МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

центр післядипломної освіти

Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший(бакалаврський)

Програмна система відстеження рівня якості програмного забезпечення у процесі розробки
(тема)

Виконав:
студент 4-го курсу, групи ПЗПІп-22-2
Садихов Т.І.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник к.т.н. доц.Кириченко І.В.
(посада, прізвище, ініціали)

Допускається до захисту:

Зав. кафедри _____ З.В. Дудар
(підпис)

2024р.

Харківський національний університет радіоелектроніки

центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Садихову Тимуру Ілхамовичу*
(прізвище, ім'я, по батькові))

1. Тема роботи: Програмна система відстеження рівня якості програмного забезпечення у процесі розробки
затверджена наказом університету від «17» *червня* 2024 р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії «24» *липня* 2024 р.
3. Вихідні дані до проекту: *у програмній системі передбачити: операції з даними які стосуються проєктів, тестових випадків, вимог, коду.. Використовувати ОС Windows 10, СРБД Microsoft SQL Server 2019, JavaScript, Visual Studio Code.*
4. Перелік питань, що потрібно опрацювати у роботі *аналіз предметної галузі, огляд існуючих рішень, концептуальне моделювання предметної області, варіанти використання, архітектура системи, структура даних, описання програмної реалізації системи.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін	Відмітки про виконання
1	Інструктаж з безпеки життєдіяльності	18.06.2024 р.	*виконано*
2	Аналіз предметної області	18.06 – 21.06	*виконано*
3	Розробка постановки задачі	22.06 – 23.06	*виконано*
4	Формування вимог до ПЗ	24.06 – 28.06	*виконано*
5	UML проектування	29.06 – 01.07	*виконано*
6	Проектування архітектури ПЗ	02.07 – 03.07	*виконано*
7	Розробка структур зберігання даних	04.07 – 05.07	*виконано*
8	Проектування прототипів інтерфейсів	06.07 – 07.07	*виконано*
9	Підготовка звіту з роботи	08.07 – 10.06	*виконано*
10	Захист роботи	24.06	

Дата видачі завдання «*17*» *червня* 2024 р.

Студент _____ Тимур Садихов

(підпис)

Керівник роботи _____ доц Кириченко І.В.

(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до звіту з кваліфікаційної роботи, стор.54, рис.8, джерел 9.

ПРОГРАМНА СИСТЕМА, ПРОГРАМНИЙ ЗАСТОСУНОК, JAVASCRIPT, SQL, РЕЛЯЦІЙНІ БАЗИ ДАНИХ

Об'єкт дослідження: організаційні та бізнес процеси підвищення якості розробляємого програмного забезпечення, що потребує автоматизації.

Мета розробки - створення програмної системи, яка забезпечить можливість ефективно проводити аналіз якості програми на стадії розробки та відстежувати якість програмного забезпечення.

Метод рішення - використання Visual Studio Code для розробки програмного забезпечення для настільних комп'ютерів. В якості системи управління базами даних обрано Microsoft SQL Server 2022.

Результати розробки - розроблена програмна система, яка дозволяє додавати, змінювати та видаляти інформацію в базі даних, а також формувати необхідні документи внутрішнього формату підприємства.

ІНФОРМАЦІЙНА СИСТЕМА ВІДСТЕЖЕННЯ ЯКОСТІ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, JAVASCRIPT, NODEJS, SQL

Object of research: Organizational and business processes for improving the quality of software under development that require automation.

Development goal: Creation of software system that enables efficient program quality analysis during the development stage and tracks software quality.

Solution Method: Use Utilizing Visual Studio Code for developing desktop software. Microsoft SQL Server 2022 was chosen as the database management system.

Development results: The developed software system that allows adding, modifying, and deleting information in the database, as well as generating necessary internal documents for the enterprise.

Я, Садихов Тимур Ілхамович, студент гр. ПЗПІп-22-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система відстеження рівня якості програмного забезпечення у процесі розробки», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному

плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Актуальність та важливість цієї роботи визначаються потребою спеціалістів у галузі програмування в контролі якості програмного забезпечення та підвищення якості наданих послуг замовникам. Розробка додатку "Програмна система відстеження рівня якості програмного забезпечення у процесі розробки" є рішенням яке задовільнить потребу в автоматизації процесів контролю якості в IT галузі.

Використання реляційних баз даних допоможе зберігати та упорядковувати велику кількість інформації про код, знайдені помилки, та інші аспекти роботи із програмним забезпеченням. Об'єктно-орієнтоване програмування забезпечує створення модульної та гнучкої системи, яка легко масштабується і розвивається разом із зростанням бізнесу.

Цей підхід до розробки програмного забезпечення сприятиме оптимізації робочих процесів, підвищенню продуктивності персоналу та забезпеченню високого рівня якості створюваних продуктів. В результаті, програма відстеження якості стане ключовим інструментом для покращення конкурентоспроможності програмного засоба на ринку.

У ході створення додатку були використані середа розробки Visual Studio Code, мова програмування JavaScript, фреймворки та білбіотеки React JS та NodeJS, та система управління базами даних Microsoft SQL Server 2022.

У сучасному світі, де цифровізація затронула повсякденне життя більшості громадян планети, кожного дня випускаються сотні програм, якими користуються тисячі користувачів. Якість програмного забезпечення стала одною із найважливійших складових частин успіху будь-якого сучасного бізнесу, оскільки вона безпосередньо впливає на ефективність роботи, задоволення клієнтів та конкурентоспроможність на ринку. Саме тому програмна система, яка надає можливість відстежувати якість програмного забезпечення має велике значення та актуальність, оскільки надає можливість відстежувати стан якості програмного забезпечення та підвищувати якість за допомогою аналізу зібраних даних. В

результаті, спеціаліст або компанія, яка створює програмне забезпечення, зможе швидше ідентифікувати проблеми із якістю продукту, ідентифікувати та виправити можливі помилки, та вивести на ринок продукт високої якості, який задовільнить користувачів та їх потреби.

Впровадження допоміжних засобів контролю якості для пришвидшення відгуку про створений продукт, або його частину, є невід'ємною складовою успіху в сучасному середовищі розробки програмних засобів. Програмна система, яка оптимізує створення програм, та забезпечує збереження даних про знайдені помилки, вимоги до програмного забезпечення, дозволяє компаніям підвищити швидкість створення якісного програмного застосунку, та підвищити рівень задоволення клієнтів завдяки ідентифікації та вирішення критичних проблем на стадії розробки.

Точність обробки даних гарантується відсутністю потреби втручань людини в процесі обробки. Обробка навіть невеликої кількості даних людиною несе в собі не тільки зниження швидкості, але й підвищує ризик помилки через людський фактор. Така помилка може мати серйозні наслідки, адже невідповідна якість продукту підвищує ризик невдоволення клієнта, а також може сприяти прямій втраті коштів клієнтів, або іншим формам збитків. Впровадження баз даних дозволяє ефективно вирішити проблему, забезпечуючи надійне сховище даних, та точність даних, наданих для обробки та аналізу. Бази даних дозволяють автоматизувати процес збереження інформації, а також дозволяють уникнути проблем, пов'язаних їх дублюванням, або помилковим введенням даних.

Зниження вартості та підвищення продуктивності програмних застосунків. Ціна виправлення помилки у програмному застосунку зростає на кожному етапі розробки, це в свою чергу приводить до того, що функціонал, який здатен відстежувати якість програмного забезпечення у процесі розробки програмного забезпечення має потенціал у підвищенні скорості розробки та зниженні ризиків, які виникають через виникнення помилок у програмному забезпеченні. Завдяки автоматизації процесів, які до цього відіймали цінний час спеціалістів, рутинні задачі будуть займати менше часу, а також виключать людський фактор у процесі

виконання. Окрім того, можливість аналізу коду одразу після завершення написання нового модулю або функціоналу, значно підвищує продуктивність спеціалістів, які користуються автоматизованим рішенням. Такі інструменти є важливою частиною сучасного бізнесу в сфері інформаційних технологій, адже вони дозволяють оптимізувати свої ресурсу та підвищувати якість наданих послуг.

Для сфери інформаційних технологій, де якість, швидкість, та гнучкість є ключовими факторами успіху, важливо мати належні засоби для оптимізації процесів. Це є важливою складовою успіху через велику кількість факторів, серед яких висока ціна спеціалістів, висока конкуренція, та рівень очікувань клієнтів від програмних застосунків.

Використання сучасних технологій значно підвищує продуктивність і якість роботи. Інтеграція інструментів розробки програмного забезпечення, платформ програмування та систем управління базами даних є важливим кроком у покращенні робочих процесів.

Спеціалізовані програмні рішення дозволяють автоматизувати рутинні операції та забезпечують точність обробки даних. Це зменшує ймовірність помилок і дозволяє зосередитися на творчих аспектах діяльності.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Аналіз предметної галузі

Компанії, які надають послуги з написання програмного забезпечення, а також поодинокі програмісти потребують ефективний засіб для аналізу якості програмного забезпечення у процесі його розробки. Система має допомагати у аналізі рівня якості програмного забезпечення, надавати інформацію щодо поточного статусу якості згідно з метриками якості.

Система матиме функціональні вимоги зазначені нижче.

Зберігання інформації про помилки, знайдені в ході тестування, включаючи кроки відтворення, очікуваний та актуальний результати, пріоритет, серйозність помилки.

Зберігання тестових випадків, включаючи кроки відтворення, очікуваний та актуальний результати, пріоритет, та до якого функціоналу відноситься даний випадок.

Аналіз якості коду за визначеними параметрами якості.

Можливість класифікації та фільтрації тестових випадків для швидкої фільтрації та пошуку.

Автоматизована генерація звітів про кількість знайдених помилок, якість коду, та інші важливі метрики.

До нефункціональних вимог можна віднести наступні вимоги.

Зручність інтерфейсу для користувача та простий доступ до інформації

Безпека та надійність даних, яка містить захист від несанкціонованого доступу та резервне копіювання даних.

Швидкість системи та продуктивність при різних обсягах навантаження.

Також треба зазначити наступні умови впровадження.

Запланована дата введення системи в експлуатацію.

Наявність потрібних ресурсів для встановлення та підтримки системи.

Проведення тренінгів для персоналу щодо роботи з новою системою.

Програма дає змогу сформувати тренд аналізу якості із часом. Вона також дозволяє налаштовувати інтерфейс і рівні доступу до даних відповідно до ролі користувача.

Наразі існує багато застосунків, які дозволяють відстежувати якість програмного застосунку у процесі розробки. Нижче наведено декілька аналогів програмної системи, які можуть використовуватись для аналізу якості програмного застосунку.

Jira: Jira – це потужний інструмент для управління проєктами та відстеження завдань, розроблений компанією Atlassian. Він широко використовується командами розробників для планування, відстеження та випуску програмного забезпечення, надаючи можливості для управління спринтами, створення звітів, налаштування робочих процесів та інтеграції з іншими інструментами.

Bugzilla – це система відстеження помилок з відкритим кодом, створена Mozilla Foundation. Вона дозволяє командам ефективно відстежувати помилки, контролювати їх виправлення та керувати змінами в програмному забезпеченні, надаючи інструменти для сортування, фільтрації та пошуку помилок.

TestRail – це комплексний інструмент для управління тестуванням, розроблений компанією Gurock. Він допомагає командам розробників планувати, організовувати та відстежувати тестові сценарії, забезпечуючи централізоване місце для зберігання тестових результатів, генерації звітів та аналітики процесу тестування.

QTest – це хмарна платформа для управління тестуванням програмного забезпечення, створена компанією QASymphony. Вона забезпечує командам можливість планувати, виконувати та відстежувати тестові випадки, інтегруючись з іншими DevOps інструментами для забезпечення безперервної інтеграції.

PractiTest[4] – це сучасний інструмент для управління процесом тестування, який дозволяє командам легко організовувати, виконувати та відстежувати тестові сценарії. Він підтримує інтеграцію з багатьма інструментами розробки та управління проєктами, а також надає можливість створювати детальні звіти та аналітику.

