager;
    private DbUserDetailsService userDetailsService;
```

```
    private JwtUtil jwtUtil;

    @PostMapping(value = "/login")
//    @ApiImplicitParam(name = "Authorization", r = true)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody LoginRequest loginRequest,
HttpServletRequest request) throws Exception {
        log.debug("Authentication attempt from " + loginRequest.getEmail());
        try {
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(loginRequest.getEmail(), loginRequest.getPassword())
            );
        } catch (BadCredentialsException e) {
            throw new Exception("Incorrect credentials");
        }
        UserDetails userDetails = userDetailsService
                .loadUserByUsername(loginRequest.getEmail());
        String jwt = jwtUtil.generateToken(userDetails);

        log.debug("Successful authentication for " + loginRequest.getEmail());
        return ResponseEntity.ok(jwt);
    }

    @GetMapping("secured")
    public String securedResource() {
        return "Hi, this is super secured resource, welcome.";
    }
}
```

edu/quiz/project/controller/QuestionController.java

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

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import edu.quiz.project.data.dto.question.NewQuestionDto;
import edu.quiz.project.data.dto.question.QuestionDto;
import edu.quiz.project.service.QuestionService;
import io.swagger.annotations.*;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
```

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

import java.util.List;

@CrossOrigin(origins = "*")
@AllArgsConstructor
@Controller
public class QuestionController {

    private final QuestionService questionService;

    @ApiOperation(value = "Отримати питання тесту за його ідентфиікатором")
    @GetMapping("/quiz/{quizId}/question")
    public ResponseEntity<List<QuestionDto>> getQuestionByTestId(
            @ApiParam(value = "Ідентифікатор тесту") @PathVariable long quizId) {
        return ResponseEntity.ok(questionService.getQuestions(quizId));
    }

    @ApiOperation(value = "Додати одне питання до тесту за ідентифікатором")
    @ResponseBody
    @PostMapping("/quiz/{quizId}/question")
    public String createNewQuestion(@PathVariable long quizId, @RequestBody NewQuestionDto newQuestionDto) {
        questionService.createNewQuestion(quizId, newQuestionDto);
        return "created";
    }

    @ApiOperation(value = "Додати питання до тесту за ідентифікатором")
    @ResponseBody
    @PostMapping("/quiz/{quizId}/questions")
    public String createNewQuestions(@PathVariable long quizId, @RequestBody List<NewQuestionDto>
newQuestionDtos) {
        questionService.createNewQuestions(quizId, newQuestionDtos);
        return "created";
    }


    @ApiOperation(value = "Додати зображення до тестовго питання за його ідентфикатором")
    @ResponseBody
```

```
    @PostMapping("/question/{questionId}/image")
    public String addImageToQuestion(@PathVariable Long questionId, @RequestPart MultipartFile file) {
        return questionService.savePicture(file,questionId);
    }

}
```

edu/quiz/project/controller/QuizController.java

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

import edu.quiz.project.data.dto.question.SubmittedQuestionDto;
import edu.quiz.project.data.dto.quiz.NewQuizDto;
import edu.quiz.project.data.dto.quiz.QuizDto;
import edu.quiz.project.data.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.data.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 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 пагініцією")
 @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){
  return quizService.saveNewQuiz(test);
 }

 @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);
  }

}
```

edu/quiz/project/controller/ResourceController.java

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

import edu.quiz.project.service.FileService;
import io.swagger.annotations.ApiOperation;
import lombok.AllArgsConstructor;
import org.springframework.http.HttpRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;

@AllArgsConstructor
@RequestMapping("/resource")
@Controller
public class ResourceController {

    private final FileService fileService;

    @ResponseBody
    @ApiOperation(value = "Get resource by internal filepath",
            notes = "Append file path to get resource. \n\rExample:
'{baseURL}/resource/test/2/question/3/1101215343.gif'")
    @GetMapping(path = "/**", produces = "image/*")
    public byte[] getResourceByPath(HttpServletRequest request) {
        String filePath = request.getRequestURI().replace("/resource/","");
        return fileService.getFileByPath(filePath);
    }
//    @ResponseBody
//    @GetMapping(value ="/resource/{*resourcePath}" ,produces = "/image/*")
//    public byte[] getResourceByPath(@PathVariable String resourcePath) {
//        return fileService.getFileByPath(resourcePath);
```

```
//    }
}
```

edu/quiz/project/controller/ResultsController.java

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

import edu.quiz.project.data.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.data.entity.User;
import edu.quiz.project.service.QuizService;
import edu.quiz.project.service.QuizUserEntryService;
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;

import java.util.List;

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

    private final QuizService quizService;
    private final QuizUserEntryService quizUserEntryService;

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

    @ResponseBody
    @GetMapping("/entries")
    public Page<QuizUserEntryDto> getUserTestEntries(@RequestParam(defaultValue = "0") int page,
                                                     @RequestParam(defaultValue = "10") int size,
                                                     Authentication authentication) {
        User user = (User) authentication.getPrincipal();
        return quizUserEntryService.getUserEntriesPage(user.getId(), page, size);
    }
```

