БАКАЛАВРСЬКА РОБОТА

БР.КІ-12.00.00.000 ПЗ

Група КІ-21-1

Комарин Михайло

2025

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій

Кафедра комп'ютерних систем і мереж

Комарин Михайло Васильович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка web-сервісу для контролю та управління життєвим циклом програмного забезпечення засобами C++, PostgreeSQL та фреймворка Qt

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ *Комарин М.В.*.
(підпис, ініціали та прізвище здобувача)

Науковий керівник *Бузоверя Н.Г., доцент*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

д.т.н.,проф. *С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025 рік

Івано-Франківський національний технічний університет нафти і газу

(повне найменування вищого навчального закладу)

Інститут *Інформаційних технологій*

Кафедра *Комп'ютерних систем та мереж*

Спеціальність 123 - *Комп'ютерна інженерія*

Освітній рівень *– бакалавр*

ЗАТВЕРДЖУЮ
Завідувач кафедри КСМ

(С.І. Мельничук)

« 25» квітня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Комарину Михайлу Васильовичу

(прізвище, ім'я, по батькові)

1. Тема дипломної роботи *Розробка web-сервісу для контролю та управління життєвим циклом програмного забезпечення засобами C++, PostgreeSQL та фреймворка Qt* .

Керівник проекту (роботи) *Бузоверя Надія Генадіївна, доцент*

затверджені наказом вищого навчального закладу від *25.04.2025 № 273/7*

2. Термін здачі студентом закінченої роботи *12 червня 2025 р* .

3. Вихідні дані до проекту (роботи) *Методичні вказівки, технічна література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз існуючих підходів та предметної області. 2. Проектування архітектури та візуалізація бізнес-логіки предметної області. 3. Програмна реалізація web-сервісу для контролю та управління життєвим циклом програмного забезпечення* .

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

_____

_____

6. Консультанти розділів роботи

Розділ	Консультант	Підпис, дата

6. Дата видачі завдання *29 січня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	*Вимоги до системи керування життєвим циклом ПЗ*	*Лютий, 2025*	
2	*Аналіз існуючих підходів та предметної області*	*Березень, 2025*	
3	*Проектування архітектури та візуалізація бізнес-логіки предметної області*	*Квітень, 2025*	
4	*Програмна реалізація web-сервісу для контролю та управління життєвим циклом програмного забезпечення*	*Травень, 2025*	
5	*Оформлення пояснювальної записки*	*Травень, 2025*	

Студент _____ *Комарин М.В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Бузоверя Н.Г.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою даної бакалаврської роботи є розробка web-сервісу для контролю та управління життєвим циклом програмного забезпечення. Реалізація здійснена на основі шаблону Model-View-Controller з використанням мови програмування C++, бібліотеки Qt та СУБД PostgreSQL. Система дозволяє керувати проєктами, завданнями, версіями, документацією, ролями користувачів, а також забезпечує авторизацію, безпеку та фільтрацію даних. Архітектура підтримує масштабованість, продуктивність і безпечну роботу з даними. Проєкт протестовано відповідно до функціональних і нефункціональних вимог. Запропоновано рефакторинг для покращення підтримуваності та продуктивності системи.

У першому розділі виконано аналіз предметної області, досліджено особливості життєвого циклу ПЗ та проведено огляд аналогів (Jira, Redmine, GitLab), визначено їх переваги та недоліки порівняно з розроблюваною системою.

У другому розділі розроблено структуру системи: сформовано UML-діаграми варіантів використання, класів, активностей і станів, які відображають функціональну архітектуру майбутнього додатку.

У третьому розділі реалізовано програмну частину системи на основі технологій C++, Qt та PostgreSQL, здійснено тестування функціональності, перевірку на відповідність вимогам і запропоновано оптимізації для підвищення продуктивності та гнучкості.

Ключові слова: життєвий цикл ПЗ, керування проєктами, Qt, C++, PostgreSQL, web-сервіс, авторизація, архітектура MVC.

