НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок для створення тестової документації

Виконала студентка IV курсу, групи ІП-02
(шифр групи)
Литвин Анастасія Вячеславівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліхоузова Т. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц. Солдатова М. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студентці

Литвин Анастасії Вячеславівні
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для створення тестової документації
керівник проєкту Ліхоузова Тетяна Анатоліївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 27 » травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, аналіз відомих програмних продуктів, аналіз відомих алгоритмічних та технічних рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних вимог, розроблення нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема структурна компонентів програмного забезпечення

3) Схема бази даних

4) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	20.03.2024	
3	Постановка та формалізація задачі	25.03.2024	
4	Розробка інформаційного забезпечення	30.03.2024	
5	Алгоритмізація задачі	05.04.2024	
6	Обґрунтування вибору використаних технічних засобів	10.04.2024	
7	Розробка програмного забезпечення	15.04.2024	
8	Налагодження програми	10.05.2024	
9	Виконання графічних документів	20.05.2024	
10	Оформлення пояснювальної записки	29.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студентка _____ Анастасія ЛИТВИН
(підпис) (ініціали, прізвище)

Керівник _____ Тетяна ЛІХОУЗОВА
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’ятьох розділів, містить 40 таблиць, 61 рисунок та 29 джерел – загалом 89 сторінок.

Дипломний проєкт присвячений розробці програмного забезпечення, що дозволяє створювати таку тестову документацію, як тестові набори, тестові випадки та звіти про дефекти, враховуючи правила їх заповнення.

Мета – надання інструментів для створення тестової документації.

Об'єкт дослідження: програмне забезпечення для створення документації.

Предмет дослідження: методи та інструменти створення документації для тестів.

У розділі передпроєктного обстеження предметної області наведено аналіз предметної області, головні визначення з неї, проаналізовано існуючі програмні та технічні рішення в області розробки тестової документації, а також сформовано постановку задачі дипломного проєкту.

Розділ розроблення вимог до програмного забезпечення містить варіанти використання програмного забезпечення, сформовані функціональні та нефункціональні вимоги.

Розділ конструювання та розроблення програмного забезпечення присвячений розробці архітектури програмного забезпечення, обґрунтуванню обраних програмних засобів, конструюванню та аналізу безпеки програмного забезпечення.

У розділі аналізу якості та тестування програмного забезпечення визначено якість розробки, протестовано розроблене програмне забезпечення відповідно до наведених тестів та наведено контрольний приклад.

Розділ розгортання та супроводу програмного забезпечення містить покроковий опис розгортання за допомогою платформи Docker та описано процес супроводу забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, ТЕСТОВА ДОКУМЕНТАЦІЯ, ТЕСТОВИЙ НАБІР, ТЕСТОВИЙ ВИПАДОК, ЗВІТ ПРО ДЕФЕКТ, ПРОГІН ТЕСТОВОГО НАБОРУ, АРХІТЕКТУРА, БАЗА ДАНИХ, ТЕСТУВАННЯ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 40 tables, 61 figures and 29 sources - a total of 89 pages.

The diploma project is devoted to the development of software that allows creating test documentation such as test suites, test cases and defect reports, considering the rules of their filling.

The goal is to provide tools for creating test documentation.

Object of research: software for creating documentation.

Subject of research: methods and tools for creating documentation for tests.

In the section of the pre-project examination of the subject area, the analysis of the subject area, the main definitions from it, the existing software and technical solutions in development of test documentation are analysed, and the task of the diploma project is presented.

The Software Requirements Development section contains software usage options, functional and non-functional requirements.

The Software Design and Development Section are devoted to the development of the software architecture, the justification of the selected software tools, and the design and analysis of software security.

In the section on software quality analysis and testing, the quality of the development is defined, the developed software is tested according to the given tests, and a control example is given.

The Software Deployment and Maintenance section provides a step-by-step description of deployment using Docker and describes the provisioning maintenance process.

KEYWORDS: WEB APPLICATION, TEST DOCUMENTATION, TEST SUITE, TEST CASE, DEFECT REPORT, TEST SUITE RUN, ARCHITECTURE, DATABASE, TESTING.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0224.045440.01.91	Технічне завдання	18	
3	А4	КПІ.ІП-0224.045440.02.81	Пояснювальна записка	89	
4	А4	КПІ.ІП-0224.045440.03.12	Текст програми	20	
5	А4	КПІ.ІП-0224.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0224.045440.05.34	Керівництво користувача	13	
7	А3	КПІ.ІП-0224.045440.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-0224.045440.00.90		
Розроб.		Литвин А.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліхоузова Т.А.					
Н.контр.		Шулькевич Т.В.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-02	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Технічне завдання

КПІ.ІП-0224.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Анастасія ЛИТВИН

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ.

Галузь застосування: інженерами з забезпечення якості програмного забезпечення при ведені тестової документації.

Наведене технічне завдання поширюється на розробку вебзастосунку для створення тестової документації «QAHelper», котрий використовується для зручної розробки тестової документації за правилами її введення та призначена для покращення процесів ведення тестової документації інженерами з забезпечення якості, мануальними тестувальниками та тих, хто лише здобуває знання в даній сфері.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку для створення тестової документації «QAHelper» є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення тестової документації.

Метою розробки є надання інструментів для створення тестової документації, такої як тестові набори, тестові випадки та звіти про дефект, з урахуванням усіх потреб тестувальників та правил заповнення даної документації.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– сторінка реєстрації відповідно до рисунку 4.1;

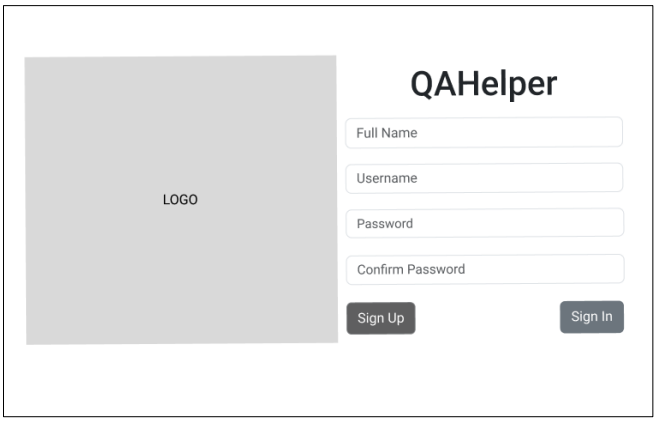


Рисунок 4.1 – Екрана форма сторінки реєстрації

– сторінка авторизації відповідно до рисунку 4.2;

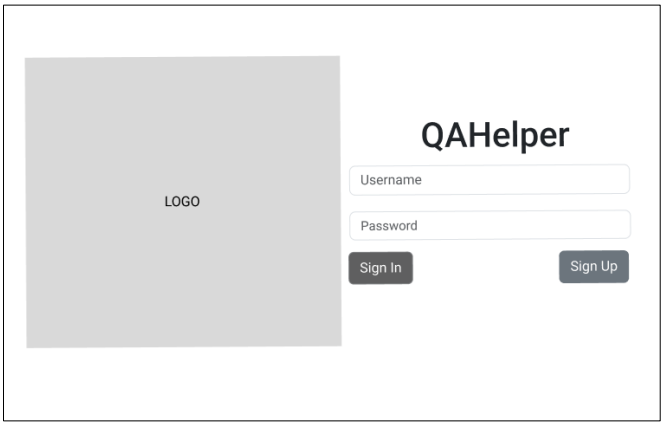


Рисунок 4.2 – Екрана форма сторінки авторизації

– сторінка з інформацією про користувача та його проєкти відповідно до рисунку 4.3;

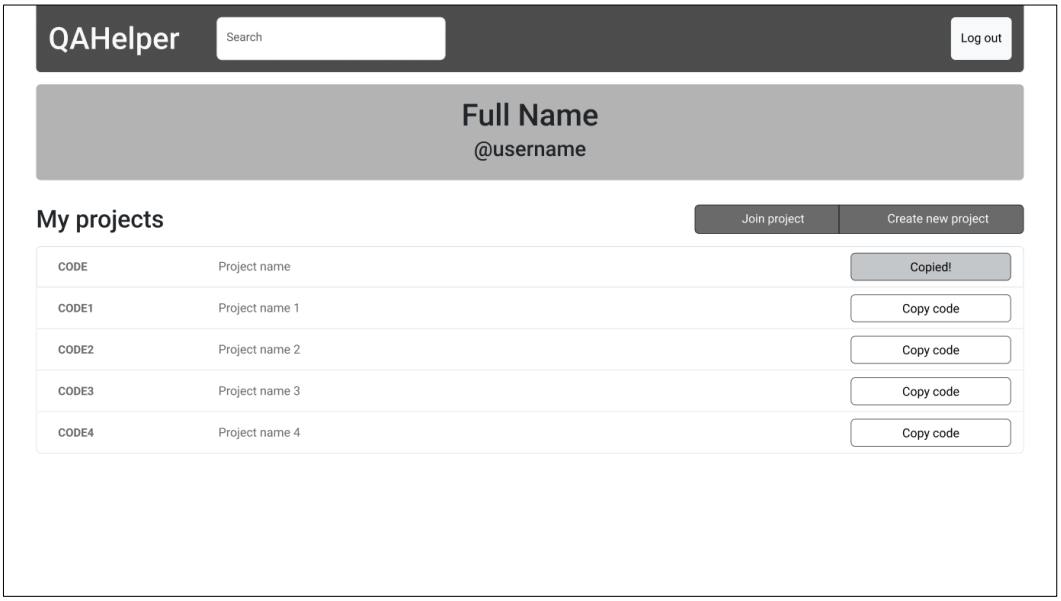


Рисунок 4.3 – Екрана форма сторінки про користувача та його проєкти

– форма створення проєкту відповідно до рисунку 4.4;

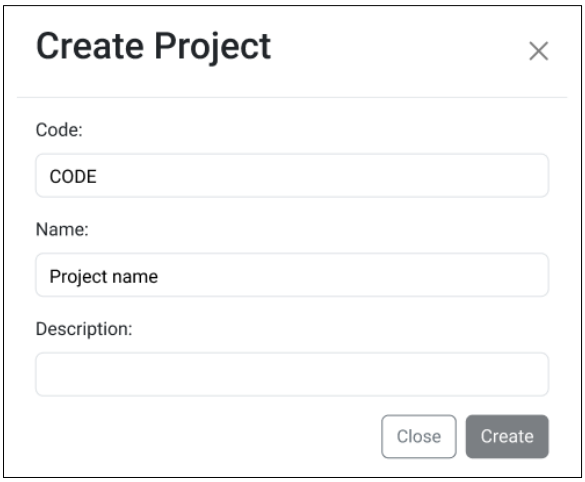


Рисунок 4.4 – Екрана форма створення проєкту

– форма доєднання до проєкту відповідно до рисунку 4.5;

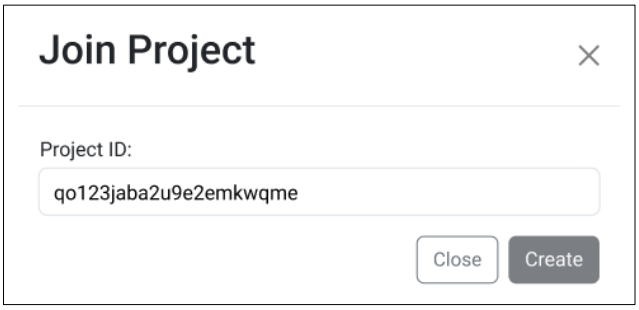


Рисунок 4.5 – Екрана форма доєднання до проєкту

– сторінка проєкту відповідно до рисунку 4.6;

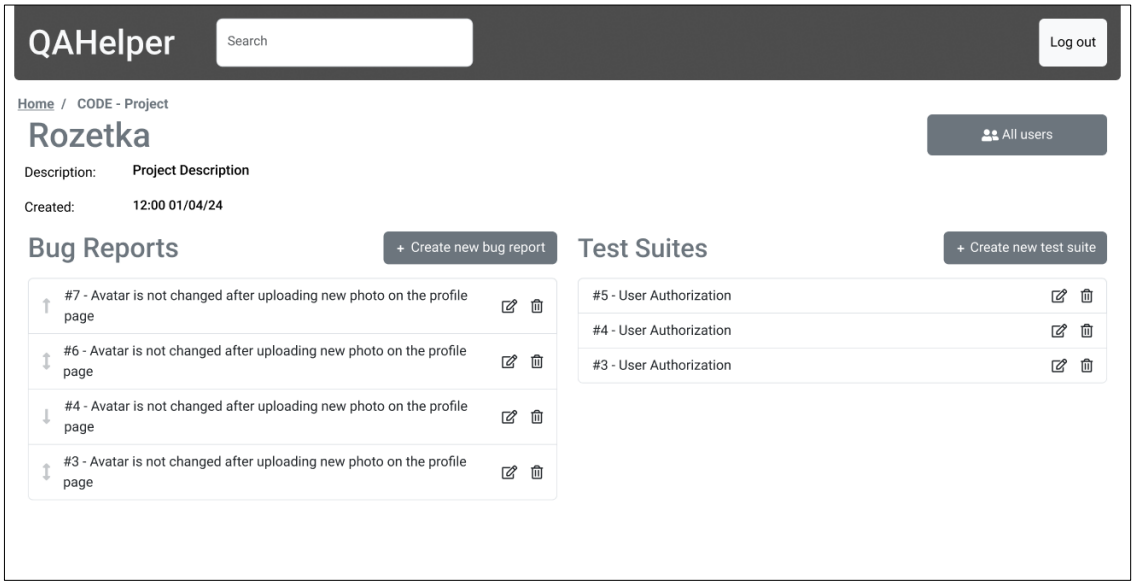


Рисунок 4.6 – Екрана форма сторінки проєкту

– сторінка усіх користувачі проєкту відповідно до рисунку 4.7;

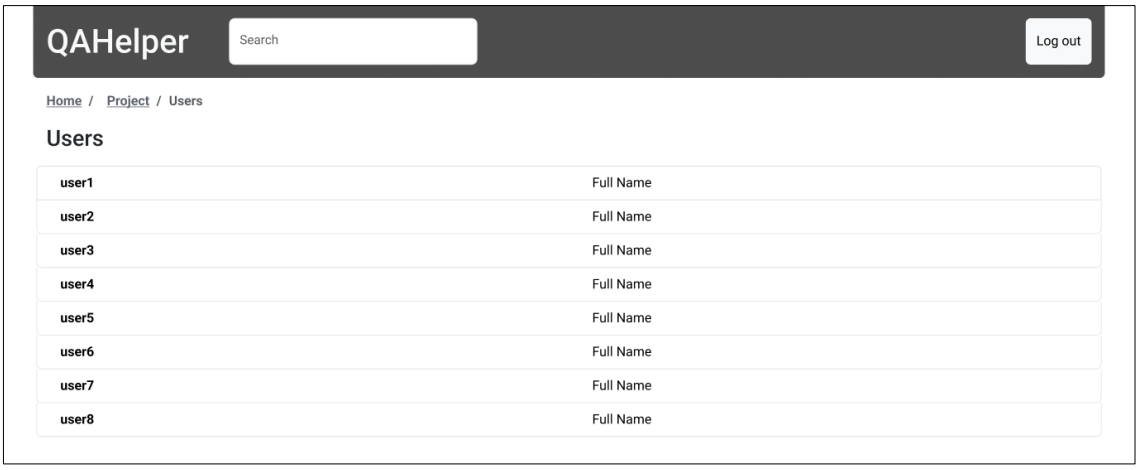


Рисунок 4.7 – Екрана форма сторінки усіх користувачів

– форма створення та редагування звіту про дефект відповідно до рисунку 4.8;

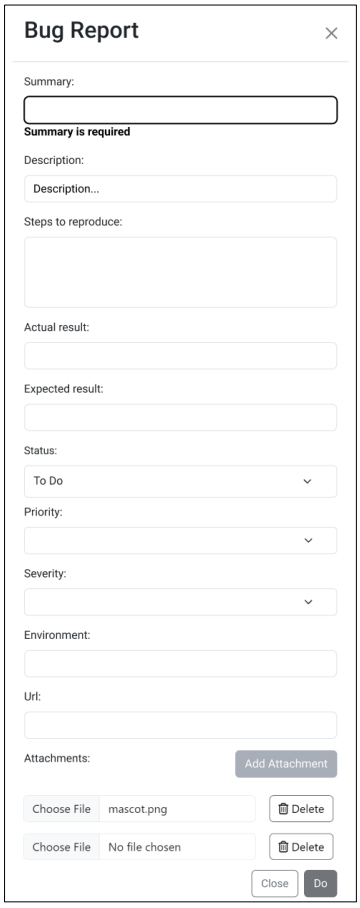


Рисунок 4.8 – Екрана форма створення та редагування звіту про дефект

– форма створення та редагування тестового набору відповідно до рисунку 4.9;

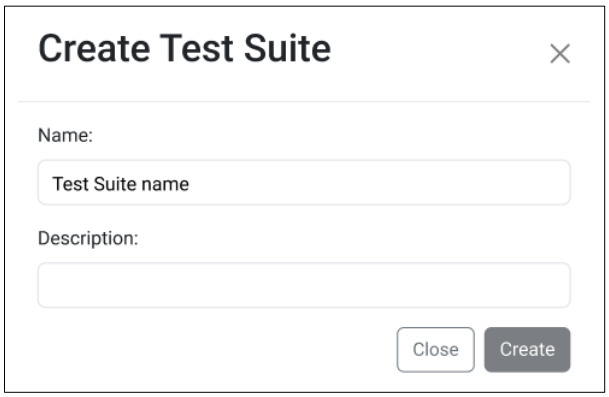


Рисунок 4.9 – Екрана форма створення та редагування тестового набору

– форма створення та редагування тестового випадку відповідно до рисунку 4.10;

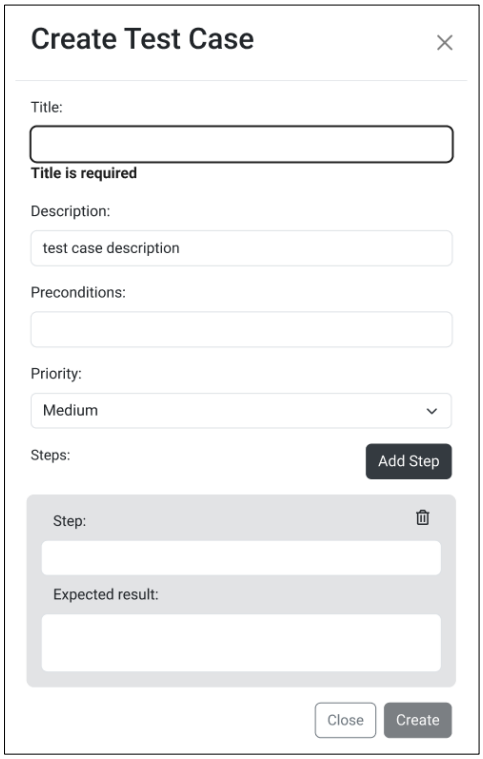


Рисунок 4.10 – Екрана форма створення та редагування тестового випадку

– сторінка звіту про дефект відповідно до рисунку 4.11;

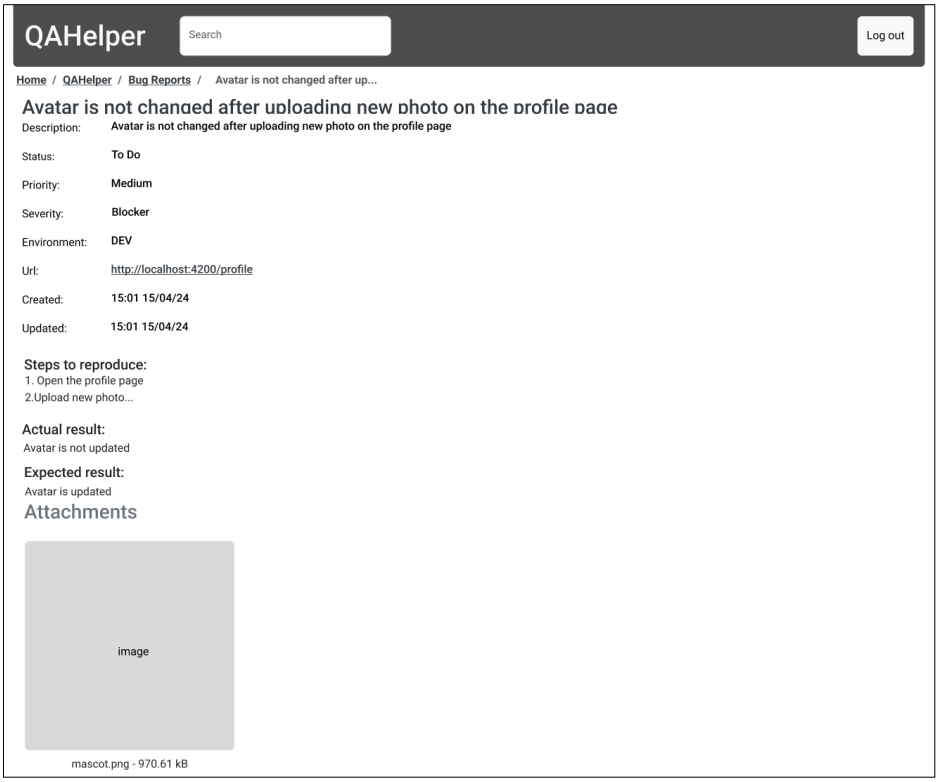


Рисунок 4.11 – Екрана форма сторінки звіту про дефект

– сторінка тестового набору відповідно до рисунку 4.12;

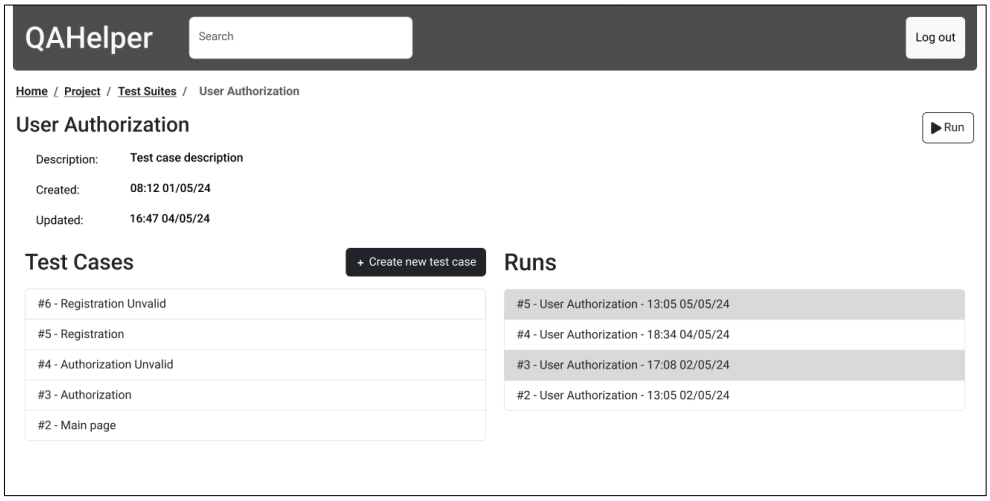


Рисунок 4.12 – Екрана форма сторінки тестового набору

– сторінка тестового випадку відповідно до рисунку 4.13;

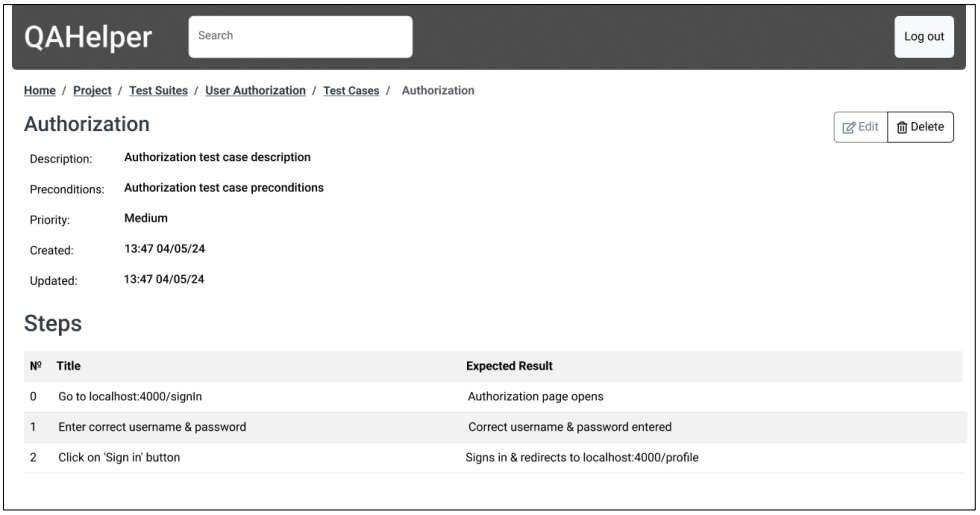


Рисунок 4.13 – Екрана форма сторінки тестового випадку

– сторінка незавершеного прогону відповідно до рисунку 4.14;

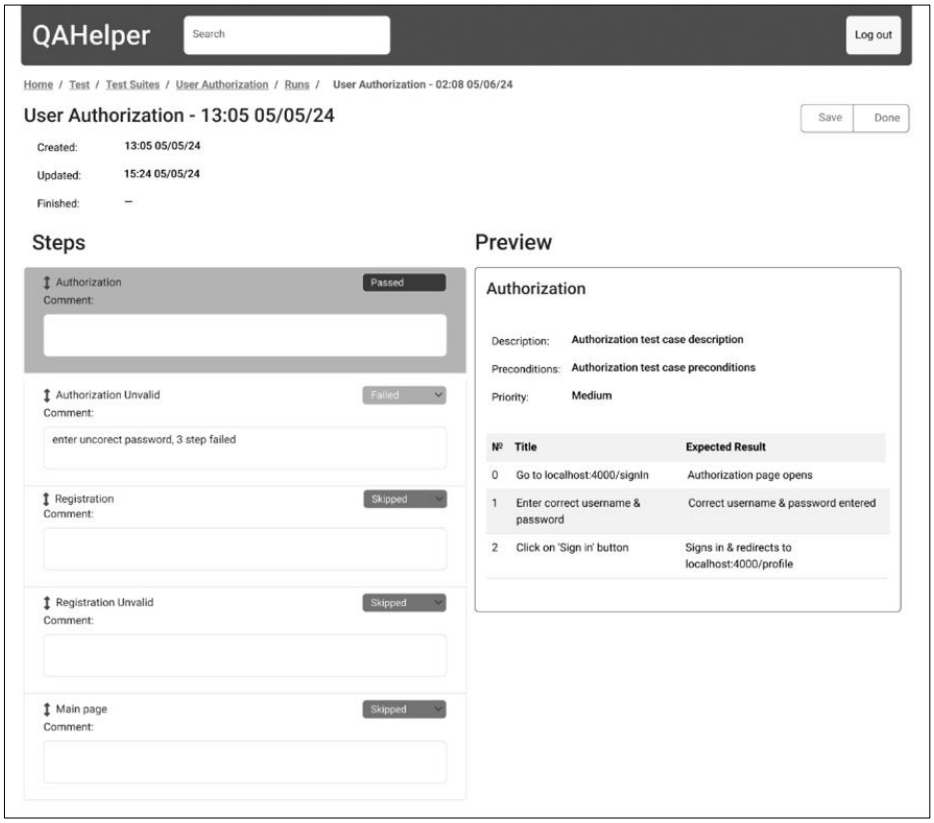


Рисунок 4.14 – Екрана форма сторінки незавершеного прогону

– сторінка виконаного прогону відповідно до рисунку 4.15;

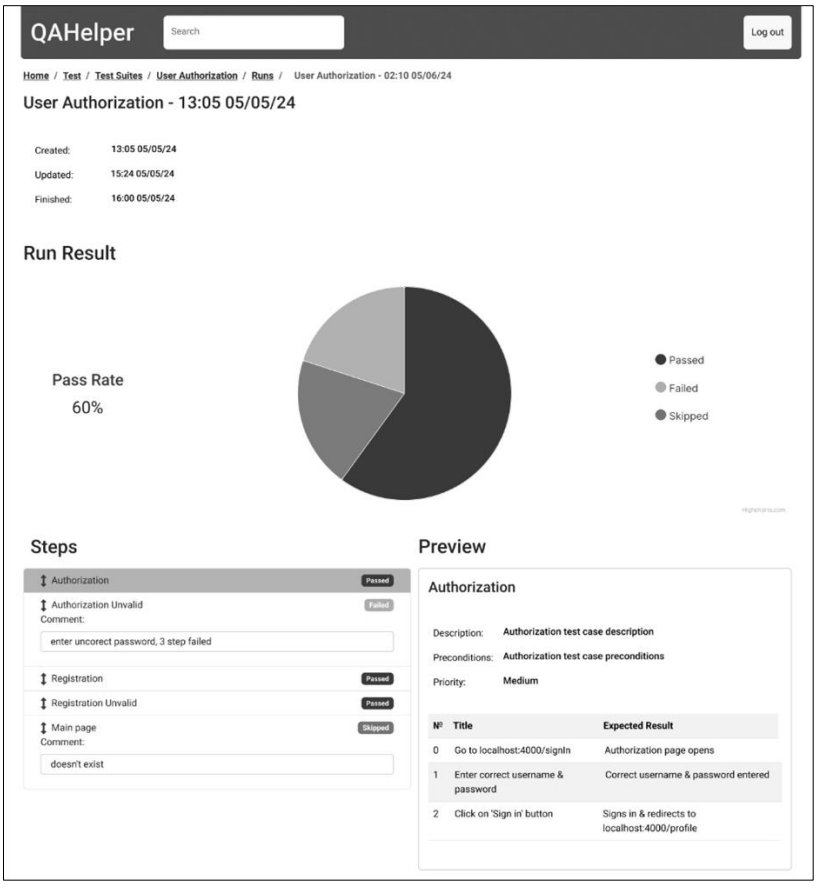


Рисунок 4.15 – Екрана форма сторінки виконаного прогону

– поле пошуку та його результатів відповідно до рисунку 4.16.

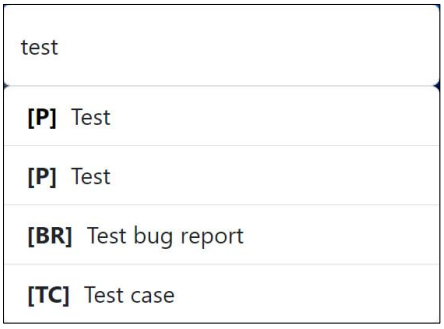


Рисунок 4.16 – Екрана форма поля пошуку та його результатів

4.1.2 Для користувача:

– можливість реєстрації;

– можливість авторизації;

– перегляд інформації про себе та свої проєкти;

– створення, доєднання до проєкту;

– створення, перегляд, редагування та видалення звітів про дефект в проєктах;

– створення, перегляд, редагування та видалення тестових наборів в проєктах;

– створення, перегляд, редагування та видалення тестових випадків в тестових наборах;

– виконання, завершення та перегляд прогонів тестових наборів;

– пошук по назвах серед проєктів, звітів про дефект, тестових наборів та тестових випадків.

4.1.3 Додаткові вимоги:

– форми для створення тестової документації мають містити усі необхідні поля відповідно до правил заповнення даної документації.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5 / AMD Ryzen 5;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 50 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 16 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows або Unix, що підтримують підключення до мережі Інтернет та можуть надати доступ до вебзастосунку через сучасний браузер.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути надані користувачем за допомогою вебзастосунку.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в форматі екранних форм на інтерфейсі користувача.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування TypeScript.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища IntelliJ IDEA.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна компонентів програмного забезпечення;
– схема бази даних;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	11.03	
2.	Аналіз існуючих методів розв’язання задачі	20.03	
3.	Постановка та формалізація задачі	25.03	Специфікації програмного забезпечення
4.	Розробка інформаційного забезпечення	30.03	Технічна документація
5.	Алгоритмізація задачі	05.04	Схема структурна програмного забезпечення та специфікація компонентів
6.	Обґрунтування вибору використаних технічних засобів	10.04	Технічна документація
7.	Розробка програмного забезпечення	15.04	Тексти програмного забезпечення
8.	Налагодження програми	10.05	Тести, результати тестування
9.	Виконання графічних документів	20.05	Графічний матеріал проєкту
10.	Оформлення пояснювальної записки	29.05	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Вебзастосунок для створення тестової документації**

КПІ.ІП-0224.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки.