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

}
```

edu/quiz/project/service/FileService.java

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

import edu.quiz.project.data.entity.Question;
import edu.quiz.project.data.entity.Quiz;
import edu.quiz.project.data.entity.User;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.function.Supplier;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

@Component
@Slf4j
public class FileService {
```

```
private final String ROOT_FILE_STORAGE_PATH;
private final String QUIZ_FILE_STORAGE_PATH = "quiz/";
private final String QUESTION_FILE_STORAGE_PATH = "question/";
private final String USER_PHOTO_CONTROL_PATH = "photoControl/";

public FileService(@Value("${quizy.pictureFolderPath}") String file_storage_path) {
   ROOT_FILE_STORAGE_PATH = file_storage_path;
}

public String saveQuestionImage(MultipartFile image, Question targetQuestion) {
   return saveFile(image, targetQuestion,
         () -> Paths.get(ROOT_FILE_STORAGE_PATH,
               QUIZ_FILE_STORAGE_PATH,
               targetQuestion.getQuiz().getQuizId().toString(),
               QUESTION_FILE_STORAGE_PATH,
               targetQuestion.getQuestionId().toString()));
}

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()
         ));
}


public String saveQuizImage(MultipartFile file, Quiz targetQuiz) {
   return saveFile(file, targetQuiz,
         () -> Paths.get(ROOT_FILE_STORAGE_PATH,
               QUIZ_FILE_STORAGE_PATH,
               targetQuiz.getQuizId().toString()));
}

public byte[] getFileByPath(String filePath) {
   Path pathToFile = Paths.get(ROOT_FILE_STORAGE_PATH,filePath);
   if (!Files.exists(pathToFile)) {
      return new byte[0];
```

```
    }
    try {
        return Files.readAllBytes(pathToFile);
    } catch (IOException ioException) {
        return new byte[0];
    }

}

private String saveFiles(MultipartFile[] files, Object targetEntity, Path directory) {
    try {
        List<MultipartFile> filesList = Arrays.asList(files);
        if (!Files.exists(directory)) {
            Files.createDirectories(directory);
        }

        List<Path> filePaths = Arrays.stream(files)
                .map(file -> {
                    String fileName = createFileName(file, targetEntity);
                    Path filePath = Paths.get(directory.toString(), fileName);
                    if (Files.exists(filePath)) {
                        int orderNumber = 2;
                        do {
                            fileName = createFileName(file, targetEntity, orderNumber++);
                            filePath = Paths.get(directory.toString(), fileName);
                        } while (Files.exists(filePath));
                    }
                    return filePath;
                })
                .collect(Collectors.toList());

        for (int a =0; a < filePaths.size(); a++) {
            Files.write(filePaths.get(a), filesList.get(a).getBytes()).toString()
                    .replace("\\", "/")
                    .replace(ROOT_FILE_STORAGE_PATH, "");
        }
        return directory.toString();
    } catch (IOException ioException) {
        log.error("Failed to save a file", ioException);
        throw new RuntimeException(ioException);
```

```
    }
}

private String saveFile(MultipartFile file, Object targetEntity, Supplier<? extends Path> pathSupplier) {
    Path questionDirPath = pathSupplier.get();
    try {
        if (!Files.exists(questionDirPath)) {
            Files.createDirectories(questionDirPath);
        }

        String fileName = createFileName(file, targetEntity);
        Path filePath = Paths.get(questionDirPath.toString(), fileName);
        if (Files.exists(filePath)) {
            int orderNumber = 2;
            do {
                fileName = createFileName(file, targetEntity, orderNumber++);
                filePath = Paths.get(questionDirPath.toString(), fileName);
            } while (Files.exists(filePath));
        }

        return Files.write(filePath, file.getBytes()).toString()
                .replace("\\","/")
                .replace(ROOT_FILE_STORAGE_PATH,"");
    } catch (IOException ioException) {
        log.error("Failed to save a file", ioException);
        throw new RuntimeException(ioException);
    }
}

private String createFileName(MultipartFile image, Object targetEntityObject) {
    return createFileName(image,targetEntityObject,0);
}


private String createFileName(MultipartFile image, Object targetEntityObject, int orderNumber) {
    StringBuilder fileNameBuilder = new StringBuilder();
    fileNameBuilder.append(Math.abs(targetEntityObject.hashCode()));
    if (orderNumber != 0) {
        fileNameBuilder.append('(').append(orderNumber).append(')');
    }
```

```
        fileNameBuilder.append(image.getOriginalFilename().replaceAll("[^/]*(?=\\.[^.]+($|\\?))", ""));
        return fileNameBuilder.toString();
    }

}
```

edu/quiz/project/service/QuestionService.java

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

import edu.quiz.project.data.dto.option.OptionDto;
import edu.quiz.project.data.dto.question.NewQuestionDto;
import edu.quiz.project.data.dto.question.QuestionDto;
import edu.quiz.project.data.entity.Option;
import edu.quiz.project.data.entity.Question;
import edu.quiz.project.data.entity.Quiz;
import edu.quiz.project.data.mapper.Mapper;
import edu.quiz.project.repository.QuestionRepository;
import edu.quiz.project.repository.QuizRepository;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

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

@Service
public class QuestionService {

    @Qualifier("NewQuestionMapper")
    private final Mapper<Question,NewQuestionDto> newQuestionMapper;
    @Qualifier(value = "QuestionMapper")
    private final Mapper<Question, QuestionDto> questionMapper;
    @Qualifier(value = "OptionMapper")
    private final Mapper<Option, OptionDto> optionMapper;



    private final QuestionRepository questionRepository;
    private final QuizRepository quizRepository;
```

```
private final QuizService quizService;
private final FileService fileService;

public QuestionService(Mapper<Question, NewQuestionDto> newQuestionMapper,
                       Mapper<Question, QuestionDto> questionMapper,
                       Mapper<Option, OptionDto> optionMapper,
                       QuestionRepository questionRepository,
                       QuizRepository quizRepository,
                       QuizService quizService, FileService fileService) {
    this.newQuestionMapper = newQuestionMapper;
    this.questionMapper = questionMapper;
    this.optionMapper = optionMapper;
    this.questionRepository = questionRepository;
    this.quizRepository = quizRepository;
    this.quizService = quizService;
    this.fileService = fileService;
}


public List<QuestionDto> getQuestions(long testId) {
    return questionRepository.getByQuizQuizId(testId).stream()
            .map(questionMapper::entityToDto)
            .collect(Collectors.toList());
}

public List<Question> createNewQuestions(long testId, List<NewQuestionDto> newQuestionDtos) {
    return newQuestionDtos.stream()
            .map(dto -> createNewQuestion(testId, dto))
            .collect(Collectors.toList());
}

public Question createNewQuestion(long testId, NewQuestionDto newQuestionDto) {
    Quiz targetQuiz = quizService.getQuizById(testId);
    Question question = newQuestionMapper.dtoToEntity(newQuestionDto);
    question.setQuiz(targetQuiz);

    return questionRepository.save(question);
}

public String savePicture(MultipartFile file, Long questionId) {
```

```
        Question targetQuestion = Optional.ofNullable(questionRepository.getById(questionId))
                .orElseThrow(() -> new RuntimeException("Question not found"));
        targetQuestion.setImagePath(
                fileService.saveQuestionImage(file, targetQuestion));
        questionRepository.save(targetQuestion);
        return targetQuestion.getImagePath();
    }

    public Question createNewQuestionWithPicture(NewQuestionDto newQuestionDto, MultipartFile file) {
        Quiz quiz = Optional.ofNullable(quizRepository.getById(newQuestionDto.getTestId()))
                .orElseThrow(() -> new RuntimeException("Test not found"));
        Question question = newQuestionMapper.dtoToEntity(newQuestionDto);
        question.setQuiz(quiz);
        question = questionRepository.save(question);

        fileService.saveQuestionImage(file, question);