ANNOTATION

This thesis project presents the development of a web service for managing the software development life cycle. The implementation follows the Model-View-Controller design pattern using C++, Qt framework, and PostgreSQL database. The system supports project and task management, version control, documentation handling, user role management, authentication, and data filtering.

In the first chapter, the subject area is analyzed, including characteristics of the software life cycle and a comparative review of existing solutions such as Jira, Redmine, and GitLab, highlighting their pros and cons.

The second chapter is dedicated to designing the system's architecture. It includes UML diagrams of use cases, classes, states, and activities, describing the structural and functional components of the service.

In the third chapter, the system is implemented using C++, Qt, and PostgreSQL. Functional testing is performed, system requirements are verified, and refactoring recommendations are proposed to improve performance and maintainability.

Keywords: software life cycle, project management, Qt, C++, PostgreSQL, web service, authentication, MVC architecture.

ЗМІСТ

БІБЛІОГРАФІЧНА ДОВІДКА

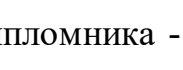
Тема бакалаврської роботи: Розробка web-сервісу для контролю та управління життєвим циклом програмного забезпечення засобами C++, PostgreeSQL та фреймворка Qt

Обсяг пояснювальної записки: 52 сторінки

8 таблиць

21 рисунок

Дата завершення роботи 12.06.2025

Підпис студента дипломника -  Михайло КОМАРИН

ВСТУП

Актуальність теми зумовлена потребою в ефективних інструментах для управління життєвим циклом програмного забезпечення, що дозволяють координувати командну роботу, відслідковувати зміни, забезпечувати контроль версій, підтримку користувачів і високу якість продукту.

Об'єктом дослідження є процес управління життєвим циклом програмного забезпечення.

Предметом дослідження є архітектурні рішення та програмна реалізація системи для управління проєктами, завданнями, документацією і версіями у рамках одного ПЗ.

Метою роботи є проєктування та реалізація web-сервісу з графічним інтерфейсом для контролю повного циклу розробки програмного забезпечення, включаючи модулі управління проєктами, користувачами, завданнями та документацією. Для досягнення мети необхідно виконати такі основні завдання роботи:

- проаналізувати предметну область і вимоги;
- розробити концептуальну модель системи;
- спроєктувати архітектуру та базу даних;
- реалізувати функціонал системи засобами C++ та Qt;
- виконати тестування та рев'ю коду на відповідність вимогам.

Методи дослідження: аналіз вимог, об'єктне моделювання, модульне програмування, ручне тестування.

Практична цінність полягає в реалізації повнофункціонального інструменту для управління програмного забезпечення, який можна адаптувати до потреб різних команд та організацій.

1 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

У сучасному IT-середовищі розробка програмного забезпечення (ПЗ) є складним процесом, який охоплює різні етапи — від формування вимог до підтримки готового продукту. Для ефективної організації цих процесів використовують системи керування життєвим циклом ПЗ. Вони дозволяють централізовано контролювати розробку, релізи, версії, задачі, документи та взаємодію між учасниками команди.

Основним завданням таких систем є забезпечення наскрізної прозорості та контрольованості всіх стадій життєвого циклу. Ключовими учасниками цього процесу виступають менеджери проєктів, технічні спеціалісти, адміністрація, клієнти та інші ролі, кожна з яких виконує свої функції в рамках системи.

У типовій компанії, що займається розробкою ПЗ, проект включає:

– завдання (task), що мають дедлайни, опис, статус;

– документи (документація, вимоги, звіти);

– версії (релізи, зміни, журнал оновлень);

– команду, що складається з менеджера, розробників, тестувальників;

– клієнта або замовника, що отримує результати роботи.

Кожна з цих сутностей тісно пов’язана з іншими. Завдання пов’язані з проектом, виконавцями та строками. Документи належать до проекту і створюються певними користувачами. Версії також пов’язані з проектом і мають змінюваний стан.

Також важливо враховувати технічні аспекти — робота з ролями, авторизація, обробка помилок, збереження цілісності даних. Система повинна бути масштабованою, тобто підтримувати зростання проєктів і команд без втрати продуктивності.