API – Application programming interface, прикладний програмний Інтерфейс.

SDLC – Software development lifecycle, цикл розробки програмного забезпечення.

QA – Quality assurance, забезпечення якості.

ER – Entity-Relation diagram.

ПЗ – Програмне забезпечення.

БД – База даних.

ER – Entity-Relation diagram.

ID – Identity document, унікальний ідентифікатор.

ВСТУП

На даний момент є доволі велика кількість вакансій для мануальних тестувальників та виникає потреба в програмному забезпеченні, що могло б містити в собі можливість розробки основної тестової документації, враховуючи усі правила її розробки; це така документація як тестові набори, тестові випадки та звіти про дефект.

Існують різні інструменти для розробки даної документації, але вони не враховують усіх потреб тестувальників, через що виникають певні незручності у роботі. До таких потреб входить відсутність усіх необхідних полів, що містить в собі згадана вище тестова документація, також інструменти можуть не містити можливості розробки усіх видів згаданої документації.

Даний дипломний проєкт передбачає розробку вебзастосунку, який підтримує різноманітну тестову документацію, що враховує потреби користувачів та існуючи правила документації.

Даний застосунок можна буде використовувати при роботі мануальних тестувальників, під час навчання на курсах з вивчення тестування або самостійному навчанні даної галузі.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Життєвий цикл (SDLC) - це процес, який виконується для проєкту в організації програмного забезпечення (ПЗ). Він складається з детального плану, який описує, як розробляти, підтримувати, замінювати або покращувати застосунки. Життєвий цикл визначає методологію покращення якості ПЗ та загального процесу розробки. Одним з його етапів є тестування, іншими словами забезпечення якості продукту [1].

Забезпечення якості (QA) — це будь-який систематичний процес визначення того, чи відповідає продукт або послуга встановленим вимогам. QA встановлює та підтримує встановлені вимоги для розробки або виробництва надійних продуктів. Система забезпечення якості покликана підвищити довіру споживачів і довіру до компанії, а також покращити робочі процеси та їх ефективність [2].

Забезпечення якості у тестуванні програмного забезпечення — це важливий процес, який гарантує, що програмні продукти відповідають найвищим вимогам якості, забезпечуючи бездоганну продуктивність і задоволеність клієнтів. Це передбачає планування, виконання та моніторинг операцій тестування для виявлення помилок, мінімізації ризиків і оптимізації загальної якості програми.

У розробці програмного забезпечення якості може бути надано шляхом тестування програмного забезпечення, щоб переконатися, що в ньому немає помилок і дефектів; написання чіткої та стислої документації, зрозумілої користувачам; проведення користувацького тестування, щоб переконатися, що програмне забезпечення є коректним та зручним для користувача [3].

Цим займається інженер із забезпечення якості, він контролює кожну фазу проєктування, розробки, тестування та передачі замовнику, щоб перевірити, чи відповідає продукт стандартам якості та вимогам. Для роботи він використовує інструменти управління якістю (TestDirector, Test Manager,

TestRail), інструменти відстеження дефектів (BugZilla, Mantis), програмне забезпечення для управління проєктами (Jira, Redmine, Backlog), інструменти тестування API (Postman, SoapUI) для автоматизованих перевірок та інші [4].

QA-інженери займаються написанням великої кількості документації, до найважливішої можна віднести таку документацію:

– Test policy (політика тестування) – документ, що описує принципи, методи та важливі цілі тестування організації;

– Test strategy (тестова стратегія) – документ, що описує рівні тестування, що мають бути виконані і тестування, що входить в ці рівні, використовується на рівні організації програм для одного чи декількох проєктів;

– Test plan (план тестування) – документ, що створюється на етапі планування тестування, він описує цілі, підходи, ресурси та графік запланованих тестових активностей;

– Test Case (тестовий випадок) – документ-інструкція по тестуванню;

– Test Suite (тестовий набір) – комплект тестових прикладів, в яких зазвичай післяумова одного використовується як передумова для наступного;

– Defect Report (звіт про дефект) – документ, що містить звіт про недолік в ПЗ, який може привести компонент чи систему до неможливості виконати потрібну функцію;

– Test summary report (підсумковий звіт про тестування) – документ, що звітує висновок по задачам та результатам тестування [5].

Опишемо детальніше інформацію та поля такої тестової документації, як тестовий випадок, тестовий набір та звіт про дефект.

Тестові випадки повинні бути розроблені таким чином, щоб повністю відображати функції ПЗ. Інженери з контролю якості повинні писати тестові випадки, щоб одночасно тестувалася лише одна функція.

Тестовий випадок має містити наступні поля:

– ID – унікальний ідентифікатор тестового випадку;

– title (назва) – короткий опис суті перевірки, виконання даного тестового випадку;

– пріоритет – вказує наскільки важливим є тестовий випадок;

– передумови – опис дій, які необхідно виконати перед основними кроками;

– тестові кроки – детальний опис послідовних дій, які необхідно виконати для завершення тесту;

– очікуваний результат – те, що ми очікуємо побачити в результаті перевірки для кожного кроку [6].

Тестовий набір є зібранням тестових випадків, тому не містить якихось специфічних полів, окрім як ID – унікального ідентифікатора тестового набору, назви даного набору, опису та тестових випадків з яких він складається.

Звіт про дефект має включати в свій опис наступні поля:

– ID – унікальний ідентифікатор звіту про дефект;

– summary (тема) – короткий опис дефекту, що складений за принципом відповідей на питання Що? Де? Коли?;

– детальний опис – більш широкий опис суті дефекту;

– статус – атрибут, що показує поточний стан дефекту;

– пріоритет – атрибут, що показує наскільки важливим є вирішення даного дефекту;

– серйозність – атрибут, що показує наскільки впливає дефект на працездатність ПЗ;

– оточення – атрибут, що містить в собі інформацію про платформу та її версію, браузер та його версію, також може містити інформацію про оточення розробки та посилання на сторінку, де спостерігається дефект;

– кроки для відтворення – точно описані кроки для відтворення дефекту;

– фактичний результат – вказується що працює не так, в якому місці продукту і за яких умов;

– очікуваний результат – вказується, як саме мала б працювати система у даному випадку;

– вкладення – скріншоти, відео, що відображають де знаходиться і як виглядає дефект [7].

Як вже згадувалось раніше, для написання даної документації використовуються різні інструменти, їх досить багато, однією з проблем є те, що кожен з інструментів зазвичай орієнтується на якийсь один вид документації, це може бути тестовий випадок або звіт про дефект, через що інженер із забезпечення якості, далі тестувальник, використовує велику кількість інструментів під час своєї роботи. Також доволі значним недоліком є відсутність усіх обов’язкових полів та вигадування нових для конкретної документації в даних інструментах, таким чином відсутня уніфікації документації створеної в різних інструментах. Це також призводить до відсутності знань у тестувальників, щодо інформації, яка необхідна для документації. Особливо це позначається на новачках, що використовують лише один програмних продукт, для розробки тестової документації, що в подальшому при зміні цього продукту, не мають досвіду в правильному заповненні документації.

Покращити дані інструменти можна шляхом створення нового забезпечення, що об’єднає в собі можливість створення декількох видів тестової документації, враховуючи усі правила її створення, що і буде виконано в рамках даного дипломного проєкту.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації вебзастосунку для створення тестової документації «QAHelper». Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.2.1 Аналіз відомих програмних продуктів

Розглянемо також відомі, вебзастосунки, що використовуються для написання тестової документації.

JIRA – це вебзастосунок, що являє собою трекінгову систему, що призначений для project-менеджменту: відстеженню помилок, виконанню задач та інше. У JIRA можна створювати проєкт, задачі, звіти про дефект, групувати їх в різні списки, підключати інтеграції з іншими застосунками для розширення функціоналу всередині даного застосунку. Його перевагами є те, що в ньому можуть знаходиться усі задачі для всіх членів команди, включаючи і звіти про дефект, також можна налаштовувати типи полів, що треба заповнювати в задачах та звітах про дефект. Недоліками є те, що за замовчуванням JIRA не підтримує розроблення такої документації як тестові набори та тестові випадки. Також JIRA не містить всіх необхідних полів, що має містити звіт про дефект. Це є значним недоліком, наприклад, від обов'язкового поля «Серйозність» для звіту про дефект JIRA вирішили відмовитись, хоча це поле є вкрай важливим, куди зручніше мати в налаштуваннях «за замовчуванням» усі необхідні поля, що приведені в правилах до тестової документації [8].

Користувацький інтерфейс JIRA наведений на рисунку 1.1.

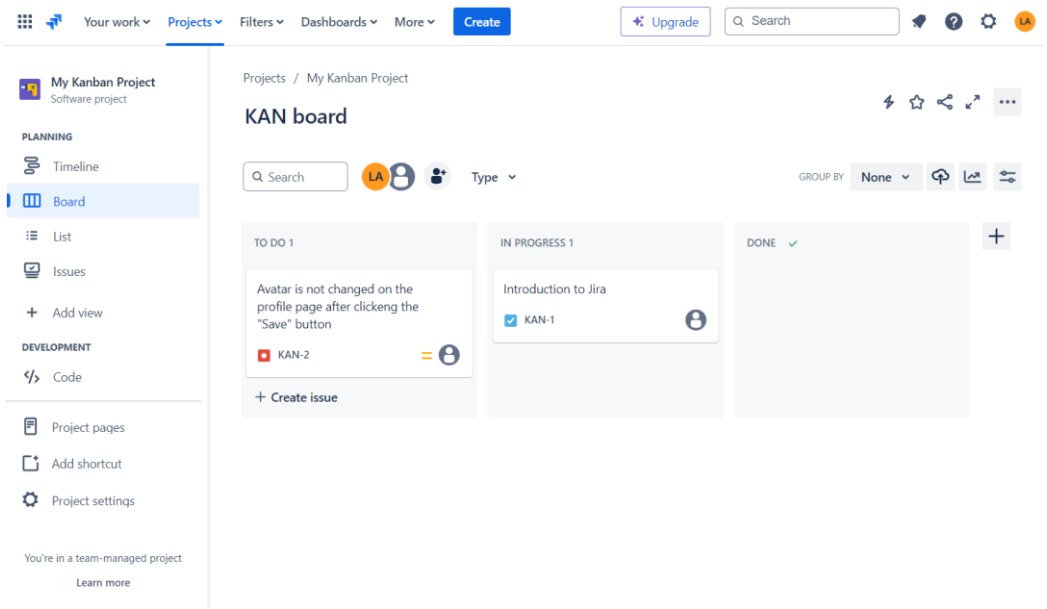


Рисунок 1.1 – Користувацький інтерфейс JIRA

TestRail – це вебзастосунок керування тестами, що призначений для роботи з тестовими випадками. У TestRail можна створювати проєкти до яких додавати секції в яких зберігати тестові випадки, виконувати тестові прогони та відстежувати їх результат, також є можливість інтегрувати його в інші застосунки. Його перевагами є широкий функціонал, що надається користувачу, коректні назви полів документації, відповідно до існуючих правил. До недоліків можна віднести відсутність повного функціоналу заведення звітів про дефект, даний функціонал існує лише для підтримок інтеграції, самі звіти є не коректними, також сам застосунки змінює назву тестових наборів на секції, що є не критичним, але впливає на розуміння та знання в області QA [9].

Користувацький інтерфейс TestRail наведений на рисунку 1.2.

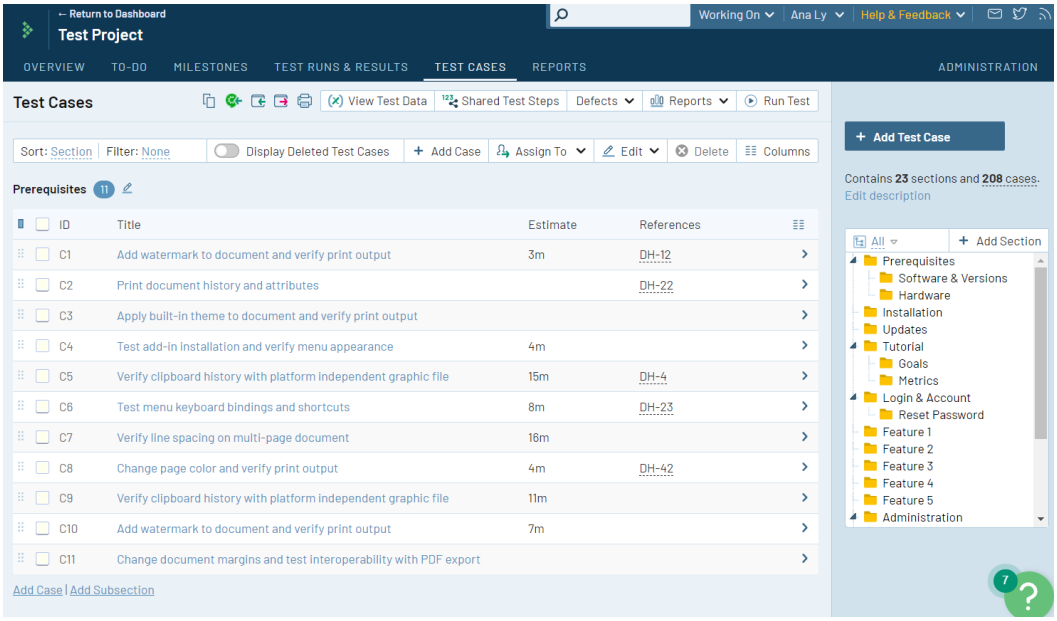


Рисунок 1.2 – Користувацький інтерфейс TestRail

Mantis Bug Tracker Demo – це вебзастосунок для відстеження помилок, що призначений для відстеження дефектів ПЗ. У Mantis Bug Tracker Demo можна створювати проєкти, створювати звіти про дефект, налаштовувати поля дефектів використовувати для project-менеджменту та створювати звіти, також є можливість інтеграції з іншими застосунками. Його переваги є простота використання, можливість створювати не лише дефекти, а й звичайні задачі, хоча їх структура буде все одно як для дефектів. До

недоліків можна віднести відсутність функціоналу підтримки такої тестової документації, як тестові набори та тестові випадки, хоча даний застосунок і є спрямований лише на звіти про дефект, він не містить «за замовчування» усіх необхідних полів для їх створення [10].

Користувацький інтерфейс Mantis Bug Tracker Demo наведений на рисунку 1.3.

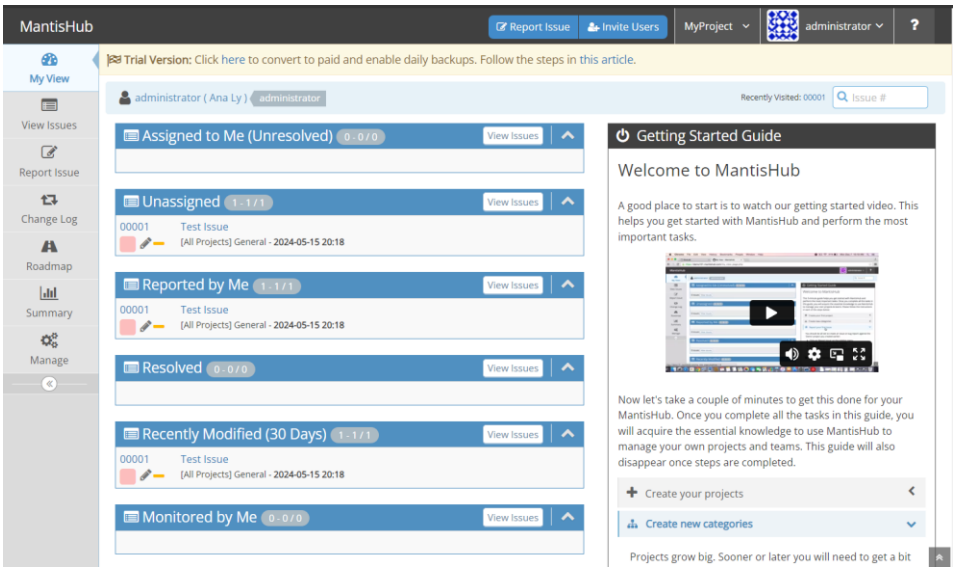


Рисунок 1.3 – Користувацький інтерфейс Mantis Bug Tracker Demo

Для порівняння проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	QA Helper	JIRA	TestRail	Mantis Bug Tracker Demo	Пояснення
Створення окремих проєктів	+	+	+	+	
Доєднання інших користувачі до проєкту	+	+	+	+	

Продовження таблиці 1.1.

Функціонал	QA Helper	JIRA	TestRail	Mantis Bug Tracker Demo	Пояснення
Створення тестових випадків	+	-	+	-	У JIRA це можливо лише за рахунок інтеграції з іншими застосунками. Mantis Bug Tracker Demo працює лише з задачами.
Виконання прогонів тестових наборів та перегляд їх результатів	+	-	+	-	У JIRA це можливо лише за рахунок інтеграції з іншими застосунками. Mantis Bug Tracker Demo працює лише з задачами.
Створення звітів про дефект	+	+	-	+	У TestRail можливо заводити лише звіти про дефект для використання їх у інтеграціях,
Наявність усіх необхідних полів тестової документації відповідно до правил	+	-	+	-	JIRA та Mantis Bug Tracker Demo мають можливість лише налаштовувати додаткові поля, за замовчування не маючи усіх правильних полів
Виконувати пошук	+	+	+	+	

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

При виборі архітектури ПЗ для порівняння було обрано клієнт-серверну та монолітну архітектуру. Модель клієнт-серверної системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Дана система є легко масштабованою та її легко підтримувати [11]. Монолітна система в свою чергу це тісно пов’язані усі компоненти системи, що у свою чергу викликає проблеми з масштабованістю, при зміні однієї частини системи скоріш за все доведеться змінювати усі інші, що займає багато часу [12]. Зважаючи на цю інформацію було обрано клієнт-серверну архітектуру.

Обираючи архітектурний патерн для серверної частини було обрано Clean architecture. Clean architecture – це зібрання принципів проєктування ПЗ, спрямовані на досягнення високої модульності, можливості тестування та незалежності коду. Даний патерн ділиться на такі рівні як рівень домену, програми, інфраструктури та презентації, у свою чергу дані рівні мають свої правила, інтерфейси та взаємодіє з іншими рівнями через абстракції [13].

1.3 Опис бізнес-процесів

Для опису бізнес процесів будуть використані BPMN моделі. Нотація моделювання бізнес-процесів (BPMN) є відкритим стандартом для діаграм бізнес-процесу, має вигляд схожий на блок-схему та використовує стандартизовану графіку для представлення учасників, вибору та перебігу процесу. Діаграми досить деталізовані - це набір символів і правил, як з’єднати ці символи для представлення бізнес-процесу, але при цьому їх все одно легко читати, що допомагає порозумітися усім учасникам бізнес-процесу [14].

Для опису основних бізнес процесу використовується BPMN модель (рисунки 1.4-1.8).

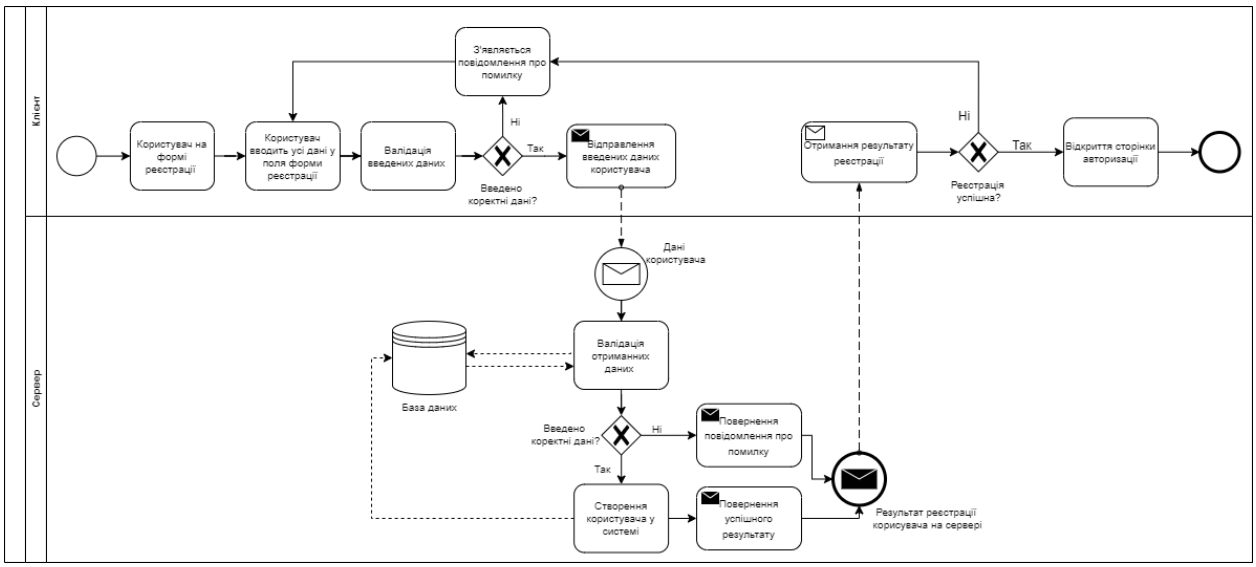


Рисунок 1.4 – Схема бізнес-процесу реєстрації

Опис послідовності реєстрації користувача:

– користувач переходить на форму реєстрації;

– користувач заповнює поля реєстрації;

– відбувається валідація даних клієнтською частиною;

– якщо валідація не пройдена, з’являється повідомлення про помилку під даними полями, користувач має вести правильні дані, якщо валідація пройдена, введені дані відправляються на сервер;

– відбувається валідація даних на сервері;

– якщо валідація пройдена, створюється новий користувач у системі та відправляється успішний результат створення користувач, якщо валідація не пройдена, відправляється повідомлення про помилку;

– клієнт отримує результат створення користувач;

– якщо реєстрація успішна, то відкривається сторінка авторизації, якщо реєстрація не успішна, з’являється повідомлення про помилку під даними полями, користувач має вести правильні дані.

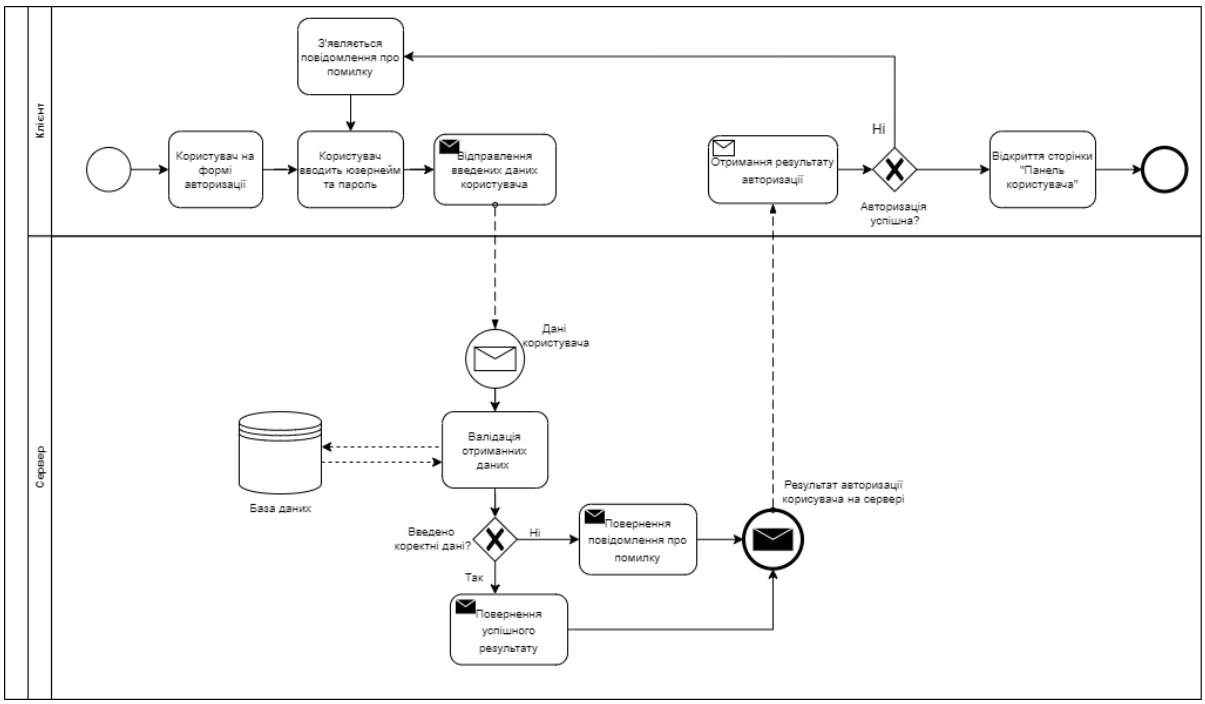


Рисунок 1.5 – Схема бізнес-процесу авторизації

Опис послідовності авторизації користувача:

– користувач переходить на форму авторизації;

– користувач заповнює поля форми авторизації - юзернейм та пароль;

– відправляються введені дані на сервер

– відбувається валідація даних на сервері;

– якщо валідація пройдена, відправляється успішний результат авторизації, якщо валідація не пройдена, відправляється повідомлення про помилку;

– клієнт отримує результат авторизації користувач;

– якщо авторизація успішна, то відкривається сторінка «Панель користувача», якщо авторизація не успішна, з’являється повідомлення про помилку під даними полями, користувач має вести правильні дані.

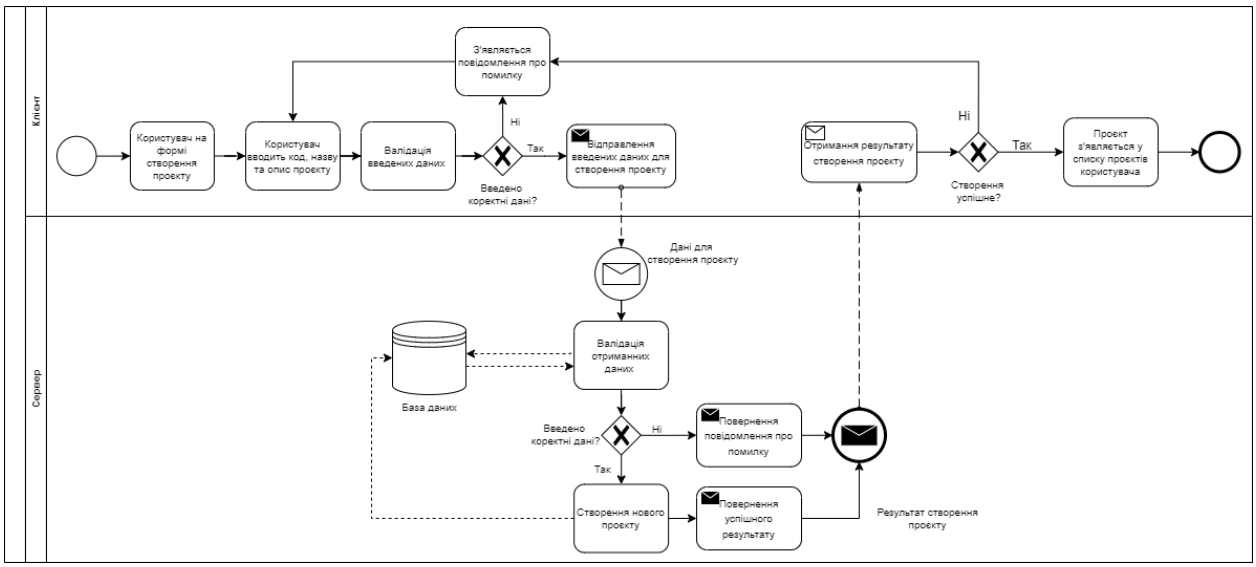


Рисунок 1.6 – Схема бізнес-процесу створення нового проєкту

Опис послідовності створення нового проєкту:

– користувач переходить на форму створення проєкту;

– користувач заповнює код, назву та опис для проєкту, що бажає створити;

– відбувається валідація даних клієнтською частиною;

– якщо валідація не пройдена, з’являється повідомлення про помилку під даними полями, користувач має вести правильні дані, якщо валідація пройдена, введені дані відправляються на сервер;

– відбувається валідація даних на сервері;

– якщо валідація пройдена, створюється новий проєкт у системі та відправляється успішний результат створення проєкту, якщо валідація не пройдена, відправляється повідомлення про помилку;

– клієнт отримує результат створення проєкту;

– якщо проєкт створено, він з’являється у списку проєктів користувача, якщо проєкт не створено, з’являється повідомлення про помилку під даними полями, користувач має вести правильні дані.

Бізнес-процеси створення звітів про дефект, тестових наборів та тестових випадків є тотожними до бізнес процесу створення проєкту.

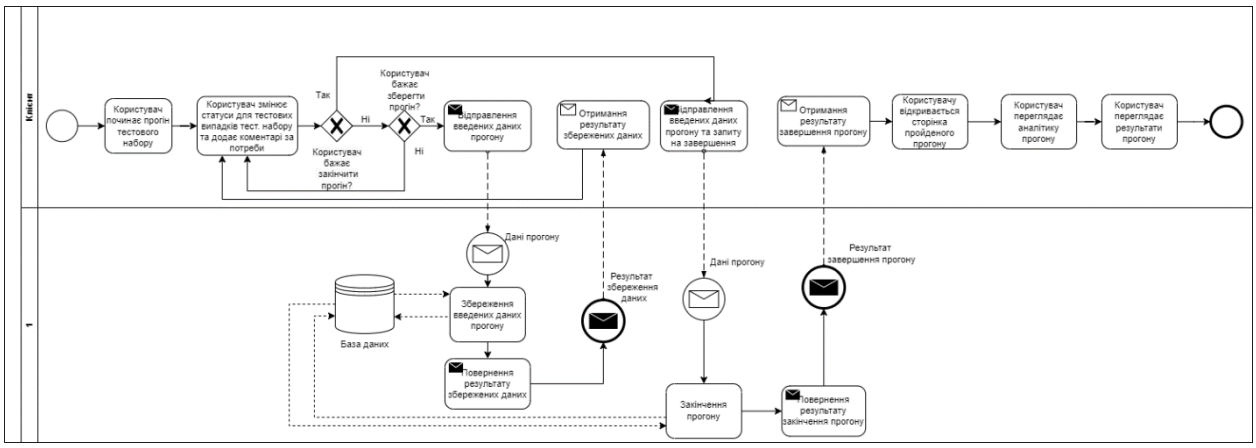


Рисунок 1.7 – Схема бізнес-процесу виконання та перегляду результатів тестового набору

Опис послідовності виконання та перегляду результатів тестового набору:

– користувач починає прогін тестового набору з його сторінки;

– користувач виставляє статуси для тестових випадків даного набору, відповідно до реального результаті їх виконання, за потреби додає коментарі;

– якщо користувач не бажає закінчити прогін, то перевіряється чи бажає користувач зберегти прогін, якщо бажає закінчити прогін, то відбувається відправка введених даних прогону та запиту на його завершення;

– якщо користувач не бажає зберегти прогін, він продовжує змінювати статуси та додавати коментарі для тестового набору, якщо користувач бажає зберегти прогін, відбувається відправка введених даних прогону;

– відбувається збереження введених даних;

– повертається результат збережених даних;

– клієнт отримує результат збереження даних;

– користувач може продовжувати змінювати статуси тестових випадків та додавати коментарі;

– коли користувач зініціював процес завершення прогону, відбувається відправка введених даних та запит на завершення прогону;

– відбувається закінчення прогону;

– повертається результат закінчення прогону;

– користувач перенаправляється на сторінку завершеного прогону;

– користувач переглядає аналітику завершеного прогону;

– користувач переглядає результати завершеного прогону.