SonarQube – це інструмент для безперервного аналізу якості коду з відкритим кодом, який підтримує різні мови програмування. Він допомагає командам розробників знаходити та виправляти вразливості, помилки та проблеми з підтримкою коду, забезпечуючи поліпшення якості та безпеки програмного забезпечення.

Кожна з цих систем має свої унікальні можливості та переваги, та можуть бути використані для специфічних потреб при створенні програмного забезпечення.

Найбільш відповідає тематиці даного проєкту – «PractiTest»

PractiTest є сучасною платформою для управління процесом тестування, що набуває все більшої популярності серед IT-компаній у всьому світі. Її широкі можливості дозволяють ефективно організовувати, виконувати та відстежувати тестові сценарії, що робить її незамінним інструментом для забезпечення якості програмного забезпечення. Однією з ключових переваг PractiTest є можливість інтеграції з багатьма іншими інструментами розробки та управління проєктами, що дозволяє гнучко налаштовувати систему під конкретні потреби команди. Платформа підтримує інтеграцію з такими популярними системами як JIRA, GitHub, Selenium та іншими, що значно спрощує процес управління тестуванням[6].

Іншою важливою перевагою PractiTest є її здатність генерувати детальні звіти та аналітику, які допомагають командам краще розуміти стан тестування та приймати обґрунтовані рішення. Платформа пропонує зручний інтерфейс для створення тестових сценаріїв, управління вимогами та записами помилок, що робить процес тестування більш прозорим та керованим. Крім того, PractiTest має розвинену систему користувацької підтримки та документації, що дозволяє швидко вирішувати питання та проблеми.

Однак, важливо зазначити деякі недоліки PractiTest. Одним з них є необхідність часу та зусиль для первинного налаштування системи, особливо для великих команд з складною структурою тестування. Крім того, для максимально

ефективного використання всіх можливостей платформи може знадобитися додаткове навчання персоналу.

Попри ці недоліки, PractiTest залишається одним з найпопулярніших та найефективніших рішень для управління тестуванням у сучасних умовах. Її гнучкість, широкі функціональні можливості та підтримка інтеграцій з іншими інструментами роблять її привабливим вибором для багатьох IT-компаній.

За допомогою PractiTest команди можуть ефективно контролювати процес тестування, відстежувати помилки та вимоги, а також аналізувати результати тестування для покращення якості програмного забезпечення.

1.2 Виявлення та вирішення проблем

Існує велика кількість компаній, що займаються розробкою програмного забезпечення, а також незалежних програмістів, які потребують відстеження якості програмних продуктів у процесі їх створення. Ці компанії можуть варіюватися від великих корпорацій зі спеціалізованими відділами контролю якості до невеликих студій або окремих розробників, які виконують всі етапи розробки самостійно. Наявність великої кількості проєктів і замовників з різними вимогами до якості програмного забезпечення створює необхідність в ефективних інструментах та методах для відстеження і контролю якості.

Незалежно від розміру компанії або індивідуального розробника, ефективне управління процесом тестування і контролю якості є критично важливим для успіху проєкту. Без належного відстеження якості програмного забезпечення виникають ризики пропуску критичних помилок, що може призвести до збоїв у роботі програмного продукту та негативно вплинути на репутацію компанії або розробника.

Наприклад, у 1999 році космічний апарат NASA Mars Climate Orbiter зазнав катастрофи через програмну помилку, що коштувала 327,6 мільйона доларів. Причиною провалу стала невідповідність між британською системою мір (фунти-сили) та метричною системою (ньютон-секунди), яку використовували різні

команди розробників. Це спричинило помилку в розрахунках траєкторії, і апарат згорів у атмосфері Марсу. Цей інцидент підкреслив важливість узгодженості та ретельного тестування програмного забезпечення.

Існує безліч інструментів для управління якістю програмного забезпечення, які допомагають виявляти та виправляти помилки на ранніх стадіях розробки. Використання таких інструментів дозволяє забезпечити високу якість продукту і зменшити час на його тестування і підтримку. Як фахівець з багаторічним досвідом у цій в сфері контролю якості, я знаходжу цю тему актуальною та цікавою.

1.3 Постановка задачи

Загальна мета кваліфікаційної роботи полягає у розробці програмної системи, призначеної для відстеження рівня якості програмного забезпечення у процесі розробки. Конкретні завдання охоплюють:

- розробку зручного та інтуїтивно зрозумілого інтерфейсу, із врахуванням що дозволить користувачам легко взаємодіяти з системою;
- реалізацію функціонала для вводу та зберігання інформації про помилки, які виникли в процесі тестування, кроки відтворення, очікуваний та реальний результати, пріоритет та серйозність;
- створення можливості класифікації та категоризації завдань для швидкого пошуку та фільтрації;
- розробку автоматизованих засобів генерації звітів про стан якості програми чи окремого функціоналу програмного засобу;
- забезпечення високої надійності та безпеки даних, включаючи захист від несанкціонованого доступу та резервне копіювання;
- інтеграцію з наявними системами управління, що дозволить ефективно обмінюватися даними та спільно використовувати ресурси.

Ці завдання спрямовані на створення системи, яка допоможе покращити продуктивність та ефективність роботи розробників програмного забезпечення,

спростить процеси тестування і відстеження помилок, а також підвищить рівень задоволеності клієнтів-користувачів програмного засобу.

Для досягнення цих цілей планується застосування передових технологій та методів розробки програмного забезпечення. Зокрема, для створення бізнес-логіки та користувацького інтерфейсу буде використано JavaScript[1] у поєднанні з NodeJS та React JS. Для управління базами даних вибрано Microsoft SQL Server[3], який забезпечить надійний і швидкий доступ до даних. Для моделювання архітектури системи та визначення взаємозв'язків між її компонентами будуть використовуватись UML-діаграми. Крім того, передбачається проведення всебічного тестування програмного забезпечення, включаючи фунцціональне, системне, інтеграційне та тестування в умовах реальної експлуатації, щоб забезпечити його високу якість.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розпочнемо формування вимог до програмної системи із функціональних вимог:

Деталізований перелік функцій програмного забезпечення, включаючи всі операції та завдання, які воно має виконувати. Визначення здатності системи аналізувати якість коду.

Опис можливостей для користувача, таких як створення та перегляд тестових випадків, генерація звітів, налаштування користувачів та інші взаємодії.

Введення тестових випадків та знайдених помилок. Система повинна надавати можливість користувачам вводити тестових випадків та знайдених помилок в системі шляхом заповнення відповідних форм.

Зберігання інформації про замовлення. Після введення тестових випадків та знайдених помилок, система повинна зберігати всю необхідну інформацію про нього, включаючи технічні деталі, користувача, що зберіг цю інформацію, дату та час реєстрації, статус тощо.

Пошук та фільтрація замовлень. Користувачам має бути надано інформацію за рядом параметрів, такими як пріоритет, статус, функіонал якого стосується цей запис тощо, а також застосовувати фільтри для точного визначення потрібної інформації.

Генерація звітів. Система повинна мати можливість генерувати звіти щодо якості програмного застосунку згідно з обраними метрик якості, такі як кількість відкритих помилок у застосунку за пріоритетом, якість коду, покриття тестами тощо, які допоможуть створити якісне програмне забезпечення для користувачів.

Керування користувачами. Адміністраторам системи має бути надана можливість змінювати права доступу користувачів, створювати нові облікові записи, блокувати або видаляти користувачів за необхідності.

Ці функціональні вимоги окреслюють ключові можливості програмної системи та її функції для задоволення потреб користувачів і досягнення цілей кваліфікаційної роботи.

Нефункціональними вимогами є опис характеристик системи, включаючи продуктивність, безпеку, надійність, доступність та інші важливі аспекти.

Продуктивність. Система має забезпечувати швидке оброблення та відповідь на запити користувачів навіть при високому обсязі даних і навантаженні на сервер.

Безпека. Гарантування високого рівня захисту даних за допомогою шифрування, аутентифікації користувачів та контролю доступу до конфіденційної інформації.

Надійність. Система повинна бути стійкою до збоїв і забезпечувати безперервну роботу навіть у разі непередбачених ситуацій, таких як відключення електроенергії або програмні помилки.

Доступність. Забезпечення доступу до системи для користувачів у будь-який час і з будь-якого місця, включаючи можливість роботи як в онлайн, так і в офлайн режимах.

Інтерфейс користувача. Створення зручного і привабливого інтерфейсу, що забезпечує легке та ефективне використання системи і є приємним для користувачів.

Сумісність. Забезпечення сумісності програмного забезпечення з різними операційними системами, браузерами та пристроями, що дозволяє користувачам працювати з системою з будь-якого пристрою за їх вибором.

Опис вимог до користувацького інтерфейсу, включаючи зручність у використанні, привабливий дизайн та сумісність з різними пристроями та розширеннями.

Також треба урахувати специфічні потреби користувачів, такі як можливість колективної роботи кількох користувачів над одним завданням.

Масштабованість. Система має бути здатною до масштабування, щоб відповідати зростаючим потребам бізнесу та обсягам даних, забезпечуючи ефективну роботу у майбутньому.

Міжплатформенність. Програмне забезпечення повинно підтримувати різні браузери, такі як Chrome, Safari, та Edge Browser. Також додаток має бути

доступним з мобільних засобів, та підтримуватись Google Chrome for mobile та Safari mobile веббраузерами.

Система резервного копіювання. Впровадження автоматичної системи резервного копіювання даних, що забезпечить збереження інформації та запобігання втратам у разі аварійних ситуацій або випадкового видалення.

Підтримка багаторівневої архітектури. Розробка програмного забезпечення з використанням багаторівневої архітектури, яка розділяє логіку бізнес-процесів, користувацький інтерфейс та роботу з базою даних, що покращує модульність та масштабованість системи.

Інтеграція з існуючими системами. Система повинна мати здатність інтегруватися з уже використовуваними на підприємстві системами управління, такими як системи управління складом, обліку замовлень та фінансові програми.

Доступність. Система має бути розроблена таким чином, щоб надавати можливість людям із вагами слуху, зору тощо, умови для комфортної праці із системою.

Ці вимоги враховують особливості контролю якості програмних застосунків, та гарантують високий рівень функціональності, ефективності та надійності програмного забезпечення.

Програмне рішення включає два основних компоненти.

Перший компонент – це база даних, яка створена за допомогою системи управління базами даних (СУБД) і використовує мову програмування SQL.

SQL бази данних мають наступні переваги.

Стандартизація: SQL (Structured Query Language) є уніфікованою мовою для роботи з реляційними базами даних, що дозволяє здійснювати розробку та управління базами даних незалежно від конкретної СУБД.

Продуктивність: SQL бази даних відрізняються високою ефективністю і швидкістю обробки запитів, завдяки оптимізації, використанню індексів та інших методів для прискорення вибірки, сортування та об'єднання даних.

Зручність використання: Інформаційна система буде інтегрована з сервером баз даних Microsoft SQL Server, який відомий своєю надійністю, швидкістю та

багатим набором функцій. Багато розробників мають досвід роботи з SQL Server, що спрощує процес розробки та підтримки системи. Інтерфейс і інструменти адміністрування SQL Server зручні та інтуїтивно зрозумілі, що сприяє швидкому освоєнню та ефективному використанню цієї технології.

Складні запити: SQL дозволяє виконувати складні запити до бази даних, такі як з'єднання, групування, фільтрація, сортування та агрегація, що робить його потужним інструментом для аналізу даних.

Безпека: SQL бази даних надають високий рівень безпеки, забезпечуючи налаштування доступу для користувачів і ролей, шифрування даних і контроль цілісності бази даних.

Масштабування: SQL бази даних можуть бути легко масштабовані для задоволення зростаючих потреб організації, підтримуючи додавання нових таблиць, індексів і реплікацію даних.

Резервне копіювання та відновлення: SQL бази даних мають вбудовані механізми для резервного копіювання та відновлення даних, що дозволяє захистити інформацію і відновити її у разі помилок або втрат.

Друга частина – це графічний інтерфейс. Додаток на основі ReactJS та NodeJS має кілька переваг:

Легкість розробки: ReactJS[2] надає розробникам простий і зрозумілий інструментарій для створення компонентів інтерфейсу користувача. Використання JSX дозволяє легко поєднувати HTML з JavaScript, що спрощує процес розробки і підтримки коду.