        return questionRepository.save(question);
    }

}
```

edu/quiz/project/service/QuizService.java

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

import edu.quiz.project.data.dto.question.QuestionDto;
import edu.quiz.project.data.dto.quiz.NewQuizDto;
import edu.quiz.project.data.dto.quiz.QuizDto;
import edu.quiz.project.data.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.data.entity.*;
import edu.quiz.project.data.mapper.Mapper;
import edu.quiz.project.repository.QuizRepository;
import edu.quiz.project.repository.QuizUserEntryRepository;
import edu.quiz.project.repository.UserRepository;
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.*;
import java.util.stream.Collectors;

@Service
public class QuizService {

    @Qualifier(value = "TestMapper")
    private final Mapper<Quiz, QuizDto> testMapper;
    @Qualifier(value = "NewTestMapper")
    private final Mapper<Quiz, NewQuizDto> newTestMapper;
    @Qualifier(value = "QuestionMapper")
    private final Mapper<Question, QuestionDto> questionMapper;

    private final QuizUserEntryRepository quizUserEntryRepository;
    private final UserRepository userRepository;
    private final QuizRepository quizRepository;
    private final FileService fileService;

    public QuizService(Mapper<Quiz, QuizDto> testMapper,
                       Mapper<QuizEntry, QuizUserEntryDto> testUserEntryDtoMapper,
                       Mapper<Quiz, NewQuizDto> newTestMapper,
                       Mapper<Question, QuestionDto> questionMapper,
                       QuizUserEntryRepository quizUserEntryRepository,
                       UserRepository userRepository,
                       QuizRepository quizRepository,
                       FileService fileService) {
        this.testMapper = testMapper;
        this.newTestMapper = newTestMapper;
        this.questionMapper = questionMapper;
        this.quizUserEntryRepository = quizUserEntryRepository;
        this.userRepository = userRepository;
        this.quizRepository = quizRepository;
        this.fileService = fileService;
    }

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

```
public QuizDto getQuizDtoById(Long testId) {
    return testMapper.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(testMapper::entityToDto)
            .collect(Collectors.toList());
}

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

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

@Transactional
public QuizDto saveNewQuiz(NewQuizDto test) {
    Quiz quizEntity = newTestMapper.dtoToEntity(test);
    return testMapper.entityToDto(quizRepository.save(quizEntity));
}

@Transactional
public QuizDto updateQuiz(QuizDto quizDtoToUpdate) {
    Quiz quizEntity = testMapper.dtoToEntity(quizDtoToUpdate);
```

```
        QuizDto quizDto = testMapper.entityToDto(quizRepository.save(quizEntity));
        quizDto.setQuestionList(quizEntity.getQuestions().stream()
                .map(questionMapper::entityToDto)
        .collect(Collectors.toList()));
        return quizDto;
    }


}
```

edu/quiz/project/service/QuizUserEntryService.java

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

import edu.quiz.project.data.constant.QuestionType;
import edu.quiz.project.data.dto.question.SubmittedQuestionDto;
import edu.quiz.project.data.dto.quiz.QuizUserEntryDto;
import edu.quiz.project.data.entity.*;
import edu.quiz.project.data.mapper.Mapper;
import edu.quiz.project.repository.QuizRepository;
import edu.quiz.project.repository.QuizUserEntryRepository;
import edu.quiz.project.repository.UserRepository;
import edu.quiz.project.service.strategy.scoring.QuestionScoringStrategy;
import org.springframework.beans.factory.annotation.Lookup;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import java.util.*;

@Component
public class QuizUserEntryService {

    @Qualifier(value = "TestUserEntryMapper")
    private final Mapper<QuizEntry, QuizUserEntryDto> testUserEntryDtoMapper;

    private final QuizUserEntryRepository quizUserEntryRepository;
```

```
    private final QuizService quizService;
    private final FileService fileService;

    public QuizUserEntryService(QuizUserEntryRepository quizUserEntryRepository,
                                UserRepository userRepository,
                                QuizRepository quizRepository,
                                Mapper<QuizEntry, QuizUserEntryDto> testUserEntryDtoMapper,
                                QuizService quizService,
                                FileService fileService) {
        this.quizUserEntryRepository = quizUserEntryRepository;
        this.testUserEntryDtoMapper = testUserEntryDtoMapper;
        this.quizService = quizService;
        this.fileService = fileService;
    }

    @Lookup
    @Qualifier("basicCheckboxQuestionScoringStrategy")
    private QuestionScoringStrategy getBasicCheckboxQuestionScoringStrategy() {
        return null;
    }
    @Lookup
    @Qualifier("basicRadiobuttonQuestionScoringStrategy")
    private QuestionScoringStrategy getBasicRadiobuttonQuestionScoringStrategy() {
        return null;
    }

    private Map<QuestionType, QuestionScoringStrategy> getQuestionScoringStrategyMap(){
        HashMap<QuestionType, QuestionScoringStrategy> questionScoringStrategyMap = new HashMap<>();
        questionScoringStrategyMap.put(QuestionType.CHECKBOX, getBasicCheckboxQuestionScoringStrategy());
        questionScoringStrategyMap.put(QuestionType.RADIO_BUTTON,
getBasicRadiobuttonQuestionScoringStrategy());
        return questionScoringStrategyMap;
    }


    @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));
    }

    private int compareResults(Quiz originalQuiz, Set<SubmittedQuestionDto> submittedQuestions) {
        Map<QuestionType, QuestionScoringStrategy> questionScoringStrategyMap =
getQuestionScoringStrategyMap();
        Map<Question,SubmittedQuestionDto> questionAnswerMap = new HashMap<>();

        originalQuiz.getQuestions().forEach(originalQuestion -> questionAnswerMap.put(originalQuestion, null));

        questionAnswerMap.entrySet().forEach(questionMapEntry -> submittedQuestions.stream()
                .filter(submittedQuestion ->
questionMapEntry.getKey().getQuestionId().equals(submittedQuestion.getQuestionId()))
                .findFirst()
                .ifPresent(questionMapEntry::setValue));

        return questionAnswerMap.entrySet().stream()
                .mapToInt(questionMapEntry -> {
                    Question question = questionMapEntry.getKey();
                    SubmittedQuestionDto answer = questionMapEntry.getValue();
                    return questionScoringStrategyMap.get(question.getType()).estimate(question,answer);
                }).sum();
    }

    public Page<QuizUserEntryDto> getUserEntriesPage(Long userId, int page, int size) {
        return quizUserEntryRepository.getByUserId(userId, PageRequest.of(page, size))
                .map(testUserEntryDtoMapper::entityToDto);
    }

    public List<String> saveUsersWebcamPhotos(MultipartFile[] photos, Long quizEntryId, User user) {
        QuizEntry quizEntry = quizUserEntryRepository.getById(quizEntryId);
```

```
        Quiz quiz = quizService.getQuizById(quizEntry.getQuiz().getQuizId());
        String photosDirectory = fileService.saveUserWebCamPhotos(photos, quiz, user);
        quizEntry.setWebcamPhoto(photosDirectory);
        return null;
    }

}
```