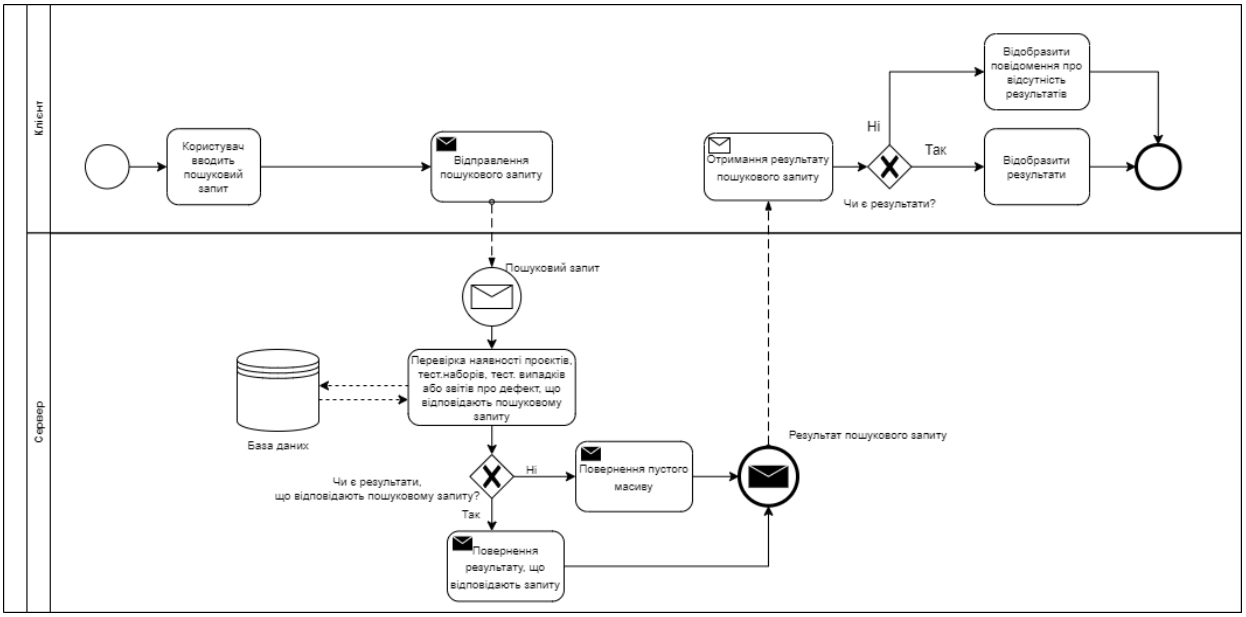


Рисунок 1.8 – Схема бізнес-процесу пошуку за назвою

Опис послідовності пошуку за назвою:

– користувач вводить пошуковий запит;

– введений пошуковий запит відправляється на сервер;

– відбувається перевірка наявності даних на сервері;

– якщо є результат, що задовольняють пошуковий запит, повернути результати, якщо результатів немає, повернути пустий масив;

– клієнт отримує результат пошукового запиту;

– якщо є результати, відобразити їх користувачу, якщо результатів немає, відобразити повідомлення про відсутність результатів.

1.4 Постановка задачі

Метою розробки є надання інструментів для створення тестової документації, такої як тестові набори, тестові випадки та звіти про дефект, з

урахуванням усіх потреб тестувальників та правил заповнення даної документації.

Необхідно розробити вебзастосунок для створення тестової документації для спрощення та покращення роботи мануальних тестувальників. Основні задачі, які він має вирішити:

- керування обліковими записами користувачів та проєктів;
- генерування різних видів тестової документації (з урахуванням існуючих правил оформлення даної документації), в тому числі:
  - створення тестових наборів;
  - створення тестових випадків;
  - виконання прогонів тестових наборів та перегляд їх результатів;
  - створення звітів про дефект;
- мати функціонал пошуку.

Висновки до розділу

У даному розділі було проаналізовано предметну область тестової документації та інструментів для її розробки. Розглянуто основні визначення по ній, правила розробки тестової документації та проблеми існуючих інструментів.

Виконано порівняння з аналогами; було визначено три аналоги, виявлено їх переваги та недоліки, порівняно функціонал. Також було сформовано вибір таких технічних рішень, як архітектура ПЗ та архітектурний патерн.

Сформовано та описано основні бізнес-процеси з використанням BPMN-діаграм.

На основі отриманої інформації було сформульовано мету та постановку задачі дипломного проєкту.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головними функціями програмного забезпечення є створення користувачем проєктів, в якому користувач може управляти тестовою документацією, такою як звіти про дефекти, тестові набори з тестовими випадками, більше функцій можна побачити на діаграмі використання, що знаходиться у графічному матеріалі, креслення 1 «Схема структурна варіантів використання».

В таблицях 2.1 - 2.14 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-01

Use case name	Реєстрація користувача в систему
Use case ID	UC-01
Goals	Зареєструвати нового користувача в систему
Actors	Неавторизований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач відкриває сторінку реєстрації. У форму реєстрації користувач вводить наступні дані: повне ім’я, юзернейм, пароль для входу в систему та повторює пароль, для його підтвердження. Після цього натискає кнопку «Зареєструватись».
Extension	У випадку введення некоректних даних, поля, що були заповнені неправильно, підсвічуються червоним та під ним з’являється повідомлення про помилку. Кнопка «Зареєструватись» неактивна.
Post-Condition	Створення нового користувача в системі. Для користувача відбувається перехід на сторінку авторизації.

Таблиця 2.2 - Варіант використання UC-02

Use case name	Авторизація користувача в системі
Use case ID	UC-02
Goals	Авторизація користувача в системі
Actors	Неавторизований користувач
Trigger	Користувач бажає увійти в застосунок
Pre-conditions	Користувач зареєстрований в системі
Flow of Events	Користувач відкриває сторінку авторизації. У форму авторизації користувач вводить власний юзернейм та пароль. Після цього натискає кнопку «Авторизуватись».
Extension	У випадку введення неправильних даних, поля підсвічуються червоним та під ними з’являється повідомлення про помилку. Кнопка «Авторизуватись» неактивна.
Post-Condition	Користувач входить в систему та переходить на сторінку «Панель користувача»

Таблиця 2.3 - Варіант використання UC-03

Use case name	Перегляд інформації про користувача
Use case ID	UC-03
Goals	Перегляд інформації про себе, що була введенаа при реєстрації користувачем
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути інформацію про себе
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває «Панель користувача» та може переглянути свій юзернейм та повне ім’я.
Extension	-
Post-Condition	-

Таблиця 2.4 - Варіант використання UC-04

Use case name	Управління проєктами
Use case ID	UC-04
Goals	Створення та перегляд проєктів користувачем
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути існуючі проєкти, учасником яких він є, або створити новий
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває «Панель користувача» та натискає кнопку «Створити проєкт», після чого відкривається форма створення проєкту. У дану форму користувач вводить назву проєкту, його опис та код для нього. Після цього натискає кнопку «Створити». Також на сторінці «Панель користувача» користувач може переглянути проєкти учасником яких він є. А натиснувши на сам проєкт може відкрити сторінку проєкту, на якій відображається вся інформація про проєкт, звіти про дефекти та тестові набори, що належать цьому проєкту.
Extension	У випадку введення некоректних даних при створенні проєкта, поля, що були заповнені неправильно, підсвічуються червоним та під ним з’являється повідомлення про помилку. Кнопка «Створити» неактивна.
Post-Condition	Користувач може переглянути створений проєкт у списку проєктів.

Таблиця 2.5 - Варіант використання UC-05

Use case name	Скопіювати унікальний код проєкта, для доєднання до проєкту інших користувачів
Use case ID	UC-05

Продовження таблиці 2.5.

Goals	Отримати унікальний код проєкта користувачем, щоб інші користувачі могли до нього доєднатись
Actors	Авторизований користувач
Trigger	Користувач бажає додати інших користувачів у проєкт
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває «Панель користувача», переходить до розділу своїх проєктів та натискає кнопку «Скопіювати» навпроти потрібного проєкта, після чого може передати даний код іншим користувачам.
Extension	-
Post-Condition	Інші користувачі доєднуються до проєкту за кодом.

Таблиця 2.6 - Варіант використання UC-06

Use case name	Доєднатися в проєкт іншого користувача за унікальним кодом
Use case ID	UC-06
Goals	Доєднатись до проєкту іншого користувача
Actors	Авторизований користувач
Trigger	Користувач бажає доєднатися до існуючого проєкту іншого користувача
Pre-conditions	Користувач авторизований, має унікальний код проєкту, що його йому надав інший користувач
Flow of Events	Користувач відкриває «Панель користувача», переходить до розділу своїх проєктів та натискає кнопку «Додатися до проєкту», після чого відкривається форма для введення коду. У дану форму користувач вводить отриманий від іншого користувача унікальний код проєкту. Після цього натискає кнопку «Доєднатись».

Продовження таблиці 2.6.

Extension	Якщо користувач вже є у даному проєкті або не існує проєкта з таким унікальним кодом, то під полем для введення коду з'являється відповідна помилка
Post-Condition	У користувача у списку проєктів з'являється новий проєкт

Таблиця 2.7 - Варіант використання UC-07

Use case name	Перегляд усіх учасників проєкту
Use case ID	UC-07
Goals	Переглянути усіх учасників певного проєкту
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути хто є учасниками відкритого проєкту
Pre-conditions	Користувач авторизований, має створений проєкт
Flow of Events	Користувач відкриває необхідний проєкт та натискає на кнопку «Усі користувачі»
Extension	-
Post-Condition	Відкривається сторінка з усіма користувачами проєкту

Таблиця 2.8 - Варіант використання UC-08

Use case name	Управління звітами про дефект
Use case ID	UC-08
Goals	Створення, перегляд, редагування та видалення звітів про дефект
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути, створити, відредагувати або видалити звіт про дефект

Продовження таблиці 2.8.

Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку проєкту в якому бажає виконати дію над звітом про дефект. Натискає кнопку «Створити новий звіт про дефект», після чого відкривається форма створення звіту. У даній формі користувач заповнює усі поля. Після цього натискає кнопку «Створити». Створений звіт з'являється у колонці звітів про дефект на сторінці проєкту. Після чого користувача перенаправляє на сторінку створеного звіту про дефект. У рядку з назвою даного звіту про дефект є кнопки редагування та видалення даного звіту. Натиснувши на кнопку «Редагувати», відкривається форма редагування звіту про дефект. У даній формі користувач змінює усі необхідні поля. Після цього натискає кнопку «Оновити». Звіт вважається оновленим. Натиснувши на кнопку «Видалити», звіт про дефект видаляється.
Extension	У випадку введення некоректних даних при створенні або редагуванні звіту про дефект, поля, що були заповнені неправильно, підсвічуються червоним та під ним з'являється повідомлення про помилку. Кнопка «Створити» або «Оновити» неактивна.
Post-Condition	-

Таблиця 2.9 - Варіант використання UC-09

Use case name	Управління тестовими випадками
Use case ID	UC-09
Goals	Створення, перегляд, редагування та видалення тестових випадків
Actors	Авторизований користувач

Продовження таблиці 2.9.

Trigger	Користувач бажає переглянути, створити, відредагувати або видалити тестовий випадок
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку тестового набору в якому бажає виконати дію над тестовим випадком. Натискає кнопку «Створити новий тестовий випадок», відкривається форма створення випадку. Користувач заповнює усі поля та натискає кнопку «Створити». Після чого користувача перенаправляє на сторінку створеного тестового випадку. Також створений випадок з’являється у списку тестових випадків на сторінці тестового набору, де можна переглянути уже існуючи тестові випадки. На сторінці тестового випадку, натиснувши на кнопку «Редагувати», відкривається форма редагування тестового випадку. У даній формі користувач змінює усі необхідні поля. Після цього натискає кнопку «Оновити». Тестовий випадок вважається оновленим. Натиснувши на кнопку «Видалити», тестовий випадок видаляється.
Extension	У випадку введення некоректних даних при створенні або редагуванні тестового випадку, поля, що були заповнені неправильно, підсвічуються червоним та під ним з’являється повідомлення про помилку. Кнопка «Створити» або «Оновити» неактивна.
Post-Condition	-

Таблиця 2.10 - Варіант використання UC-10

Use case name	Управліннями тестовими наборами
Use case ID	UC-10

Продовження таблиці 2.10.

Goals	Створення, перегляд, редагування та видалення тестових наборів
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути, створити, відредагувати або видалити тестовий набір
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку проєкту в якому бажає виконати дію над тестовим набором. Натискає кнопку «Створити новий тестовий набір», після чого відкривається форма створення набору. У даній формі користувач заповнює усі поля. Після цього натискає кнопку «Створити». Створений набір з’являється у колонці тестових наборів на сторінці проєкту. Після чого користувача перенаправляє на сторінку створеного тестового набору. У рядку з назвою даного набору є кнопки редагування та видалення набору». Натиснувши на кнопку «Редагувати», відкривається форма редагування тестового набору. У даній формі користувач змінює усі необхідні поля. Після цього натискає кнопку «Оновити». Набір вважається оновленим. Натиснувши на кнопку «Видалити», тестовий набір та його тестові випадки видаляються.
Extension	У випадку введення некоректних даних при створенні або редагуванні тестового набору, поля, що були заповнені неправильно, підсвічуються червоним та під ними з’являється повідомлення про помилку. Кнопка «Створити» або «Оновити» неактивна.
Post-Condition	-

Таблиця 2.11 - Варіант використання UC-11

Use case name	Виконати прогін тестового набору
Use case ID	UC-11
Goals	Виконання прогону тестового набору
Actors	Авторизований користувач
Trigger	Користувач бажає виконати прогін тестового набору
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку тестового набору в якому бажає виконати прогін. Натискає кнопку «Почати», після чого відкривається нова сторінка з прогоном. На сторінці користувач обирає для кожного тестового випадку тестового набору відповідний статус та за потреби додає коментарі. Для збереження інформації натискає кнопку «Зберегти», для закінчення виконання прогону натискає кнопку «Завершити».
Extension	-
Post-Condition	Після натискання кнопки «Завершити» прогін вважається виконаним

Таблиця 2.12 - Варіант використання UC-12

Use case name	Перегляд аналітики прогону тестового набору
Use case ID	UC-12
Goals	Перегляд аналітики виконаного прогону
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути аналітику готового прогону
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку виконаного прогону та може переглянути кругові діаграми разом з кількістю пройдених тестових випадків, також відсоток проходження тестових випадків

Продовження таблиці 2.12.

Extension	-
Post-Condition	-

Таблиця 2.13 - Варіант використання UC-13

Use case name	Перегляд результатів прогону тестового набору
Use case ID	UC-13
Goals	Перегляд результату виконаного прогону
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути результати готового прогону
Pre-conditions	Користувач авторизований
Flow of Events	Користувач відкриває сторінку виконаного прогону та може переглянути результати прогону, тобто статуси тестових випадків
Extension	-
Post-Condition	-

Таблиця 2.14 - Варіант використання UC-14

Use case name	Пошук за іменем проєкту, тестового набору, тестового випадку та звітам про дефекти
Use case ID	UC-14
Goals	Пошук проєктів, тестових наборів, тестових випадків та звітів про дефект, що відповідають пошуковому запиту
Actors	Авторизований користувач
Trigger	Користувач бажає знайти проєкти, тестові набори, тестові випадки та звіти про дефект, що відповідають пошуковому запиту
Pre-conditions	Користувач авторизований

Продовження таблиці 2.14.

Flow of Events	Користувач переходить в поле пошуку та вводить бажану назву. З’являється випадаючий список з проєктами, тестовими наборами, тестовими випадками та звітами про дефект, що відповідають пошуковому запиту
Extension	У випадку відсутності сутностей, що задовольняють пошуковий запит, користувачу відображається повідомлення про порожній список
Post-Condition	Користувач може перейти на сторінку сутності, що шукалась

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.15 наведено загальну модель вимог, а в таблиці 2.16 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.15 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Можливість мати акаунт в застосунку	FR-1	Високий	Високий
1.1	Реєстрація користувача	FR-2	Високий	Високий
1.2	Авторизація користувача	FR-3	Високий	Високий
1.3	Вихід із акаунту	FR-4	Високий	Високий
2	Перегляд інформації про свій акаунт	FR-5	Низький	Низький
3	Перегляд проєктів користувача	FR-6	Високий	Низький
3.1	Створення проєкту	FR-7	Високий	Низький

Продовження таблиці 2.15.

№	Назва	ID вимоги	Пріоритети	Ризики
3.2	Надавати унікальний код проєкту, що можна скопіювати, для додавання до проєкту інших користувачів	FR-8	Середній	Низький
3.3	Доєднання до проєкту іншого користувача	FR-9	Середній	Середній
3.4	Переглядати усіх учасників проєкту	FR-10	Низький	Низький
4	Перегляд звітів про дефект, що належать певному проєкту	FR-11	Високий	Низький
4.1	Створення звіту про дефект	FR-12	Високий	Низький
4.2	Редагування звіту про дефект	FR-13	Високий	Низький
4.3	Видалення звіту про дефект	FR-14	Середній	Низький
4.4	Наявність усіх полів звіту про дефект відповідно до правил заповнення документації	FR-15	Високий	Низький
4.5	Відображення пріоритету звіту про дефект відповідними символами на сторінці проєкту	FR-16	Низький	Низький
5	Перегляд тестових наборів, що належать певному проєкту	FR-17	Високий	Низький

Продовження таблиці 2.15.

№	Назва	ID вимоги	Пріоритети	Ризики
5.2	Редагування тестового набору	FR-19	Середній	Низький
5.3	Видалення тестового набору	FR-20	Середній	Низький
5.4	Можливість створювати тестові випадки всередині набору	FR-21	Високий	Низький
6	Перегляд тестових випадків, що належать певному тестовому набору	FR-22	Високий	Низький
6.1	Створення тестового випадку	FR-23	Високий	Низький
6.2	Редагування тестового випадку	FR-24	Середній	Низький
6.3	Видалення тестового випадку	FR-25	Середній	Низький
6.4	Наявність усіх полів тестового випадку відповідно до правил заповнення документації	FR-26	Високий	Низький
7	Прогін тестового набору	FR-27	Високий	Середній
7.1	Запуск прогону тестового набору	FR-28	Високий	Низький
7.2	Присвоєння статусу для тестового випадку, що є частиною тестового набору, в даному прогоні	FR-29	Середній	Низький

Продовження таблиці 2.15.

№	Назва	ID вимоги	Пріоритети	Ризики
7.3	Можливість продовжити тестовий прогін через певний час, поки той незавершений	FR-30	Низький	Низький
7.4	Завершити прогін	FR-31	Середній	Низький
7.5	Переглянути аналітику запуску даного прогону	FR-32	Середній	Низький
7.6	Переглянути результати прогону для тестового набору та тестових випадків, що їх включає набір	FR-33	Високий	Середній
8	Здійснювати пошук по назвам	FR-34	Середній	Низький
8.1	Здійснювати пошук за назвою серед проєктів	FR-35	Середній	Низький
8.2	Здійснювати пошук за назвою серед тестових наборів	FR-36	Середній	Низький
8.3	Здійснювати пошук за назвою серед тестових випадків	FR-37	Середній	Низький
8.4	Здійснювати пошук за назвою серед тем звітів про дефект	FR-38	Середній	Низький

Таблиця 2.16 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Можливість мати акаунт в застосунку Користувач, якщо він неавторизований може зареєструватись та увійти в акаунт, якщо він авторизований, то вийти з акаунту
FR-2	Реєстрація користувача Неавторизований користувач може зареєструватись у застосунку, вводячи своє повне ім’я, юзернейм, пароль та підтвердження паролю
FR-3	Авторизація користувача Неавторизований користувач може увійти у застосунок, вводячи власний юзернейм та пароль
FR-4	Вихід із акаунту Авторизований користувач може вийти з застосунку, натиснувши кнопку виходу в заголовку сторінки
FR-5	Перегляд інформації про свій акаунт Авторизований користувач може переглянути інформацію про себе на сторінці «Панель користувача»
FR-6	Перегляд проєктів користувача Авторизований користувач може переглядати проєкти, де він є учасником, на сторінці «Панель користувача»
FR-7	Створення проєкту Авторизований користувач може створити новий проєкт на сторінці «Панель користувача», вводячи код, опис та назву для нового проєкту
FR-8	Надавати унікальний код проєкту, що можна скопіювати, для додавання до проєкту інших користувачів Авторизований користувач може отримати унікальний код проєкту, щоб надати його іншим користувачам застосунку

Продовження таблиці 2.16.

ID вимоги	Назва та опис
FR-9	Доєднання до проєкту іншого користувача Авторизований користувач може додатися до проєкту іншого користувача, використавши унікальний код потрібного проєкту
FR-10	Переглядати усіх учасників проєкту Авторизований користувач може переглядати усіх учасників проєкту, де він є учасником
FR-11	Перегляд звітів про дефект, що належать певному проєкту Авторизований користувач може перейти на потрібний проєкт та переглянути наявні в ньому звіти про дефект
FR-12	Створення звіту про дефект Авторизований користувач може створити новий звіт про дефект на сторінці потрібного проєкту
FR-13	Редагування звіту про дефект Авторизований користувач може відредагувати існуючий звіт про дефект
FR-14	Видалення звіту про дефект Авторизований користувач може видалити існуючий звіт про дефект
FR-15	Наявність усіх полів звіту про дефект відповідно до правил заповнення документації Авторизований користувач може побачити усі необхідні поля при створенні, редагуванні та перегляді звіту про дефект
FR-16	Відображення пріоритету звіту про дефект відповідними символами на сторінці проєкту Авторизований користувач може зрозуміти пріоритет звіту про дефект за допомогою символів напроти заголовку звіту про дефект на сторінці проєкту

Продовження таблиці 2.16.

ID вимоги	Назва та опис
FR-17	Перегляд тестових наборів, що належать певному проєкту Авторизований користувач може перейти на потрібний проєкт та переглянути наявні в ньому звіти про дефект
FR-18	Створення тестового набору Авторизований користувач може створити новий тестовий набір на сторінці потрібного проєкту
FR-19	Редагування тестового набору Авторизований користувач може відредагувати існуючий тестовий набір
FR-20	Видалення тестового набору Авторизований користувач може видалити існуючий тестовий набір
FR-21	Можливість створювати тестові випадки всередині набору Авторизований користувач може відкрити тестовий набір і всередині нього створювати тестові випадки, що мають належати даному набору
FR-22	Перегляд тестових випадків, що належать певному тестовому набору Авторизований користувач може перейти на потрібний тестовий набір та переглянути наявні в ньому тестові випадки
FR-23	Створення тестового випадку Авторизований користувач може створити новий тестовий випадок на сторінці потрібного тестового набору
FR-24	Редагування тестового випадку Авторизований користувач може відредагувати існуючий тестовий випадок

Продовження таблиці 2.16.

ID вимоги	Назва та опис
FR-25	Видалення тестового випадку Авторизований користувач може видалити існуючий тестовий випадок
FR-26	Наявність усіх полів тестового випадку відповідно до правил заповнення документації Авторизований користувач може побачити усі необхідні поля при створенні, редагуванні та перегляді тестового випадку
FR-27	Прогін тестового набору Авторизований користувач може відкрити необхідний тестовий набір та почати, провести та закінчити прогін
FR-28	Запуск прогону тестового набору Авторизований користувач може запустити прогін зі сторінки тестового набору
FR-29	Присвоєння статусу для тестового випадку, що є частиною тестового набору, в даному прогоні Авторизований користувач може відкрити незавершений прогін та обрати статуси для тестових випадків в залежності від результату проходження тестових випадків
FR-30	Можливість продовжити тестовий прогін через певний час, поки той незавершений Авторизований користувач може відкрити незавершений прогін і далі змінювати інформацію в ньому, після чого натиснути кнопку «Зберегти»
FR-31	Завершити прогін Авторизований користувач може завершити прогін натиснувши кнопку «Завершити» на сторінці прогону

Продовження таблиці 2.16.

ID вимоги	Назва та опис
FR-32	Переглянути аналітику запуску даного прогону Авторизований користувач може відкрити сторінку завершеного прогону та переглянути аналітику проходження тестових випадків
FR-33	Переглянути результати прогону для тестового набору та тестових випадків, що їх включає набір Авторизований користувач може відкрити сторінку завершеного прогону та переглянути результати проходження тестових випадків
FR-34	Здійснювати пошук по назвам Авторизований користувач може перейти в поле пошуку, ввести потрібну назву та переглянути можливі варіанти
FR-35	Здійснювати пошук за назвою серед проєктів Авторизований користувач в полі пошуку може ввести назву проєкту, після чого під полем з’являться проєкти, що відповідають на пошуковий запит, користувач може натиснути на назву та перейти на сторінку даного проєкту
FR-36	Здійснювати пошук за назвою серед тестових наборів Авторизований користувач в полі пошуку може ввести назву тестового набору, після чого під полем з’являться тестові набори, що відповідають на пошуковий запит, користувач може натиснути на назву та перейти на сторінку даного тестового набору

Продовження таблиці 2.16.

ID вимоги	Назва та опис
FR-37	Здійснювати пошук за назвою серед тестових випадків Авторизований користувач в полі пошуку може ввести назву тестового випадку, після чого під полем з’являться тестові випадки, що відповідають на пошуковий запит, користувач може натиснути на назву та перейти на сторінку даного тестового випадку
FR-38	Здійснювати пошук за назвою серед тем звітів про дефект Авторизований користувач в полі пошуку може ввести тему звіту про дефект, після чого під полем з’являться звіти про дефект, що відповідають на пошуковий запит, користувач може натиснути на тему та перейти на сторінку даного звіту про дефект

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26	FR-27	FR-28	FR-29	FR-30	FR-31	FR-32	FR-33	FR-34	FR-35	FR-36	FR-37	FR-38
UC-1		+																																				
UC-2	+		+	+																																		
UC-3					+																																	
UC-4						+	+																															
UC-5								+																														
UC-6									+																													
UC-7										+																												
UC-8											+	+	+	+	+	+																						
UC-9																						+	+	+	+	+												
UC-10																	+	+	+	+	+																	
UC-11																											+	+	+	+	+							
UC-12																																+						
UC-13																																	+					
UC-14																																		+	+	+	+	+

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Для створюваного вебзастосунку було сформовано наступні вимоги:

– зручність користування: простий та інтуїтивно зрозумілий інтерфейс;

– сумісність з сучасними браузерами, такими як Google Chrome, Mozilla Firefox, Opera Browser;

– надійність: забезпечення цілісності інформації в БД;

– локалізація: весь функціонал подається англійською за правилами заповнення документації.

Висновки до розділу

У даному розділі було розроблено діаграму варіантів використання, описано дані варіанти використання, сформовано функціональні та нефункціональні вимоги для вебзастосунку для створення тестової документації.

Для побудови діаграми варіантів використання було визначено акторів та як вони взаємодіють з варіантами використання. Також на основі проаналізованих аналогів в попередньому розділі та описаним бізнес-процесам було сформовано та описано функціональні вимоги. Враховуючи потреби користувачів при використанні вебзастосунків було сформовано нефункціональні вимоги.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Раніше було обрано клієнт-серверну архітектуру для реалізації ПЗ. Тобто у нас є Клієнт – рівень представлення даних користувачеві, що відправляє запити на Сервер – рівень, що реалізує логіку ПЗ і обробляє необхідні дані, він реагує та у разі потреби дістає або зберігає дані з БД – рівень управління даними, та Сервер надсилає відповідь Клієнту з результатом виконання його запиту. Діаграма компонентів, що показує архітектуру ПЗ знаходиться у графічному матеріалі, креслення 2 «Схема структурна компонентів програмного забезпечення».

У якості архітектурного патерну серверної частини було обрано Clean architecture з власним модифікаціями, маємо наступних рівнів:

– Entities – це рівень, що складається з основних бізнес-об’єктів – сутностей;

– Domain Services – це рівень, що спілкується з базою даних;

– Use Cases – це рівень бізнес-логіки ПЗ, що в своїй роботі також використовує попередньо згаданий рівень;

– Infrastucture – це рівень, що відповідає за обробку та взаємодію з іншими частинами системи (клієнтом або API), отримавши запит від клієнта він викликає попередньо згаданий рівень варіантів використання [13].

Даний патерн визначає, що кожен його рівень залежить лише від наступного рівня.

Візуальне представлення архітектурного патерну Clean architecture зображено на рисунку 3.1.

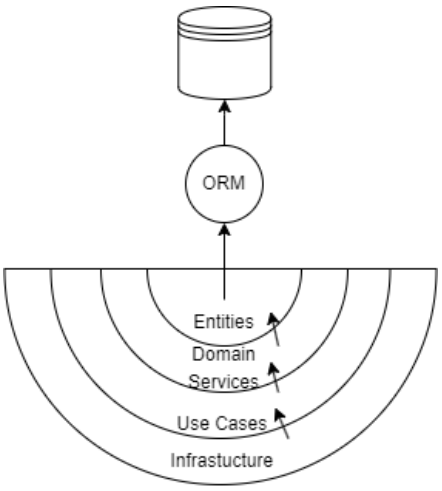


Рисунок 3.1 – Візуальне представлення реалізованого архітектурного патерну Clean architecture

Діаграма пакетів серверної частини наведено в додатку Б.

Клієнтська частина було виконана з використанням компонентної архітектури. Ця архітектура сприяє модульності коду, багаторазовому використанню та зручності обслуговування. Клієнтська частина буде складатись з дерева компонентів, де один кореневий компонент представляє всю програму. Ці компоненти вкладаються один в одного, утворюючи ієрархічну структуру [15].

Для даного рішення база даних має бути швидкою та надійною. Розглянемо два варіанти БД, що може бути використана у даному ПЗ це PostgreSQL та MongoDB. За своєю суттю це зовсім різні БД, дані в них представляються по різному, адже PostgreSQL – це реляційна БД, дані в якій зберігаються в табличному вигляді, MongoDB – це нереляційна БД, кожен запис в якій зберігаються в окремий документ. Обидві БД є досить швидкими, але через свою структуру MongoDB є швидшим, але якщо згадувати про безпеку даних, то реляційна БД перевершує нереляційну, своїми жорсткими правилами, що регулюють структуру, але незначно сповільнюють саму роботу з БД. Також MongoDB не підтримує зовнішні ключі. Отже, враховуючи всі вище зазначені фактори, буде використано PostgreSQL, адже нам потрібно зберігати дані і при цьому підтримувати їх безпеку [16].

Також в роботі буде використано ORM – спосіб узгодження програмного коду зі структурами бази даних. Механізм ORM дає змогу адресувати об’єкти, отримувати доступ до них і маніпулювати ними, не враховуючи, як ці об’єкти пов’язані зі своїми джерелами даних, також надається можливість виконувати операції з БД без використання SQL. Завдяки цьому зростає швидкість та зручність розробки [17].

У якості ORM використано TypeORM – доволі проста ORM, яка має ряд переваг, вона автоматичною створює схеми таблиць БД на основі описаних моделей, надає можливість швидко виконувати прості дії над сутностями БД, підтримує створення зв’язків між таблицями [18].

3.2 Обґрунтування вибору засобів розробки

Розробка вебзастосунку не передбачає використання якоїсь конкретної мови програмування. Тому для розробки даної предметної області вибір стояв між фреймворком Express.js для мови JavaScript та фреймворком Nest.js для мови TypeScript. Nest.js є більш надійним і добре підходить для розробки великомасштабних програм, у свою чергу Express.js є асинхронним, що дозволяє виконувати кілька операцій незалежно. Беручи за пріоритет надійність було обрано Nest.js [19].

Для клієнтської частини також немає вимог, тому був вибір між фреймворком Reactjs для мови JavaScript та фреймворком Angular для мови TypeScript. Reactjs добре підходить для проєктів з великою вкладеністю, Angular же для модульних проєктів, спираючись на предметну область, було визначено потребу у модульності клієнтської частини, тому було обрано Angular [20].

Для документування API було використано Swagger — це набір правил, специфікацій та інструментів із відкритим кодом для розробки й опису RESTful API. Специфікації API зазвичай містять таку інформацію, як підтримувані операції, параметри та результати, вимоги до авторизації, доступні кінцеві точки та необхідні ліцензії. Swagger може автоматично

генерувати цю інформацію з вихідного коду, попросивши API повернути файл документації з його анотацій. Swagger допомагає користувачам створювати, документувати, тестувати та використовувати веб-сервіси RESTful [21].

Для автентифікації було використано бібліотеку Passport – найпопулярнішу бібліотеку автентифікації node.js. Дана бібліотека автентифікує користувача, підтверджуючі його облікові дані [22].