Система, яка розробляється в рамках цієї роботи, повинна відповідати таким вимогам:

– забезпечення авторизації користувачів з різними рівнями доступу;

– централізоване управління проєктами та задачами;

– створення та редагування документів у межах проєкту;

– підтримка оновлення версій з журналами змін;

– інтерфейс для взаємодії клієнтів з командою;

– резервне копіювання даних та захист від несанкціонованого доступу.

Таким чином, предметна область включає технічну (архітектура, безпека, продуктивність) і логічну (взаємодія користувачів, структури даних) складову, що робить завдання реалізації web-сервісу міждисциплінарним.

1.2 Огляд аналогів

Існує чимало систем, призначених для керування життєвим циклом ПЗ, кожна з яких має свої особливості. Найвідоміші з них — Jira, Redmine, GitLab, Trello, Asana.

Jira — одна з найпоширеніших систем, що орієнтована на гнучке управління проєктами (Scrum, Kanban). Її основні функції включають створення задач, етапів (sprint), контроль релізів та звітність. Проте Jira є платною і часто занадто складною для невеликих команд.

Redmine — безкоштовна система з відкритим кодом. Підтримує управління проєктами, задачами, журналами, документацією. Має модульну структуру, але менш гнучкий інтерфейс у порівнянні з Jira. Інтерфейс застарілий, а налаштування потребує досвіду.

GitLab — комплексне середовище, що об’єднує контроль версій (Git), CI/CD, керування задачами та релізами. Потужна, але потребує серйозних ресурсів і адміністративного супроводу. Складність впровадження може бути бар'єром.

Trello — простий візуальний менеджер завдань. Підходить для невеликих команд і особистих проєктів. Не має глибокої інтеграції з системами керування кодом або версіями. Використовується більше як «to-do board», ніж як система контролю всього циклу розробки.

Asana — хмарна система для керування задачами з інтеграцією в календарі та комунікаційні сервіси. Менше орієнтована на розробників, не має підтримки контролю версій або CI.

Зіставлення з розроблюваною системою наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика розроблюваної системи із наявними аналогами

Критерій	Jira	Redmine	GitLab	Поточна система
Контроль версій	частково	ні	так	так
Управління задачами	так	так	так	так
Підтримка документації	обмежено	так	частково	так
Автентифікація	так	так	так	так
Безкоштовна ліцензія	ні	так	частково	так
Платформа	веб	веб	веб	десктоп/web
Підтримка української мови	ні	обмежено	ні	так

Перевагами власної розробки є повний контроль над функціональністю, гнучкість у кастомізації, локалізація під специфіку організації, використання новітніх технологій та зменшення залежності від сторонніх сервісів.

1.3 Постановка завдання

Метою проєкту є створення програмної системи, яка дозволяє здійснювати управління проєктами, задачами, користувачами, документацією та версіями в межах повного життєвого циклу розробки програмного забезпечення. Система повинна забезпечувати надійність, безпеку, зручний інтерфейс, масштабовану архітектуру та функціональність відповідно до сучасних вимог IT-індустрії.

Вихідні дані:

– вимоги до системи керування життєвим циклом ПЗ;

– типові бізнес-процеси команди розробників;

– огляд аналогічних систем (Jira, GitLab, Redmine);

– технічне середовище: мова програмування C++, фреймворк Qt, СУБД PostgreSQL;

– архітектурний шаблон MVC;

– вимоги до ролей користувачів, рівнів доступу та структурованих даних.

Функціональні обмеження:

– система повинна працювати в середовищі ОС Windows;

–взаємодія з користувачем відбувається через графічний інтерфейс (QML/Qt);

– взаємодія з базою даних – через бібліотеку pqxx;

– підтримка аутентифікації користувачів та ролей;

– підтримка журналу змін при оновленні версій;

–реалізація CRUD-операцій для проєктів, задач, користувачів та документів.