Широкий набір компонентів: ReactJS пропонує великий вибір готових компонентів, які можна використовувати для побудови інтерактивного інтерфейсу користувача. Це дозволяє швидко створювати різноманітні форми, таблиці та інші елементи інтерфейсу, що задовольняють потреби користувачів.

Висока продуктивність: ReactJS використовує віртуальний DOM, що значно підвищує продуктивність додатків шляхом оптимізації оновлення інтерфейсу. Це дозволяє створювати швидкі та оптимізовані інтерфейси навіть при великому навантаженні.

Інтеграція з NodeJS: Використання NodeJS на серверній стороні забезпечує можливість створення повноцінних вебдодатків. NodeJS дозволяє обробляти запити до бази даних, здійснювати аутентифікацію користувачів та інші серверні функції, що робить розробку додатків більш гнучкою та потужною.

Підтримка подій - ReactJS надає зручний механізм обробки подій, що дозволяє легко реагувати на взаємодію користувачів з додатком. Це включає обробку кліків, введення даних та інших дій користувачів, що дозволяє створювати інтуїтивно зрозумілий інтерфейс.

Кросплатформеність - ReactJS дозволяє створювати додатки, які можуть працювати на різних платформах, включаючи веб, мобільні пристрої (з використанням React Native) та інші середовища. Це робить ReactJS ідеальним вибором для розробки кросплатформенних додатків.

Активна спільнота - ReactJS має велику і активну спільноту розробників, що забезпечує доступ до великої кількості ресурсів, бібліотек, документації та форумів, які допомагають розробникам у процесі створення додатків.

Ці переваги роблять ReactJS і NodeJS популярним вибором для розробки сучасних вебдодатків з графічним інтерфейсом. Використовуючи вебінтерфейс, ми забезпечуємо зручність взаємодії з програмою.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У вебдодатку будуть присутні основні функціональні модулі для авторизації, створення записів у базі даних, та формування звітів тощо. Вебдодаток на ReactJS включатиме кілька ключових компонентів, таких як керування користувачами, управління дизайнами, звітність тощо.

Компоненти авторизації та управління користувачами забезпечуватимуть аутентифікацію користувачів та передачу інформації до інших компонентів. Ці компоненти включатимуть функції для:

- додавання та управління інформацією про користувачів;
- розподілу завдань і обов'язків між користувачами;
- управління можливостями та ролями користувачів.

Компонент управління дизайнами дозволятиме створювати, редагувати та видаляти дизайни. Він також буде взаємодіяти з базою даних для збереження та отримання даних, забезпечуючи зручний і ефективний інтерфейс для роботи з дизайнами.

Компонент звітності та інтеграції з іншими відділами дозволить генерувати звіти та технічні специфікації, необхідні для технологічного процесу підприємства. Цей компонент забезпечить ефективну комунікацію між різними відділами, використовуючи переваги ReactJS для створення інтерактивних та динамічних звітів.

Створення архітектури та проєктування програмного забезпечення є важливими етапами розробки інформаційної системи. Вони визначають структуру додатка, взаємодію між компонентами та способи їх реалізації. У цьому розділі розглядаються ключові аспекти архітектури та проєктування програмного забезпечення для системи відстеження рівня якості програмного забезпечення у процесі розробки.

3.1 Архітектурні підходи

При розробці архітектури системи була використона багаторівнева архітектура, яка складається з кількох основних рівнів:

- рівень клієнта (Presentation layer);
- рівень бізнес-логіки;
- рівень даних (Data access layer);
- рівень інтеграції.

3.1.1 Рівень клієнта (Presentation layer)

Цей рівень включає все, що користувач бачить і взаємодіє з ним у браузері. Він відповідає за рендеринг інтерфейсу користувача, обробку користувацьких подій і відображення даних, отриманих з серверу. Основні технології, що використовуються на цьому рівні, включають HTML, CSS, JavaScript[5], а також фронтенд-фреймворки та бібліотеки, такі як ReactJS, Angular або Vue.js.

ReactJS є однією з найпопулярніших бібліотек для створення користувацьких інтерфейсів завдяки своїй модульній архітектурі та ефективному рендерингу компонентів. Використання ReactJS дозволяє створювати компоненти, які можуть бути повторно використані у різних частинах програми, що сприяє зменшенню дублювання коду та підвищенню його підтримуваності.

Основні компоненти рівня клієнта включають компонент App.js який виступає як головний компонент додатку, що містить маршрутизацію та компоненти навігації, а також компоненти для керування користувачами такі як Login, Users, що забезпечують функції аутентифікації та управління користувачами. В свою чергу для управління проектами, вимогами, тестовими випадками та звітністю були створені такі компоненти як Projects, Requirements, TestCases, Dashboards.

3.1.2 Рівень бізнес-логіки

Цей рівень відповідає за обробку бізнес-правил і логіки додатку. Він приймає запити від рівня клієнта, обробляє їх, взаємодіє з базою даних і повертає відповідні дані назад на клієнтський рівень. Для даного проєкту була обрана Node.js.

Основними технологіями, які використовуються на цьому рівні, є Express.js для створення веб-серверу та обробки HTTP-запитів та Middleware для аутентифікації, авторизації та обробки запитів.

Основні компоненти рівня бізнес-логіки включають:

Маршрути (routes): Вони відповідають за прийом HTTP-запитів, їх обробку та передачу даних до відповідних контролерів. В цьому проекті використовуються наступні маршрути: userRoutes, projectRoutes, requirementRoutes, testCaseRoutes, issueRoutes, dashboardRoutes, exportRoutes.

Контролери містять логіку обробки запитів та взаємодії з моделями. Вони отримують запити від маршрутів, виконують необхідні бізнес-операції та повертають результати назад маршрутам.

Моделі (models): представляють структуру даних та містять методи для взаємодії з базою даних. Вони відповідають за визначення схем таблиць бази даних, встановлення відношень між таблицями та виконання CRUD-операцій (створення, читання, оновлення, видалення).

3.1.3 Рівень доступу до даних

Цей рівень відповідає за доступ до даних, їх збереження та обробку. Він включає бази даних, системи управління базами даних (СУБД) та інші сховища даних. Основні технології, що використовуються на цьому рівні, включають реляційні та нереляційні бази даних. Для цього проєкту обрана реляційна база даних MSSQL

Реляційна база даних MSSQL забезпечує надійне зберігання даних та підтримку складних запитів за допомогою мови SQL. Основні елементи рівня доступу до даних включають:

- таблиці: Для зберігання структурованої інформації про користувачів, проекти, вимоги, тестові випадки, проблеми та результати аналізу коду;
- відношення: Встановлюють зв’язки між різними таблицями, забезпечуючи цілісність даних і можливість виконання складних запитів.
- індекси: Використовуються для оптимізації продуктивності запитів, забезпечуючи швидкий доступ до даних;
- тригери: Автоматизують певні операції в базі даних, наприклад, при додаванні або зміні записів;
- збережені процедури: Використовуються для виконання складних бізнес-логічних операцій безпосередньо на рівні бази даних, що підвищує продуктивність і безпеку системи.

3.1.4 Рівень даних (Data access layer)

Цей рівень забезпечує взаємодію між різними системами та сервісами, такими як API, мікросервіси або зовнішні сервіси. Він включає в себе інтерфейси для інтеграції з іншими системами і може включати middleware для обробки повідомлень, маршрутизації запитів і трансформації даних. Технології, що використовуються на цьому рівні, включають RESTful API, GraphQL, gRPC, та інші інтеграційні фреймворки.

3.2 Проєктування архітектури

Проєктування архітектури включає створення діаграм, що ілюструють структуру системи та взаємодію між її компонентами. Для цього використовуються діаграми UML.

3.2.1 Діаграма класів

Діаграма класів (див.рис. 3.1) відображає основні класи системи та їх взаємозв'язки. Основні класи включають:

– клас User для зберігання інформації про користувачів;
– клас Issue для зберігання інформації про знайдені дефекти;
– клас Report для зберігання інформації щодо створення звітів;
– клас Metric для зберігання інформації про метрік якості програмного застосунку.

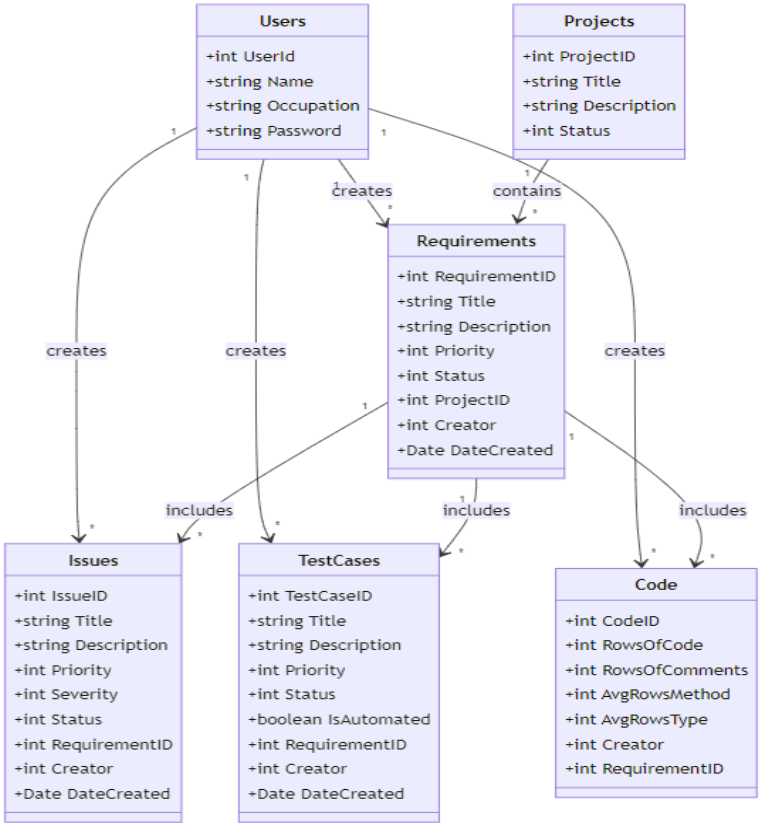


Рисунок 3.1 – Діаграма класів застосунку

3.2.2 Діаграма компонентів

Діаграма компонентів (див.рис. 3.2) відображає основні компоненти системи та їх взаємодію. Основні компоненти включають:

– компонент рівня презентації;
– компонент серверної частини додатку;
– компонент доступу до даних;
– компонент інтеграції.

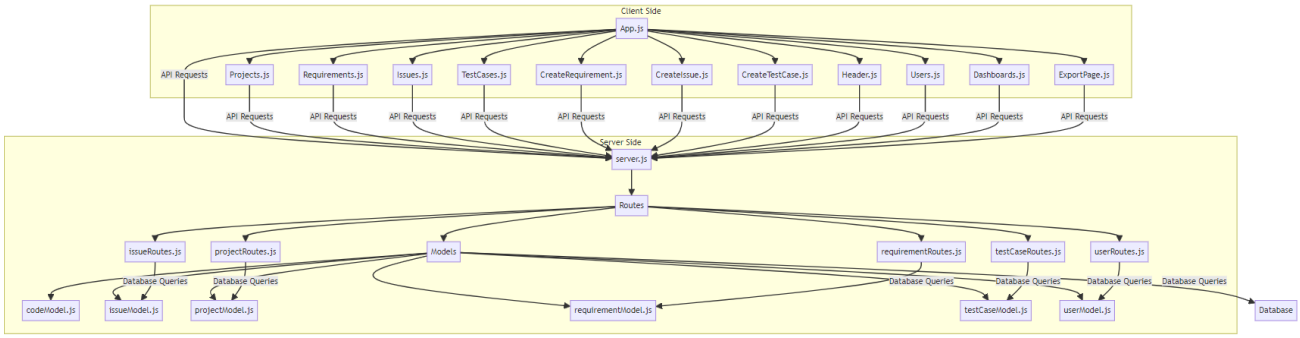


Рисунок 3.2 – Діаграма компонентів застосунку

3.3 Проєктування структури зберігання даних

Проєктування структури зберігання даних включає розробку схеми бази даних[7]. Схема бази (див.рис. 3.3) даних включає:

– таблиці для зберігання інформації про користувачів, знайдені дефекти, тестові випадки, результати аналізу коду;
– зв'язки між таблицями;
– індекси для оптимізації запитів;
– тригери та процедури для забезпечення цілісності даних.