Тематика вебзастосунку потребує збереження зображень, що будуть завантажувати користувачі, для цього було обрано MinIO – систему зберігання об’єктів. Дана система є гарно масштабованою, має високу продуктивність, високий рівень захисту даних [23].

При розробці деякої частини клієнту було використано NgRx – фреймворк для створення реактивних програм, що дозволяє управляти станом застосунку. В особливості було використано Store – для глобального керування станом, та Effects – модель ефектів, що використовує в Store [24].

Також для розробки клієнтської частини використано Bootstrap – CSS фреймворк, що допомагає розробляти полегшено шаблони, що бачить користувач. Даний фреймворк спрощує написання стилів шаблону шляхом існуючих компонентів, таким чином створення адаптивних додатків є досить швидким процесом [25].

Клієнтська частина відображає графіки, для цього було обрано бібліотеку highcharts – бібліотека, що містить інструменти для візуалізації даних, вона легко інтегрується з Angular та добре задокументовано, тому її буде легко використовувати при візуальному представленні даних [26].

У якості IDE для розробки веб-застосунку була обрана IntelliJ IDEA, так як вона має зручний інтерфейс, який також надає більше можливостей з системою контролю версій, також доступне оригінальне рішення налаштування codestyle у проєктах, на відміну від Visual Studio Code.

У якості системи керування репозиторіями програмного коду було обрано GitHub.

3.3 Конструювання програмного забезпечення

Було спроєктовано базу даних для програмного забезпечення, її ER-діаграма знаходиться у графічному матеріалі, креслення 3 «Схема бази даних».

В якості системи управління базами даних використовується Postgres. База даних серверу призначена для зберігання користувачів, а також даних їх проєктів, звітів про дефекти, тестових наборів, тестових випадків, історію їх прогонів та результати прогонів. Опис таблиць бази даних наведено у таблицях 3.1 - 3.14.

Таблиця 3.1 – Опис таблиці user

Назва поля	Тип даних	Опис
id	uuid	ідентифікатор користувача
name	varchar	повне ім’я користувача
username	varchar	юзернейм користувача
password_hash	varchar	хеш паролю користувача
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.2 – Опис таблиці project

Назва поля	Тип даних	Опис
id	uuid	ідентифікатор проєкту
name	varchar	назва проєкту
code	varchar	код проєкту
description	varchar	опис проєкту

Продовження таблиці 3.2.

Назва поля	Тип даних	Опис
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.3 – Опис таблиці user_project

Назва поля	Тип даних	Опис
user_id	uuid	ідентифікатор користувача
project_id	uuid	ідентифікатор проєкту, учасником якого є користувач

Таблиця 3.4 – Опис таблиці bug_report

Назва поля	Тип даних	Опис
id	integer	ідентифікатор звіту про дефект
summary	varchar	тема звіту про дефект
description	varchar	опис дефекту
steps_to_reproduce	varchar	кроки для відтворення дефекту
expected_result	varchar	очікуваний результат реакції системи на певні кроки
actual_result	varchar	фактичний результат реакції системи на певні кроки
status	varchar	статус виконання дефекту
priority	varchar	пріоритет дефекту
severity	varchar	серйозність дефекту

Продовження таблиці 3.4.

Назва поля	Тип даних	Опис
environment	varchar	середовище дефекту
url	varchar	посилання за яким спостерігається дефект
project_id	uuid	ідентифікатор проєкту в якому відтворюється дефект
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.5 – Опис таблиці test_suite

Назва поля	Тип даних	Опис
id	integer	ідентифікатор тестового набору
name	varchar	назва тестового набору
description	varchar	опис тестового набору
project_id	uuid	ідентифікатор проєкту до якого належить тестовий набір
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.6 – Опис таблиці test_case

Назва поля	Тип даних	Опис
id	integer	ідентифікатор тестового випадку
title	varchar	тема тестового випадку
description	varchar	опис тестового випадку

Продовження таблиці 3.6.

Назва поля	Тип даних	Опис
priority	varchar	пріоритет тестового випадку
preconditions	varchar	передумови для виконання тестового випадку
test_suite_id	integer	ідентифікатор тестового набору до якого належить тестовий випадок
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.7 – Опис таблиці test_case_step

Назва поля	Тип даних	Опис
id	integer	ідентифікатор кроку тестового випадку
title	varchar	заголовок кроку тестового випадку
expected_result	varchar	очікуваний результат кроку
test_case_id	integer	ідентифікатор тестового випадку до якого належить крок
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.8 – Опис таблиці run

Назва поля	Тип даних	Опис
id	integer	ідентифікатор прогону

Продовження таблиці 3.8.

Назва поля	Тип даних	Опис
test_suite_id	integer	ідентифікатор тестового набору до якого належить прогін
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД
finished_at	timestamp	час завершення прогону

Таблиця 3.9 – Опис таблиці run_step

Назва поля	Тип даних	Опис
run_id	integer	ідентифікатор прогону
test_case_id	integer	ідентифікатор тестового випадку, що належить до тестового набору прогону
status	varchar	статус тестового випадку
name	varchar	ім’я тестового випадку
comment	varchar	коментар до тестового випадку в даному прогоні
created_at	timestamp	час створення запису в БД
updated_at	timestamp	час оновлення запису в БД

Таблиця 3.10 – Опис таблиці attachments

Назва поля	Тип даних	Опис
id	nanoid	ідентифікатор вкладення
name	varchar	назва вкладення

Продовження таблиці 3.10.

Назва поля	Тип даних	Опис
size	integer	розмір вкладення
bug_report_id	integer	ідентифікатор звіту про дефект до якого належить вкладення
extension	varchar	розширення файлу вкладення
mime_type	varchar	медіа тип вкладення
created_at	timestamp	час створення запису в БД

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.11.

Таблиця 3.11 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ IDEA	Середовище розробки програмного забезпечення.
2	NestJS	Фреймворк для мови TypeScript. Використовувався для розробки вебсервісу.
3	Angular	Фреймворк для мови TypeScript. Використовувався для створення демонстраційного клієнтського застосунку.
4	Postgres	База даних програмного забезпечення.
5	TypeORM	ORM для роботи з БД для серверної частини.
6	Swagger	Інструмент автоматичного опису API.

Продовження таблиці 3.11.

№ п/п	Назва утиліти	Опис застосування
7	Passport	Бібліотека, що була використана для реалізації автентифікації.
8	MinIO	Система для зберігання об’єктів інформації: зображення, відео, файли і т.д.
9	NgRx	Фреймворк для управління станом застосунку.
10	Bootstrap	Фреймворк для розробки шаблонів стилю користувацького застосунку.
11	Highcharts	Npm пакет для візуального відображення даних на клієнтській частині.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

У застосунку, де є поділ на користувачів з їх особистими даними, їх безпека вкрай важлива. Тому при реєстрації користувача у системі буде зберігатись не його пароль, а хеш його пароля.

При авторизації користувачу видається JWT-токен доступу, що дійсний лише 1 годину. Усі методи API, окрім методів реєстрації та авторизації є непублічними, це означає, що при використання даних методів неможливо отримати дані про інших користувачів, не маючи відповідного токену.

Висновки до розділу

У даному розділі було спроєктовано вебзастосунок, що розробляється. Було описано архітектуру усього застосунку та надано діаграму його

компонентів. Також було описано обраний архітектурний патерн, показано його структуру. Враховуючи цю інформацію, було побудовано діаграму пакетів серверної частини застосунку. Було описано розробку клієнтської частини застосунку. Також порівняно рішення для можливих баз даних, на основі чого було обрано ту, що задовольняє вимоги – PostgreSQL.

Обрано технічні засоби, провівши порівняння можливих засобів, що могли бути використані, для клієнтської частини застосунку було обрано фреймворк Angular, для серверної NestJS, розробка буде проводитись в IDE IntelliJ IDEA. Описано використанні бібліотеки та утиліти.

Виконано конструювання бази даних, виконано її ER-діаграму та описано таблиці. Також виконано аналіз безпеки даних даного вебзастосунку.

Реалізовано вебзастосунок для створення тестової документації «QAHelper».

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Після розробки програмного забезпечення варто оцінити якість даної розробки та виконати тестування.

Метриками для оцінки якості ПЗ обрано наступні:

– надійність;

– зручність підтримки;

– безпека;

– рівень дублювання коду.

Для того, щоб оцінити дані метрики було використано сервіс SonarQube, що призначений для безперервного аналізу та вимірювання якості програмного продукту [27]. Для цього в нього було завантажено сервісну та клієнтську частину застосунку. Результати аналізу зображенні на рисунку 4.1–4.2.

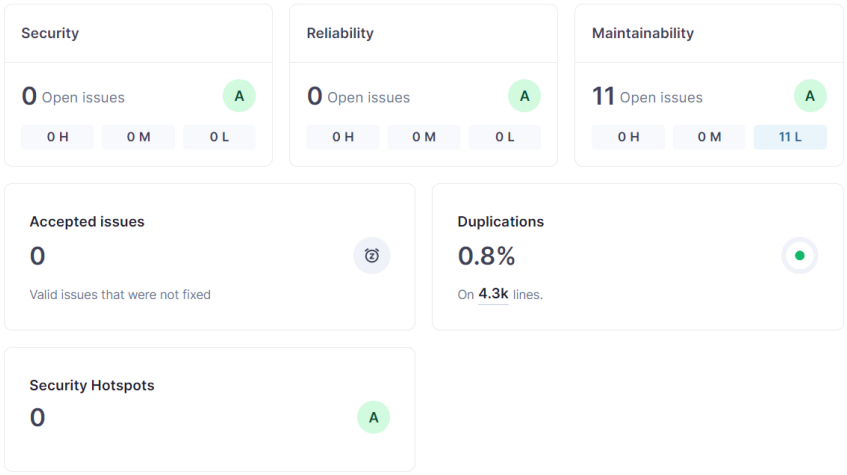


Рисунок 4.1 – Аналіз серверної частини застосунку сервісом SonarQube

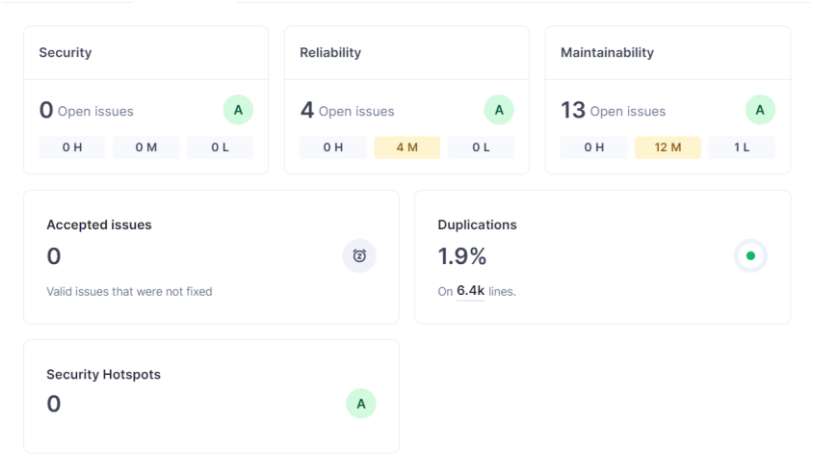


Рисунок 4.2 – Аналіз клієнтської частини застосунку сервісом SonarQube

Аналізуючи результати, можна сказати, що проєкт відповідає високій якості, адже кожен критерій аналізу оцінено максимально, він потребує мінімальних покращень, що не впливають на його якісь, також досить низький рівень дублікації, що пояснюється використанням однакових стилів для розробки шаблонів клієнтської частини та схожим документування для сутностей серверної частини.

Також для аналізу написаного ПЗ було використано ESLint – інструмент, що дозволяє аналізувати код, шляхом пошуку помилок і автоматичного їх виправлення в деяких випадках.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування». Метою тестування є:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox);

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.12.

Таблиця 4.1 – Тест 1.1 Реєстрація користувача з коректними даними

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Повне ім’я користувача, юзернейм, пароль, підтвердження паролю
Опис проведення тесту	Користувач вводить коректні дані у відповідні поля: своє повне ім’я, юзернейм, що такого ще не існує в системі, пароль, що містить 8 або більше символів та підтвердження паролю. Після цього натискає кнопку «Зареєструватись».
Очікуваний результат	Користувач зареєстрований в системі та перенаправляється на сторінку авторизації.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Авторизація користувача з коректними даними

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Юзернейм та пароль користувача
Опис проведення тесту	Користувач вводить коректні дані у відповідні поля: свій юзернейм та пароль, що був введений при реєстрації. Після цього натискає кнопку «Авторизуватись».
Очікуваний результат	Користувач авторизований та перенаправляється на сторінку «Панель користувача».
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 2.1 Створення нового проєкту з коректними даними

Початковий стан системи	Користувач авторизований, знаходиться на сторінці «Панель користувача»
Вхідні дані	Назва проєкту, його код та опис
Опис проведення тесту	Користувач натискає на кнопку «Створити новий проєкт», відкривається модальне вікно. Користувач вводить коректні код та назву проєкту, що таких не існує в системі, також вводить опис для даного проєкту. Натискає кнопку «Створити».
Очікуваний результат	Проєкт створено, він з’явився у списку проєктів користувача на сторінці «Панель користувача», користувача перенаправляє на сторінку створеного проєкту.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 2.2 Доєднання до проєкту іншого користувача

Початковий стан системи	Користувач авторизований, знаходиться на сторінці «Панель користувача»
Вхідні дані	Ідентифікатор проєкту, що його отримано від іншого користувача
Опис проведення тесту	Користувач натискає на кнопку «Доєднатися до проєкту», відкривається модальне вікно. Користувач вводить коректний ідентифікатор існуючого проєкту. Натискає кнопку «Доєднатись».
Очікуваний результат	Користувач доєднався до проєкту іншого користувача, він з’явився у списку проєктів користувача на сторінці «Панель користувача».
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 2.3 Перегляд усіх користувачів проєкту

Початковий стан системи	Користувач авторизований, знаходиться на сторінці проєкту
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку «Усі користувачі», що знаходиться напроти назви проєкту.
Очікуваний результат	Відкривається сторінка «Користувачі» та відображається таблиця з юзернеймами та повними іменами користувачів, що є учасниками даного проєкту.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 3.1 Створення нового тестового набору з коректними даними

Початковий стан системи	Користувач авторизований, знаходиться на сторінці проєкту
Вхідні дані	Назва для тестового набору та опис за потреби
Опис проведення тесту	Користувач натискає на кнопку «Створити тестовий набір», відкривається модальне вікно. Користувач вводить коректну назву для тестового набору та його опис за бажанням. Натискає кнопку «Створити».
Очікуваний результат	Тестовий набір створено, він з’явився у списку тестових наборів проєкту на сторінці даного проєкту, користувача перенаправляє на сторінку створеного тестового набору.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 3.2 Створення нового тестового випадку з коректними даними

Початковий стан системи	Користувач авторизований, знаходиться на сторінці тестового набору обраного проєкту
Вхідні дані	Назву тестового випадку, опис для нього, пріоритет тестового випадку, передумови, кроки тестового випадку та відповідні очікуванні результати для кожного кроку.
Опис проведення тесту	Користувач натискає на кнопку «Створити тестовий випадок», відкривається модальне вікно. Користувач вводить коректну назву для тестового випадку, опис тестового випадку, обирає пріоритет для тестового випадку, заповнює передумови для виконання тестового випадку, додає інформацію про крок та очікуваний результат виконання даного кроку, за потреби додає новий крок за допомогою кнопки «Додати крок» та заповнює ново доданий крок з очікуваним результатом. Натискає кнопку «Створити».
Очікуваний результат	Тестовий випадок створено, він з’явився у списку тестових наборів проєкту на сторінці даного проєкту, користувача перенаправляє на сторінку створеного тестового випадку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 3.3 Виконання прогону тестового набору

Початковий стан системи	Користувач авторизований, знаходиться на сторінці тестового набору обраного проєкту
Вхідні дані	У тестовому наборі створено хоча б один тестовий випадок

Продовження таблиці 4.8.

Опис проведення тесту	Користувач натискає на кнопку «Запустити» на сторінці тестового набору. Відкривається сторінка прогону. На сторінці прогону користувач натискає на тестовий випадок для якого треба обрати статус та може побачити його попередній перегляд, після виконання відповідних дій в необхідному середовищі, користувач обирає статус виконання даного тестового випадку з можливих: пропущено, пройдено, не вдало, за потреби додає коментар. Користувач зберігає поточний стан прогону, натиснувши на кнопку «Зберегти» напроти назви прогону. Користувач продовжує виконувати тестові набори, надаючи їм відповідний статус, та додаючи за потреби коментарі. Після того, як користувач виконав все, що було треба, натискає кнопку «Завершити».
Очікуваний результат	Прогін тестового набору вважається закінчений, поле «Час завершення» вказує на час, коли користувач закінчив прогін. Відображаються результати виконання прогону.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 3.4 Перегляд результатів виконаного прогону тестового набору

Початковий стан системи	Користувач авторизований, знаходиться на сторінці тестового набору обраного проєкту
Вхідні дані	-
Опис проведення тесту	Користувач натискає на закінчений тестовий прогін, той що не підсвічується червоним, на сторінці тестового набору. Користувачу відкривається сторінка прогону.

Продовження таблиці 4.9.

Очікуваний результат	На сторінці тестового прогону відображається час створення, останнього оновлення та закінчення прогону. Відображаються результати прогону, що включають в себе аналітику та статуси для кожного тестового випадку. Аналітика містить в собі інформацію таку як відсоток пройдених тестових випадків, а також кругова діаграма, що відображає відношення пройдених, не вдалих та пропущених тестових наборів, також містить інформацію про кількість даних статусів для тестових наборів. Під аналітикою користувачу відображаються тестові набори, що містить даний прогін, напроти них знаходиться їх статус виконання та коментарі, де вони були додані.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 4.1 Створення нового звіту про дефект з коректними даними

Початковий стан системи	Користувач авторизований, знаходиться на сторінці проєкту
Вхідні дані	Тема, детальний опис, кроки для відтворення, актуальний результат, очікуваний результат, статус, пріоритет, серйозність, оточення, посилання, де відтворюється дефект, для звіту про дефект та за потреби вкладення.

Продовження таблиці 4.10.

Опис проведення тесту	Користувач натискає на кнопку «Створити звіт про дефект», відкривається модальне вікно. Користувач вводить коректні тему, детальний опис, кроки для відтворення, актуальний результат, очікуваний результат, статус, пріоритет, серйозність, оточення, посилання, де відтворюється дефект, додає при наявності зображення. Натискає кнопку «Створити».
Очікуваний результат	Звіт про дефект створено, він з’явився у списку звітів про дефект проєкту на сторінці даного проєкту, користувача перенаправляє на сторінку створеного звіту про дефект.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 5.1 Пошук за назвою існуючого проєкту

Початковий стан системи	Користувач авторизований
Вхідні дані	Назва проєкту, що користувач хоче знайти
Опис проведення тесту	Користувач у поле пошуку, що знаходиться у заголовку сторінки, вводить назву існуючого проєкту, учасником якого він є.
Очікуваний результат	У випадаючому списку під полем пошуку з’являється шуканий проєкт, а також проєкти, тестові набори, тестові випадки, звіти про дефект, що задовольняють пошуковий запит користувача.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 5.2 Пошук за назвою неіснуючого проєкту

Початковий стан системи	Користувач авторизований
Вхідні дані	Назва проєкту, що користувач хоче знайти
Опис проведення тесту	Користувач у поле пошуку, що знаходиться у заголовку сторінки, вводить назву неіснуючого проєкту.
Очікуваний результат	У випадаючому списку під полем пошуку з’являється повідомлення «Пустий список», що повідомляє про відсутність проєктів, тестових наборів, тестових випадків та звітів про дефект з такою назвою.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

До контрольного прикладу проєкту віднесено повний сценарій роботи з вебзастосунком:

- реєстрація користувача;
- авторизація;
- створення нового проєкту;
- доєднання до існуючого проєкту;
- перегляд користувачів проєкту;
- створення тестового набору в проєкті;
- перегляд тестового набору проєкту;
- створення тестового випадку в тестовому наборі;
- перегляд тестового випадку;
- створення прогону тестового набору;
- виконання прогону тестового набору;
- зберігання прогону тестового набору;
- завершення прогону тестового набору;

– перегляд результатів тестового набору;
– створення звіту про дефект в проєкті;
– перегляд створеного звіту про дефект;
– пошук за назвою проєкту.

Перейдемо до опису контрольного прикладу.

Користувач знаходиться на сторінці реєстрації, що зображена на рисунку 4.3, кнопка «Зареєструватись» є неактивною, доки усі поля будуть не заповненні.

Рисунок 4.3 – Сторінка реєстрації

Користувач вводить коректне повне ім’я, юзернейм та пароль, що містить менше 8 символів, користувачу відображається помилка під полем паролю, що пароль має містить 8 або більше символів, а також користувач вводить в поле для підтвердження паролю, пароль, що не співпадає з введеним раніше, під полем підтвердження паролю, відображається відповідна помилка (рис. 4.4).

Рисунок 4.4 – Сторінка реєстрації з введеними некоректними паролями

Користувач вводить коректні дані у поля повне ім’я, пароль та підтвердження паролю, а в поле юзернейму вводить юзернейм іншого користувача, після чого натискає кнопку «Зареєструватись». З’являється відповідна помилка (рис. 4.5).

Рисунок 4.5 – Сторінка реєстрації з введеними юзернеймом іншого користувача

Користувач вводить коректні дані у всі поля (рис. 4.6).

Рисунок 4.6 – Сторінка реєстрації з введеними коректними даними

Користувач натискає кнопку «Зареєструватись», він успішно зареєструвався, відкривається сторінка авторизації, кнопка «Авторизуватись» неактивна (рис. 4.7).

Рисунок 4.7 – Сторінка авторизації

Користувач вводить неправильний юзернейм та(або) пароль, натискає кнопку «Авторизуватись», з’являється помилка про неправильно введений юзернейм або пароль (рис. 4.8).

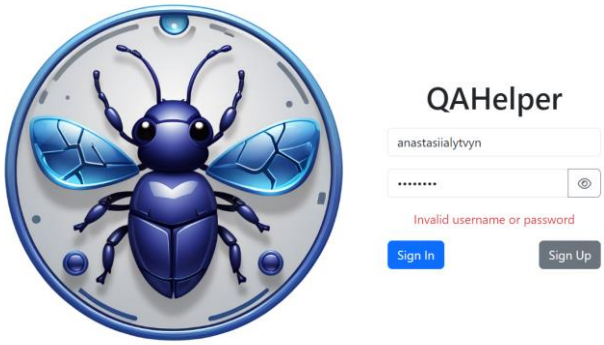


Рисунок 4.8 – Сторінка авторизації з введеними неправильними даними

Користувач вводить правильний юзернейм та пароль (рис. 4.9).

Рисунок 4.9 – Сторінка авторизації з введеними коректними даними

Користувач натискає кнопку «Авторизуватись», він успішно авторизований, відкривається сторінка «Панель користувача» (рис. 4.10).

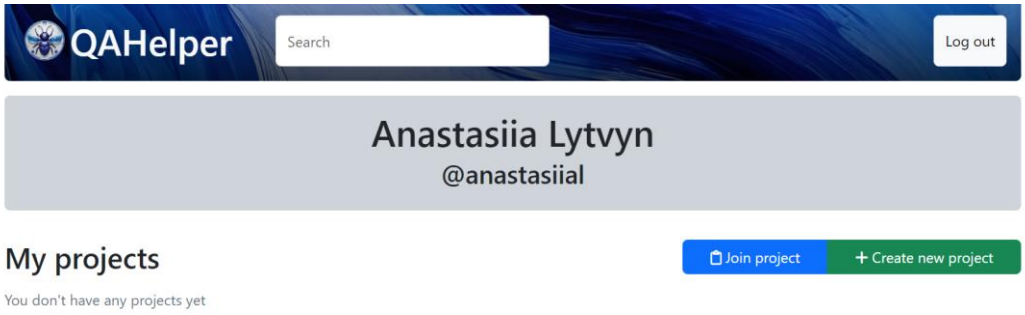


Рисунок 4.10 – Сторінка «Панель користувача»

Користувач натискає на кнопку «Створити проєкт», відкривається модальне вікно, кнопка «Створити» неактивна (рис. 4.11).

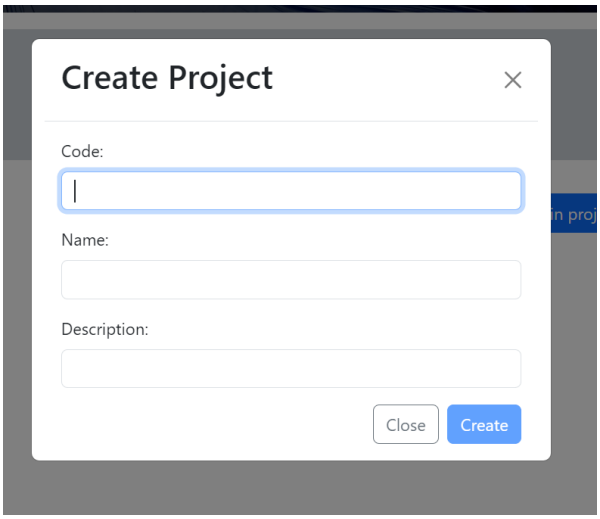


Рисунок 4.11 – Форма створення проєкту

Користувач вводить код, що вже існує для іншого проєкту, також коректні назву та опис для проєкту, з’являється відповідна помилка (рис. 4.12).

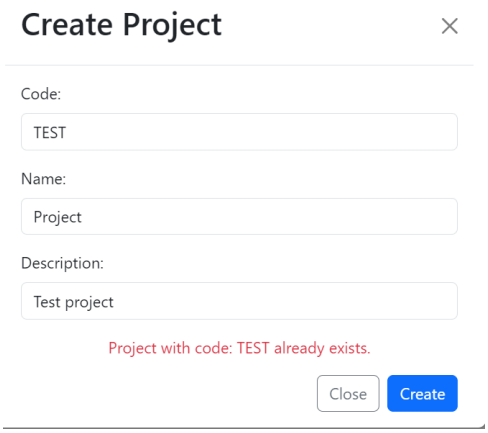


Рисунок 4.12 – Форма створення проєкту з кодом, що вже існує

Користувач вводить назву, що вже існує для іншого проєкту, також коректні код та опис для проєкту, з’являється відповідна помилка (рис. 4.13).

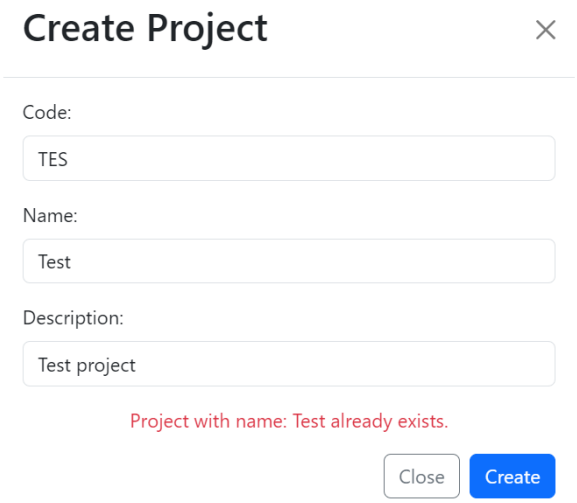


Рисунок 4.13 – Форма створення проєкту з назвою, що вже існує

Користувач вводить коректні дані у всі поля (рис. 4.14).

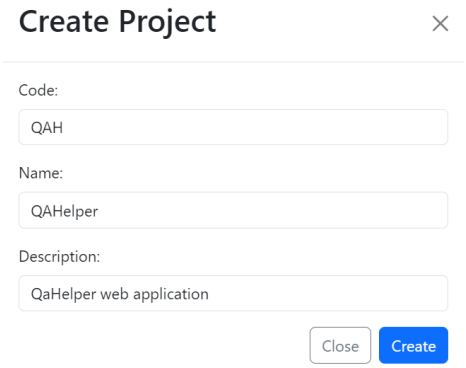


Рисунок 4.14 – Форма створення проєкту з коректними даними

Користувач натискає кнопку «Створити». Проєкт створено, користувача перенаправлено на сторінку створеного проєкту (рис. 4.15).

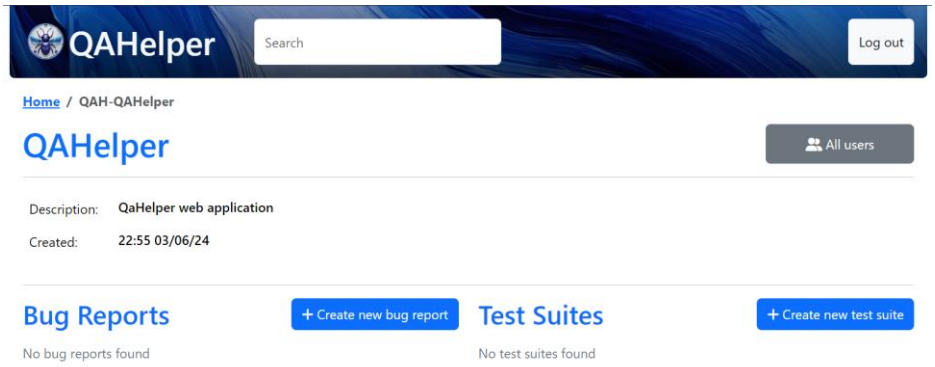


Рисунок 4.15 – Сторінка створеного проєкту

Користувач доєднується до існуючого проєкту, натиснувши на сторінці «Панель користувача» на кнопку «Доєднатись до проєкту», відкривається модальне вікно, кнопка «Доєднатись» неактивна (рис. 4.16).

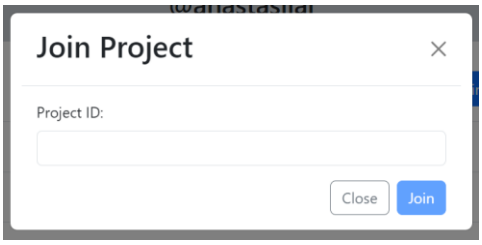


Рисунок 4.16 – Форма доєднання до проєкту

Користувач вводить ідентифікатор проєкту, учасником якого він вже є, з’являється відповідна помилка (рис. 4.17).

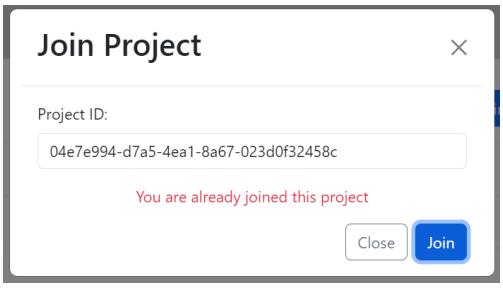


Рисунок 4.17 – Форма доєднання до проєкту за ID проєкту, учасником якого користувач вже є

Користувач вводить ідентифікатор проєкту, якого не існує, з’являється відповідна помилка (рис. 4.18).

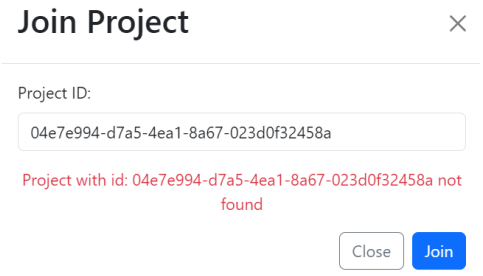


Рисунок 4.18 – Форма доєднання до проєкту за ID проєкту, що не існує

Користувач вводить набір випадкових символів, з’являється відповідна помилка (рис. 4.19).

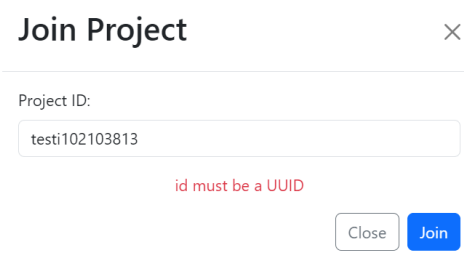


Рисунок 4.19 – Форма доєднання до проєкту за ID складеного з випадкових символів

Користувач вводить ідентифікатор проєкту іншого користувача (рис. 4.20).