Функції, що підлягають реалізації:

1. Авторизація та реєстрація користувачів з розмежуванням ролей.
2. Створення, редагування та видалення проєктів.
3. Додавання, перегляд і зміна статусів завдань.
4. Оновлення версій проєктів із фіксацією журналу змін.
5. Робота з документацією: створення, редагування, збереження.
6. Перегляд та фільтрація даних у зручному інтерфейсі.
7. Захист даних і обробка помилок.
8. Тестування працездатності системи та звітність.

Результати, що очікуються:

– повнофункціональний десктопний web-сервіс;

– реалізована архітектура з дотриманням принципів MVC;

– інтерактивний інтерфейс користувача;

– система, здатна масштабуватися під більші команди;

– звіт про тестування із підтвердженням відповідності вимогам.

1.4 Розробка концептуальної моделі предметної області

Концептуальна модель предметної області відображає відображає основні аспекти предметної області розробки програмних систем, включаючи взаємодії, атрибути та події, які визначають робочий процес фірми. Візуалізація концептуальної моделі предметної області зображена на рисунку 1.1.

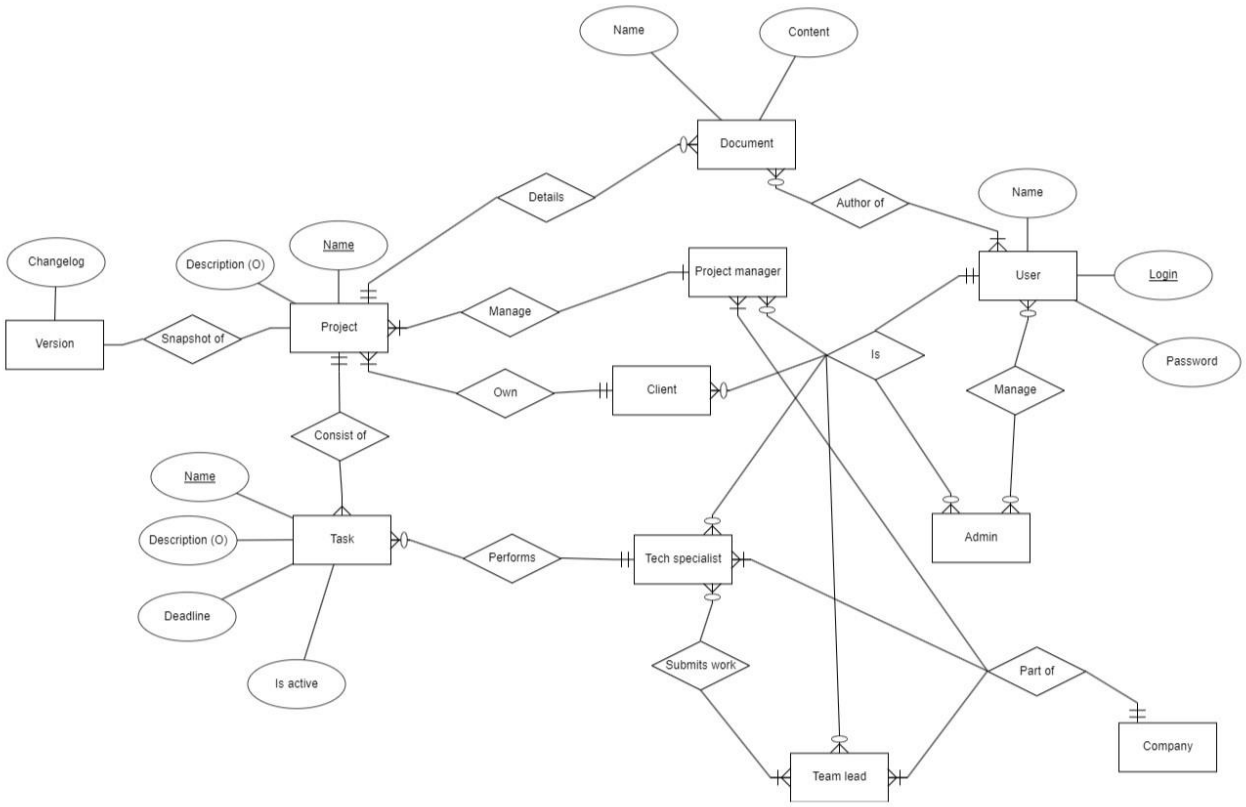


Рисунок 1.1 - Концептуальна модель предметної області

Сутності:

Project (Проект): Представляє проект в компанії. Має атрибути, такі як назва, опис, та менеджера проекту.