edu/quiz/project/service/strategy/scoring/basic/BasicCheckboxQuestionScoringStrategy.java

```
package edu.quiz.project.service.strategy.scoring.basic;

import edu.quiz.project.data.dto.option.SubmittedOptionDto;
import edu.quiz.project.data.dto.question.SubmittedQuestionDto;
import edu.quiz.project.data.entity.Option;
import edu.quiz.project.data.entity.Question;
import edu.quiz.project.service.strategy.scoring.QuestionScoringStrategy;
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

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

@Component
@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class BasicCheckboxQuestionScoringStrategy implements QuestionScoringStrategy {

    @Override
    public int estimate(Question targetQuestion, SubmittedQuestionDto submittedQuestion) {
        byte score = 0;
        List<Long> sourceOptionIds = targetQuestion.getOptions().stream()
                .filter(Option::getIsRight)
                .map(Option::getOptionId)
                .collect(Collectors.toList());
        List<Long> submittedOptionIds = submittedQuestion.getSubmittedOptionDtos().stream()
                .filter(SubmittedOptionDto::isSelected)
                .map(SubmittedOptionDto::getOptionId)
                .collect(Collectors.toList());
```

```
        if (submittedOptionIds.size() != 0 && sourceOptionIds.containsAll(submittedOptionIds)) {
            score = 1;
        }

        return score;
    }

}
```

edu/quiz/project/service/strategy/scoring/basic/BasicRadiobuttonQuestionScoringStrategy.java

```
package edu.quiz.project.service.strategy.scoring.basic;

import edu.quiz.project.data.dto.option.SubmittedOptionDto;
import edu.quiz.project.data.dto.question.SubmittedQuestionDto;
import edu.quiz.project.data.entity.Option;
import edu.quiz.project.data.entity.Question;
import edu.quiz.project.service.strategy.scoring.QuestionScoringStrategy;
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class BasicRadiobuttonQuestionScoringStrategy implements QuestionScoringStrategy {

    @Override
    public int estimate(Question targetQuestion, SubmittedQuestionDto submittedQuestion) {
        byte score = 0;
        Option sourceOption = targetQuestion.getOptions().stream()
                .filter(Option::getIsRight)
                .findFirst()
                .get();
        SubmittedOptionDto submittedOption = submittedQuestion.getSubmittedOptionDtos().stream()
                .filter(SubmittedOptionDto::isSelected)
                .findFirst()
                .orElse(null);
        if (submittedOption != null && sourceOption.getOptionId().equals(submittedOption.getOptionId())) {
```

```
            score = 1;
        }

        return score;
    }

}
```

src/main/resources/application.properties

```
#spring.datasource.url=jdbc:h2:mem:quizH2 #H2 DataSource

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

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

quizy.pictureFolderPath = E:/PicturesStorage/

spring.servlet.multipart.max-file-size=100MB
spring.servlet.multipart.max-request-size=100MB
```

Додаток Г

(обов’язковий)

Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ

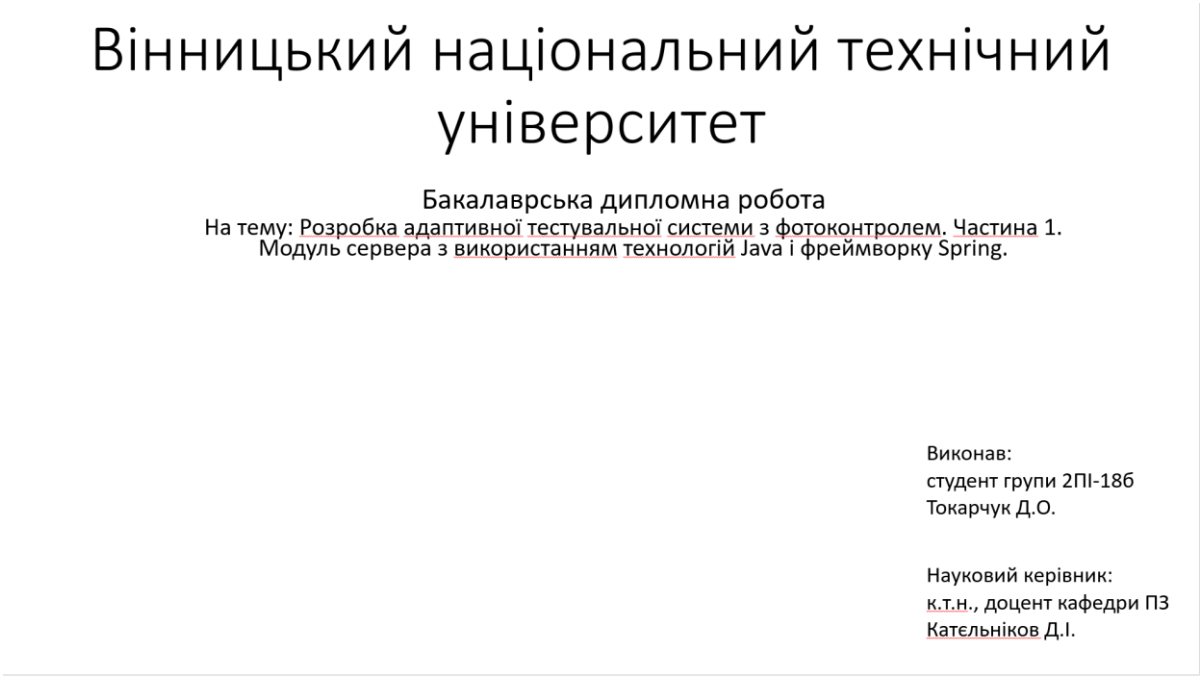


Рисунок Г.1 – Титульний слайд

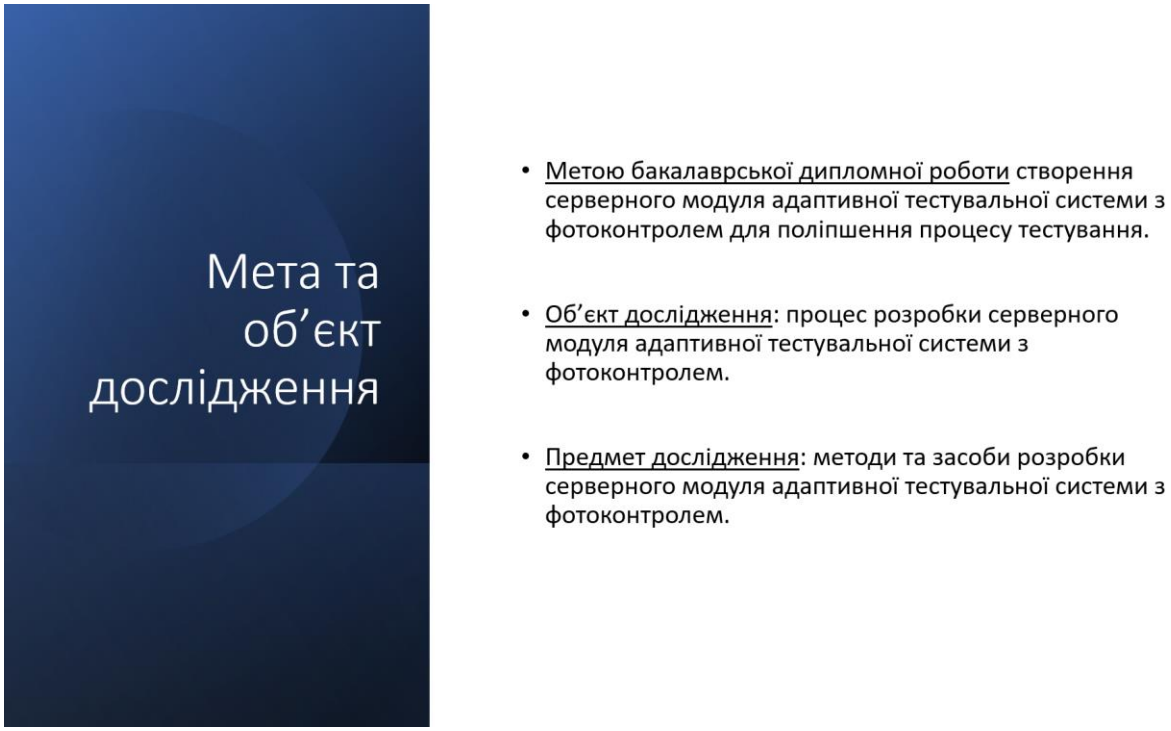


Рисунок Г.2 – Мета та об’єкт дослідження

Рисунок Г.3 – Використані технології

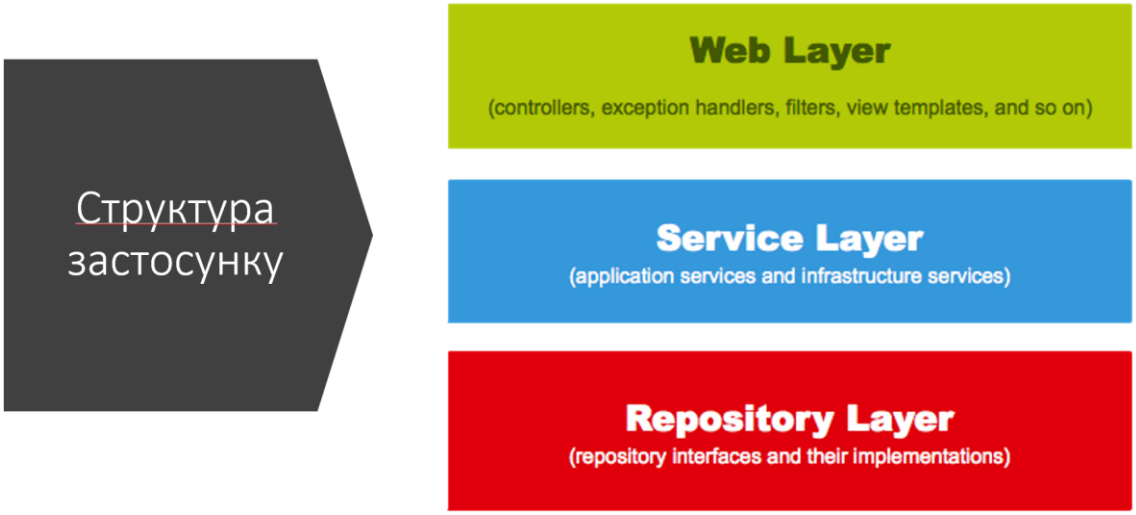


Рисунок Г.4 – Структура застосунку

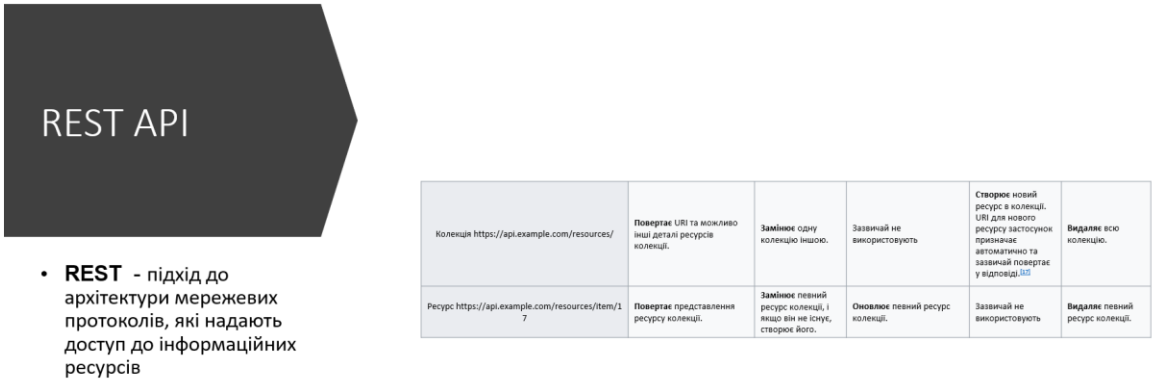