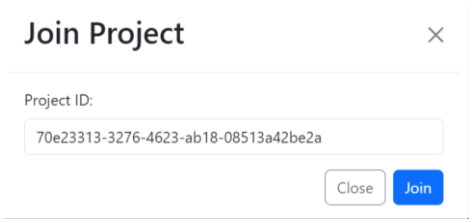


Рисунок 4.20 – Форма доєднання до проєкту за ID проєкту іншого користувача

Користувач натискає кнопку «Доєднатись», користувач доєднується до проєкту, проєкт з’являється в списку проєктів користувача (рис. 4.21).

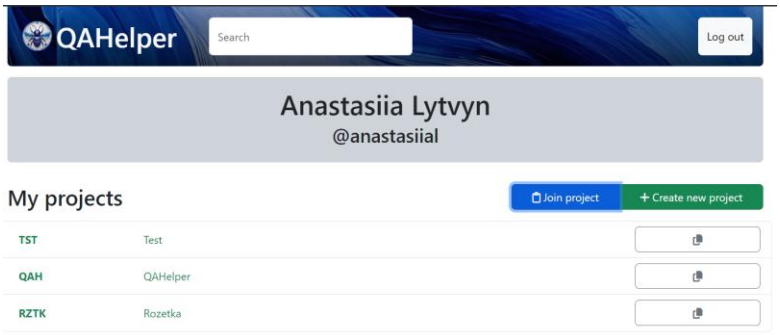


Рисунок 4.21 – Сторінка «Панель користувача» з проєктами користувача

Користувач відкриває сторінку проєкта. Користувач натискає на кнопку «Усі користувачі». Відкривається сторінка «Користувачі» проєкту (рис. 4.22).

Рисунок 4.22 – Сторінка «Користувачі» проєкту

Користувач повертається на сторінку проєкту за допомогою «хлібних крихт», що знаходяться під заголовком сторінки та натискає кнопку «Створити тестовий набір», відкривається модальне вікно, кнопка «Створити» неактивна (рис. 4.23).

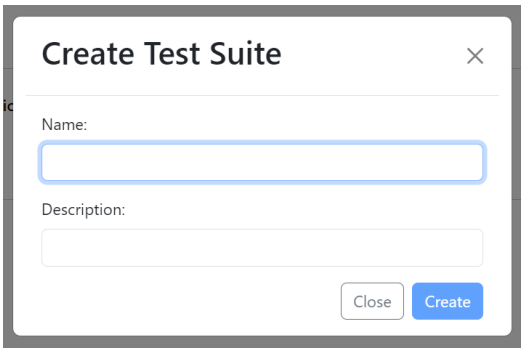


Рисунок 4.23 – Форма створення тестового набору

Користувач вводить назву та опис для тестового набору (рис. 4.24).

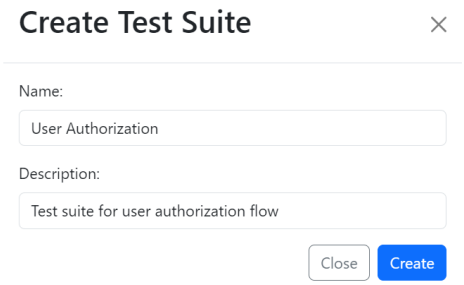


Рисунок 4.24 – Форма створення проєкту з кодом, що вже існує

Користувач натискає кнопку «Створити». Тестовий набір створено, користувача перенаправлено на сторінку створеного тестового набору (рис. 4.25).

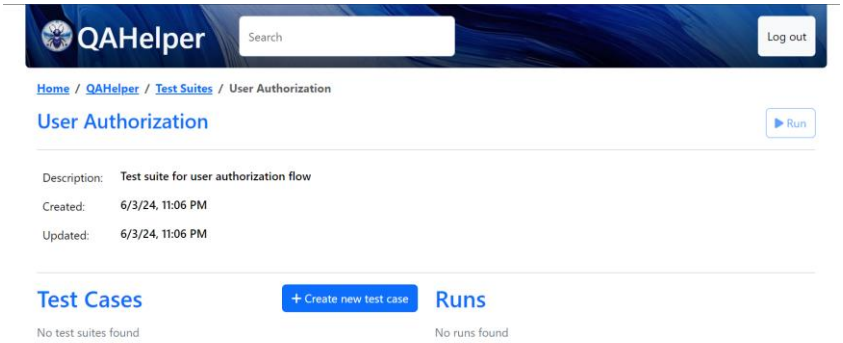


Рисунок 4.25 – Сторінка тестового набору

Користувач натискає на кнопку «Створити тестовий випадок», відкривається модальне вікно, кнопка «Створити» неактивна (рис. 4.26).

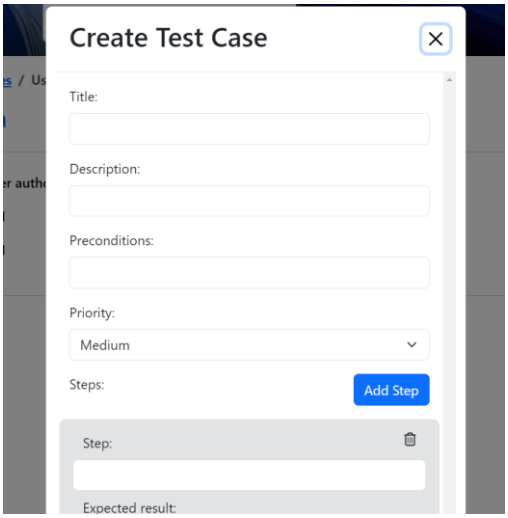


Рисунок 4.26 – Форма створення тестового випадку

Користувач видаляє крок, що існує за замовчування, з’являється помилка, що має існувати хоча б один крок тестового випадку (рис. 4.27).

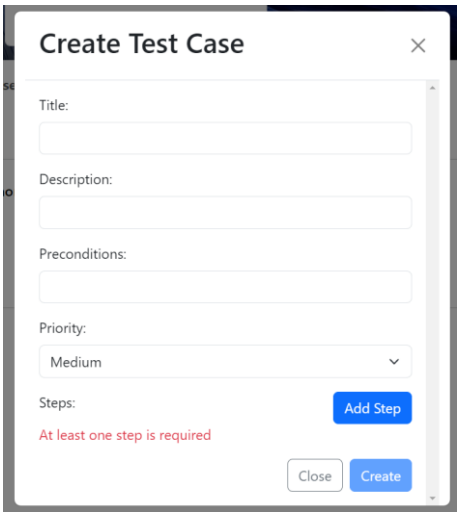


Рисунок 4.27 – Форма створення тестового випадку без кроків

Користувач заповнює усі поля коректно (рис. 4.28).

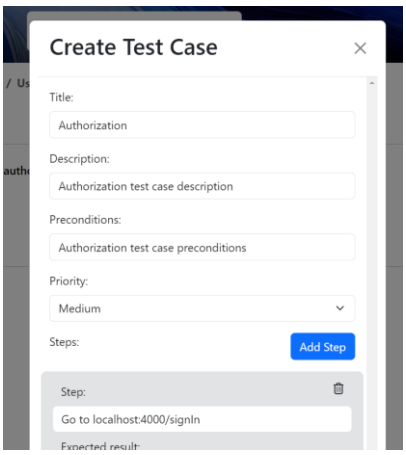


Рисунок 4.28 – Форма створення тестового набору з коректними даними

Користувач натискає кнопку «Створити». Тестовий випадок створено, користувача перенаправлено на сторінку створеного тестового випадку (рис. 4.29).

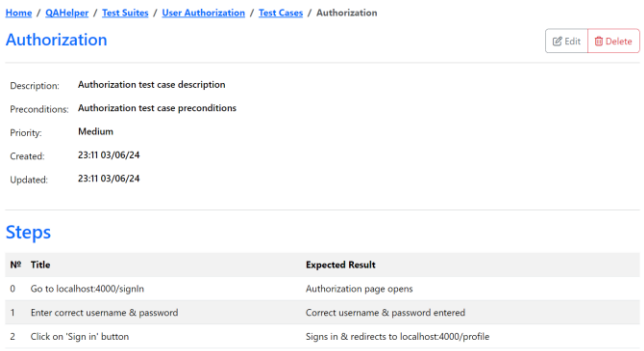


Рисунок 4.29 – Сторінка тестового випадку

Користувач повертається на сторінку тестового набору. Користувач натискає кнопку «Запустити», відкривається сторінка прогону тестового набору (рис. 4.30).

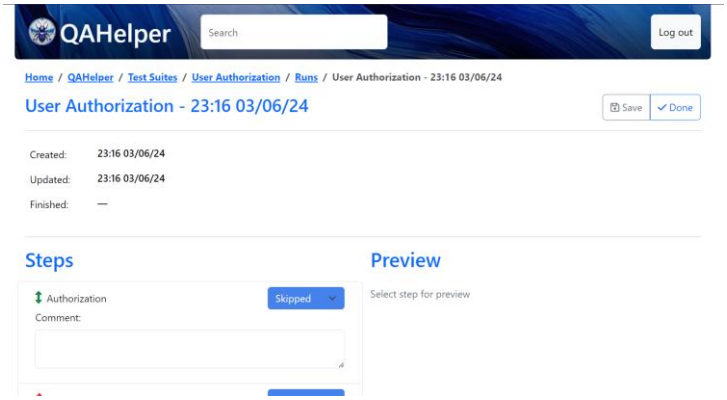


Рисунок 4.30 – Сторінка прогону тестового набору

Користувач натискає на тестовий випадок, відкривається «Попередній перегляд» тестового випадку (рис. 4.31).

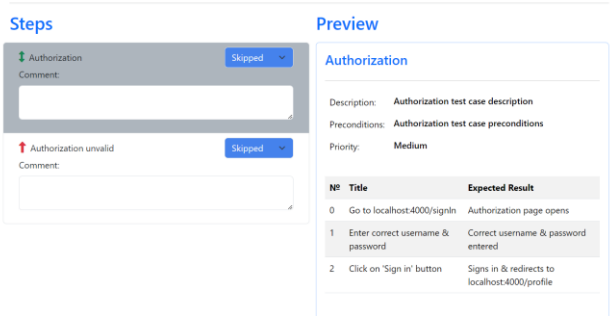


Рисунок 4.31 – Вигляд попереднього перегляду тестового випадку на сторінці прогону тестового набору

Користувач обирає відповідний статус для даного тестового випадку та додає коментар (рис. 4.32).

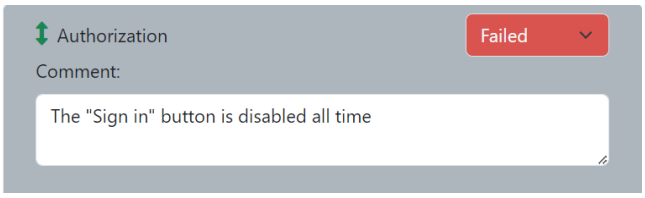


Рисунок 4.12 – Зміна статусу та додання коментарю до тестового набору в даному прогоні

Користувач зберігає прогін натиснувши кнопку «Зберегти», поле «Час оновлення» змінюється на новий час (рис. 4.33).

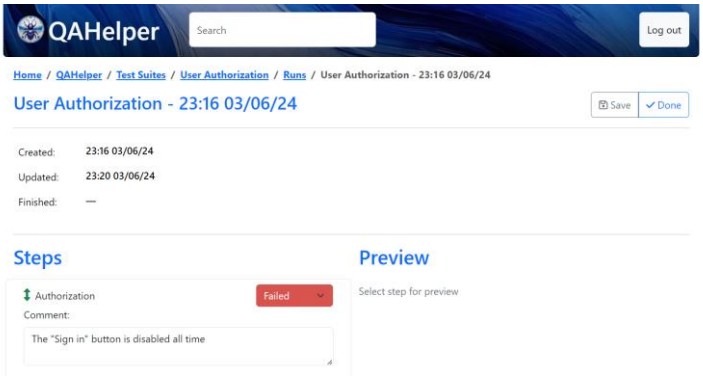


Рисунок 4.33 – Стан сторінки прогону після його збереження

Користувач продовжує змінювати статуси тестових наборів та додавати за потреби коментарі (рис. 4.34).

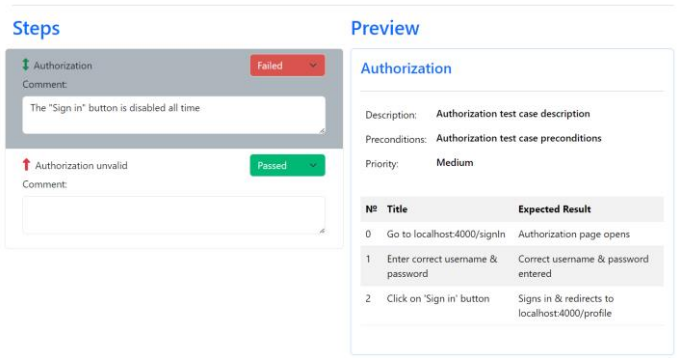


Рисунок 4.34 – Стан статусів тестових випадків для тестового набору в даному прогоні

Користувач завершує прогін натиснувши кнопку «Готово», на сторінці оновлюється «Час закінчення» прогону та з’являються результати прогону (рис. 4.35 – 4.36).

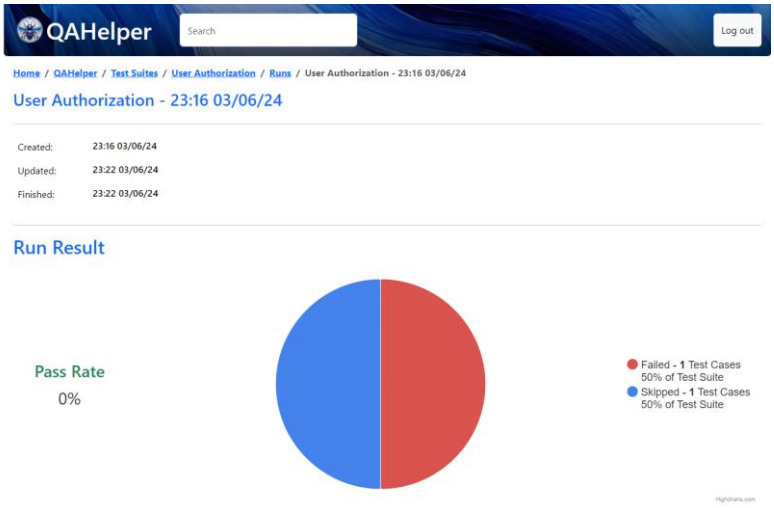


Рисунок 4.35 – Аналітика виконаного прогону

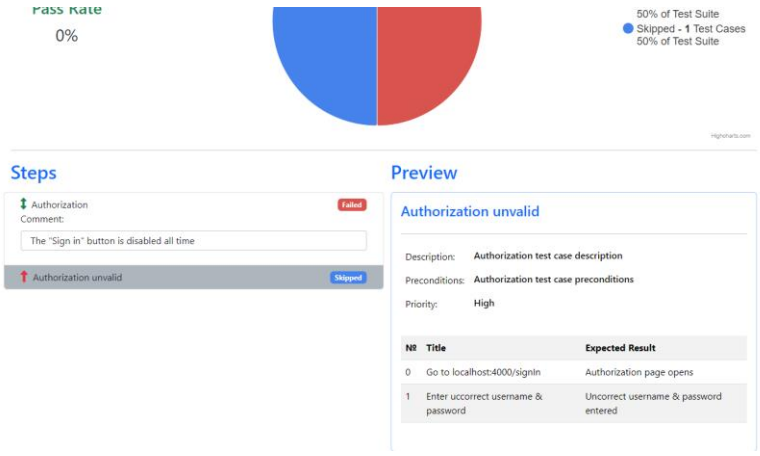


Рисунок 4.36 – Результати виконаного прогону

Користувач повертається на сторінку проєкта та натискає кнопку «Створити звіт про дефект», відкривається модальне вікно, кнопка «Створити» неактивна (рис. 4.37).

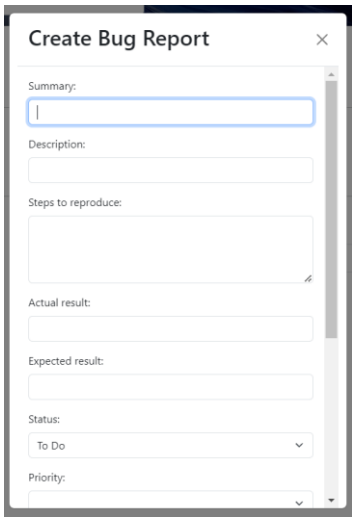


Рисунок 4.37 – Форма створення звіту про дефект

Користувач заповнює усі поля коректними даними (рис. 4.38).

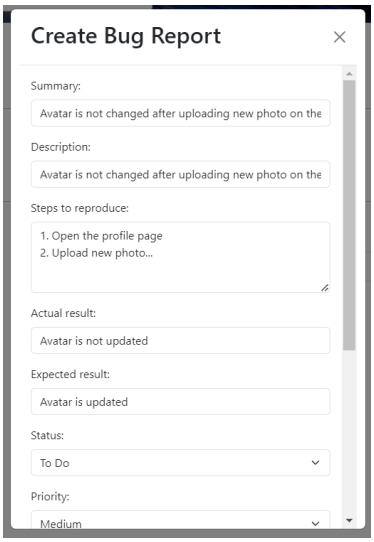


Рисунок 4.38 – Форма створення звіту про дефект з коректними даними

Користувач додає зображення за потреби, для цього він натискає кнопку «Додати вкладення», з’являється поле для додавання зображення (рис. 4.39).

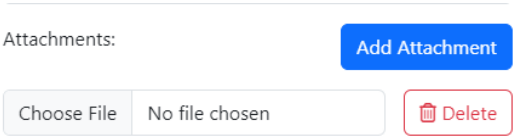


Рисунок 4.39 – Поле для додавання зображень на формі створення звіту про дефект

Користувач обирає зображення, що бажає додати (рис. 4.40).

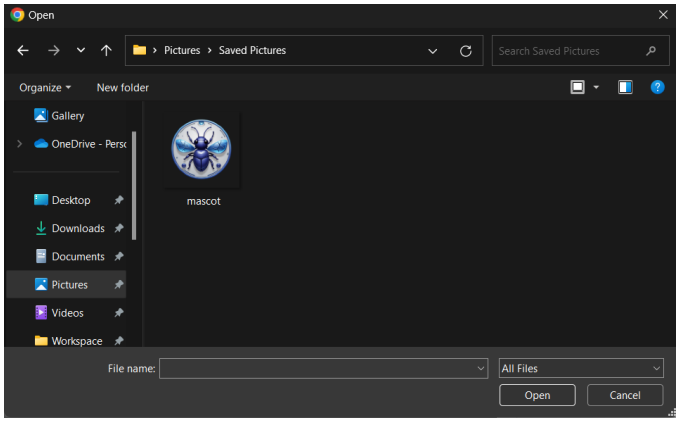


Рисунок 4.40 – Вибір зображення, що буде доєднано до звіту про дефект

Користувач натискає кнопку «Створити». Звіт про дефект створено, користувача перенаправлено на сторінку створеного звіту про дефект (рис. 4.41).

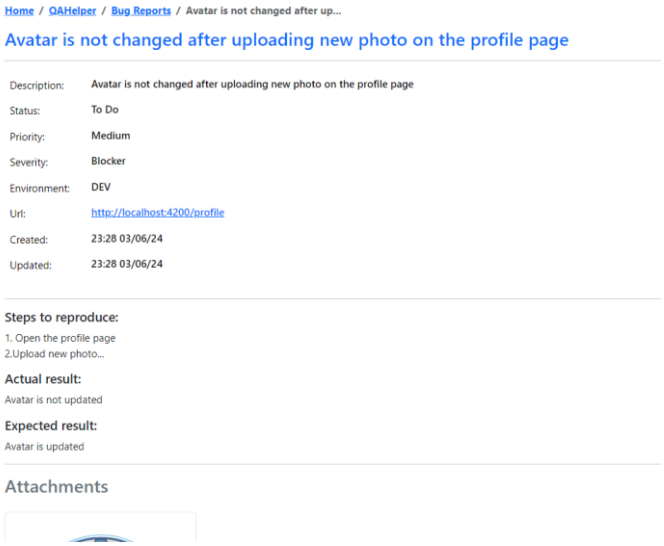


Рисунок 4.41 – Сторінка звіту про дефект

Користувач вводить у поле пошуку випадкові символи, з’являється повідомлення «Пустий список» (рис. 4.42).

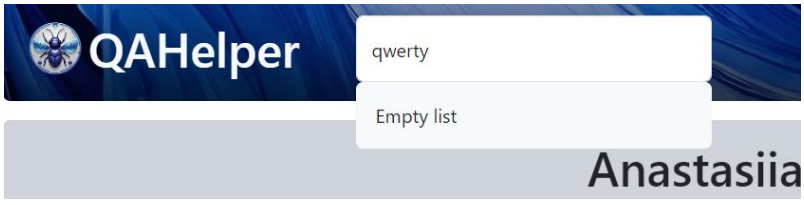


Рисунок 4.42 – Пошук за випадковими символами

Користувач вводить у поле пошуку назву створеного раніше проєкту, з’являється випадаючий список з проєктами, тестовими наборами, тестовими випадками та звітами про дефект, що задовольняють пошуковому запиту (рис. 4.43).

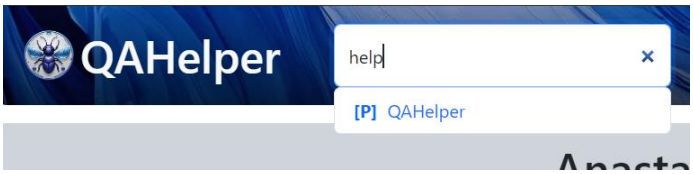


Рисунок 4.43 – Пошук за назвою проєкта

Висновки до розділу

У даному розділі було проаналізовано якість ПЗ та виконано його мануальне тестування для основного функціоналу. Під час розробки використовувався ESLint, що допомогло уникнути багатьох помилок, тому використовуючи SonarCloud було триманого гарний результат: було створено надійне, не вразливе, безпечне і недубльоване ПЗ.

Під час мануального тестування було більш детально розглянуто основний функціонал, протестувавши його використовуючи різні дані. Тестування показало, що для всіх тестів очікуваний і фактичний результат є однаковим, а це означає, що ПЗ відповідає функціональним вимогам, що від нього і вимагалося.

Також було описано контрольний приклад, що містив у собі усі повний опис того, як реагує система на дії користувача. Контрольний приклад містить у собі опис реєстрації, авторизації, роботи з проєктами, звітами про дефект, тестовими наборами, тестовими випадками та прогонів тестового набору, а також виконання пошуку.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання ПЗ було використано Docker - це програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації, що формує додатки в контейнери [28].

Для того, щоб почати працювати з розгортанням в контейнері необхідно створити Dockerfile - файл, в якому необхідно описати, як буде створюватися образ, на серверній та на клієнтській частині ПЗ. Вміст Dockerfile для ПЗ зображені на рисунках 5.1, 5.2.

Dockerfile

```
FROM node:20.11

WORKDIR /app

COPY package*.json ./

RUN npm ci

COPY . .

RUN npm run build


CMD [ "npm", "run", "start:prod" ]
```

Рисунок 5.1 – Вміст Dockerfile для серверної частини ПЗ

Dockerfile

```
FROM node:20.11

WORKDIR /app

COPY package*.json ./

RUN npm ci

COPY . .

EXPOSE 4200

CMD [ "npm", "start" ]
```

Рисунок 5.2 – Вміст Dockerfile для клієнтської частини ПЗ

Також створимо файл docker-compose.yml. Даний файл використовують для налаштування служб ПЗ, містить назви додатків, імена контейнерів і т. д., він дозволяє розгорнути ПЗ використавши одну команду. Дане ПЗ містить в собі такі сервіси, як сервіс бази даних postgres, сервіс для

зберігання вкладень minio, сервіс сервісної та клієнтської частини додатку. Вміст файлу docker-compose.yml знаходиться у додатку В.

Для того, щоб розгорнути програмне забезпечення, необхідно прописати у командний рядок проєкту команду, що представлена на рисунку 5.3, він сформує докер-контейнер серверу, після чого написати команду, що представлена на рисунку 5.4, він сформує докер-контейнер клієнту. Після цього можна розгорнути ПЗ за допомогою команди, представленої на рисунку 5.5.

```
$ docker build . -t qahelper/backend:v1.0.0 -t qahelper/backend:latest
[+] Building 1.7s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 198B
 => [internal] load metadata for docker.io/library/node:20.11
 => [auth] library/node:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 54B
 => [1/6] FROM docker.io/library/node:20.11@sha256:e06aae17c40c7a6b5296ca6f942a02e6737ae61bbbf3e2158624bb0f887991b5
 => [internal] load build context
 => => transferring context: 194.59kB
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] COPY package*.json ./
 => CACHED [4/6] RUN npm ci
 => CACHED [5/6] COPY . .
 => CACHED [6/6] RUN npm run build
 => exporting to image
 => => exporting layers
 => => writing image sha256:99fa5f7eba733707fcdb5b191f22b89154f19b0c4be6b50b142d905c445c7ecd
 => => naming to docker.io/qahelper/backend:v1.0.0
 => => naming to docker.io/qahelper/backend:latest
```

Рисунок 5.3 – Формування докер-контейнеру серверу

```
$ docker build . -t qahelper/frontend:v1.0.0 -t qahelper/frontend:latest
[+] Building 6.6s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 162B
 => [internal] load metadata for docker.io/library/node:20.11
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/node:20.11@sha256:e06aae17c40c7a6b5296ca6f942a02e6737ae61bbbf3e2158624bb0f887991b5
 => [internal] load build context
 => => transferring context: 4.19MB
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY package*.json ./
 => CACHED [4/5] RUN npm ci
 => CACHED [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:16467665443cdd8c0f46bcf6224791e6fc53e5dc7a95e952878df951f7711b7e
 => => naming to docker.io/qahelper/frontend:v1.0.0
 => => naming to docker.io/qahelper/frontend:latest
```

Рисунок 5.4 – Формування докер-контейнеру клієнту

```
$ docker compose up
[+] Running 4/0
 ✓ Container backend-db-1        Created
 ✓ Container backend-minio-1     Created
 ✓ Container backend-backend-1   Created
 ✓ Container backend-frontend-1  Created
Attaching to backend-1, db-1, frontend-1, minio-1
```

Рисунок 5.5 – Розгортання програмного забезпечення

5.2 Супровід програмного забезпечення

Для супроводу та підтримки ПЗ буде використано Docker Hub, для цього необхідно розмістити сформовані контейнери на даному сервісі, тому треба мати акаунт в ньому та створені репозиторії для ПЗ (рис. 5.6). У командному рядку прописати команди, що зображені на рисунках 5.7-5.8.

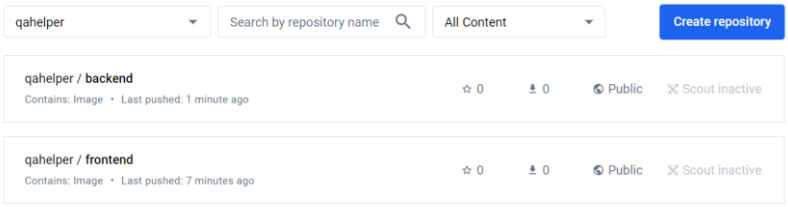


Рисунок 5.6 –Docker Hub репозиторії

```
$ docker push qahelper/backend:latest
The push refers to repository [docker.io/qahelper/backend]
0f5b93f098b6: Pushed
9e6a61774f6b: Pushed
3778779a5755: Pushed
32ed324d8954: Pushed
0801cd446f1c: Mounted from qahelper/frontend
ac68e27ae9cc: Mounted from qahelper/frontend
9cef422ea209: Mounted from qahelper/frontend
09ddcd01d2dc: Mounted from qahelper/frontend
5358370f44ab: Mounted from qahelper/frontend
21e1c4948146: Mounted from qahelper/frontend
68866beb2ed2: Mounted from qahelper/frontend
e6e2ab10dba6: Mounted from qahelper/frontend
0238a1790324: Mounted from qahelper/frontend
latest: digest: sha256:cda539a5431e617e7e5877cced709711f9bde106ea544d85eece8b592e714f31 size: 3052
```

Рисунок 5.7 – Команда для додавання докер-контейнеру сервісу в Docker Hub репозиторій

```
$ docker push qahelper/frontend:latest
The push refers to repository [docker.io/qahelper/frontend]
bf2bab4987a6: Pushed
4dda6161d9d8: Pushed
d511fdd30512: Pushed
0801cd446f1c: Pushed
ac68e27ae9cc: Pushed
9cef422ea209: Mounted from library/node
09ddcd01d2dc: Mounted from library/node
5358370f44ab: Pushed
21e1c4948146: Mounted from library/node
68866beb2ed2: Mounted from library/node
e6e2ab10dba6: Mounted from library/node
0238a1790324: Mounted from library/node
latest: digest: sha256:e4a00fff11c2284828789ee060140e7d47b1b3f70d1b7f62b0f1d157bf3c2893 size: 2846
```

Рисунок 5.8 – Команда для додавання докер-контейнеру клієнту в Docker Hub репозиторій

Таким чином було розміщено дані контейнери на сервісі.

Для того, щоб користувач міг отримати програмне забезпечення та його останню версію, йому необхідно мати встановлений Docker на своєму пристрої, отримати контейнери з Docker Hub, виконавши docker-compose.yml

файл, використавши команду «docker compose up» в директорії розміщення даного файлу (рис. 5.5).

Користувач успішно отримав останню версію програмного забезпечення.

Інструкція користувача наведена в окремому документі.

Висновки до розділу

Даний розділ описує впровадження і підтримку ПЗ. Було описано та виконано розгортання програмного забезпечення за використанням відповідного ПЗ – Docker.

Було наведено вміст необхідних файлів для розгортання з використанням згаданого вище сервісу, наданий порядок дій, що допоможуть розгорнути ПЗ.

Також було описано процес супроводу ПЗ, він включає в себе створення контейнерів на сервісі Docker Hub та додавання туди контейнерів, що були створенні раніше. Для того, щоб користувачу отримати саме ПЗ та його останні версії необхідно мати на своєму пристрої наданий раніше файл docker-compose.yml файл та виконати його, використавши наведену команду.

ВИСНОВКИ

У результаті виконання дипломного проєкту було реалізовано вебзастосунок для створення тестової документації «QAHelper», що призначений для розробки тестової документації, такої як тестові набори, тестові випадки та звіти про дефект, з урахуванням усіх правил її заповнення та потреб тестувальників.

Для вебзастосунку було обрано клієнт-серверну архітектуру з використанням патерну Clean architecture з незначними модифікаціями. Розробка виконувалась за допомогою мови програмування TypeScript, використано фреймворки NestJS та Angular для серверної та клієнтської частини застосунку відповідно. У якості БД обрано PostgreSQL. Для збереження вкладень, що передбачає даний вебзастосунок, було використано об’єктне сховище MinIO.

Після реалізації застосунку виконано аналіз його якості та мануальне тестування. Для аналізу якості використано сервіс SonarQube, що містить оцінки обраних метрик. Для мануального тестування розроблено тести для виконання. Даний аналіз та тестування підтвердили ефективність та правильність розробленого ПЗ.

Розроблений вебзастосунок може бути використаний працівниками сфери інформаційних технології, а саме інженерами з забезпечення якості ПЗ, а також тими, хто лише вивчає дану сферу та використовує додатки для тестування. Вебзастосунок надає зручний інтерфейс та дотримується усіх правил заповнення тестової документації, що робить його зручним для використання як новачкам, так і вже спеціалістам з тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) - M. I. H. Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. International Journal For Multidisciplinary Research. 2023. Т. 5, № 5. URL: https://doi.org/10.36948/ijfmr.2023.v05i05.6223 (дата звернення: 15.04.2024).

2) Gillis A. S. What is Quality Assurance? - Definition from WhatIs.com. Software Quality. URL: https://www.techtarget.com/searchsoftwarequality/definition/quality-assurance (дата звернення: 15.04.2024).

3) Testsigma. What is Quality Assurance (QA) in Software Testing?. Testsigma Blog. URL: https://testsigma.com/guides/quality-assurance/ (дата звернення: 15.04.2024).

4) QATestLab. Ролі QA інженерів. Онлайн-курси від компанії QATestLab. URL: https://training.qatestlab.com/blog/helpful-materials/roles-of-quality-assurance-engineers-skills-tools-and-responsibilities-in-the-test-team/ (дата звернення: 15.04.2024).