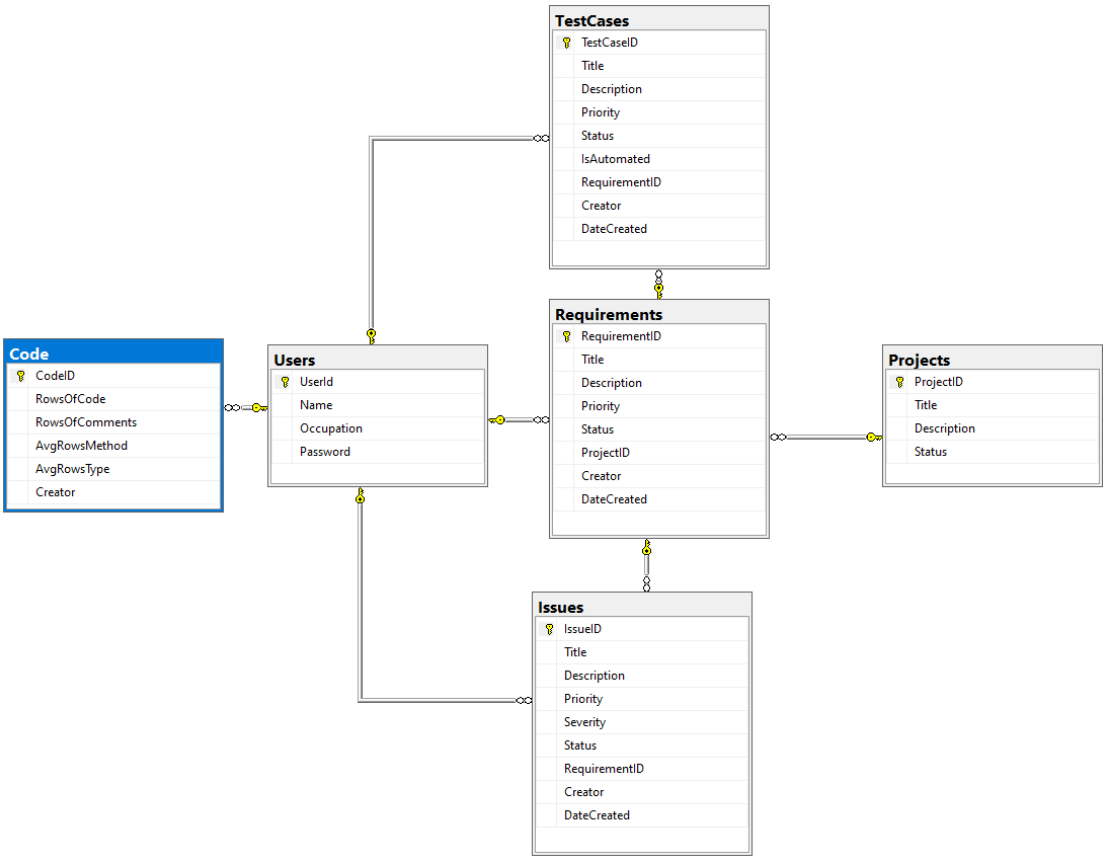


Рисунок 3.3 – Схема бази даних

3.4 Створення UI/UX

Проєктування користувацького інтерфейсу включає:

– розробку макетів основних форм програми;

– використання принципів зручності та інтуїтивності інтерфейсу;

– забезпечення доступності та ергономіки інтерфейсу.

Таким чином, архітектура (див.рис. 3.4) та проєктування програмного забезпечення забезпечують надійну основу для розробки ефективної та масштабованої системи, що відповідає вимогам замовника та забезпечує зручність використання.

Рисунок 3.4 – Use-case діаграма застосунку

Користувач може увійти в систему за допомогою логіну та пароля. У головному меню вибрати один із напрямків робот, яка може виконуватись в системі. Робота може включати в себе створення та перегляд тестових випадків, створення та перегляд записів про знайдені помилки, та завантажити код для аналізу системою, а також сформувати звіти за обраними метриками.

Адміністратор може увійти у систему за допомогою логіну та пароля. У головному меню адміністратор може обрати один із напрямків робот. Робота може включати в себе надання, редагування, та видалення прав користувачам, а також створення проєктів в системі.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Програмне рішення для зберігання даних

Реляційна база даних є системою зберігання та організації доступу до взаємопов'язаних елементів інформації. Ця база даних заснована на реляційній моделі, яка представляє інформацію у вигляді таблиць. Кожен рядок у таблиці має унікальний ідентифікатор - ключ, а стовпці містять атрибути даних, що дозволяє встановлювати зв'язки між елементами даних. Реляційна модель бази даних є найбільш ефективною для вирішення задач цього курсового проекту.

Нормальна форма є вимогою до структури таблиць у теорії реляційних баз даних, яка спрямована на усунення зайвих функціональних залежностей між атрибутами. У даному проекті під час нормалізації таблиці були приведені до третьої нормальної форми. Відношення перебуває у третій нормальній формі, коли воно знаходиться у першій та другій нормальній формі, тобто всі атрибути є атомарними, кожен неключовий атрибут повністю функціонально залежить від первинного ключа відношення, і між неключовими атрибутами немає транзитивних залежностей.

У всіх відношеннях всі атрибути є атомарними, тому вони знаходяться в першій нормальній формі.

При зведенні до другої нормальної форми було доведено, що всі відношення не містять неповних функціональних залежностей, тобто в складі потенційного ключа відсутня менша підмножина атрибутів, від якої можна також вивести дану функціональну залежність.

При зведенні до третьої нормальної форми була проведена перевірка на відсутність транзитивних функціональних залежностей.

4.2 Програмне рішення для створення серверної частини

Серверна частина розробленого проекту побудована на основі фреймворку Express для Node.js і використовує Microsoft SQL Server для зберігання даних. У цьому розділі детально розглянуто основні кроки створення серверної частини, включаючи налаштування середовища розробки, встановлення необхідних пакетів, а також реалізацію основних функціональних модулів.

Для початку розробки необхідно підготувати робоче середовище, встановивши Node.js та npm (Node Package Manager). Це дозволяє управляти залежностями проекту та запускати сервер[9]. Завантаження та встановлення Node.js здійснюється з офіційного сайту Node.js, після чого перевіряється коректність встановлення за допомогою команд «node -v» та «npm -v».

4.2.1 Ініціалізація проекту

Після встановлення Node.js, створюється новий проект за допомогою команди npm init -y, яка автоматично створює файл package.json з базовими налаштуваннями проекту. Далі, для керування залежностями та створення основних файлів проекту, необхідно встановити Express та інші необхідні пакети за допомогою командної строки Microsoft Windows cmd:

```
npm install express
npm install mssql
npm install dotenv
npm install body-parser
```

4.2.2 Налаштування бази даних

Проект використовує Microsoft SQL Server для зберігання даних. Для підключення до бази даних використовується пакет mssql. Налаштування підключення до бази даних зберігаються у файлі .env, що дозволяє захистити конфіденційну інформацію. Приклад конфігураційного файлу dbConfig.js виглядає наступним чином:

```
// dbConfig.js
module.exports = {
    user: 'TymurS',
    password: 'T24S02I90f@',
    server: 'localhost\\SQLExpress',
    database: 'ProjectTrackingSystem',
    options: {
        encrypt: false, // Use this if you're on Windows Azure
        enableArithAbort: true
       }
};
```

4.2.3 Реалізація основних модулів

Основний серверний файл server.js відповідає за налаштування базового функціоналу сервера. Він забезпечує підключення до бази даних, налаштування проміжного програмного забезпечення (middleware), та ініціалізацію всіх необхідних маршрутів для обробки запитів. У цьому файлі задаються основні параметри сервера, такі як порт, на якому він буде слухати запити, та підключення до бази даних.

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const sql = require('mssql');
const dbConfig = require('./dbConfig');
const app = express();
const port = process.env.PORT || 3001;
app.use(bodyParser.json());
app.use(cors());
sql.connect(dbConfig, err => {
```

```
 if (err) {
    console.error('Database connection failed: ', err);
     return;
  }
     console.log('Connected to the database');
     });
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
const projectRoutes = require('./routes/projectRoutes');
const requirementRoutes = require('./routes/requirementRoutes');
const testCaseRoutes = require('./routes/testCaseRoutes');
const issueRoutes = require('./routes/issueRoutes');
const codeRoutes = require('./routes/codeRoutes');
const userRoutes = require('./routes/userRoutes');
const dashboardRoutes = require('./routes/dashboardRoutes');
const exportRoutes = require('./routes/exportRoutes'); // Import the
export routes
app.use('/api/projects', projectRoutes);
app.use('/api/requirements', requirementRoutes);
app.use('/api/testcases', testCaseRoutes);
app.use('/api/issues', issueRoutes);
app.use('/api/code', codeRoutes);
app.use('/api/users', userRoutes);
app.use('/api/dashboard', dashboardRoutes);
app.use('/api/export', exportRoutes); // Use the export routes
```

Маршрутизація в проекті реалізується через окремі файли для кожної сутності, такі як користувачі, проекти, вимоги тощо. Наприклад, файл routes/userRoutes.js містить маршрути для обробки CRUD операцій з користувачами. Цей файл визначає маршрути для отримання списку користувачів, додавання нового користувача, оновлення інформації про користувача та видалення користувача. Такий підхід забезпечує модульність та зручність у підтримці коду.

```
// routes/userRoutes.js
const express = require('express');
const router = express.Router();
const userModel = require('../models/userModel');
router.get('/', async (req, res) => {
    const users = await userModel.getAllUsers();
    res.json(users);
});
router.get('/:id', async (req, res) => {
    const user = await userModel.getUserById(req.params.id);
    res.json(user);
});
router.post('/', async (req, res) => {
    const result = await userModel.createUser(req.body);
    res.json({ rowsAffected: result });
```

```
});
router.put('/:id', async (req, res) => {
    const result = await userModel.updateUser(req.params.id,
req.body);
    res.json({ rowsAffected: result });
});
router.delete('/:id', async (req, res) => {
    const result = await userModel.deleteUser(req.params.id);
    res.json({ rowsAffected: result });
});
module.exports = router;
```

Моделі даних у проекті представляють структуру таблиць у базі даних та містять методи для взаємодії з цими таблицями. Наприклад, модель models/userModel.js містить функції для отримання списку користувачів, пошуку користувачів за ідентифікатором, створення нового користувача, оновлення існуючого та видалення користувача. Використання моделей даних дозволяє абстрагуватися від безпосереднього взаємодії з базою даних, забезпечуючи чітке розділення логіки обробки даних та бізнес-логіки.