Рисунок Г.5 – REST API

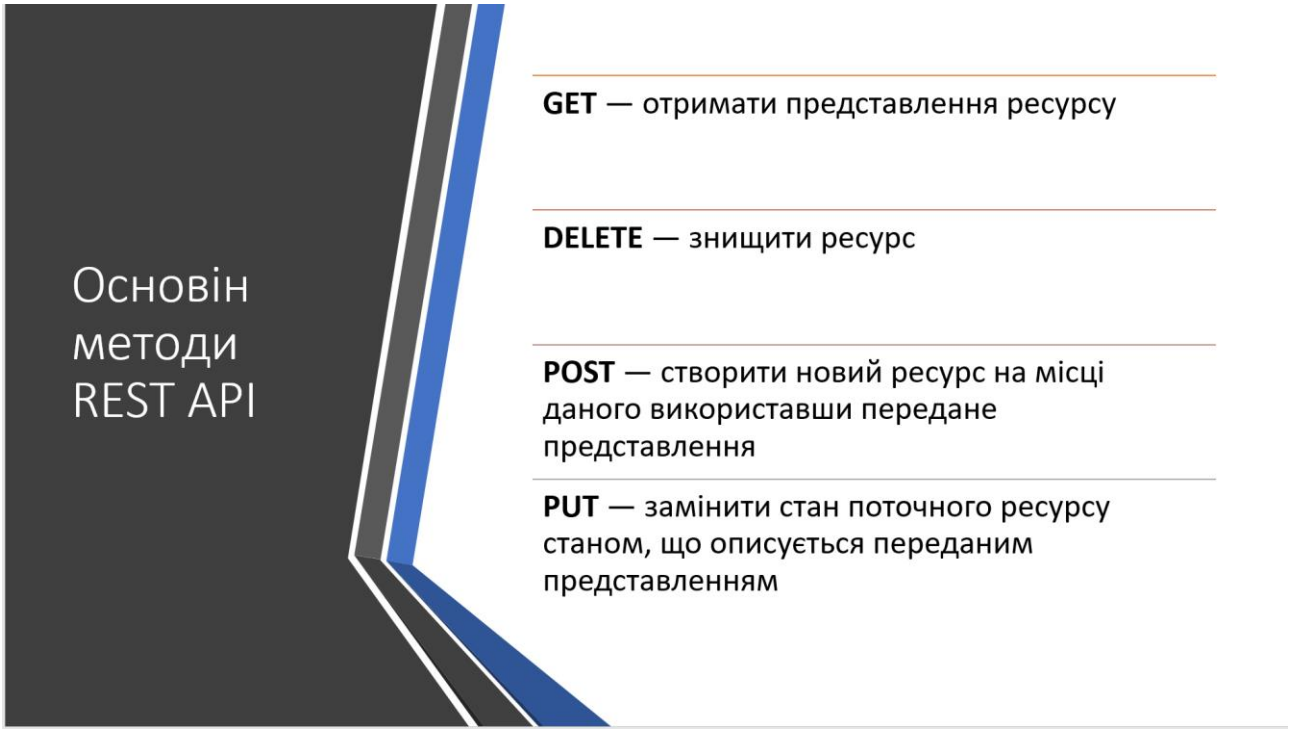


Рисунок Г.6 – Основні методи REST API

Рисунок Г.7 – Вихідні точки розробленої програми

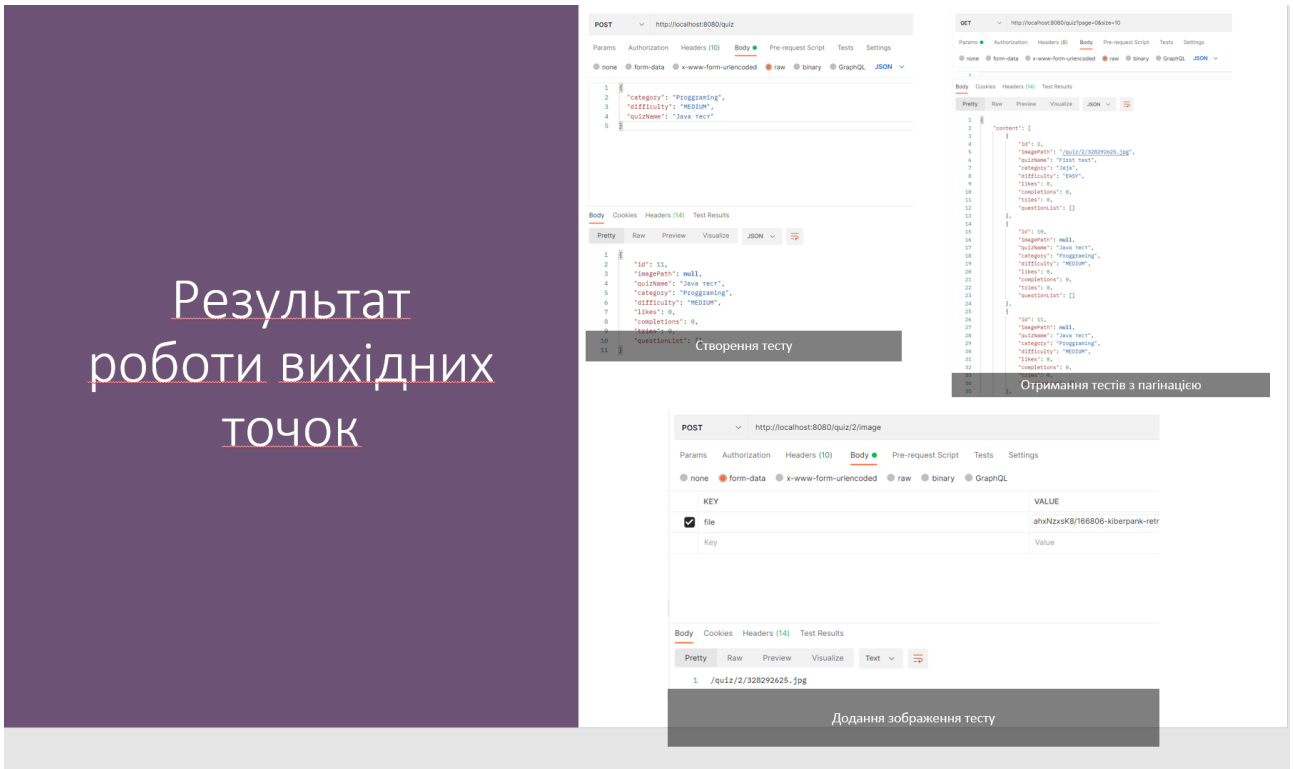


Рисунок Г.8 – Результати роботи вихідних точок

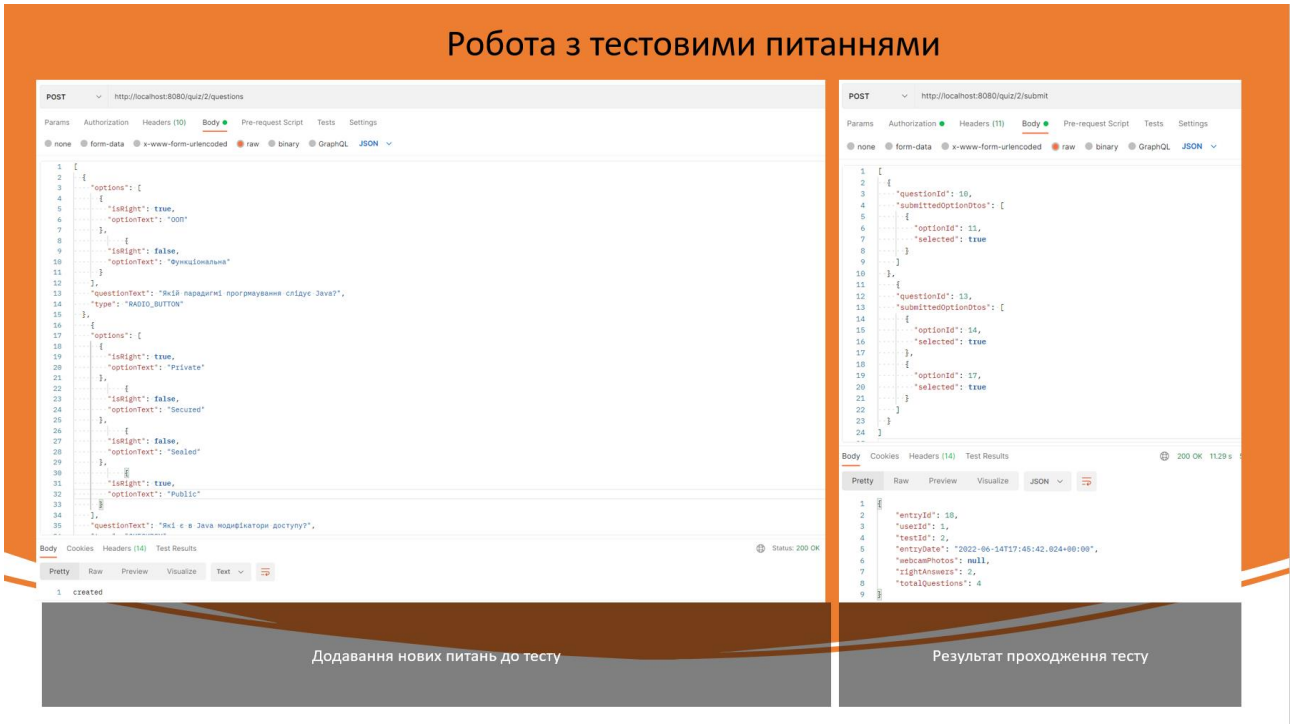


Рисунок Г.9 – Робота з тестовими питаннями

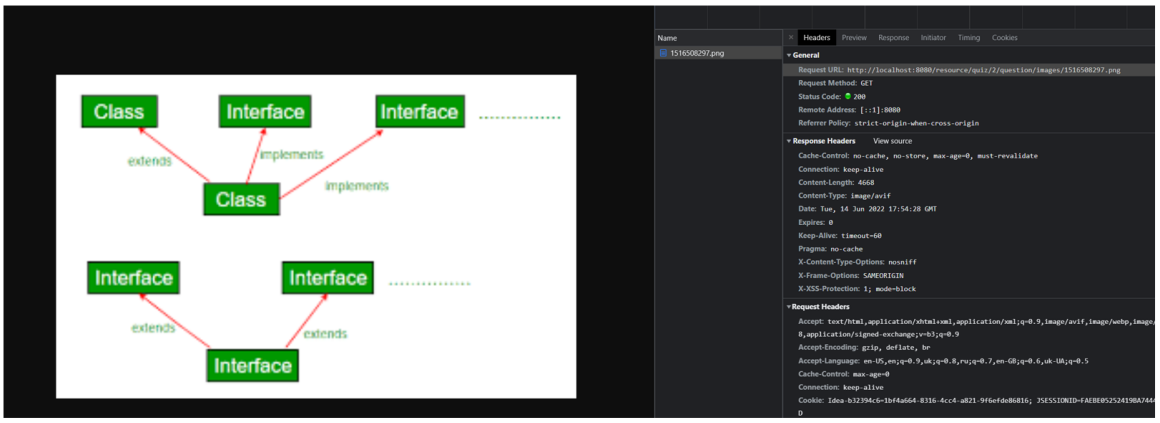