5) Hamilton T. Test Documentation in Software Testing (Example). Guru99. URL: https://www.guru99.com/testing-documentation.html (дата звернення: 15.04.2024).

6) Brush K. What is a Test Case?. Software Quality. URL: https://www.techtarget.com/searchsoftwarequality/definition/test-case (дата звернення: 15.04.2024).

7) QATestLab. Основні атрибути баг-репорта. Онлайн-курси від компанії QATestLab. URL: https://training.qatestlab.com/blog/course-materials/attributes-bug-report/ (дата звернення: 15.04.2024).

8) Rai A. Calculating and Understanding the Usefulness of JIRA Developers Communities. International Journal for Research in Applied Science and Engineering Technology. 2021. Т. 9, № VII. С. 2829–2839. URL: https://doi.org/10.22214/ijraset.2021.36983 (дата звернення: 25.04.2024).

9) J.B.A Gemunu Priyadarshana. Software Test Management Tool. Master of Information Technology - 2017, 19 черв. 2017 р. URL: http://hdl.handle.net/123456789/3949 (дата звернення: 25.04.2024).

10) V.B Singh1, Krishna Kumar Chaturvedi. Bug Tracking and Reliability Assessment System (BTRAS). International Journal of Software Engineering and Its Applications. 2011. Т. 5, № 4 (дата звернення: 25.04.2024).

11) Awati R., Wigmore I. What is monolithic architecture in software?. WhatIs. URL: https://www.techtarget.com/whatis/definition/monolithic-architecture (дата звернення: 01.05.2024).

12) Client-Server Based LBS Architecture / M. A. Nabhan та ін. International Journal of Handheld Computing Research. 2010. Т. 1, № 3. С. 1–18. URL: https://doi.org/10.4018/jhcr.2010070101 (дата звернення: 01.05.2024).

13) Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson Education, Limited, 2017 (дата звернення: 01.05.2024).

14) Hashemi-Pour C., Wright G. What Is Business Process Modeling Notation (BPMN)? | Definition from TechTarget. CIO. URL: https://www.techtarget.com/searchcio/definition/Business-Process-Modeling-Notation (дата звернення: 04.05.2024).

15) Kumar P. Exploring Component Architecture in Angular: A Comprehensive Guide with Examples. Medium. URL: https://codewithpawan.medium.com/exploring-component-architecture-in-angular-a-comprehensive-guide-with-examples-b22297dffc09 (дата звернення: 10.05.2024).

16) Ravoof S. MongoDB vs PostgreSQL: 15 Critical Differences. Kinsta. URL: https://kinsta.com/blog/mongodb-vs-postgresql/ (дата звернення: 10.05.2024).

17) What is an ORM (Object Relational Mapper)?. Prisma's Data Guide. URL: https://www.prisma.io/dataguide/types/relational/what-is-an-orm (дата звернення: 10.05.2024).

18) TypeORM - Amazing ORM for TypeScript and JavaScript. TypeORM. URL: https://typeorm.io/ (дата звернення: 10.05.2024).

19) Bhardwaj P. NestJS vs ExpressJS: Which Framework To Choose?. SoluteLabs. URL: https://www.solutelabs.com/blog/nestjs-vs-expressjs (дата звернення: 10.05.2024).

20) Raval N. React vs Angular: Which One is Best for Your Next Front-end Project?. Radixweb. URL: https://radixweb.com/blog/react-vs-angular (дата звернення: 10.05.2024).

21) Swagger Documentation. API Documentation & Design Tools for Teams | Swagger. URL: https://swagger.io/docs/ (дата звернення: 10.05.2024).

22) Passport Documentation. Passport.js. URL: https://www.passportjs.org/docs/ (дата звернення: 10.05.2024).

23) MinIO Object Storage for Kubernetes. MinIO | S3 & Kubernetes Native Object Storage for AI. URL: https://min.io/docs/minio/kubernetes/upstream/index.html (дата звернення: 10.05.2024).

24) NgRx Docs. NgRx Docs. URL: https://ngrx.io/docs (дата звернення: 10.05.2024).

25) Bootstrap Introduction. Bootstrap. URL: https://getbootstrap.com/docs/4.1/getting-started/introduction/ (дата звернення: 10.05.2024).

26) Highcharts Documentation. Highcharts - Interactive Charting Library for Developers. URL: https://www.highcharts.com/docs/index (дата звернення: 10.05.2024).

27) SonarQube 10.5. SonarQube. URL: https://docs.sonarsource.com/sonarqube/latest/ (дата звернення: 15.05.2024).

28) Docker Documentation. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 20.05.2024).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016318303

Дата перевірки:
06.06.2024 06:49:10 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.06.2024 08:32:43 EEST

ID користувача:
76913

Назва документа: **ІП-02_Литвин_ПЗ**

Кількість сторінок: **75** Кількість слів: **10979** Кількість символів: **87257** Розмір файлу: **8.78 MB** ID файлу: **1016116308**

11.2%
Схожість

Найбільша схожість: **4.61%** з джерелом з Бібліотеки (ID файлу: **1016106058**)

2.98% Джерела з Інтернету	167	Сторінка 77
11% Джерела з Бібліотеки	290	Сторінка 78

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

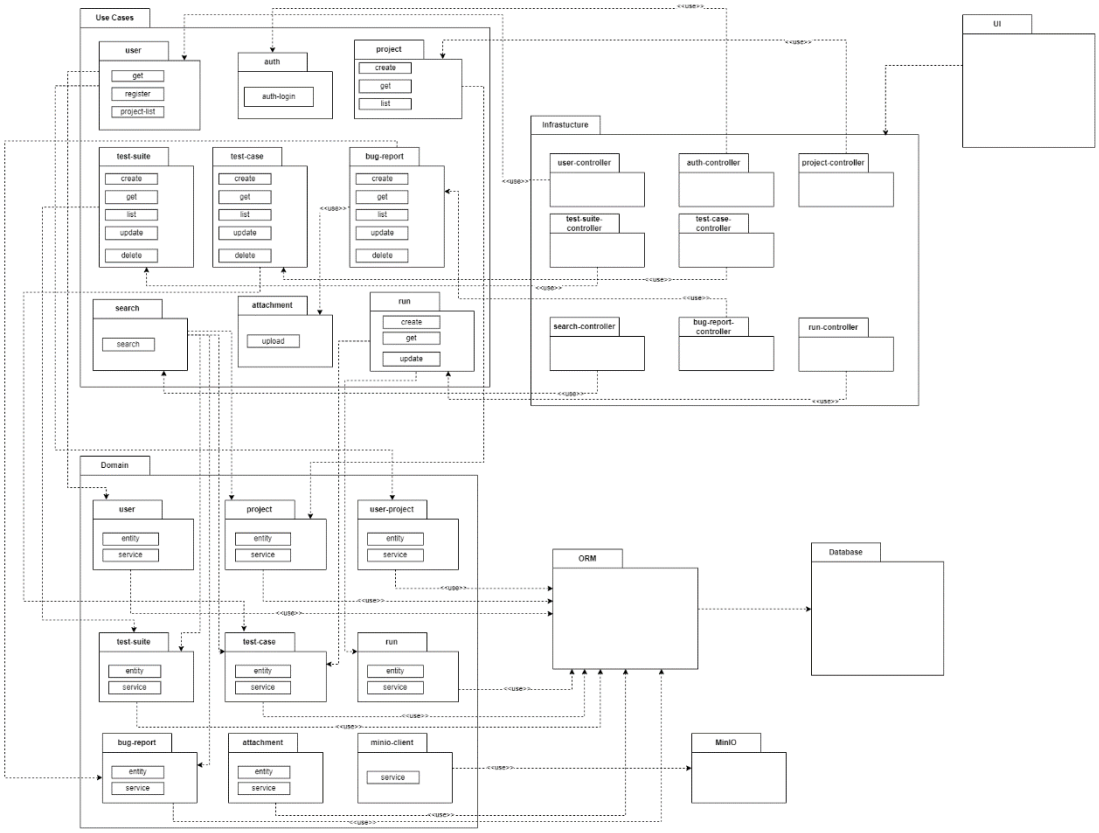
Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4

ДОДАТОК Б ДІАГРАМА ПАКЕТІВ СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ



ДОДАТОК В ВМІСТ ФАЙЛУ DOCKER-COMPOSE.YML ДЛЯ РОЗГОРТАННЯ ВЕБЗАСТОСУНКУ

Dockerfile

```
version: '3.1'

services:
  frontend:
    image: qahelper/frontend:latest
    restart: always
    ports:
      - "4200:4200"
    depends_on:
      - backend
    networks:
      - qa-helper
  backend:
    image: qahelper/backend:latest
    restart: always
    ports:
      - "3000:3000"
    environment:
      DATABASE_HOST: db
      DATABASE_PORT: 5432
      DATABASE_NAME: postgres
      DATABASE_USERNAME: postgres
      DATABASE_PASSWORD: postgres

      JWT_SECRET: superdupersecretkey
      JWT_EXPIRES_IN: 60m

      MINIO_BUCKET_NAME: qa-helper
      MINIO_ACCESS_KEY: bOwl0SnsNvknRX4ILxsm
      MINIO_SECRET_KEY: Qz6l65POZ2re7UZ8OtWnH2rjPQjhWOLMRyucbdrz

    depends_on:
      - db
      - minio
    networks:
      - qa-helper
  db:
    image: postgres
    restart: unless-stopped
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USERNAME: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - "db-data:/var/lib/postgresql/data"
    networks:
      - qa-helper
  minio:
    image: "bitnami/minio"
    restart: unless-stopped
    ports:
      - "9000:9000"
      - "9001:9001"
    volumes:
      - "minio-data:/bitnami/minio/data"
    networks:
      - qa-helper

volumes:
  db-data:
    driver: local
  minio-data:
    driver: local

networks:
  qa-helper:
    driver: bridge
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Текст програми

КПІ.ІП-0224.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Анастасія ЛИТВИН

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/...

Файл executable_objects.cpp

Реалізація функціональної вимоги "Можливість мати акаунт в застосунку", "Реєстрація користувача", "Авторизація користувача", "Вихід із акаунту":

Файл user.controller.ts

```
export class UserController {

  @Inject() private userMeUsecase: UserMeUsecase;
  @Inject() private userProjectListUsecase: UserProjectListUsecase;
  @Inject() private userRegisterUsecase: UserRegisterUsecase;

  @ApiOperation({ summary: "Get user projects" })
  @ApiResponse({ type: [Project] })
  @UseJwtGuard()
  @Get("projects")
  public projects(@Request() { user }: UserRequestInput): Promise<Project[]> {
    return this.userProjectListUsecase.run({ id: user.id });
  }

  @ApiOperation({ summary: "Get authorized user" })
  @UseJwtGuard()
  @Get()
  public me(@Request() { user }: UserRequestInput): Promise<User | null> {
    return this.userMeUsecase.run({ id: user.id });
  }

  @ApiOperation({ summary: "Register a new user" })
  @ApiResponse({ type: User })
  @Post()
  public register(@Body() body: UserRegisterInput): Promise<any> {
    return this.userRegisterUsecase.run(body);
  }
}
```

Файл user-register.usecase.ts

```
export class UserRegisterUsecase extends Usecase<UserRegisterInput, User> {

  @Inject() private userService: UserService;
  private readonly PASSWORD_SALT = 10;

  protected async execute(input: UserRegisterInput): Promise<User> {

    const userByUsername = await this.userService.findOneByUsername(input.username);

    if (userByUsername) {
      throw new ConflictException(`User with username: ${input.username} already exists.`);
    }

    const passwordHash = await hash(input.password, this.PASSWORD_SALT);

    const user = await this.userService.create({
      ...input,
      passwordHash,
    });

    return this.userService.get({ id: user.id });
  }
}
```

Файл user-project-list.usecase.ts

```
export class UserProjectListUsecase extends Usecase<UserOneInput, UserProject[], Project[]> {

  @Inject() private userProjectService: UserProjectService;

  protected async execute(input: UserOneInput): Promise<UserProject[]> {
    return this.userProjectService.list(input);
  }

  protected async format(userProjects: UserProject[]): Promise<Project[]> {
    return userProjects.map((userProject: UserProject) => userProject.project);
  }
}
```

Файл user-me.usecase.ts

```
export class UserMeUsecase extends Usecase<UserOneInput, User | null> {

  @Inject() private userService: UserService;

  protected async execute(input: UserOneInput): Promise<User | null> {
    return this.userService.get(input);
  }
}
```

Файл user.service.ts

```
export class UserService {

  @InjectRepository(User) private userRepository: Repository<User>;

  public async validateUser(username: string, password: string): Promise<any> {
    const user = await this.findOneByUsername(username); // TODO move from here?
    if (!user) {
      return null;
    }
    const isMatch = await compare(password, user.passwordHash);
    if (isMatch) {
      // eslint-disable-next-line @typescript-eslint/no-unused-vars
      const { passwordHash, ...result } = user;
      return result;
    }
    return null;
  }

  public findOneByUsername(username: string): Promise<User | null> {
    return this.userRepository.findOneBy({ username });
  }

  public get(input: UserOneInput): Promise<User | null> {
    return this.userRepository.findOne({ where: { id: input.id } });
  }

  public create(input: DeepPartial<User>): Promise<User> {
    return this.userRepository.save(input);
  }
}
```

Файл header.component.ts

```
public logout(): void {
  sessionStorage.clear();
  this.rootStoreService.clear();
  void this.router.navigate(["/signIn"]);
}
```

Реалізація функціональної вимоги "Перегляд інформації про свій акаунт":

Файл dashboard.component.ts

```
export class DashboardComponent implements OnInit, OnDestroy {

  constructor(
    private rootStoreService: RootStoreService,
    private dashboardStoreService: DashboardStoreService,
    private modalService: NgbModal,
    private changeDetectorRef: ChangeDetectorRef,
  ) {
  }

  public user$!: Observable<User | undefined>;
  public projects$!: Observable<Project[]>;
  public copiedIndex: number = -1;

  public ngOnInit(): void {
    this.dashboardStoreService.loadUserProjects();
    this.user$ = this.rootStoreService.selectUser$;
    this.projects$ = this.dashboardStoreService.selectUserProjects$;
  }
}
```

Файл dashboard.component.html

```
<div
  *ngIf="user$| async as user"
  class="row my-3 mx-0 p-3 rounded text-center text-dark bg-dark-subtle  rounded m-3 p-3"
>
```

```
    <div class="h3 fs-1 font-weight-bold mb-1">{{ user.name }}</div>
    <div class="h4 fs-3 font-weight-bold">@{{ user.username }}</div>
</div>
```

Реалізація функціональної вимог "Перегляд проєктів користувача", "Створення проєкту", "Доєднання до проєкту іншого користувача", "Переглядати усіх учасників проєкту":

Файл project.controller.ts

```
export class ProjectController {

    @Inject() private projectCreateUsecase: ProjectCreateUsecase;
    @Inject() private projectListUsecase: ProjectListUsecase;
    @Inject() private projectGetOneUsecase: ProjectGetOneUsecase;
    @Inject() private projectUpdateUsecase: ProjectUpdateUsecase;
    @Inject() private projectDeleteUsecase: ProjectDeleteUsecase;
    @Inject() private projectJoinUsecase: ProjectJoinUsecase;
    @Inject() private projectUsersUsecase: ProjectUsersUsecase;

    @ApiOperation({ summary: "Create new project" })
    @ApiResponse({ type: Project })
    @Post()
    public create(
        @Request() { user }: any,
        @Body() body: ProjectCreateInput,
    ): Promise<Project> {
        return this.projectCreateUsecase.runInTransaction({ userId: user.id, ...body });
    }

    @ApiOperation({ summary: "Get paginated list of projects" })
    @ApiPaginatedResponse(Project)
    @Get()
    @UsePagination()
    public list(@Query() query: ProjectListInput): Promise<PaginationResults<Project>> {
        return this.projectListUsecase.run(query);
    }

    @ApiOperation({ summary: "Get project by ID" })
    @ApiResponse({ type: Project })
    @Get(":id")
    public get(@Param() param: ProjectOneInput): Promise<Project | null> {
        return this.projectGetOneUsecase.run(param);
    }

    @ApiOperation({ summary: "Update project by ID" })
    @Patch(":id")
    public update(
        @Param() param: ProjectOneInput,
        @Body() body: ProjectUpdateInput,
    ): Promise<Project | null> {
        return this.projectUpdateUsecase.runInTransaction({ ...param, ...body });
    }

    @ApiOperation({ summary: "Delete project by ID" })
    @Delete(":id")
    public delete(@Param() param: ProjectOneInput): Promise<boolean> {
        return this.projectDeleteUsecase.run(param);
    }

    @ApiOperation({ summary: "Join project by ID" })
    @ApiResponse({ type: Project })
    @UseJwtGuard()
    @Post(":id/join")
    public me(
        @Request() { user }: UserRequestInput,
        @Param() param: ProjectOneInput,
    ): Promise<Project> {
        return this.projectJoinUsecase.run({ userId: user.id, id: param.id });
    }

    @ApiOperation({ summary: "Get project users by ID" })
    @ApiResponse({ type: ProjectUserListOutput })
    @Get(":id/users")
    public users(@Param() param: ProjectOneInput): Promise<ProjectUserListOutput> {
        return this.projectUsersUsecase.run(param);
```

```
  }
}
```

Файл project-create.usecase.ts

```
export class ProjectCreateUsecase extends Usecase<ProjectCreateInput & { userId: string }, Project> {

  @Inject() private projectService: ProjectService;
  @Inject() private userProjectService: UserProjectService;

  protected async execute(input: ProjectCreateInput & { userId: string }): Promise<Project> {
    await this.checkExisting(input);

    const project = await this.projectService.create(input);

    await this.userProjectService.create({ projectId: project.id, userId: input.userId });

    return project;
  }

  private async checkExisting(input: ProjectCreateInput): Promise<void> {
    const projectByName: Project | null = await this.projectService.findOneByName(input.name);

    if (projectByName) {
      throw new ConflictException(`Project with name: ${input.name} already exists.`);
    }

    const projectByCode: Project | null = await this.projectService.findOneByCode(input.code);

    if (projectByCode) {
      throw new ConflictException(`Project with code: ${input.code} already exists.`);
    }
  }
}
```

Файл project-get-one.usecase.ts

```
export class ProjectGetOneUsecase extends Usecase<ProjectOneInput, Project | null> {

  @Inject() private projectService: ProjectService;

  protected async execute(input: ProjectOneInput): Promise<Project | null> {
    const project = await this.projectService.getOne(input.id);

    if (!project) {
      throw new NotFoundException(`Project with id: ${input.id} not found`);
    }

    return project;
  }
}
```

Файл project-join.usecase.ts

```
export class ProjectJoinUsecase extends Usecase<{ userId: string } & ProjectOneInput, Project> {

  @Inject() private projectService: ProjectService;
  @Inject() private userProjectService: UserProjectService;

  protected async execute(input: { userId: string } & ProjectOneInput): Promise<Project> {
    const { userId, id: projectId } = input;
    const project = await this.projectService.getById(projectId);
    if (!project) {
      throw new NotFoundException(`Project with id: ${projectId} not found`);
    }
    const userProject = await this.userProjectService.get(userId, projectId);
    console.log(userProject);
    if (userProject) {
      throw new BadRequestException("You are already joined this project");
    }
    await this.userProjectService.create({
      projectId,
      userId,
    });
    return project;
  }
}
```

Файл project-list.usecase.ts

```
export class ProjectListUsecase extends Usecase<ProjectListInput, PaginationResults<Project>> {

  @Inject() private projectService: ProjectService;
```

```
    protected async execute(input: ProjectListInput): Promise<PaginationResults<Project>> {
        return this.projectService.list(input);
    }
}
```

Файл project-users.usecase.ts

```
export class ProjectUsersUsecase extends Usecase<ProjectOneInput, ProjectUserListOutput> {

    @Inject() private projectService: ProjectService;
    @Inject() private userProjectService: UserProjectService;

    protected async execute(input: ProjectOneInput): Promise<ProjectUserListOutput> {

        const project = await this.projectService.getOne(input.id);

        if (!project) {
            throw new NotFoundException(`Project with id: ${input.id} not found`);
        }

        const userProjects = await this.userProjectService.getUsersByProjectId(input.id);

        return {
            project,
            users: userProjects.map((userProject: UserProject) => userProject.user),
        };
    }
}
```

Файл project.service.ts

```
export class ProjectService {
    @InjectRepository(Project)
    private projectRepository: Repository<Project>;

    public async create(input: ProjectCreateInput): Promise<Project> {
        return this.projectRepository.save(input);
    }

    public list(query: ProjectListInput): Promise<PaginationResults<Project>> {
        const queryBuilder = this.projectRepository.createQueryBuilder();
        return paginate(queryBuilder, query.limit, query.page);
    }

    public getById(id: string): Promise<Project | null> {
        return this.projectRepository.findOneBy({ id });
    }

    public getOne(id: string): Promise<Project | null> {
        return this.projectRepository.findOne({
            where: { id },
            order: {
                bugReports: {
                    id: "DESC",
                },
                testSuites: {
                    id: "DESC",
                },
            },
            relations: ["bugReports", "testSuites"],
        });
    }

    public async update(
        project: Project,
        updates: ProjectUpdateInput,
    ): Promise<Project | null> {
        const updatedProject = this.projectRepository.merge(project, updates);

        return this.projectRepository.save(updatedProject);
    }

    public async delete(id: string): Promise<boolean> {
        const { affected } = await this.projectRepository.delete(id);

        return !!affected;
    }

    public async findOneByName(name: string): Promise<Project | null> {
        return this.projectRepository.findOneBy({ name });
    }
```

```
  public async findOneByCode(code: string): Promise<Project | null> {
    return this.projectRepository.findOneBy({ code });
  }

  public findByName(name: string): Promise<Pick<Project, "id" | "name">[]> {
    return this.projectRepository.find({
      where: {
        name: ILike(`%${name}%`),
      },
      select: {
        id: true,
        name: true,
      },
    });
  }
}
```

Файл user-project.service.ts

```
export class UserProjectService {

  @InjectRepository(UserProject) private userProjectRepository: Repository<UserProject>;

  public create(input: UserProjectOneInput): Promise<UserProject> {
    return this.userProjectRepository.save(input);
  }

  public get(userId: string, projectId: string): Promise<UserProject | null> {
    return this.userProjectRepository.findOneBy({
      userId,
      projectId,
    });
  }

  public list(input: UserOneInput): Promise<UserProject[]> {
    return this.userProjectRepository.find({ where: { userId: input.id }, relations: ["project"] });
  }

  public async getUsersByProjectId(id: string): Promise<UserProject[]> {
    return this.userProjectRepository.find({
      where: {
        projectId: id,
      },
      relations: ["user", "project"],
    });
  }
}
```

Файл project.component.ts

```
export class ProjectComponent implements OnInit {

  constructor(
    private route: ActivatedRoute,
    private projectStoreService: ProjectStoreService,
  ) {
  }

  public project$!: Observable<Project | undefined>;

  public ngOnInit(): void {
    this.project$ = this.projectStoreService.project$;
    this.route.params.pipe(
      map((params: Params) => params["id"]),
    ).subscribe((id: string) => this.projectStoreService.loadProject(id));
  }
}
```

Реалізація функціональної вимоги "Надавати унікальний код проєкту, що можна скопіювати, для додавання до проєкту інших користувачів":

Файл dashboard.component.ts

```
public copyToClipboard(project: Project, index: number): void {
  void navigator.clipboard.writeText(project.id);
  this.copiedIndex = index;
  setTimeout(() => {
    this.copiedIndex = -1;
    this.changeDetectorRef.markForCheck();
  }, 5000);
}
```

Файл dashboard.component.html

```
<button
   class="btn btn-outline-secondary borderless col-2"
   type="button"
   (click)="copyToClipboard(project, index)"
   data-bs-toggle="modal"
>
   <i
      *ngIf="copiedIndex !== index"
      class="fa-solid fa-copy"
   ></i>
   <span *ngIf="copiedIndex === index">Copied to clipboard!</span>
</button>
```

Реалізація функціональної вимоги "Перегляд звітів про дефект, що належать певному проєкту", "Створення звіту про дефект", "Редагування звіту про дефект", "Видалення звіту про дефект", "Наявність усіх полів звіту про дефект відповідно до правил заповнення документації":

Файл bug-report.controller.ts

```
export class BugReportController {

   @Inject() private bugReportCreateUsecase: BugReportCreateUsecase;
   @Inject() private bugReportListUsecase: BugReportListUsecase;
   @Inject() private bugReportGetUsecase: BugReportGetUsecase;
   @Inject() private bugReportUpdateUsecase: BugReportUpdateUsecase;
   @Inject() private bugReportDeleteUsecase: BugReportDeleteUsecase;

   @ApiOperation({ summary: "Create new bug report" })
   @ApiResponse({ type: BugReport })
   @ApiConsumes("multipart/form-data")
   @ApiExtraModels(BugReportCreateInput)
   @UseInterceptors(FilesInterceptor("attachments"))
   @Post()
   public create(
      @Body() body: BugReportCreateInput,
      @UploadedFiles() files: Express.Multer.File[],
   ): Promise<BugReport> {
      return this.bugReportCreateUsecase.runInTransaction({
         ...body,
         files,
      });
   }

   @ApiOperation({ summary: "Get paginated list of bug reports" })
   @ApiPaginatedResponse(BugReport)
   @Get()
   @UsePagination()
   public list(@Query() query: BugReportListInput): Promise<PaginationResults<BugReport>> {
      return this.bugReportListUsecase.run(query);
   }

   @ApiOperation({ summary: "Get bug report by ID" })
   @ApiResponse({ type: BugReport })
   @Get(":id")
   public get(@Param() param: BugReportOneInput): Promise<BugReport> {
      return this.bugReportGetUsecase.run(param);
   }

   @ApiOperation({ summary: "Update test suite by ID" })
   @Patch(":id")
   public update(
      @Param() param: BugReportOneInput,
      @Body() body: BugReportUpdateInput,
   ): Promise<BugReport | null> {
      return this.bugReportUpdateUsecase.runInTransaction({ ...param, ...body });
   }

   @ApiOperation({ summary: "Delete bug report by ID" })
   @Delete(":id")
   public delete(@Param() param: TestSuiteOneInput): Promise<boolean> {
      return this.bugReportDeleteUsecase.run(param);
   }
}
```

Файл bug-report-create.usecase.ts

```
export class BugReportCreateUsecase extends Usecase<BugReportCreateInput & { files: Express.Multer.File[] }, BugReport> {

  @Inject() private projectService: ProjectService;
  @Inject() private bugReportService: BugReportService;
  @Inject() private attachmentUploadUsecase: AttachmentUploadUsecase;

  protected async execute(input: BugReportCreateInput & { files: Express.Multer.File[] }): Promise<BugReport> {
    const project = await this.projectService.getById(input.projectId);

    if (!project) {
      throw new NotFoundException(`Project with id: ${input.projectId} not found.`);
    }

    const { files, ...data } = input;
    let attachments: Attachment[] = [];
    if (files) {
      attachments = await this.attachmentUploadUsecase.run({ files });
    }
    return this.bugReportService.create({
      ...data,
      attachments,
    });
  }
}
```

Файл bug-report-delete.usecase.ts

```
export class BugReportDeleteUsecase extends Usecase<BugReportOneInput, boolean> {

  @Inject() private bugReportService: BugReportService;

  protected async execute(input: TestSuiteOneInput): Promise<boolean> {
    return this.bugReportService.delete(input);
  }
}
```

Файл bug-report-get.usecase.ts

```
export class BugReportGetUsecase extends Usecase<BugReportOneInput, BugReport> {

  @Inject() private bugReportService: BugReportService;

  protected async execute(input: BugReportOneInput): Promise<BugReport> {
    const bugReport = await this.bugReportService.getById(input.id);

    if (!bugReport) {
      throw new NotFoundException(`Bug report with id: ${input.id} not found.`);
    }

    return bugReport;
  }
}
```

Файл bug-report-list.usecase.ts

```
export class BugReportGetUsecase extends Usecase<BugReportOneInput, BugReport> {

  @Inject() private bugReportService: BugReportService;

  protected async execute(input: BugReportOneInput): Promise<BugReport> {
    const bugReport = await this.bugReportService.getById(input.id);

    if (!bugReport) {
      throw new NotFoundException(`Bug report with id: ${input.id} not found.`);
    }

    return bugReport;
  }
}
```

Файл bug-report-update.usecase.ts

```
export class BugReportGetUsecase extends Usecase<BugReportOneInput, BugReport> {

  @Inject() private bugReportService: BugReportService;

  protected async execute(input: BugReportOneInput): Promise<BugReport> {
    const bugReport = await this.bugReportService.getById(input.id);

    if (!bugReport) {
      throw new NotFoundException(`Bug report with id: ${input.id} not found.`);
    }

    return bugReport;
```

```
  }
}
```

Файл bug-report.service.ts

```
export class BugReportService {

  @InjectRepository(BugReport) private bugReportRepository: Repository<BugReport>;

  public async create(input: DeepPartial<BugReport>): Promise<BugReport> {
    return this.bugReportRepository.save(input);
  }

  public list(query: BugReportListInput): Promise<PaginationResults<BugReport>> {
    const queryBuilder = this.bugReportRepository.createQueryBuilder("br");

    queryBuilder.andWhere("br.projectId = :projectId", { projectId: query.projectId });

    return paginate(queryBuilder, query.limit, query.page);
  }

  public getById(id: number): Promise<BugReport | null> {
    return this.bugReportRepository.findOne({
      where: { id },
      relations: ["project", "attachments"],
    });
  }

  public async update(
    id: number,
    updates: Partial<BugReport>,
  ): Promise<UpdateResult> {
    return this.bugReportRepository.update(id, updates);
  }

  public async delete(param: BugReportOneInput): Promise<boolean> {
    const { affected } = await this.bugReportRepository.delete({ id: param.id });

    return !!affected;
  }

  public findBySummary(summary: string): Promise<Pick<BugReport, "id" | "summary">[]> {
    return this.bugReportRepository.find({
      where: {
        summary: ILike(`%${summary}%`),
      },
      select: {
        id: true,
        summary: true,
      },
    });
  }
}
```

Файл bug-report.component.ts

```
export class BugReportComponent implements OnInit {
  constructor(
    private route: ActivatedRoute,
    private bugReportRestService: BugReportRestService,
  ) {
  }

  public bugReport$!: Observable<BugReport>;

  public ngOnInit(): void {
    this.bugReport$ = this.route.params.pipe(
      map((params: Params) => params["id"]),
      switchMap((id: string) => this.bugReportRestService.loadBugReport(id)),
    );
  }
}
```

Реалізація функціональної вимоги "Відображення пріоритету звіту про дефект відповідними символами на сторінці проєкту":

Файл bug-report-list.component.html

```
<div
  class="me-2"
  [ngSwitch]="bugReport.priority"
```

```
>
   <i
      *ngSwitchCase="bugReportPriorities.HIGH"
      class="fa-solid fa-up-long me-1 text-danger fs-5"
   ></i>
   <i
      *ngSwitchCase="bugReportPriorities.MEDIUM"
      class="fa-solid fa-up-down me-1 text-success fs-5"
   ></i>
   <i
      *ngSwitchCase="bugReportPriorities.LOW"
      class="fa-solid fa-down-long me-1 text-primary fs-5"
   ></i>
</div>
```

Реалізація функціональної вимоги "Перегляд тестових наборів, що належать певному проєкту", "Створення тестового набору", "Редагування тестового набору", "Видалення тестового набору":

Файл test-suite.controller.ts

```
export class TestSuiteController {

   @Inject() private testSuiteCreateUsecase: TestSuiteCreateUsecase;
   @Inject() private testSuiteListUsecase: TestSuiteListUsecase;
   @Inject() private testSuiteGetUsecase: TestSuiteGetUsecase;
   @Inject() private testSuiteUpdateUsecase: TestSuiteUpdateUsecase;
   @Inject() private testSuiteDeleteUsecase: TestSuiteDeleteUsecase;

   @ApiOperation({ summary: "Create new test suite" })
   @ApiResponse({ type: TestSuite })
   @Post()
   public create(@Body() body: TestSuiteCreateInput): Promise<TestSuite> {
      return this.testSuiteCreateUsecase.runInTransaction(body);
   }