```
// models/userModel.js
const sql = require('mssql');
const dbConfig = require('../dbConfig');
const getAllUsers = async () => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().query('SELECT * FROM
Users');
        return result.recordset;
    } catch (err) {
        console.error('Error fetching users:', err);
    }
};
const getUserById = async (id) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().input('id', sql.Int,
id).query('SELECT * FROM Users WHERE UserID = @id');
        return result.recordset[0];
    } catch (err) {
        console.error('Error fetching user by id:', err);
    }
};
const createUser = async (data) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request()
            .input('Name', sql.NVarChar, data.Name)
            .input('Occupation', sql.NVarChar, data.Occupation)
            .input('Password', sql.NVarChar, data.Password)
```

```
            .query('INSERT INTO Users (Name, Occupation, Password)
VALUES (@Name, @Occupation, @Password)');
        return result.rowsAffected;
    } catch (err) {
        console.error('Error creating user:', err);
    }
};
const updateUser = async (id, data) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request()
            .input('id', sql.Int, id)
            .input('Name', sql.NVarChar, data.Name)
            .input('Occupation', sql.NVarChar, data.Occupation)
            .input('Password', sql.NVarChar, data.Password)
            .query('UPDATE Users SET Name = @Name, Occupation =
@Occupation, Password = @Password WHERE UserID = @id');
        return result.rowsAffected;
    } catch (err) {
        console.error('Error updating user:', err);
    }
};
const deleteUser = async (id) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().input('id', sql.Int,
id).query('DELETE FROM Users WHERE UserID = @id');
        return result.rowsAffected;
    } catch (err) {
        console.error('Error deleting user:', err);
    }
};
module.exports = {
    getAllUsers,
    getUserById,
    createUser,
    updateUser,
    deleteUser
};
```

Таким чином, процес створення серверної частини включає налаштування середовища розробки, встановлення необхідних пакетів, реалізацію моделей даних та маршрутизації. Використання Express для Node.js забезпечує гнучкість та зручність у розробці, а Microsoft SQL Server надає надійну систему для зберігання та управління даними.

4.3 Програмне рішення для створення клієнтської частини

4.3.1 Ініціалізація проекту

Клієнтська частина розробленого проекту побудована на основі бібліотеки React для JavaScript[8] і використовує ряд сучасних інструментів та технологій для забезпечення інтерактивності та зручності у користуванні. У цьому розділі детально розглянуто основні кроки створення клієнтської частини, включаючи налаштування середовища розробки, встановлення необхідних пакетів, а також реалізацію основних функціональних модулів.

Після встановлення Node.js, створюється новий проект за допомогою команди npx create-react-app client[6], яка автоматично створює структуру директорій і файлів для React додатка. Далі необхідно встановити додаткові пакети для управління станом, маршрутизації та HTTP-запитів:

```
npm install react-router-dom axios
```

Клієнтська частина проекту організована у вигляді компонентів, що знаходяться у директорії src/components. Кожен компонент відповідає за окрему частину функціональності додатка.

4.3.2 Реалізація основних компонентів

Кожен компонент реалізує окремий функціональний блок додатка, забезпечуючи взаємодію з користувачем та сервером. Розглянемо основні компоненти та їх функціональні можливості.

Компонент Requirements.js відповідає за відображення списку вимог для обраного проекту. Використовуючи axios, він здійснює HTTP-запити до сервера для отримання даних про вимоги, пов'язані з конкретним проектом, і відображає їх у вигляді списку. Кожна вимога представлена як посилання, що дозволяє перейти

до сторінки з детальною інформацією про цю вимогу. Якщо вимог не знайдено, відображається відповідне повідомлення. Цей компонент забезпечує користувачів зручним інтерфейсом для перегляду та управління вимогами, надаючи можливість легко знаходити та переглядати необхідні дані.

```
// src/components/Requirements.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Link, useParams } from 'react-router-dom';
import { format } from 'date-fns';
const Requirements = () => {
    const { projectId } = useParams();
    const [requirements, setRequirements] = useState([]);
    useEffect(() => {
        if (projectId) {
axios.get(`http://localhost:3001/api/requirements?projectId=${proje
ctId}`)
                .then(response => {
                    setRequirements(response.data);
                })
                .catch(error => {
                    setRequirements([]);
                });
        }
    }, [projectId]);

    return (
        <div className="container mt-4">
            <h1>Requirements for Project {projectId}</h1>
            <ul className="list-group">
                {Array.isArray(requirements)                &&
requirements.length > 0 ? (
                    requirements.map(requirement => (
                        <li     key={requirement.RequirementID}
className="list-group-item">
                                <Link
to={`/requirement/${requirement.RequirementID}`}>{requirement.Title
}</Link> -
                                {format(new
Date(requirement.DateCreated), 'dd-MM-yyyy HH:mm')}
                            </li>
                    ))
                ) : (
                    <p>No    requirements    found    for    this
project.</p>
                )}
            </ul>
        </div>
    );
};
export default Requirements;
```

Компонент IssuesPerMonthByPriority.js відповідає за відображення статистики знайдених багів за місяць з урахуванням пріоритетів. Цей компонент використовує бібліотеку axios для отримання даних з сервера та react-chartjs-2 для побудови інтерактивних графіків. Інтерфейс компонента включає форму для вибору проекту, діапазону дат та статусу завдань, що дозволяє користувачам фільтрувати дані за різними критеріями. Після відправки форми, завантажені дані відображаються у вигляді лінійних та кругових діаграм, що надають візуальне представлення розподілу завдань за пріоритетами протягом обраного періоду. Це допомагає користувачам краще розуміти тенденції та ефективно управляти завданнями у проекті.

Наведений нижче фрагмент коду з компонента IssuesPerMonthByPriority.js виконує наступні дії:

- використовує хуки useState для зберігання списку проектів, даних форми та даних для графіків;
- завантажує список проектів з сервера за допомогою axios у useEffect під час першого рендерингу компонента;
- оновлює стан форми при зміні значень у полях введення та вибору дат.
- відправляє дані форми на сервер для отримання статистики завдань при надсиланні форми;
- генерує дані для побудови лінійного та кругового графіків на основі отриманої статистики.

```
// src/components/IssuesPerMonthByPriority.js
const IssuesPerMonthByPriority = () => {
    const [projects, setProjects] = useState([]);
    const [formData, setFormData] = useState({
        projectId: '',
        dateFrom: new Date(),
        dateTo: new Date(),
        status: '0'
    });
    const [chartData, setChartData] = useState(null);
    useEffect(() => {
        axios.get('http://localhost:3001/api/projects')
            .then(response => {
                setProjects(response.data);
            })
```

```
            .catch(error => {
                console.error('There was an error fetching the
projects!', error);
            });
    }, []);
    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData({
            ...formData,
            [name]: value,
        });
    };
    const handleDateChange = (name, date) => {
        setFormData({
            ...formData,
            [name]: date,
        });
    };
    const handleSubmit = (e) => {
        e.preventDefault();
        const data = {
            ...formData,
            dateFrom: format(formData.dateFrom, 'yyyy-MM-dd'),
            dateTo: format(formData.dateTo, 'yyyy-MM-dd')
        };
        axios.post('http://localhost:3001/api/dashboard/issues-per-
month', data)
            .then(response => {
                setChartData(response.data);
            })
            .catch(error => {
                console.error('There was an error fetching the chart
data!', error);
            });
    };

    const generateLineChartData = () => {
        if (!chartData) return {};
        const labels = chartData.months;
        const                          datasets                          =
Object.keys(chartData.issuesByPriority).map((priority, index) => ({
            label: priorityOptions[priority],
            data: chartData.issuesByPriority[priority],
            borderColor:    ['#ff6384',    '#36a2eb',    '#cc65fe',
'#ffce56'][index],
            fill: false
        }));
        return {
            labels,
            datasets
        };
    };
 const generatePieChartData = () => {
    if (!chartData) return {};
    const                             labels                             =
Object.keys(chartData.totalIssuesByPriority).map(priority         =>
priorityOptions[priority]);
    const data = Object.values(chartData.totalIssuesByPriority);
```

```
    const backgroundColor = ['#ff6384', '#36a2eb', '#cc65fe',
'#ffce56'];
    return {
        labels,
        datasets: [{
            data,
            backgroundColor
        }]
    };
};
```

В той же час омпонент ExportPage.js надає користувачам можливість експортувати дані проекту у форматі Excel. Використовуючи axios для запитів до сервера та бібліотеки file-saver та xlsx для обробки та збереження файлів, цей компонент забезпечує зручний інтерфейс для вибору проекту, типу даних (тестові випадки, завдання, вимоги) та діапазону дат для експорту. Форма включає випадаючі списки та календарі для точного налаштування параметрів експорту. Після натискання кнопки "Export", дані завантажуються з сервера і автоматично зберігаються на локальному комп'ютері користувача у вигляді Excel-файлу. Цей інструмент є важливим для збереження та аналізу даних поза межами системи.

Цей фрагменту код з компонента ExportPage.js виконує наступні дії:

- використовує хуки useState для зберігання списку проектів, вибраного проекту, типу експорту та діапазону дат;
- завантажує список проектів з сервера за допомогою axios у useEffect під час першого рендерингу компонента;
- формує URL-адресу залежно від типу експорту та надсилає запит на сервер для отримання даних;
- після отримання даних формує Excel-файл за допомогою бібліотеки XLSX і зберігає його на локальному комп'ютері користувача за допомогою бібліотеки file-saver.

```
const ExportPage = () => {
    const [projects, setProjects] = useState([]);
    const [selectedProject, setSelectedProject] = useState('');
    const [exportType, setExportType] = useState('');
    const [dateFrom, setDateFrom] = useState(new Date());
    const [dateTo, setDateTo] = useState(new Date());
```

```
    useEffect(() => {
        axios.get('http://localhost:3001/api/projects')
            .then(response => {
                setProjects(response.data);
            })
            .catch(error => {
                console.error('There was an error fetching the projects!',
error);
            });
    }, []);