Task (Завдання): Представляє завдання в рамках проекту. Має атрибути, такі як назва, опис, крайній термін, активність та виконавець завдання.

Document (Документ): Представляє документ в рамках проекту. Має атрибути, такі як назва, вміст.

User (Користувач): Представляє користувача системи. Має атрибути, такі як ім'я, опис та пароль входу.

Team Lead (Керівний розробник): Представляє керівного розробника в компанії, який перевіряє роботу розробників.

Tech Specialist (Технічний Спеціаліст): Представляє фахівця, який відповідає за виконання завдань у компанії.

Project Manager (Менеджер Проекту): Представляє керівника проекту, який відповідає за планування та управління реалізацією проектів компанії.

Admin (Адміністратор): Представляє адміністратора системи, відповідального за управління користувачами системи компанії.

Version (Версія): Представляє версію програмної проекту, Має атрибут: журнал змін

Company (Компанія): Представляє компанію, в якій існують усі вищезазначені сутності

Client (Клієнт): Представляє замовника або клієнта, який взаємодіє з компанією для отримання програмних продуктів

Зв'язки:

Project (Проект) - Task (Завдання): Кожен проект може мати багато завдань, але кожне завдання належить лише одному проекту.

Project (Проект) - Document (Документ): Кожен проект може мати багато документів, але кожен документ належить лише одному проекту.

Team Lead (Керівний розробник), Tech Specialist (Технічний Спеціаліст), Project Manager (Менеджер Проекту), Admin (Адміністратор) та Client (Клієнт) є користувачами (User).

Project (Проект) - Project Manager (Менеджер Проекту): Кожен проект має одного менеджера проекту, але один і той же менеджер проекту може керувати багатьма проектами.

Project (Проект) - Team Lead (Лідер команди): Кожен проект має одного керівного розробника, який відповідає за виконання завдань розробниками.

Task (Завдання) - Tech Specialist (Технічний Спеціаліст): Кожне завдання виконується одним технічним спеціалістом, але один і той же технічний спеціаліст може виконувати багато завдань.

Document (Документ) - User (Користувач): Кожен документ подається одним користувачем, але один і той же користувач може подати багато документів.

Version (Версія) - Project (Проект): Кожна версія пов'язана з одним проектом, але кожен проект може мати багато версій.

Team Lead (Керівний розробник), Tech Specialist (Технічний Спеціаліст), Project Manager (Менеджер Проекту), Admin (Адміністратор) є співробітниками компації(Company).

Admin (Адміністратор) - User (Користувач) : адміністратор керує повноваження інших користувачів, декілька адміністраторів можуть керувати повноваженнями однієї групи користувачів.

1.5 Аналіз вимог

Система контролю життєвого циклу програмного забезпечення - це комплекс інструментів, процесів та методів, що призначений для керування усіма аспектами розробки, тестування, впровадження та підтримки програмного забезпечення протягом всього його життєвого циклу. Система контролю життєвого циклу програмного забезпечення допомагає розробникам та командам забезпечувати високу якість, надійність та безпеку ПЗ та ефективно керувати проектами.

Система контролю життєвого циклу програмного забезпечення повинна забезпечувати ряд функцій, щоб забезпечити ефективне управління програмними продуктами.

Функціональні вимоги:

1. Управління проектами і ресурсами:
   - планування та відстеження проектів розробки ПЗ;
   - розподіл ресурсів, включаючи персонал та обладнання;
   - моніторинг строків та робочих завдань.
2. Тестування та контроль якості:

- планування та тестування програмного продукту;
- відстеження якості коду та звітність про тестування.