   @ApiOperation({ summary: "Get paginated list of test suites" })
   @ApiPaginatedResponse(TestSuite)
   @Get()
   @UsePagination()
   public list(@Query() query: TestSuiteListInput): Promise<PaginationResults<TestSuite>> {
      return this.testSuiteListUsecase.run(query);
   }

   @ApiOperation({ summary: "Get test suite by ID" })
   @Get(":id")
   public get(@Param() param: TestSuiteOneInput): Promise<TestSuite | null> {
      return this.testSuiteGetUsecase.run(param);
   }

   @ApiOperation({ summary: "Update test suite by ID" })
   @Patch(":id")
   public update(
      @Param() param: TestSuiteOneInput,
      @Body() body: TestSuiteUpdateInput,
   ): Promise<TestSuite | null> {
      return this.testSuiteUpdateUsecase.runInTransaction({ ...param, ...body });
   }

   @ApiOperation({ summary: "Delete test suite by ID" })
   @Delete(":id")
   public delete(@Param() param: TestSuiteOneInput): Promise<boolean> {
      return this.testSuiteDeleteUsecase.run(param);
   }
}
```

Файл test-suite-create.usecase.ts

```
export class TestSuiteCreateUsecase extends Usecase<TestSuiteCreateInput, TestSuite> {

   @Inject() private projectService: ProjectService;
   @Inject() private testSuiteService: TestSuiteService;

   protected async execute(input: TestSuiteCreateInput): Promise<TestSuite> {
      const project = await this.projectService.getById(input.projectId);

      if (!project) {
         throw new NotFoundException(`Project with id: ${input.projectId} not found.`);
      }
```

```
        return this.testSuiteService.create({
            ...input,
            project,
        });
    }
}
```

Файл test-suite-delete.usecase.ts

```
export class TestSuiteDeleteUsecase extends Usecase<TestSuiteOneInput, boolean> {

    @Inject() private testSuiteService: TestSuiteService;

    protected async execute(input: TestSuiteOneInput): Promise<boolean> {
        return this.testSuiteService.delete(input);
    }
}
```

Файл test-suite-get.usecase.ts

```
export class TestSuiteGetUsecase extends Usecase<TestSuiteOneInput, TestSuite | null> {

    @Inject() private testSuiteService: TestSuiteService;

    protected async execute(input: TestSuiteOneInput): Promise<TestSuite | null> {
        return this.testSuiteService.getOne(input.id);
    }
}
```

Файл test-suite-list.usecase.ts

```
export class TestSuiteListUsecase extends Usecase<TestSuiteListInput, PaginationResults<TestSuite>> {

    @Inject() private testSuiteService: TestSuiteService;

    protected async execute(input: TestSuiteListInput): Promise<PaginationResults<TestSuite>> {
        return this.testSuiteService.list(input);
    }
}
```

Файл test-suite-update.usecase.ts

```
export class TestSuiteUpdateUsecase extends Usecase<TestSuiteOneInput & TestSuiteUpdateInput, TestSuite> {

    @Inject() private testSuiteService: TestSuiteService;

    protected async execute(input: TestSuiteOneInput & TestSuiteUpdateInput): Promise<TestSuite> {
        const testSuite = await this.testSuiteService.getById(input.id);

        if (!testSuite) {
            throw new NotFoundException(`Test suite with id: ${input.id} not found.`);
        }

        const updated = await this.testSuiteService.update(input.id, input);

        if (!!updated.affected) {
            return {
                ...testSuite,
                ...input,
            };
        } else {
            throw new BadRequestException("Test suite was not updated.");
        }
    }
}
```

Файл test-suite.service.ts

```
export class TestSuiteService {

    @InjectRepository(TestSuite) private testSuiteRepository: Repository<TestSuite>;

    public async create(input: TestSuiteCreateInput & { project: Project }): Promise<TestSuite> {
        return this.testSuiteRepository.save(input);
    }

    public list(input: TestSuiteListInput): Promise<PaginationResults<TestSuite>> {
        const queryBuilder = this.testSuiteRepository.createQueryBuilder("ts");

        queryBuilder.andWhere("ts.projectId = :projectId", { projectId: input.projectId });

        return paginate(queryBuilder, input.limit, input.page);
    }

    public getById(id: number): Promise<TestSuite | null> {
        return this.testSuiteRepository.findOneBy({ id });
```

```
    }

    public getOne(id: number): Promise<TestSuite | null> {
        return this.testSuiteRepository.findOne({
            where: { id },
            relations: ["testCases", "project", "runs"],
            order: {
                runs: {
                    id: "DESC",
                },
                testCases: {
                    id: "DESC",
                },
            },
        });
    }

    public async update(
        id: number,
        updates: Partial<TestSuite>,
    ): Promise<UpdateResult> {
        return this.testSuiteRepository.update(id, updates);
    }

    public async delete(param: TestSuiteOneInput): Promise<boolean> {
        const { affected } = await this.testSuiteRepository.delete({ id: param.id });

        return !!affected;
    }

    public findByName(name: string): Promise<Pick<TestSuite, "id" | "name">[]> {
        return this.testSuiteRepository.find({
            where: {
                name: ILike(`%${name}%`),
            },
            select: {
                id: true,
                name: true,
            },
        });
    }
}
```

Файл test-suite.component.ts

```
export class TestSuiteComponent implements OnInit {

    constructor(
        private route: ActivatedRoute,
        private testSuiteRestService: TestSuiteRestService,
    ) {
    }

    public testSuite$!: Observable<TestSuite>;

    public ngOnInit(): void {
        this.testSuite$ = this.route.params.pipe(
            map((params: Params) => params["id"]),
            switchMap((id: string) => this.testSuiteRestService.loadTestSuite(id)),
        );
    }
}
```

Реалізація функціональної вимоги "Можливість створювати тестові випадки всередині набору", "Перегляд тестових випадків, що належать певному тестовому набору", "Створення тестового випадку", "Редагування тестового випадку", "Видалення тестового випадку", "Наявність усіх полів тестового випадку відповідно до правил заповнення документації":

Файл test-case.controller.ts

```
export class TestCaseController {

    @Inject() private testCaseCreateUsecase: TestCaseCreateUsecase;
    @Inject() private testCaseGetUsecase: TestCaseGetUsecase;
    @Inject() private testCaseUpdateUsecase: TestCaseUpdateUsecase;
    @Inject() private testCaseDeleteUsecase: TestCaseDeleteUsecase;
```

```
  @ApiOperation({ summary: "Create new test case" })
  @ApiResponse({ type: TestCase })
  @Post()
  public create(@Body() body: TestCaseCreateInput): Promise<TestCase> {
    return this.testCaseCreateUsecase.run(body);
  }

  @ApiOperation({ summary: "Get test case by ID" })
  @Get(":id")
  public get(@Param() param: TestCaseOneInput): Promise<TestCase> {
    return this.testCaseGetUsecase.run(param);
  }

  @ApiOperation({ summary: "Update test case by ID" })
  @Patch(":id")
  public update(
    @Param() param: TestCaseOneInput,
    @Body() body: TestCaseUpdateInput,
  ): Promise<TestCase | null> {
    return this.testCaseUpdateUsecase.runInTransaction({ ...param, ...body });
  }

  @ApiOperation({ summary: "Delete test case by ID" })
  @Delete(":id")
  public delete(@Param() param: TestCaseOneInput): Promise<boolean> {
    return this.testCaseDeleteUsecase.run(param);
  }
}
```

Файл test-case-create.usecase.ts

```
export class TestCaseCreateUsecase extends Usecase<TestCaseCreateInput, TestCase> {

  @Inject() private testCaseService: TestCaseService;
  @Inject() private testSuiteService: TestSuiteService;

  protected async execute(input: TestCaseCreateInput): Promise<TestCase> {
    const project = await this.testSuiteService.getById(input.testSuiteId);

    if (!project) {
      throw new NotFoundException(`Project with id: ${input.testSuiteId} not found.`);
    }

    return this.testCaseService.create(input);
  }
}
```

Файл test-case-delete.usecase.ts

```
export class TestCaseDeleteUsecase extends Usecase<TestCaseOneInput, boolean> {

  @Inject() private testCaseService: TestCaseService;

  protected async execute(input: TestSuiteOneInput): Promise<boolean> {
    return this.testCaseService.delete(input);
  }
}
```

Файл test-case-get.usecase.ts

```
export class TestCaseGetUsecase extends Usecase<TestCaseOneInput, TestCase> {

  @Inject() private testCaseService: TestCaseService;

  protected async execute(input: TestCaseOneInput): Promise<TestCase> {
    const testCase = await this.testCaseService.getOne(input.id);

    if (!testCase) {
      throw new NotFoundException(`Test case with id: ${input.id} not found`);
    }

    return testCase;
  }
}
```

Файл test-case-update.usecase.ts

```
export class TestCaseUpdateUsecase extends Usecase<TestCaseOneInput & TestCaseUpdateInput, TestCase> {

  @Inject() private testCaseService: TestCaseService;
  @Inject() private testCaseGetUsecase: TestCaseGetUsecase;

  protected async execute({ id, ...updates }: TestCaseOneInput & TestCaseUpdateInput): Promise<TestCase> {
```

```
    const updatedTestCase = await this.testCaseService.update(id, updates);

    return this.testCaseGetUsecase.run({ id: updatedTestCase.id });
  }
}
```

Файл test-case.service.ts

```
export class TestCaseService {

  @InjectRepository(TestCase) private testCaseRepository: Repository<TestCase>;

  public async create(input: DeepPartial<TestCase>): Promise<TestCase> {
    return this.testCaseRepository.save(input);
  }

  public getOne(id: number): Promise<TestCase | null> {
    return this.testCaseRepository.findOne({
      where: {
        id,
      },
      relations: ["steps", "testSuite", "testSuite.project"],
    });
  }

  public async update(
    id: number,
    updates: DeepPartial<TestCase>,
  ): Promise<TestCase> {
    return this.testCaseRepository.save({ id, ...updates });
  }

  public async delete(param: TestCaseOneInput): Promise<boolean> {
    const { affected } = await this.testCaseRepository.delete({ id: param.id });

    return !!affected;
  }

  public findByTitle(title: string): Promise<Pick<TestCase, "id" | "title">[]> {
    return this.testCaseRepository.find({
      where: {
        title: ILike(`%${title}%`),
      },
      select: {
        id: true,
        title: true,
      },
    });
  }

  public findByTestSuiteId(testSuiteId: number): Promise<TestCase[]> {
    return this.testCaseRepository.findBy({ testSuiteId });
  }
}
```

Файл test-case.component.ts

```
export class TestCaseComponent implements OnInit {

  constructor(
    private route: ActivatedRoute,
    private modalService: NgbModal,
    private testCaseRestService: TestCaseRestService,
  ) {
  }

  public testCase$!: Observable<TestCase>;

  public ngOnInit(): void {
    this.testCase$ = this.route.params.pipe(
      map((params: Params) => params["id"]),
      switchMap((id: string) => this.testCaseRestService.loadTestCase(id)),
    );
  }

  public updateTestCase(testCase: TestCase): void {
    this.testCase$ = of(testCase);
    this.modalService.dismissAll();
  }
}
```

Реалізація функціональної вимоги "Прогін тестового набору", "Запуск прогону тестового набору", "Присвоєння статусу для тестового випадку, що є частиною тестового набору, в даному прогоні", "Можливість продовжити тестовий прогін через певний час, поки той незавершений", "Завершити прогін", "Переглянути аналітику запуску даного прогону", "Переглянути результати прогону для тестового набору та тестових випадків, що їх включає набір":

Файл run.controller.ts

```
export class RunController {

  @Inject() private runCreateUsecase: RunCreateUsecase;
  @Inject() private runGetOneUsecase: RunGetUsecase;
  @Inject() private runSaveUsecase: RunSaveUsecase;
  @Inject() private runDoneUsecase: RunDoneUsecase;
  @Inject() private runGetAnalyticsUsecase: RunGetAnalyticsUsecase;

  @ApiOperation({ summary: "Create run for test suite" })
  @ApiResponse({ type: Run })
  @Post()
  public create(@Body() body: RunCreateInput): Promise<Run> {
    return this.runCreateUsecase.run(body);
  }

  @ApiOperation({ summary: "Get run by ID" })
  @ApiResponse({ type: Run })
  @Get(":id")
  public getOne(@Param() param: RunOneInput): Promise<Run | null> {
    return this.runGetOneUsecase.run(param);
  }

  @ApiOperation({ summary: "Save run" })
  @ApiResponse({ type: Run })
  @Post(":id/save")
  public save(
    @Param() param: RunOneInput,
    @Body() body: RunSaveInput,
  ): Promise<Run | null> {
    return this.runSaveUsecase.run({
      ...param,
      ...body,
    });
  }

  @ApiOperation({ summary: "Done run" })
  @ApiResponse({ type: Run })
  @Post(":id/done")
  public done(
    @Param() param: RunOneInput,
    @Body() body: RunSaveInput,
  ): Promise<Run | null> {
    return this.runDoneUsecase.run({
      ...param,
      ...body,
    });
  }

  @ApiOperation({ summary: "Run  get analytics" })
  @Get(":id/analytics")
  public analytics(@Param() param: RunOneInput): Promise<any> {
    return this.runGetAnalyticsUsecase.run(param);
  }
}
```

Файл run-create.usecase.ts

```
export class RunCreateUsecase extends Usecase<RunCreateInput, Run> {

  @Inject() private runService: RunService;
  @Inject() private testCaseService: TestCaseService;

  protected async execute(input: RunCreateInput): Promise<Run> {
    const testCases = await this.testCaseService.findByTestSuiteId(input.testSuiteId);

    if (!testCases.length) {
      throw new BadRequestException("Can't run test suite with no test cases");
```

```
    }

    const runSteps = testCases.map((testCase: TestCase): Partial<RunStep> => ({
      name: testCase.title,
      testCaseId: testCase.id,
    }));

    return this.runService.create({
      ...input,
      runSteps,
    });
  }
}
```

Файл run-done.usecase.ts

```
export class RunDoneUsecase extends Usecase<RunOneInput & RunSaveInput, Run | null> {

  @Inject() private runService: RunService;
  @Inject() private runGetOneUsecase: RunGetUsecase;

  protected async execute(input: RunOneInput & RunSaveInput): Promise<Run | null> {
    const { id, ...updates } = input;

    const run = await this.runService.getOne(id);

    if (!run) {
      throw new NotFoundException(`Run with id: ${id} not found`);
    }

    if (run.finishedAt) {
      throw new BadRequestException(`Run with id: ${id} is already finished`);
    }

    await this.runService.update(id, updates);

    await this.runService.updateFinishedAt(id);

    return this.runGetOneUsecase.run({ id });
  }
}
```

Файл run-get.usecase.ts

```
export class RunGetUsecase extends Usecase<RunOneInput, Run | null> {

  @Inject() private runService: RunService;

  protected async execute(input: RunOneInput): Promise<Run | null> {
    return this.runService.getOne(input.id);
  }
}
```

Файл run-get-analytics-usecase.service.ts

```
export class RunGetAnalyticsUsecase extends Usecase<RunOneInput, Run | null> {

  @Inject() private runService: RunService;

  protected async execute(input: RunOneInput): Promise<any> {
    const analytics = await this.runService.getOneAnalytics(input.id);

    const total = analytics.reduce((acc: number, item: any) => acc + Number(item.count), 0);

    return analytics.map((item: any) => ({
      ...item,
      percent: parseFloat((item.count / total * 100).toFixed(2)),
    }));
  }
}
```

Файл run-save.usecase.ts

```
export class RunSaveUsecase extends Usecase<RunOneInput & RunSaveInput, Run | null> {

  @Inject() private runService: RunService;
  @Inject() private runGetOneUsecase: RunGetUsecase;

  protected async execute(input: RunOneInput & RunSaveInput): Promise<Run | null> {

    const { id, ...updates } = input;

    const run = await this.runService.getOne(id);
```

```
        if (!run) {
            throw new NotFoundException(`Run with id: ${id} not found`);
        }

        if (run.finishedAt) {
            throw new BadRequestException(`Run with id: ${id} is already finished`);
        }

        await this.runService.update(id, updates);

        return this.runGetOneUsecase.run({ id });
    }
}
```

Файл run.service.ts

```
export class RunService {

    @InjectRepository(Run) private runEntityRepository: Repository<Run>;

    public create(input: DeepPartial<Run>): Promise<Run> {
        return this.runEntityRepository.save(input);
    }

    public getOne(id: number): Promise<Run | null> {
        return this.runEntityRepository.findOne({
            where: {
                id,
            },
            relations: ["runSteps", "runSteps.testCase", "runSteps.testCase.steps", "testSuite", "testSuite.project"],
            order: {
                runSteps: {
                    id: "ASC",
                },
            },
        });
    }

    public getOneAnalytics(id: number): Promise<any[]> {
        return this.runEntityRepository.createQueryBuilder("run")
            .leftJoinAndSelect("run.runSteps", "runSteps")
            .select("runSteps.status", "status")
            .addSelect("COUNT(runSteps.status)", "count")
            .where("run.id = :id", { id })
            .groupBy("runSteps.status")
            .getRawMany();

    }

    public update(id: number, updates: DeepPartial<Run>): Promise<Run> {
        return this.runEntityRepository.save({
            id,
            ...updates,
        });
    }

    public updateFinishedAt(id: number): Promise<UpdateResult> {
        return this.runEntityRepository
            .createQueryBuilder("run")
            .update<Run>(Run, { finishedAt: "now()" })
            .where("run.id = :id", { id })
            .returning(["*"])
            .updateEntity(true)
            .execute();
    }
}
```

Файл run.component.ts

```
export class RunComponent implements OnInit {

    constructor(
        private route: ActivatedRoute,
        private runRestService: RunRestService,
        private changeDetectorRef: ChangeDetectorRef,
    ) {
    }

    public selectedStep: RunStep | null = null;
    public run: Run;

    public ngOnInit(): void {
```

```
    this.route.params.pipe(
      map((params: Params) => params["id"]),
      switchMap((id: string) => this.runRestService.loadRun(id)),
    ).subscribe((run: Run) => {
      this.run = run;
      this.changeDetectorRef.markForCheck();
    });
  }

  public onSelectStep(runStep: RunStep): void {
    this.selectedStep = runStep;
  }

  public onRunStepChange(runStep: RunStep): void {
    this.run = {
      ...this.run,
      runSteps: [
        ...this.run.runSteps.map((item: RunStep) => {
          if (item.testCaseId === runStep.testCaseId) {
            return runStep;
          }
          return item;
        }),
      ],
    };
  }

  public onSave(): void {
    this.runRestService.saveRun({
      id: this.run.id,
      runSteps: this.run.runSteps,
    }).subscribe((run: Run) => {
      this.run = run;
      this.changeDetectorRef.markForCheck();
    });
  }


  public onDone(): void {
    this.runRestService.doneRun({
      id: this.run.id,
      runSteps: this.run.runSteps,
    }).subscribe((run: Run) => {
      this.run = run;
      this.changeDetectorRef.markForCheck();
    });
  }
}
```

Реалізація функціональних вимог "Здійснювати пошук по назвам", "Здійснювати пошук за назвою серед проєктів", "Здійснювати пошук за назвою серед тестових наборів", "Здійснювати пошук за назвою серед тестових випадків", "Здійснювати пошук за назвою серед тем звітів про дефект":

Файл search.controller.ts

```
export class SearchController {

  @Inject() private searchUsecase: SearchUsecase;

  @ApiOperation({ summary: "Search projects, test suites, test cases, bug reports by name" })
  @Get()
  public search(@Query() query: SearchInput): Promise<any> {
    return this.searchUsecase.run(query);
  }
}
```

Файл search.usecase.ts

```
export class SearchUsecase extends Usecase<SearchInput, SearchOutput[]> {

  @Inject() private projectService: ProjectService;
  @Inject() private bugReportService: BugReportService;
  @Inject() private testSuiteService: TestSuiteService;
  @Inject() private testCaseService: TestCaseService;
```

```
  protected async execute(input: SearchInput): Promise<SearchOutput[]> {
    console.log(input.query);
    const projects = await this.projectService.findByName(input.query);
    const bugReports = await this.bugReportService.findBySummary(input.query);
    const testSuites = await this.testSuiteService.findByName(input.query);
    const testCases = await this.testCaseService.findByTitle(input.query);
    console.log(projects);
    return [
      ...projects.map((project: Pick<Project, "id" | "name">) => ({
        id: project.id,
        title: project.name,
        type: SearchOutputType.PROJECT,
      })),
      ...bugReports.map((bugReport: Pick<BugReport, "id" | "summary">) => ({
        id: bugReport.id,
        title: bugReport.summary,
        type: SearchOutputType.BUG_REPORT,
      })),
      ...testSuites.map((testSuite: Pick<TestSuite, "id" | "name">) => ({
        id: testSuite.id,
        title: testSuite.name,
        type: SearchOutputType.TEST_SUITE,
      })),
      ...testCases.map((testCase: Pick<TestCase, "id" | "title">) => ({
        id: testCase.id,
        title: testCase.title,
        type: SearchOutputType.TEST_CASE,
      })),
    ];
  }
}
```

Файл header.component.ts

```
public search(event: string): void {
  this.items$ = of(event).pipe(
    filter((query: string) => query.length >= 3),
    switchMap((query: string) => this.searchRestService.search(query)),
  );
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Програма та методика тестування

КПІ.ІП-0224.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Анастасія ЛИТВИН

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є вебзастосунок «QAHelper», який використовується інженерами із забезпечення якості програмного забезпечення та мануальними тестувальниками для зручного та правильного створення тестової документації.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox);

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– нефункціональне тестування – полягає у перевірці властивостей, що не пов’язані з функціональним тестуванням та вимірюються різними величинами;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;

– тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на пристроях з різною роздільною здатністю екрану;

– тестування на пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування інтерфейсу користувача;

– тестування зручності використання;

– тестування локалізації.

Тестування виконується статично з використанням інструментів, що містять в собі метрики необхідні для перевірки програми і документації на відповідність дотримання стандартів.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Керівництво користувача

КПІ.ІП-0224.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Анастасія ЛИТВИН

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«QAHelper» - це вебзастосунок для створення тестової документації з урахуванням правил її оформлення. Програмне забезпечення надає механізм авторизації, реєстрації, створення проєктів, тестових наборів, тестових випадків та звітів про дефекти, а також виконання прогонів тестового набору.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішного локального розгортання даного застосунку необхідне виконання наступних вимог:

– наявність ПК з типом процесора Intel Core i5 / AMD Ryzen 5 або вище (рекомендований тип процесору Intel Core i7) та(або) мобільного пристрою;

– обсяг оперативної пам’яті не менше 8 ГБ (рекомендований об’єм оперативної пам’яті 16 ГБ);

– наявність доступу до Інтернету зі швидкістю від 50 мегабіт (рекомендована швидкість Інтернету від 100 мегабіт);

– наявність сучасного браузеру на пристрої;

– наявність встановленої платформи Docker.

2.2 Завантаження застосунку

На даний момент вебзастосунок можна встановити власноруч, використовуючи docker-compose.yml, що наданий в пояснювальні записці. Для цього необхідні в командному рядку виконати команду «docker compose up».

2.3 Перевірка коректної роботи

По завершенню встановлення додатка користувач має переконатись, що контейнери з виконаного файлу успішно запустились, звернувши увагу на результат виконання команди в командному рядку. Якщо все гаразд, то відкрити браузер та перейти на сторінку http://localhost:4200, користувачу відкривається сторінка авторизації.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску програмного забезпечення користувачу буде відображено сторінку авторизації за допомогою якої він може авторизуватись у свій акаунт(рис. 3.1).

Рисунок 3.1 – Сторінка авторизації вебзастосунку

Користувач має змогу відкрити сторінку реєстрації та зареєструватись в вебзастосунку (рис. 3.2).

Рисунок 3.2 – Сторінка реєстрації вебзастосунку

Користувач має змогу після авторизації відкрити сторінку «Панель користувача» (рис. 3.3).

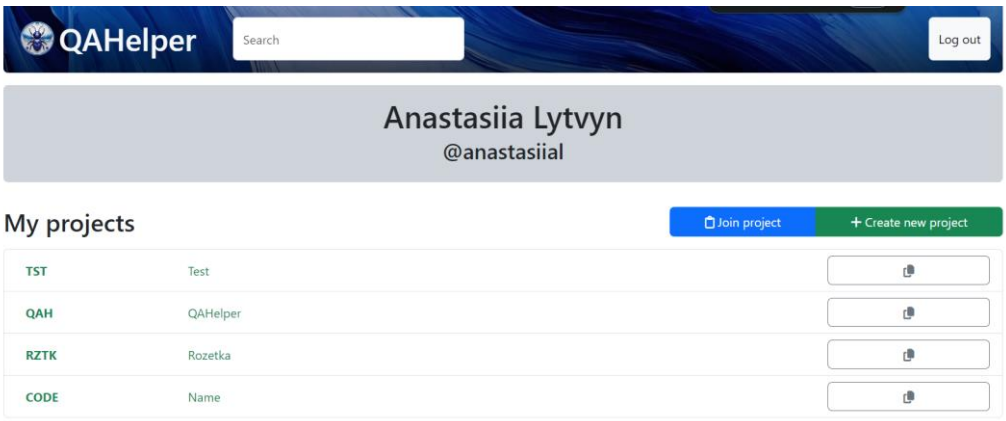


Рисунок 3.3 – Сторінка «Панель користувача» вебзастосунку

Користувач має змогу створити новий проєкт за допомогою форми створення проєкту (рис. 3.4).

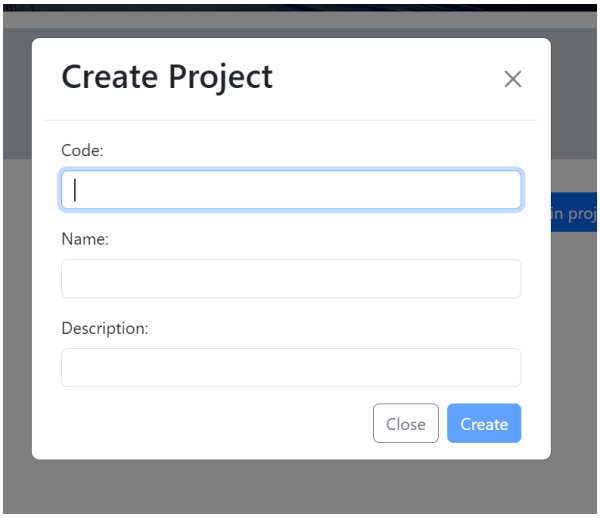


Рисунок 3.4 – Форма створення проєкту вебзастосунку

Користувач має змогу доєднується до існуючого проєкту за допомогою форми доєднання до проєкту, натиснувши на сторінці «Панель користувача» на кнопку «Доєднатись до проєкту», відкривається модальне вікно, кнопка «Доєднатись» неактивна (рис. 3.5).

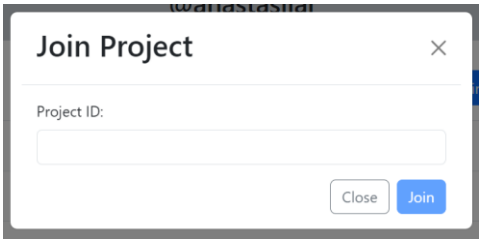


Рисунок 3.5 – Форма доєднання до проєкту вебзастосунку

Користувач має змогу відкрити сторінку проєкта (рис. 3.6).

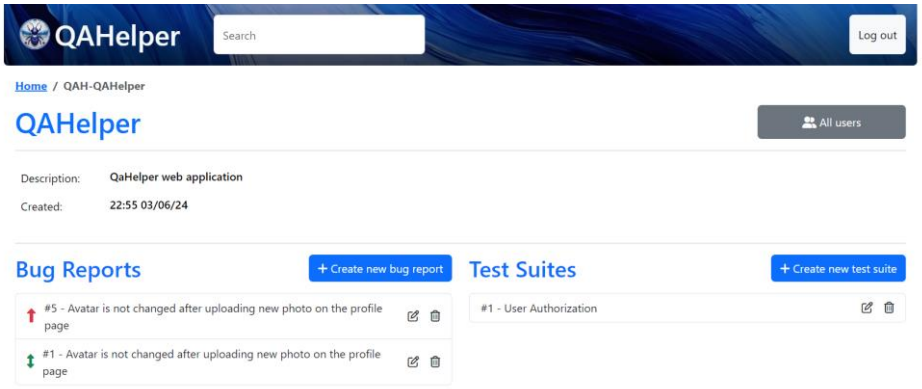


Рисунок 3.6 – Сторінка проєкту вебзастосунку

Користувач має змогу переглянути учасників проєкту (рис. 3.7).

Рисунок 3.7– Сторінка «Користувачі» проєкту вебзастосунку

Користувач має змогу створити тестовий набір за допомогою форми створення тестового набору (рис. 3.8).

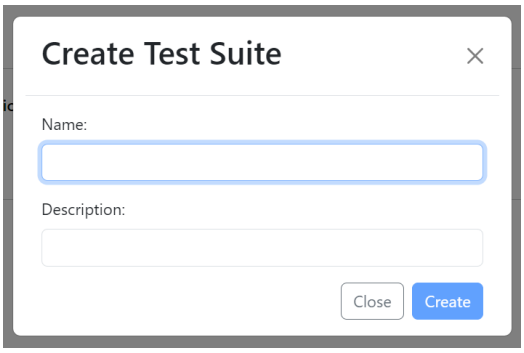


Рисунок 3.8 – Форма створення тестового набору

Користувач має змогу редагувати тестовий набір за допомогою форми редагування тестового набору (рис. 3.9), та видалити його, натиснувши відповідну кнопку напроти назви тестового набору на сторінці проєкту (рис. 3.10).

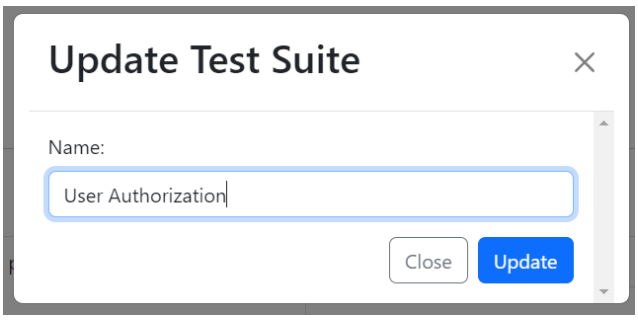


Рисунок 3.9 – Форма редагування тестового набору

Рисунок 3.10 – Кнопки редагування та видалення тестового набору

Користувач має змогу відкрити сторінку тестового набору (рис. 3.11).

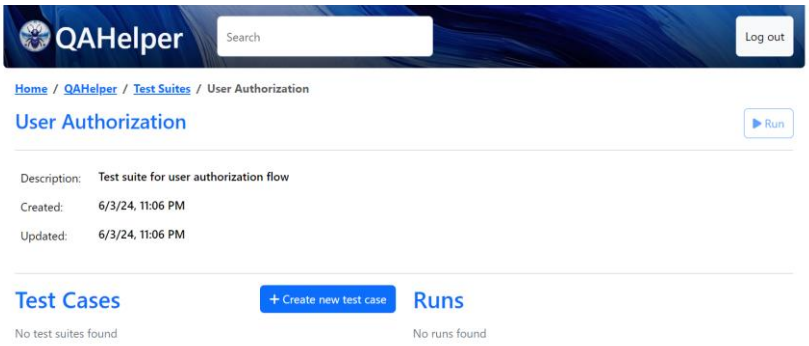


Рисунок 3.11 – Сторінка тестового набору

Користувач має змогу створити тестовий випадок за допомогою форми створення на сторінці тестового набору (рис. 3.12).