    const handleExport = () => {
        let url = '';
        if (exportType === 'testcases') {
            url = 'http://localhost:3001/api/export/testcases';
        } else if (exportType === 'issues') {
            url = 'http://localhost:3001/api/export/issues';
        } else if (exportType === 'requirements') {
            url = 'http://localhost:3001/api/export/requirements';
        }
        axios.post(url, {
            projectId: selectedProject,
            dateFrom: dateFrom.toISOString(),
            dateTo: dateTo.toISOString()
        })
        .then(response => {
            const data = response.data;
            const worksheet = XLSX.utils.json_to_sheet(data);
            const workbook = XLSX.utils.book_new();
            XLSX.utils.book_append_sheet(workbook, worksheet, 'Data');
            const excelBuffer = XLSX.write(workbook, { bookType: 'xlsx',
type: 'array' });
            const blob = new Blob([excelBuffer], { type:
'application/octet-stream' });
            saveAs(blob, `${exportType}.xlsx`);
        })
        .catch(error => {
            console.error('There was an error exporting the data!', error);
        });
    };
```

Таким чином, процес створення клієнтської частини включає налаштування середовища розробки, встановлення необхідних пакетів, розробку компонентів та стилізацію інтерфейсу. Використання React та сучасні інструменти та бібліотеки сприяють ефективній та інтерактивній роботі з додатком. Клієнтська частина надає користувачам інтуїтивний інтерфейс для управління проектами, вимогами, знайденими багами та тестовими випадками, забезпечуючи зручний доступ до необхідної інформації та функцій.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Мануальне тестування програмного забезпечення

Мануальне тестування – це процес перевірки програмного забезпечення шляхом виконання тестових випадків вручну без використання автоматизованих інструментів. Основною метою мануального тестування є виявлення дефектів у програмному забезпеченні та забезпечення відповідності вимогам користувача. Мануальне тестування дозволяє оцінити функціональність, зручність використання та надійність системи з точки зору кінцевого користувача.

Для проведення мануального тестування було розроблено набір тестових випадків, що охоплюють основні функціональні можливості системи. Тестові випадки включали сценарії аутентифікації користувача, створення та редагування проектів, управління вимогами, завданнями та тестовими випадками, а також експорт даних. Кожен тестовий випадок складався з дій, які необхідно виконати, очікуваних результатів та фактичних результатів, що фіксувалися під час тестування.

Також було проведено дослідницьке тестування програмного застосунку без використання тестових артефактів, що призвело до знаходження трьох помилок, створених у процесі розробки, які також були виправлені у подальших версіях.

В результаті мануального тестування було виявлено декілька дефектів, які були виправлені в процесі розробки.

Система успішно аутентифікує користувачів з валідними обліковими даними та відображає повідомлення про помилку при введенні невірних даних (див.рис. 5.1).

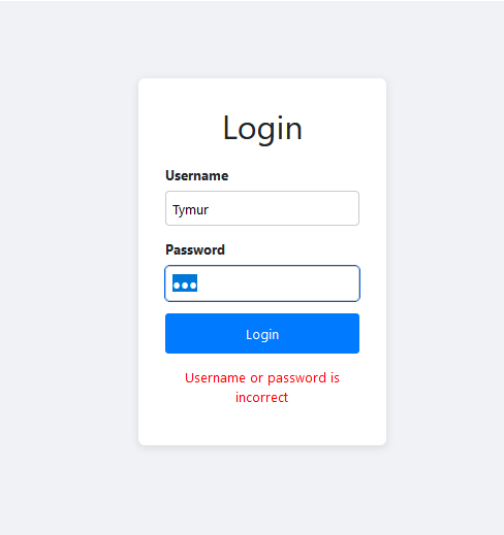


Рисунок 5.1 – Приклад введення некоректного паролю

Функціональність створення, редагування (див.рис. 5.2) та видалення проектів, вимог, завдань та тестових випадків працює коректно.

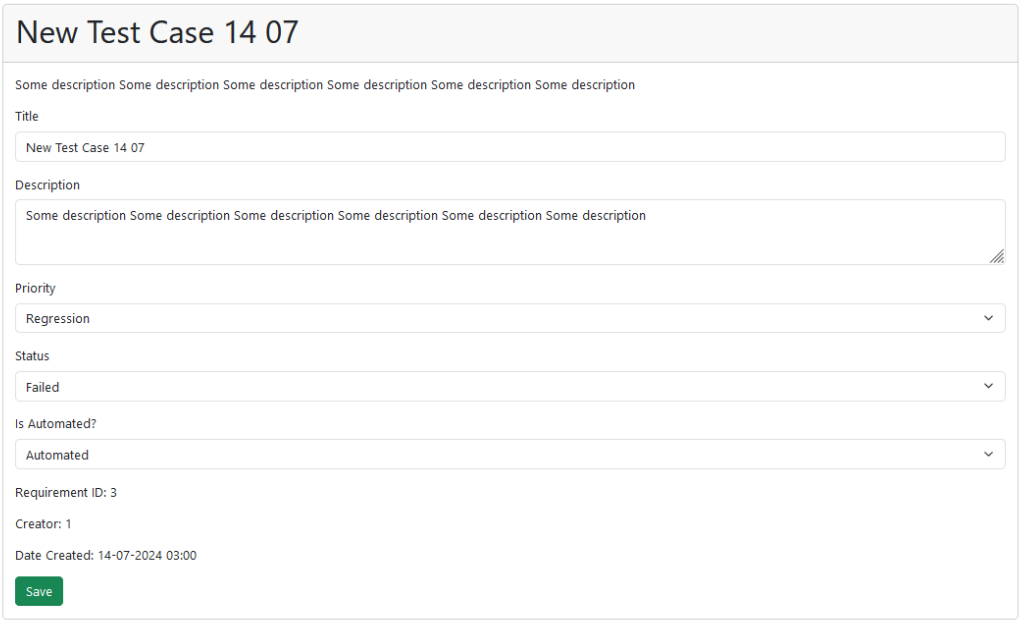


Рисунок 5.2 – Форма редагування існуючого тестового випадку

Відображення списків проектів, вимог, завдань та тестових випадків відповідає очікуванням, дані коректно оновлюються (див.рис. 5.3) після внесення змін.

Експорт даних у форматі Excel здійснюється без помилок, збережені дані відповідають очікуваним.

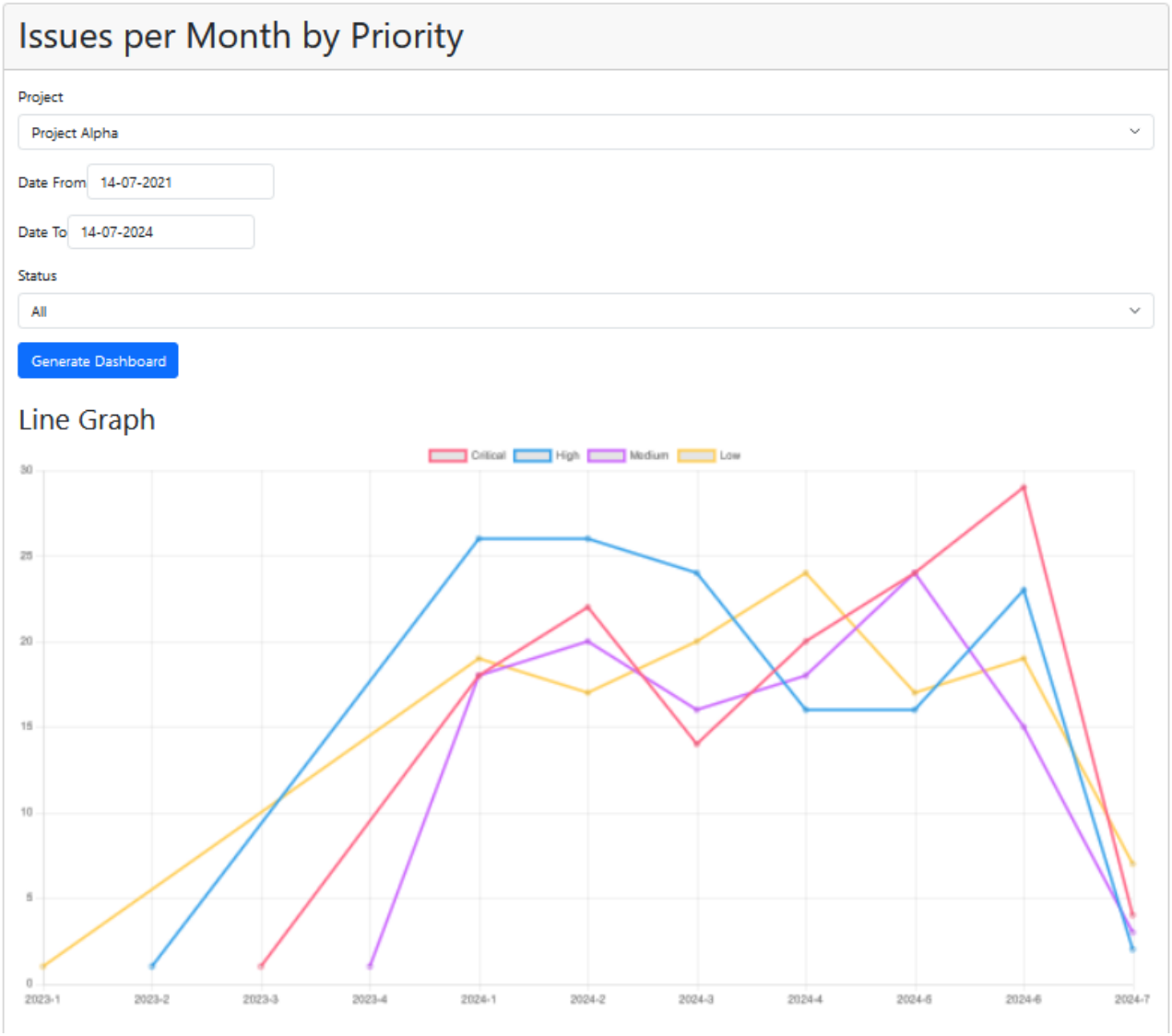


Рисунок 5.3 – Сторінка генерації графиків пов’язаних із звітами знайдених помилок (багів)

Мануальне тестування дозволило перевірити не лише функціональність системи, але й її зручність для кінцевих користувачів.

Користувачі оцінили інтуїтивність інтерфейсу, що значно покращило загальний користувацький досвід. Виявлені під час тестування дефекти були

оперативно виправлені, що сприяло підвищенню надійності та якості програмного забезпечення.

5.2 Тестування серверної частини із застосуванням Postman

Тестування API (Application Programming Interface) є невід'ємною частиною розробки програмного забезпечення, що дозволяє перевірити коректність роботи взаємодії між різними компонентами системи. API-тестування включає перевірку функціональності, продуктивності, безпеки та надійності API. Postman – це потужний інструмент для тестування API, який дозволяє автоматизувати процеси відправлення HTTP-запитів, перевірки відповідей та управління колекціями тестів.

Для тестування API нашого проекту було створено колекцію тестів у Postman, що охоплює основні ендпоінти системи. Тестові випадки включали перевірку операцій аутентифікації, управління проектами, вимогами, завданнями та тестовими випадками, а також експорт даних. Кожен тестовий випадок складався з відправлення HTTP-запиту до API, перевірки статус-коду відповіді та вмісту відповіді на відповідність очікуваним результатам.

В результаті тестування API із використанням Postman (див.рис. 5.4) було досягнуто наступних результатів. Всі основні ендпоінти системи успішно обробляють запити та повертають коректні відповіді. Тести на аутентифікацію підтвердили безпечність та коректність процесу входу користувача. Операції створення, редагування та видалення проектів, вимог, завдань та тестових випадків працюють згідно з очікуваннями.

Експорт даних здійснюється без помилок, забезпечуючи точність та цілісність збережених файлів

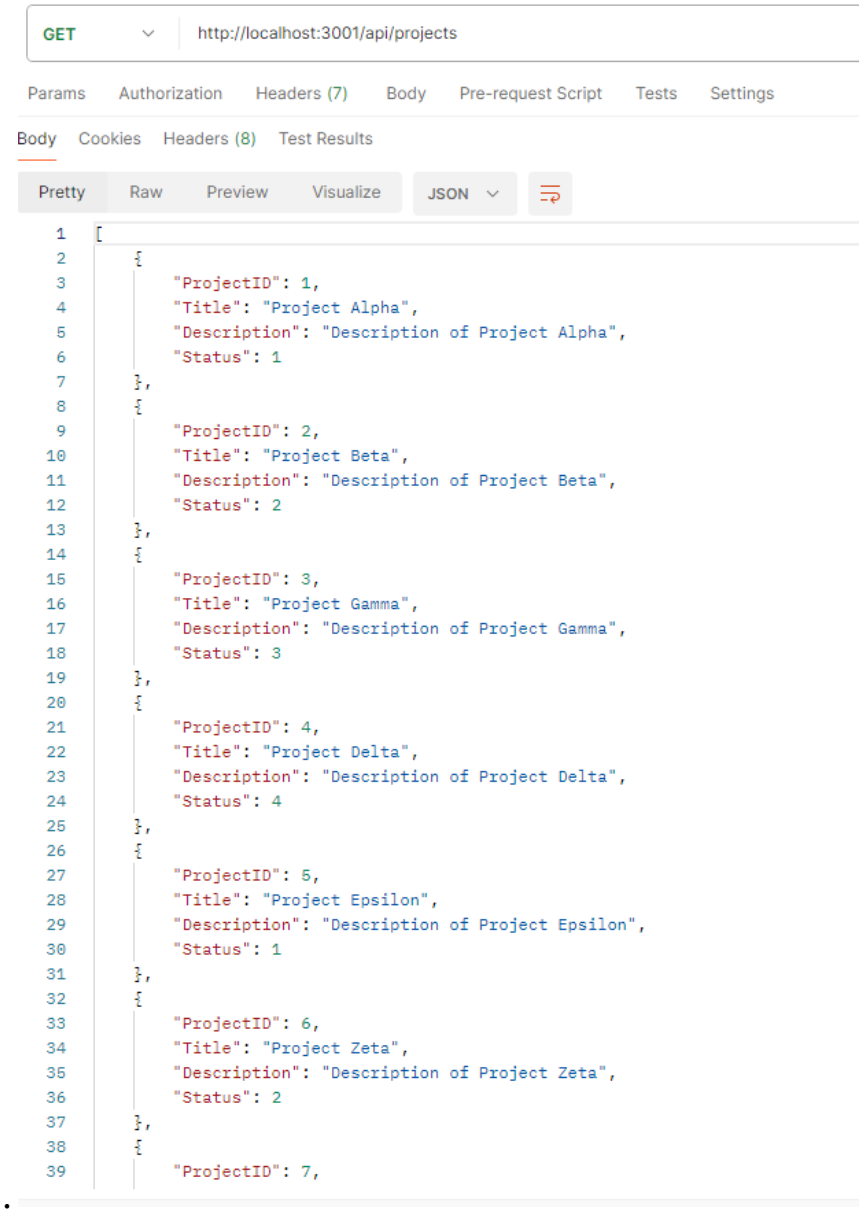


Рисунок 5.4 – Результат винонання запиту у Postman

Тестування API дозволило впевнитися в коректній роботі серверної частини системи та її здатності обробляти запити відповідно до специфікації. Використання Postman для тестування API значно спростило процес виявлення та виправлення дефектів, що покращило загальну якість програмного забезпечення.