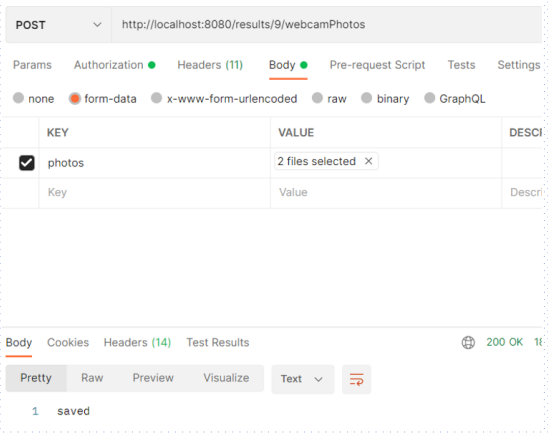


Рисунок Г.10 – Отримання медіа даних

Робота з фотокнтролем

Збереження фотографій з веб камери



Отримання результатів проходження тестування

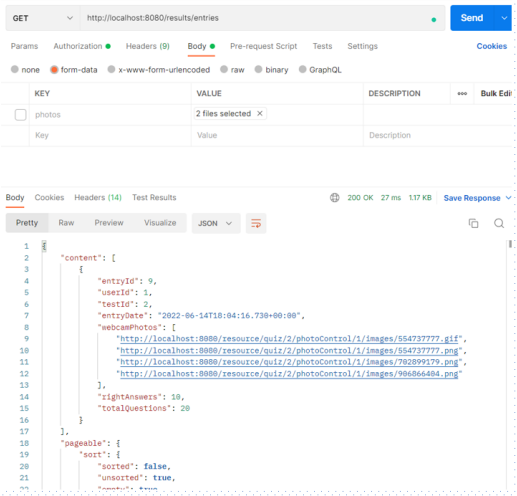


Рисунок Г.11 – Робота з фотоконтролем

JSON Web Token

- Це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи.