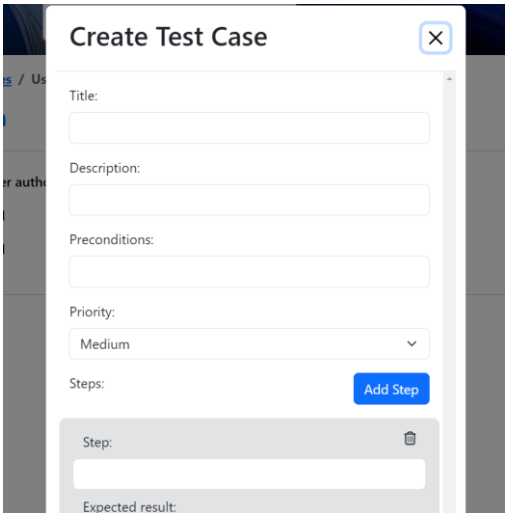


Рисунок 3.12 – Форма створення тестового випадку

Користувач має змогу переглянути сторінку тестового випадку (рис. 3.13).

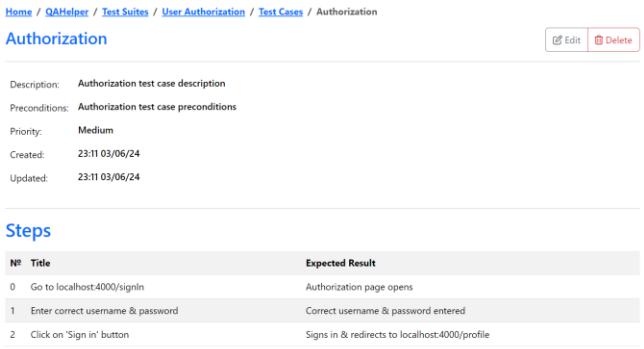


Рисунок 3.13 – Сторінка тестового випадку

Користувач має змогу редагувати тестовий випадок за допомогою форми редагування тестового випадку (рис. 3.14), та видалити його, натиснувши відповідну кнопку напроти назви тестового випадку на сторінці тестового випадку (рис. 3.15).

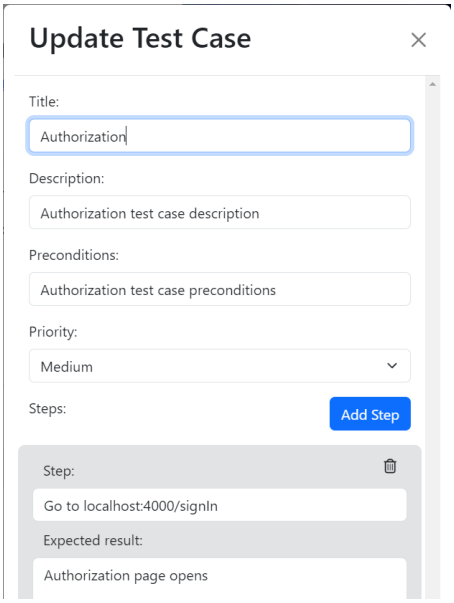


Рисунок 3.14 – Форма редагування тестового випадку

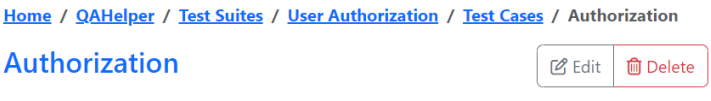


Рисунок 3.15 – Кнопки редагування та видалення тестового випадку

Користувач має змогу виконати прогін тестового набору за допомогою кнопки «Запустити» (рис. 3.16).

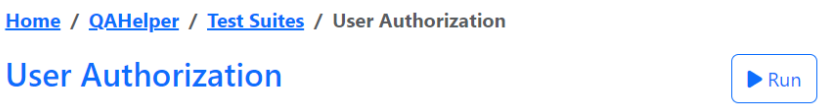


Рисунок 3.16 – Кнопка запуску прогону тестового набору

Користувач має змогу переглянути початий прогін тестового набору (рис 3.17).

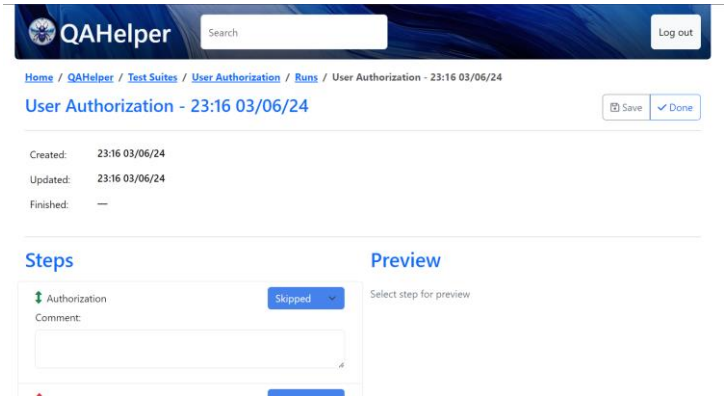


Рисунок 3.17 – Сторінка прогону тестового набору

Користувач має змогу попереднього переглянути вміст кожного тестового випадку в даному прогоні, натиснувши на тестовий випадок (рис. 3.18).

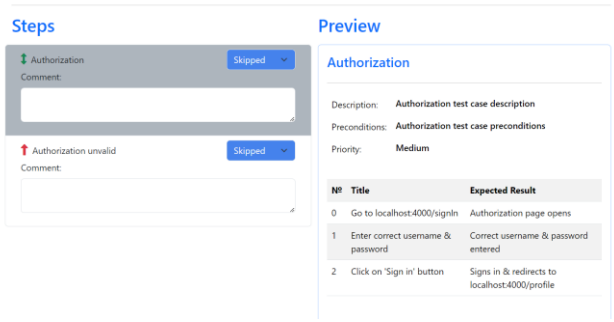


Рисунок 3.18 – Вигляд попереднього перегляду тестового випадку на сторінці прогону тестового набору

Користувач має змогу обрати відповідний статус для даного тестового випадку та додати коментар (рис. 3.19).

Рисунок 3.19 – Зміна статусу та додання коментарю до тестового набору в даному прогоні

Користувач має змогу зберегти та завершити прогін за допомогою відповідних кнопок (рис. 3.20).

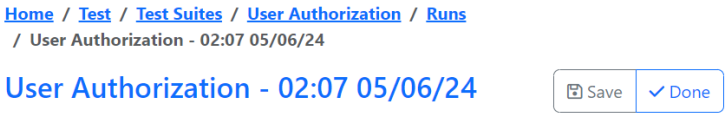


Рисунок 3.20 – Кнопки збереження та завершення прогону

Користувач має змогу переглянути завершений прогін (рис. 3.21–3.22).

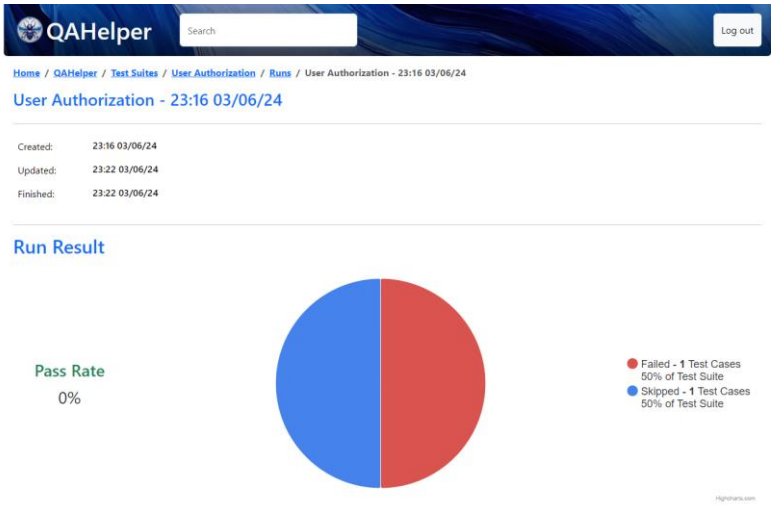


Рисунок 3.21 – Аналітика виконаного прогону

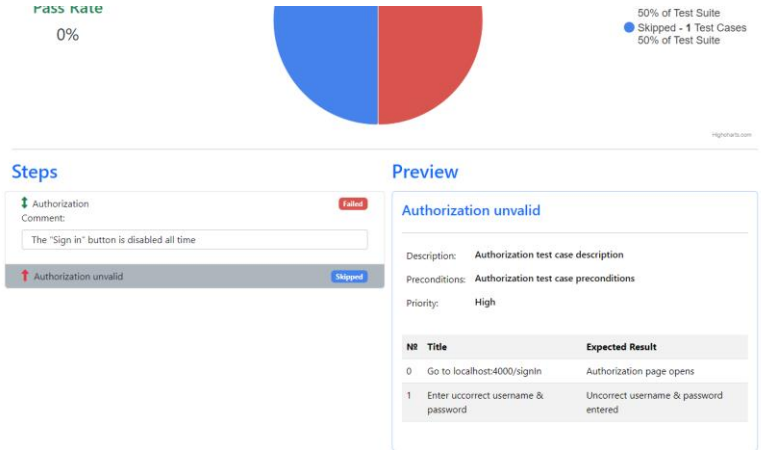


Рисунок 3.22 – Результати виконаного прогону

Користувач має змогу створити звіт про дефект за допомогою форми створення(рис. 3.23).

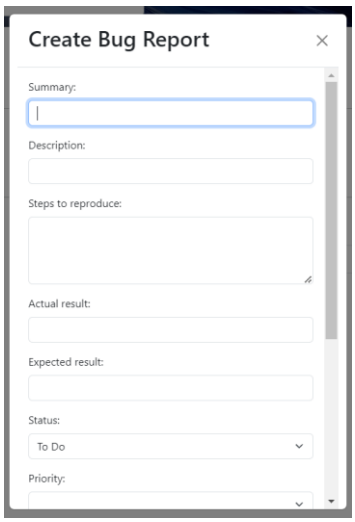


Рисунок 3.23 – Форма створення звіту про дефект

Користувач має змогу редагувати звіт про дефект за допомогою форми редагування тестового набору (рис. 3.24), та видалити його, натиснувши відповідну кнопку напроти назви звіту про дефект на сторінці проєкту (рис. 3.25).

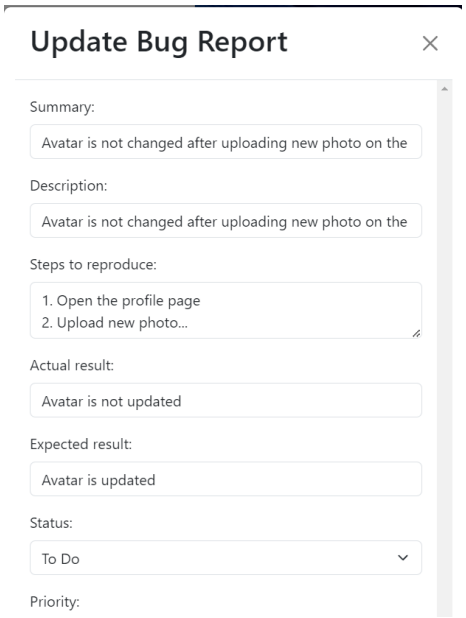


Рисунок 3.24 – Форма редагування звіту про дефект

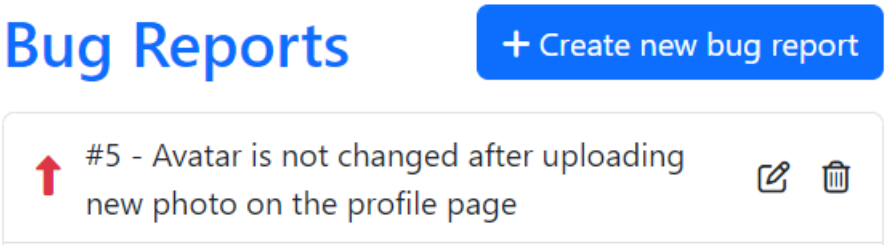


Рисунок 3.25 – Кнопки редагування та видалення звіту про дефект

Користувач має змогу відкрити сторінку звіту про дефект (рис. 3.26).

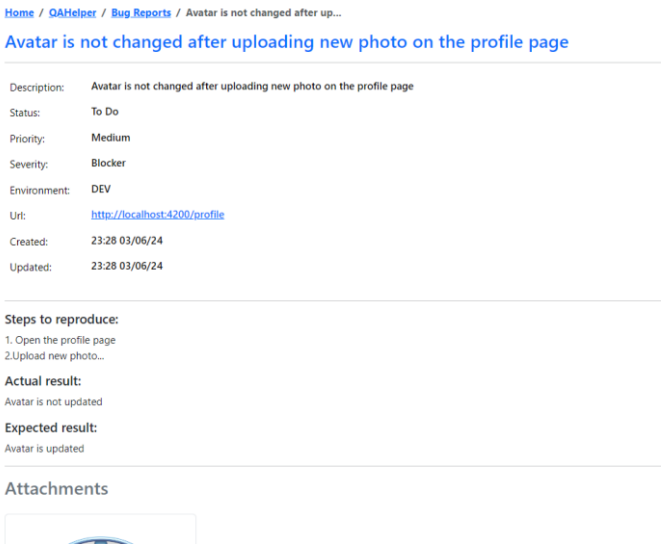


Рисунок 3.26 – Сторінка звіту про дефект

Користувач має змогу знайти за назвою необхідний проєкт, тестовий набір, тестовий випадок або звіт про дефект за допомогою поля пошуку (рис. 3.27).

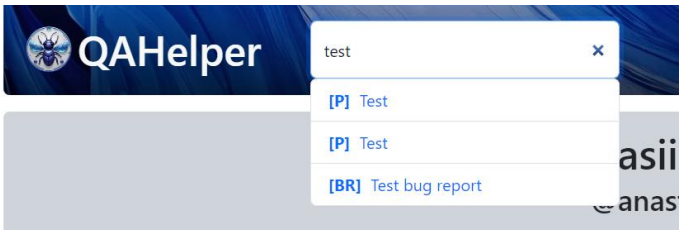


Рисунок 3.27 – Пошук за назвою

Користувач має змогу вийти з акаунту за допомогою кнопки «Вийти» в заголовку сторінки (рис. 3.28).

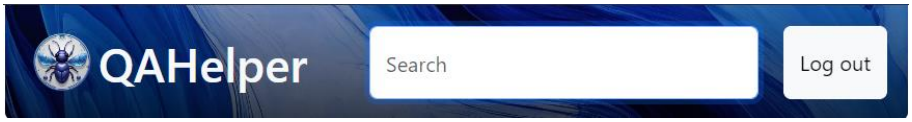


Рисунок 3.28 – Розташування кнопки виходу з акаунту в заголовку сторінки

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ СТВОРЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Графічний матеріал

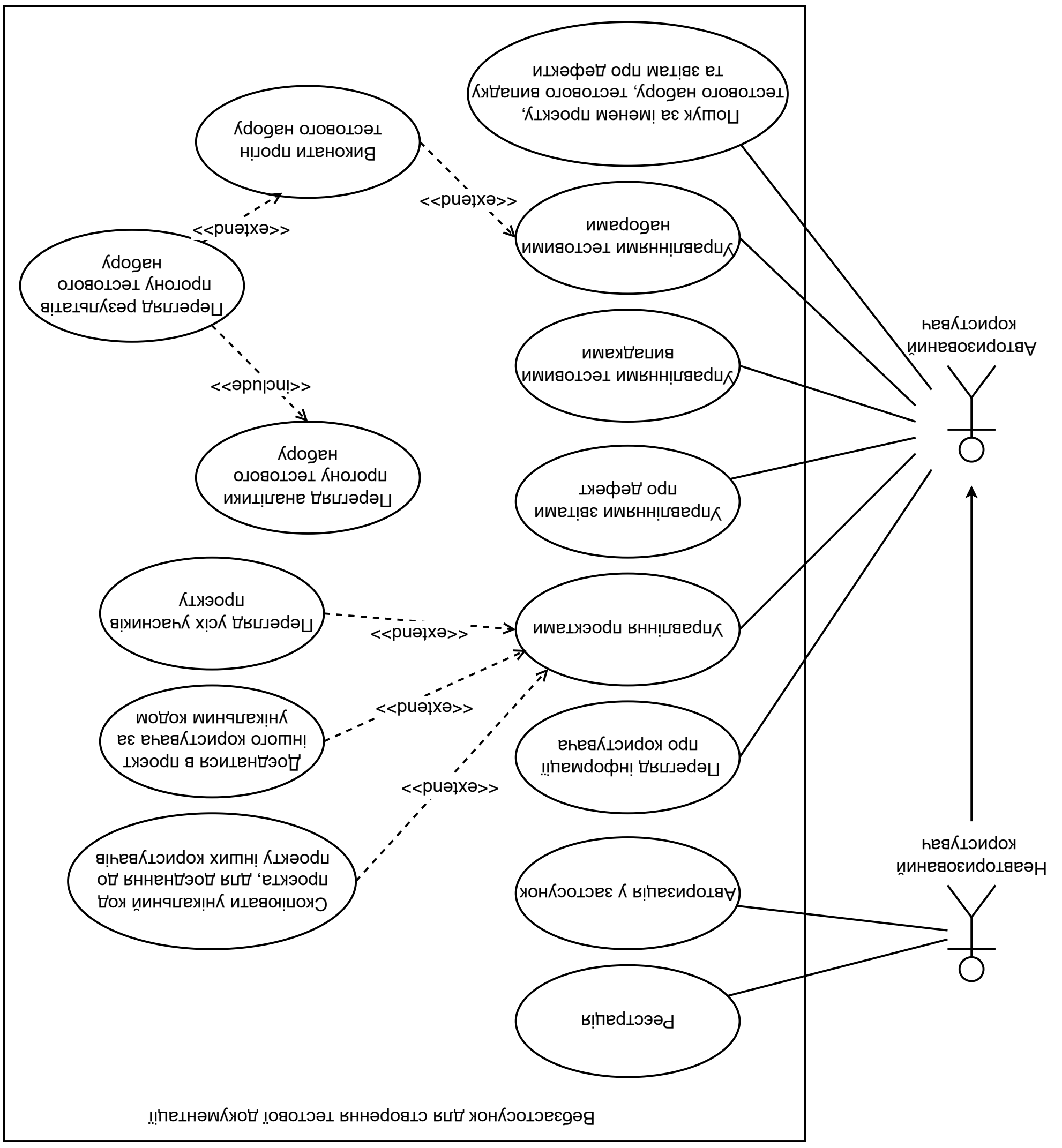
КПІ.ІП-0224.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Анастасія ЛИТВИН

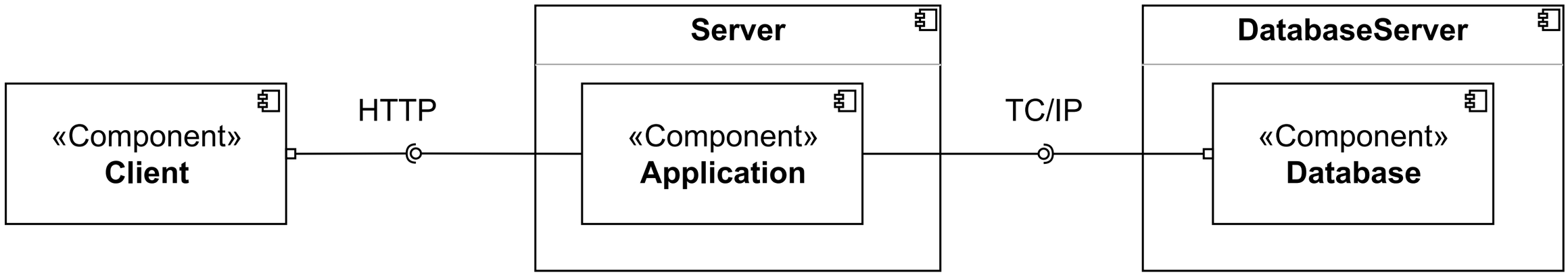
Київ – 2024



КПІ.ІП-0224.045440.06.99.ССВ

3м.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-0224.045440.06.99.ССВ				
Розробив		Литвин А.В.							
Перевірив		Лихоузова Т.А.			Схема структура варіантів використань				
Т. контр.									
Н. контр.		Шупькевич Т.В.							
Затвердив		Жаріков Е.В.			Вебзастосунок для створення тестової документації				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02
						Літера	Маса	Масштаб	
						Аркуш 1		Аркушів 4	

КПІ.ІП-0224.045440.06.99.ССМ



					КПІ.ІП-0224.045440.06.99.ССМ		
					Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Литвин А.В.				Аркуш 2	Аркушів 4
Перевірив		Ліхоузова Т.А.			Вебзастосунок для створення тестової документації	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02	
Т. контр.							
Н. контр.		Шулькевич Т.В.					
Затвердив		Жаріков Е.В.					

attachments

PK	**id** nanoid **NOT NULL**	
	name varchar NOT NULL	
FK	size integer NOT NULL	
	bug_report_id integer NOT NULL	
	extension varchar NOT NULL	
	mime_type varchar NOT NULL	
	created_at timestamp NOT NULL	

bug_reports

PK	**id** integer **NOT NULL**	
	summary varchar NOT NULL	
	description varchar NOT NULL	
	steps_to_reproduce varchar NOT NULL	
	expected_result varchar NOT NULL	
	actual_result varchar NOT NULL	
	status varchar NOT NULL	
	priority varchar NOT NULL	
	severity varchar NOT NULL	
	environment varchar NOT NULL	
	url varchar NOT NULL	
FK	project_id uuid NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

test_suites

PK	**id** integer **NOT NULL**	
	name varchar NOT NULL	
	description varchar	
FK	project_id uuid NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

runs

PK	**id** integer **NOT NULL**	
FK	test_suite_id integer NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	
	finished_at timestamp	

projects

PK	**id** uuid **NOT NULL**	
	name varchar NOT NULL	
	description varchar NOT NULL	
	code varchar NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

users_projects

PK, FK	project_id uuid NOT NULL	
PK, FK	user_id uuid NOT NULL	

users

PK	**id** uuid **NOT NULL**	
	name varchar NOT NULL	
	username varchar NOT NULL	
	password_hash varchar NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

test_cases

PK	**id** integer **NOT NULL**	
	title varchar NOT NULL	
	description varchar NOT NULL	
FK	test_suite_id integer NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	
	priority varchar NOT NULL	
	preconditions varchar	

run_steps

PK, FK	run_id integer NOT NULL	
PK, FK	test_case_id integer NOT NULL	
	status varchar NOT NULL	
	name varchar NOT NULL	
	comment varchar	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

test_case_steps

PK	**id** integer **NOT NULL**	
	title varchar NOT NULL	
	expected_result varchar NOT NULL	
FK	test_case_id integer NOT NULL	
	created_at timestamp NOT NULL	
	updated_at timestamp NOT NULL	

КПІ.ІП-0224.045440.06.99.СБД

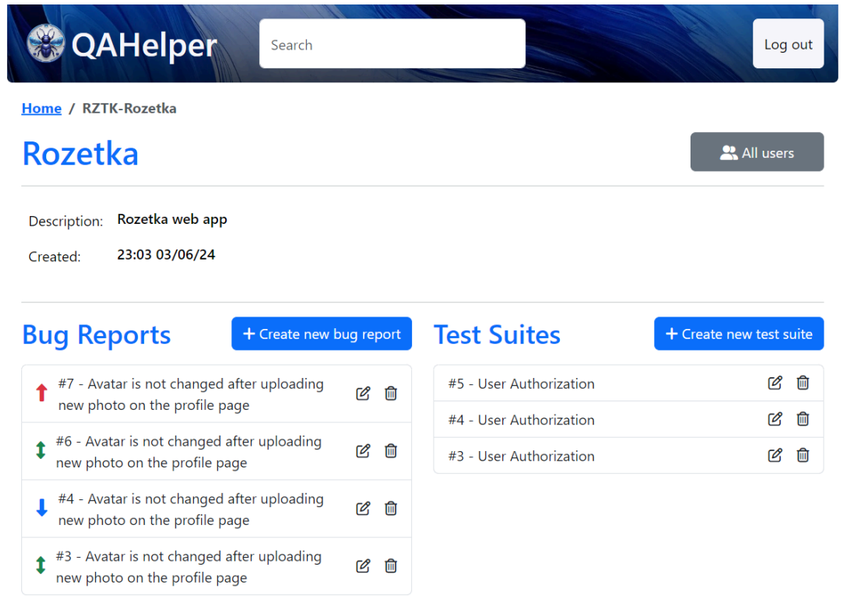
Зм.	Арк.	№ докумен та	Підпис	Дата	КПІ.ІП-0224.045440.06.99.СБД
Розробив		Литвин А.В.			
Перевірив		Лихоузова Т.А.			Схема бази даних
Т. контр.					
Н. контр.		Шулькевич Т.В.			
Затвердив		Жарікова Е.В.			

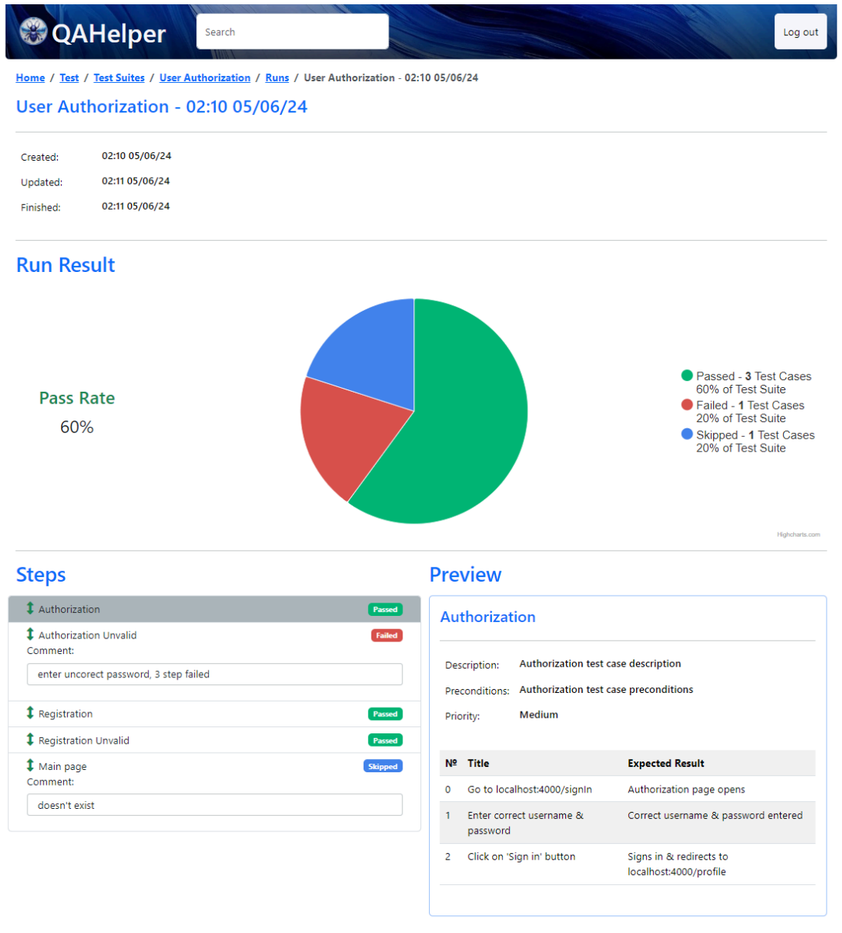
Літера	Маса	Масштаб
Аркуш 3		Аркушів 4

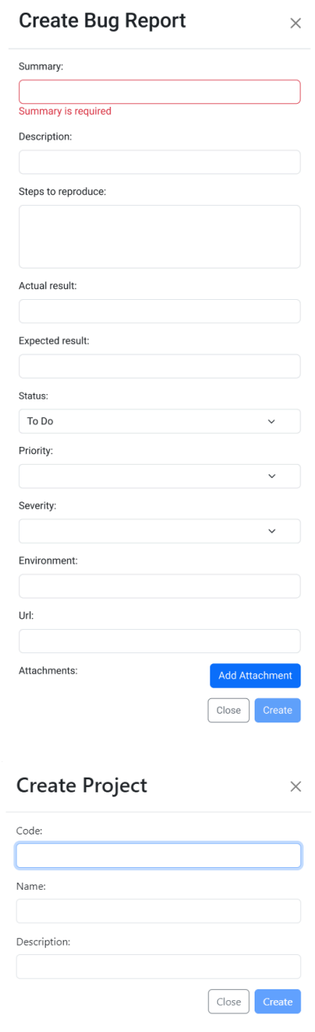
Відомість посилань для створення тестової документації

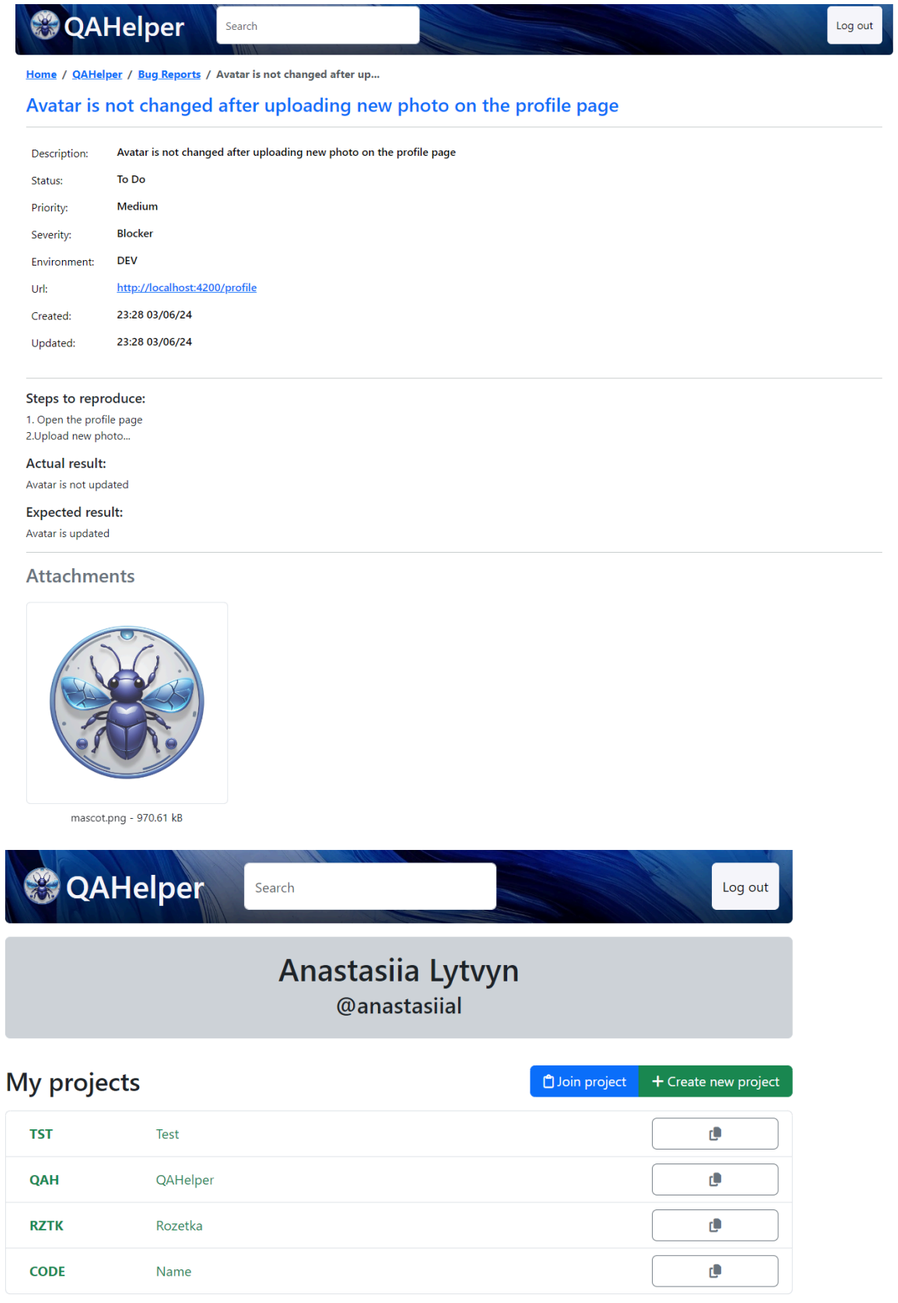
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-02

КПІ.ІП-0224.045440.06.99.КЕ









					КПІ.ІП-0224.045440.06.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Литвин А.В.				
Перевірив		Ліхоузова Т.А.				Аркуш 4 / Аркушів 4
Т. контр.						
					Вебзастосунок для створення тестової документації	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02
Н. контр.		Шулькевич Т.В.				
Затвердив		Жаріков Е.В.				