ВИСНОВКИ

Виконання даного проекту з розробки та тестування системи управління проектами виявилося успішним та дозволило досягти поставлених цілей. Програмне забезпечення було створено на основі сучасних технологій, таких як React для клієнтської частини та Express для серверної частини, що забезпечило високу продуктивність та зручність у використанні.

Клієнтська частина системи була реалізована за допомогою бібліотеки React, що забезпечило модульність та підтримуваність коду. Використання компонентного підходу дозволило створити інтуїтивно зрозумілий інтерфейс, який спрощує взаємодію користувача із системою. Реалізовано основні функціональні модулі, включаючи управління проектами, вимогами, завданнями та тестовими випадками, а також експорт даних.

Серверна частина була побудована на базі фреймворку Express для Node.js з використанням Microsoft SQL Server для зберігання даних. Налаштовані маршрути для обробки CRUD операцій для основних сутностей системи. Використання ORM забезпечило безпеку та ефективність роботи з базою даних.

Система пройшла два етапи тестування: мануальне тестування та тестування API із використанням Postman. Мануальне тестування дозволило перевірити функціональність та зручність використання інтерфейсу користувача, виявити та виправити дефекти. Тестування API із використанням Postman підтвердило коректність роботи серверної частини та її здатність обробляти запити відповідно до специфікації.

В результаті мануального тестування та тестування API було виявлено та виправлено декілька дефектів, що дозволило підвищити якість програмного забезпечення. Система успішно обробляє запити користувачів, забезпечує надійне зберігання та обробку даних, а також надає інтуїтивно зрозумілий інтерфейс для управління проектами та завданнями.

Розробка та тестування даного проекту дозволило підтвердити ефективність використання сучасних технологій для створення складних програмних систем.

Використання React для клієнтської частини та Express для серверної частини забезпечило високу продуктивність, масштабованість та підтримуваність коду. Процес тестування дозволив виявити та виправити дефекти, що сприяло підвищенню надійності та якості програмного забезпечення.

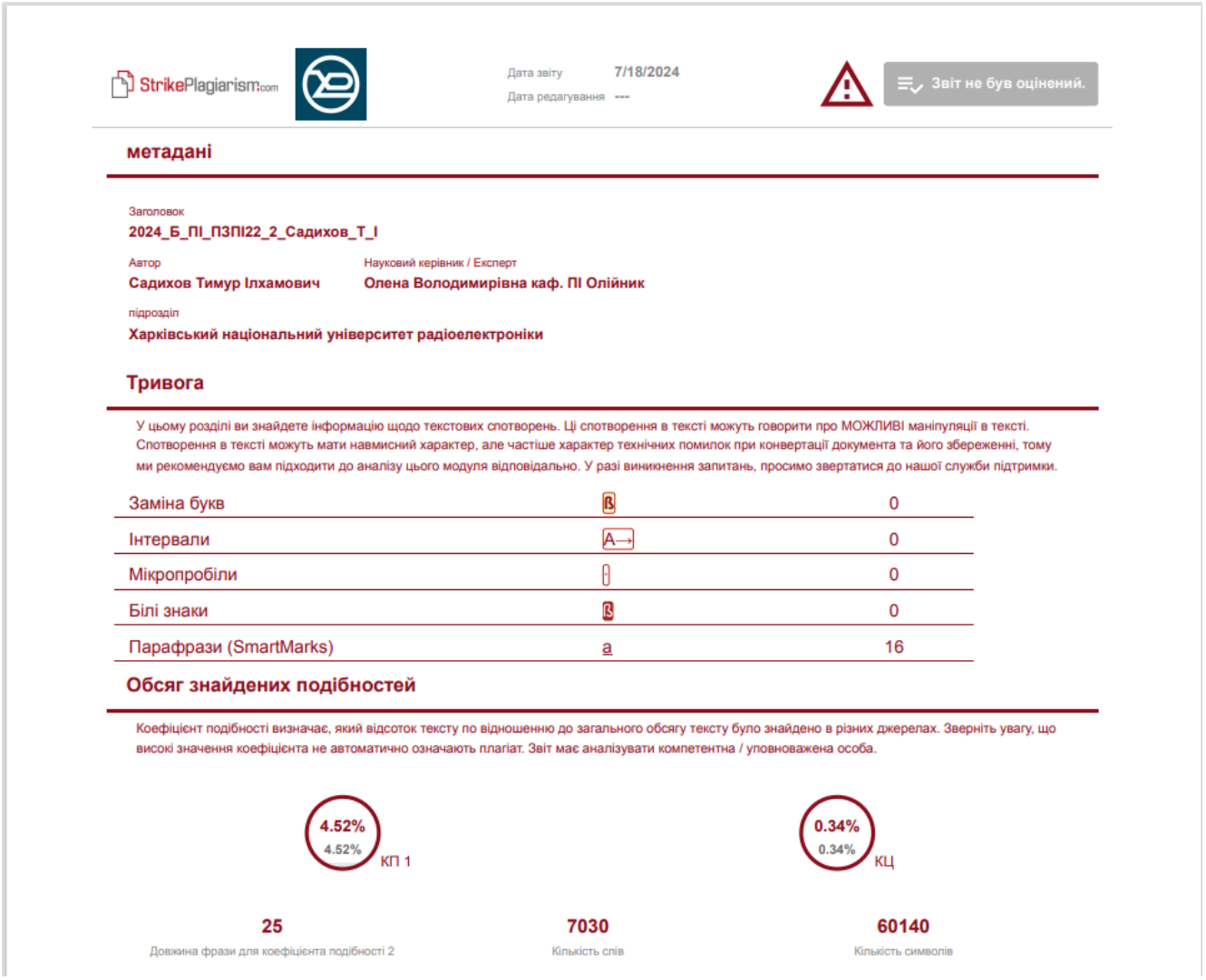
Подальший розвиток системи може включати інтеграцію з іншими інструментами для управління проектами, розширення функціональних можливостей, а також автоматизацію тестування для підвищення ефективності процесу розробки. Виконання цього проекту стало важливим кроком у набутті практичного досвіду розробки та тестування програмного забезпечення, що є надзвичайно важливим для подальшої професійної діяльності у сфері інформаційних технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mozilla Developer Network (MDN) - JavaScript Guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
2. ReactJS - Офіційний сайт документації. URL: https://reactjs.org/docs/getting-started.html
3. W3Schools - SQL Tutorial. URL: https://www.w3schools.com/sql/
4. PractiTest - Офіційна документація. URL: https://www.practitest.com/help/
5. JavaScript Info - Повний посібник з JavaScript. URL: https://javascript.info/
6. React for Beginners [Електронний ресурс] - URL: https://reactforbeginners.com/
7. Microsoft Docs - SQL Server Documentation. URL: https://docs.microsoft.com/en-us/sql/sql-server/
8. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. O'Reilly Media, 2020. 704 p.
9. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2020. 350 p.

ДОДАТОК А

Висновок щодо плагіату



StrikePlagiarism.com

Дата звіту 7/18/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІ22_2_Садихов_Т_І

Автор
Садихов Тимур Ілхамович

Науковий керівник / Експерт
Олена Володимирівна каф. ПІ Олійник

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	16

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

4.52% 4.52% КП 1

0.34% 0.34% КЦ

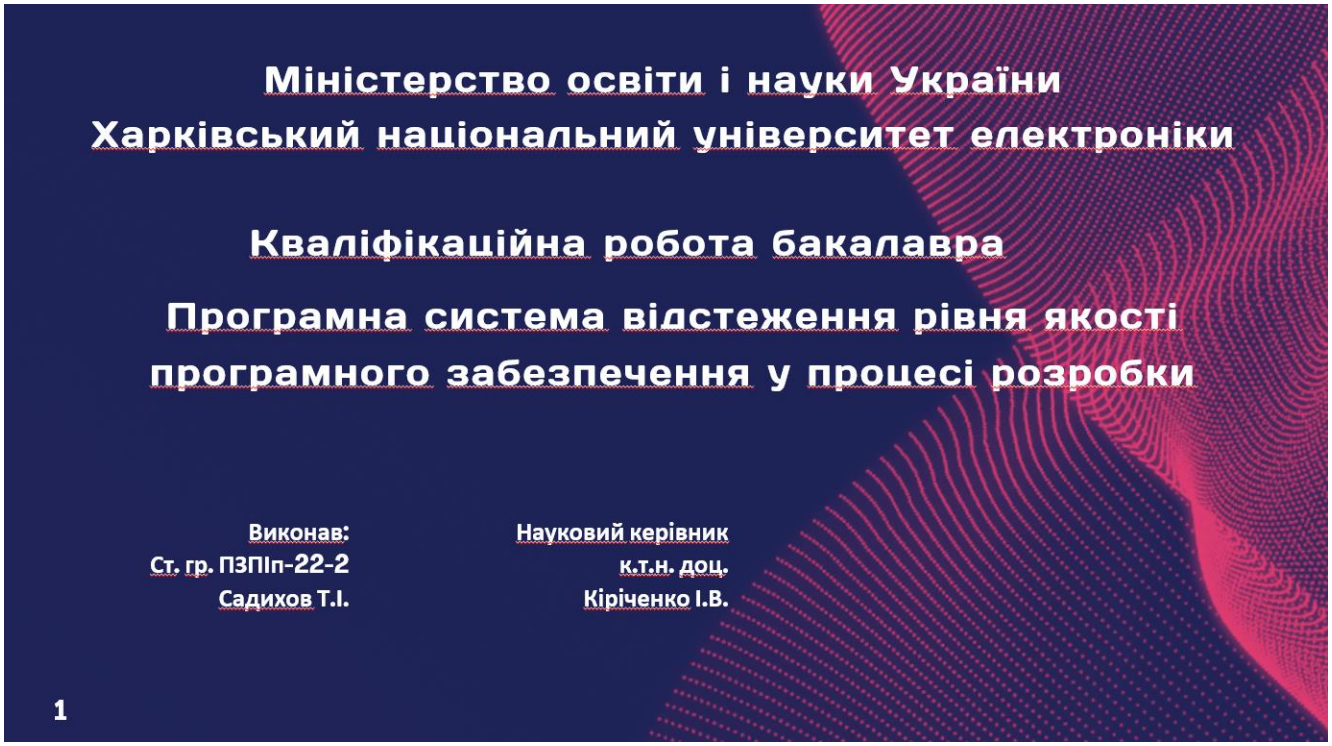
25
Довжина фрази для коефіцієнта подібності 2

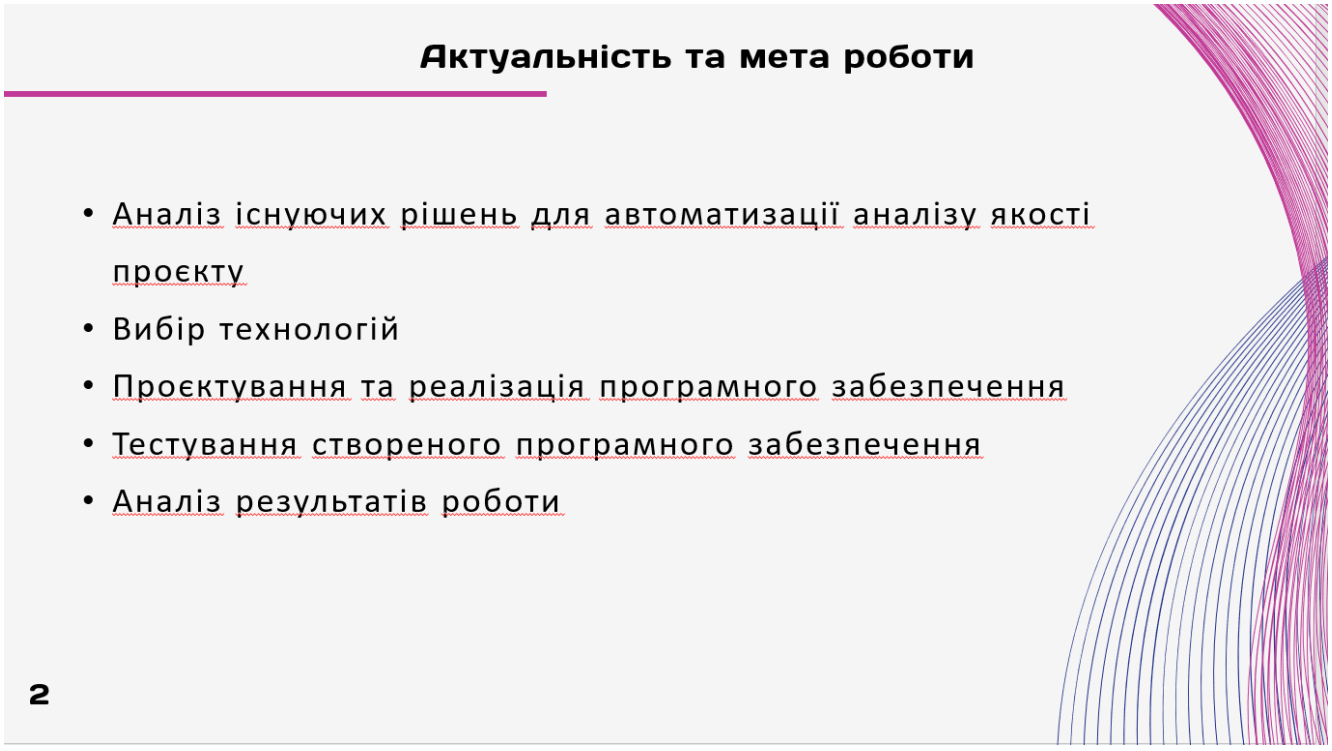
7030
Кількість слів

60140
Кількість символів

ДОДАТОК Б

Слайди презентації





Аналіз існуючих рішень

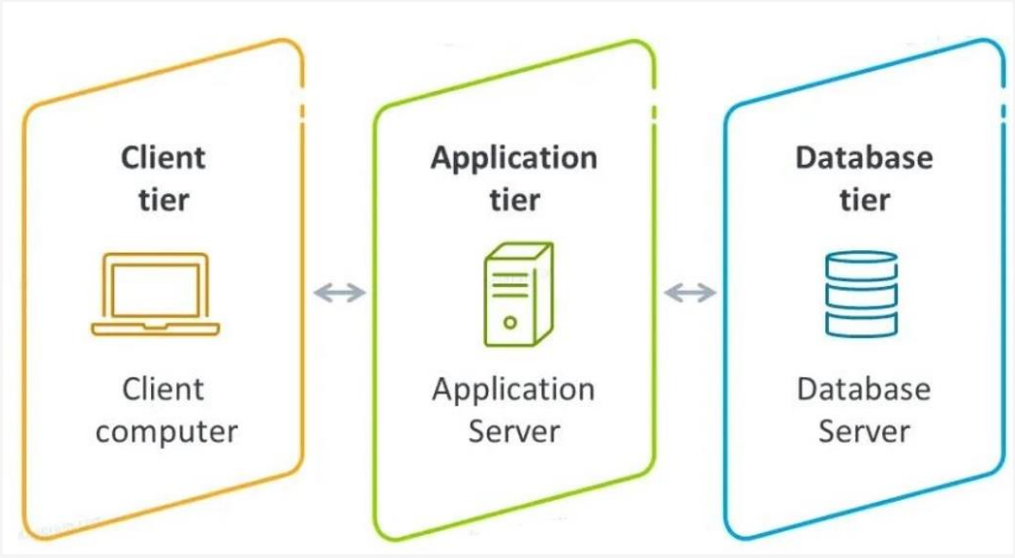
Система	Переваги	Недоліки
JIRA	• Jira пропонує широкий спектр функцій для управління проектами, включаючи створення та відстеження завдань, управління вимогами, планування спринтів та звітування про виконання. • Забезпечує налаштування робочих процесів відповідно до потреб різних команд та проектів.	• Може бути складною для нових користувачів через великий набір функцій і налаштувань, що може вимагати часу на навчання. • Ліцензування Jira може бути дорогим для невеликих команд або стартапів.
Bugzilla	• Підтримує основні функції для управління багами, включаючи відстеження помилок, створення звітів і налаштування робочих процесів. • Відкритий вихідний код і безкоштовне використання, що робить його доступним для невеликих команд.	• Інтерфейс користувача застарілий і менш зручний у порівнянні з сучасними інструментами. • Обмежені можливості інтеграції з іншими сучасними інструментами розробки. • Відсутність офіційної підтримки може ускладнити вирішення проблем та налаштування.
Practitest	• Спеціалізується на управлінні тестуванням, пропонуючи функції для створення, виконання та відстеження тестових випадків. • Інтуїтивно зрозумілий інтерфейс, який полегшує роботу з тестовими сценаріями та звітами. • Підтримує інтеграцію з багатьма інструментами, такими як Jira, Jenkins, Selenium, що спрощує процес тестування.	• Висока вартість ліцензії, особливо для невеликих команд. • Фокусується виключно на управлінні тестуванням, що може вимагати додаткових інструментів для управління іншими аспектами проекту. • Хоча інтерфейс інтуїтивно зрозумілий, для повного використання всіх функцій може знадобитися час на навчання.

3

Використані технології

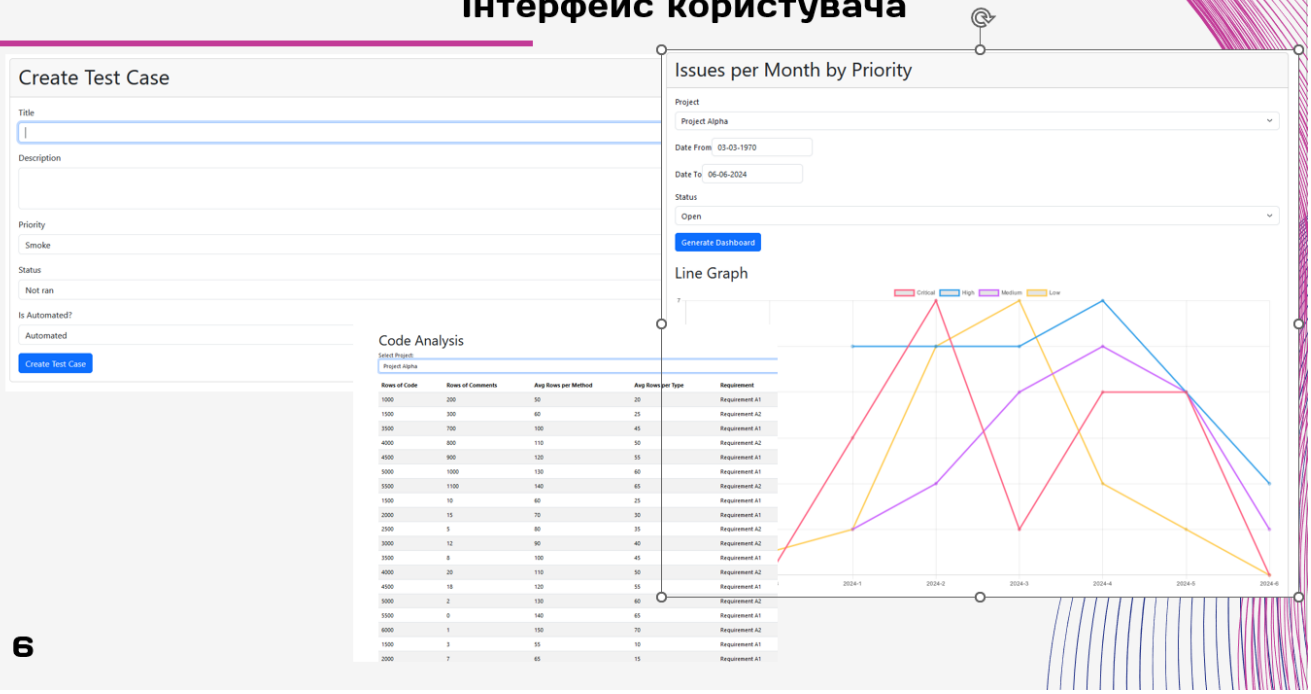
4

Опис розробленої системи



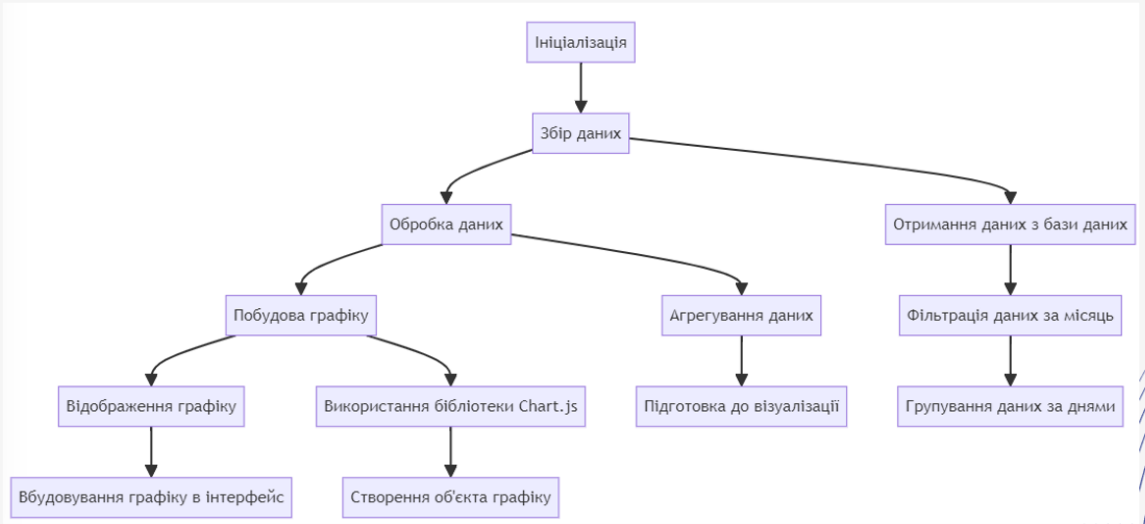
5

Інтерфейс користувача



6

Ключові функції системи



7

Інтеграція з базою даних