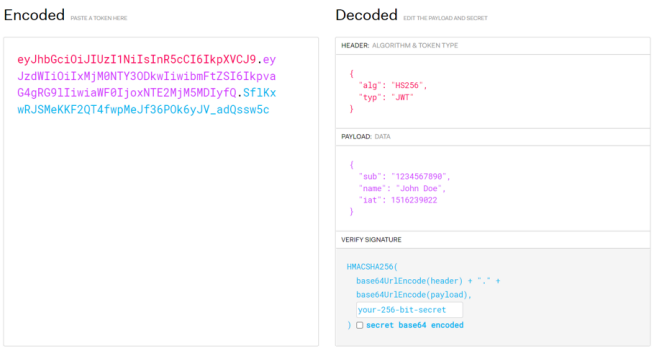


Рисунок Г.12 – JSON Web Token

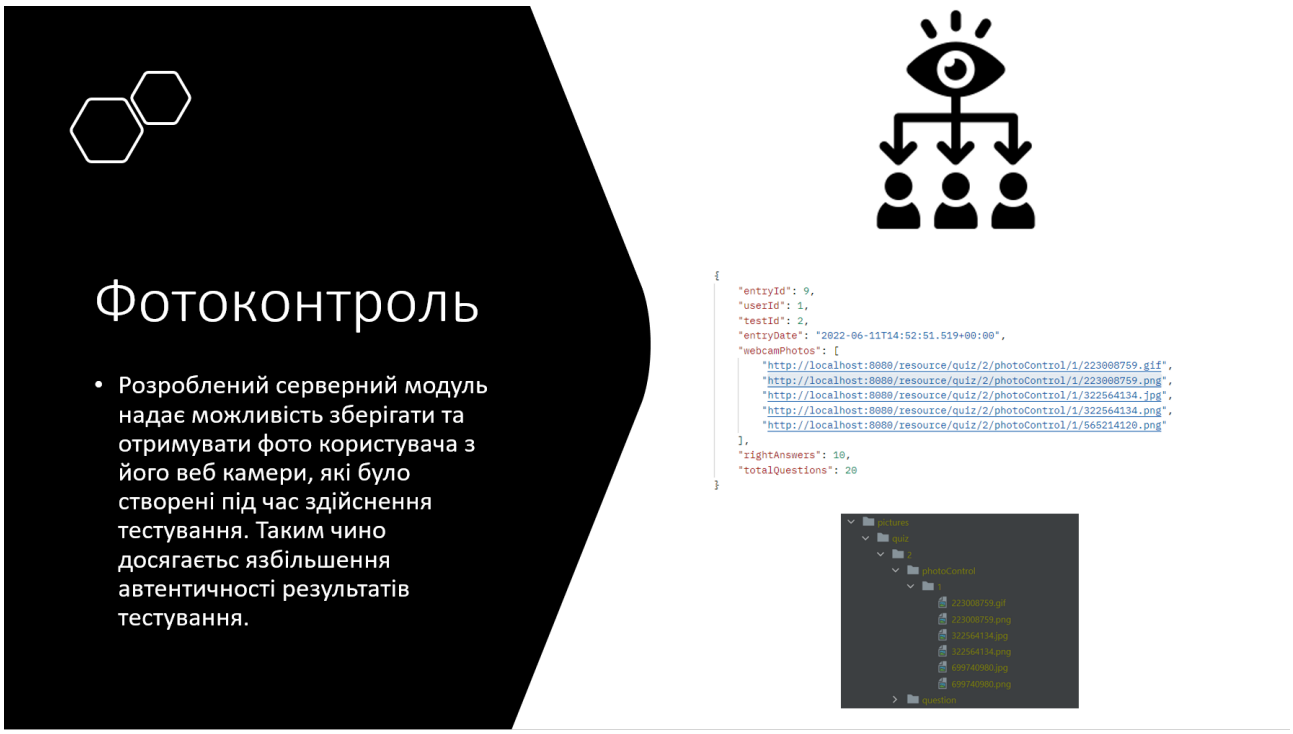


Рисунок Г.13 - Фотоконтроль

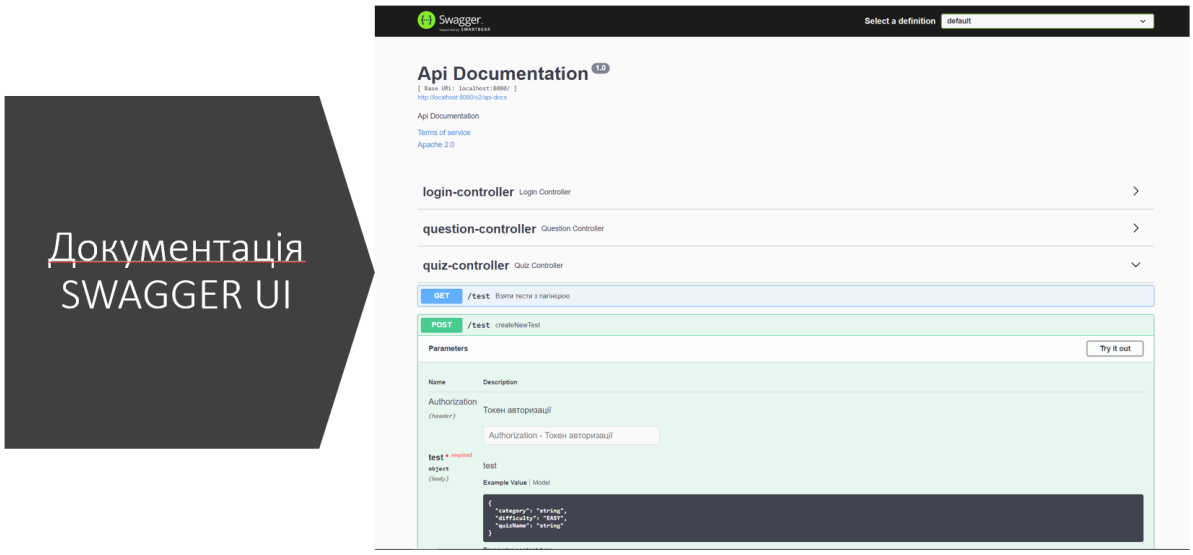


Рисунок Г.14 – Документація Swagger UI

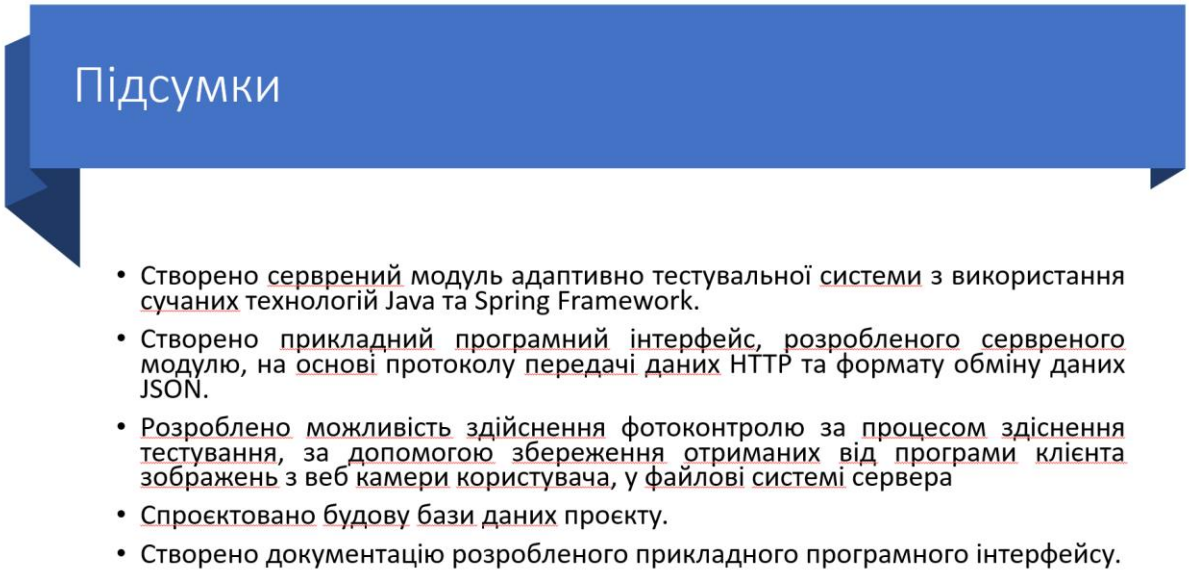


Рисунок Г.15 – Підсумки