3. Відстеження змін та завдань:
   - відстеження змін у програмному продукті;
   - управління списком завдань і пріоритетами.
4. Управління релізами:
   - планування релізів і випусків програмного продукту;
   - моніторинг релізів.
5. Підтримка користувачів і вирішення проблем:
   - система звітів та обробки звернень користувачів;
   - відстеження та вирішення помилок та проблем користувачів;
   - надання підтримки.
6. Безпека і автентифікація:
   - захист даних та доступу до системи;
   - автентифікація та авторизація користувачів;
   - захист від потенційних загроз безпеці.
7. Моніторинг та звітність:
   - збір та аналіз даних щодо роботи програмного продукту;
   - створення звітів для прийняття рішень щодо подальшого розвитку і підтримки.
8. Адміністрування системи:
   - управління правами користувачів та ролями;
   - моніторинг та управління ресурсами серверів та обладнання.

Нефункціональні вимоги:

1. Безпека:
   - система повинна забезпечувати високий рівень безпеки для захисту конфіденційності, цілісності та доступності даних користувачів;
   - всі дані, передавані між користувачем і системою, повинні бути шифрованими;

- система повинна мати механізми автентифікації та авторизації з можливістю управління правами доступу користувачів.

2. Витривалість та резервне копіювання:

- система повинна регулярно створювати резервні копії даних для забезпечення можливості відновлення у випадку втрати інформації або системних збоїв;

- забезпечити можливість відновлення до попередніх версій проектів та документації.

3. Продуктивність:

- система повинна мати ефективний механізм керування ресурсами для забезпечення стабільної та високої продуктивності при великій кількості користувачів та об'ємі даних.

4. Масштабованість:

- система повинна бути масштабованою, здатною розширюватися при збільшенні обсягу роботи та кількості користувачів;

- забезпечити підтримку для паралельної обробки завдань та розподіленого обчислення.

5. Доступність:

- забезпечити високий рівень доступності системи, уникати довгих періодів відключення та мінімізувати час відновлення після аварій.

6. Інтерфейс:

- забезпечити інтуїтивно зрозумілий та ергономічний інтерфейс для користувачів.

7. Захист від вірусів та атак:

- необхідно забезпечити заходи захисту від вірусів, зловмисних атак та інших загроз безпеці системи.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА ВІЗУАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Діаграма модулів

На рисунку 2.1 представлена архітектурна схема інформаційної системи, яка складається з двох основних рівнів: системного рівня та бекенду. У верхній частині схеми розміщено компонент «User Interface», який відповідає за надання графічного інтерфейсу користувача (GUI) для чотирьох підсистем: управління проєктами, моніторинг і звітність, комунікація та співпраця, а також управління змінами. Кожна з цих підсистем отримує дані через блок «Security and Access Control», який забезпечує безпеку доступу до інформації. Блок безпеки, у свою чергу, авторизує транзакції до бази даних, яка знаходиться на найнижчому рівні архітектури. Вся система побудована за принципом багаторівневої взаємодії між інтерфейсом користувача, функціональними модулями та централізованою системою управління доступом і збереженням даних.