```
const getAllTestCases = async () => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().query('SELECT * FROM TestCases');
        return result.recordset;
    } catch (err) {
        console.error('Error fetching test cases:', err);
        return []; // Return an empty array in case of error
    }
};

const getTestCasesByRequirementId = async (requirementId) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().input('requirementId', sql.Int, requirementId).query('SELECT * FROM TestCases WHERE RequirementID = @requirementId');
        return result.recordset;
    } catch (err) {
        console.error('Error fetching test cases by requirementId:', err);
        return []; // Return an empty array in case of error
    }
};

const getTestCaseById = async (id) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request().input('id', sql.Int, id).query('SELECT * FROM TestCases WHERE TestCaseID = @id');
        return result.recordset[0];
    } catch (err) {
        console.error('Error fetching test case by id:', err);
        return null; // Return null in case of error
    }
};

const createTestCase = async (data) => {
    try {
        const pool = await sql.connect(dbConfig);
        const result = await pool.request()
            .input('Title', sql.NVarChar, data.Title)
            .input('Description', sql.NVarChar, data.Description)
            .input('Priority', sql.TinyInt, data.Priority)
            .input('Status', sql.TinyInt, data.Status)
            .input('IsAutomated', sql.Bit, data.IsAutomated)
            .input('RequirementID', sql.Int, data.RequirementID)
            .input('Creator', sql.Int, data.Creator)
            .input('DateCreated', sql.Date, new Date()) // Use current date
            .query('INSERT INTO TestCases (Title, Description, Priority, Status, IsAutomated, RequirementID, Creator, DateCreated) VALUES (@Title, @Description, @Priority, @Status, @IsAutomated, @RequirementID, @Creator,
            @DateCreated)');
        return result.rowsAffected;
    } catch (err) {
        console.error('Error creating test case:', err);
        return 0; // Return 0 in case of error
    }
};
```

8

Тестування

- Мануальне тестування
- Тестування API за допомогою Postman

9

Можливості використання

Поточна реалізація дозволяє слідкувати за станом якості створюваного проєкту, вести проєктну документацію, аналізувати метрики, експортувати документацію

Перспективи проєкту включають в себе розвиток та популярізацію програми в середовищі програмистів. Перспективит ехнічного розвитку програими включають в себе інтеграцію із популярними системами, Jira, Confluence, Azure DevOps та інші. Розширення функціоналу та автоматизація збору метрік. Налаштування регулярних звітів на пошту користувачей.

10