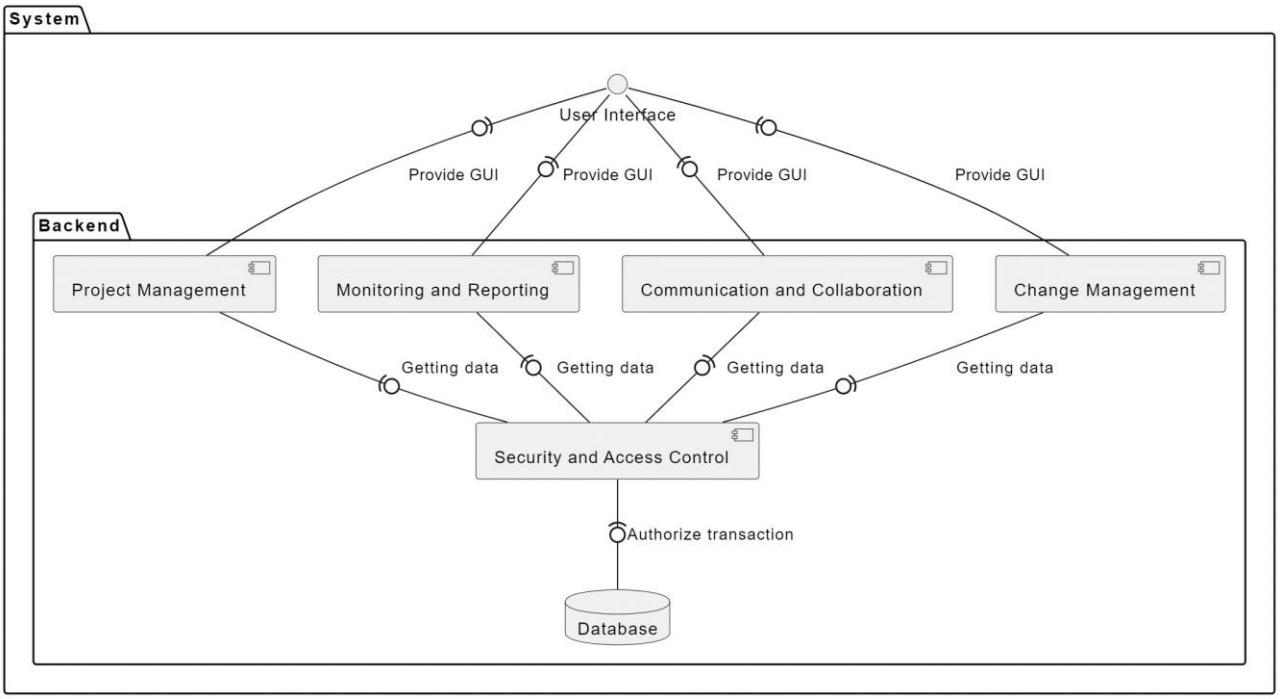


Рисунок 2.1 - Діаграма модулів

Опишемо детальніше кожен із модулів системи:

1. Користувацький інтерфейс (User Interface): Цей модуль включає в себе інтерфейс користувача, який використовується для взаємодії з системою.

2. Модуль керування проектами (Project Management): Цей модуль дозволяє створювати та відстежувати проекти. Він включає в себе інструменти для планування, управління ресурсами, моніторингу виконання проектів та генерації звітів.

3. Модуль моніторингу та звітності (Monitoring and Reporting): Цей модуль включає в себе інструменти для моніторингу та генерації звітів про стан проектів.

4. Модуль безпеки та прав доступу (Security and Access Control): Цей модуль забезпечує захист від несанкціонованого доступу до даних та дій в системі. Він включає в себе засоби аутентифікації, авторизації та шифрування.

5. Модуль зв'язку та спілкування (Communication and Collaboration): Цей модуль допомагає організовувати спілкування між учасниками проекту, обмін інформацією та спільну роботу.

6. Модуль управління змінами (Change Management): Цей модуль допомагає впорядковувати та керувати змінами в проектах.

7. Модуль бази даних (Database): Цей модуль відповідає за зберігання, організацію та доступ до даних, пов'язаних з проектами. Модуль бази даних також включає в себе засоби резервного копіювання та відновлення даних для забезпечення їхньої надійності та доступності.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (рисунок 2.2) допомагає візуалізувати функціональність системи та взаємодію між різними акторами.

Історії користувачів, для кращого розуміння потреб користувачів, які є акторами діаграми варіантів використання наведено нижче для адміністратора, технічного спеціаліста, клієнта та менеджера проекту.

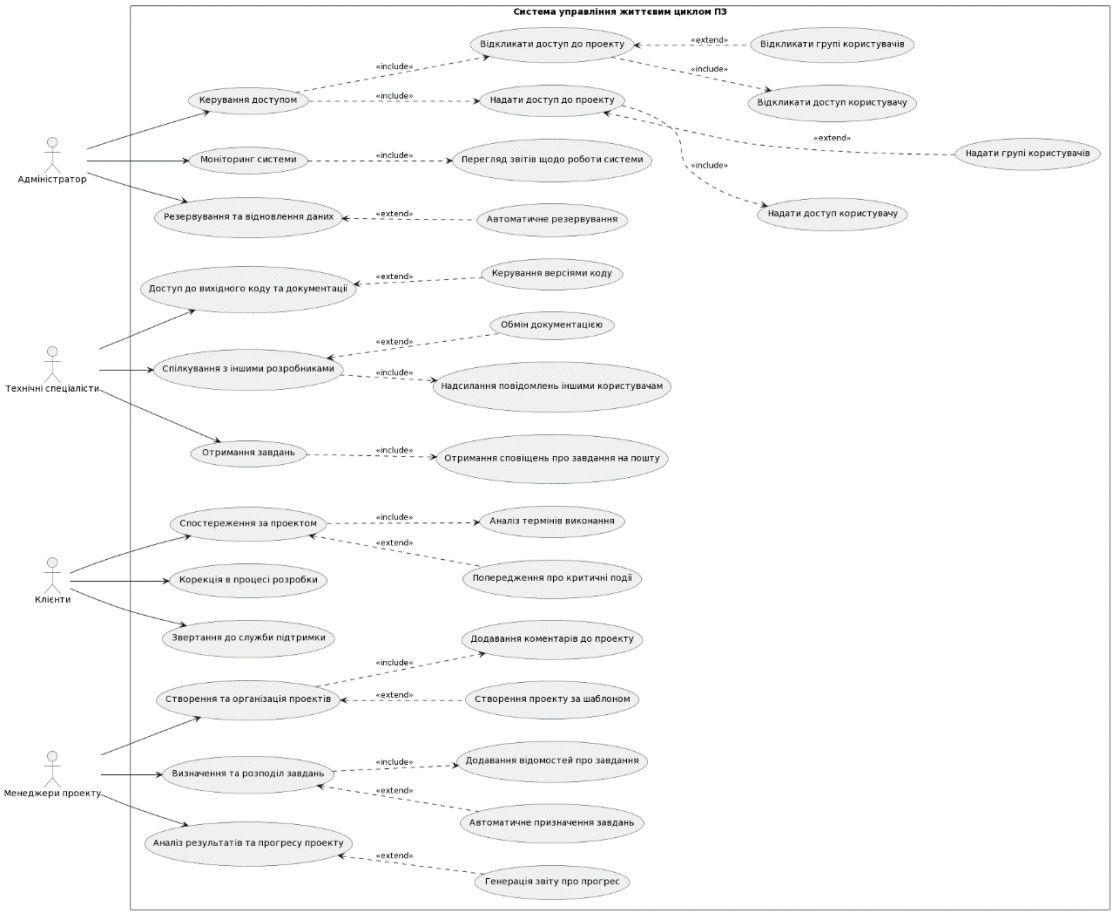


Рисунок 2.2 - Діаграма варіантів використання

Історії корисувачів для адміністраторів.

Як адміністратор, я хочу мати можливість керувати доступом користувачів до системи, щоб забезпечити безпеку та конфіденційність даних.

Як адміністратор, я хочу мати можливість моніторити роботу системи, щоб вчасно виявляти та вирішувати можливі проблеми.

Як адміністратор, я хочу мати можливість резервувати та відновлювати дані системи, щоб запобігти втраті важливої інформації.

Історії корисувачів для технічних спеціалістів.

Як технічний спеціаліст, я хочу мати можливість доступу до вихідного коду та документації проекту, щоб ефективно працювати над внесенням змін та вдосконаленням системи.

Як технічний спеціаліст, я хочу мати можливість спілкуватися з іншими розробниками, щоб обмінюватися ідеями та кращими практиками розробки.

Як технічний спеціаліст, я хочу мати можливість отримувати сповіщення про призначення мені нових завдань, щоб мати змогу вчасно виконувати роботу.

Історії корисувачів для клієнтів.

Як клієнт, я хочу мати можливість спостерігати перебіг проекту, щоб отримувати актуальну інформацію про стан програмного продукту.

Як клієнт, я хочу мати можливість вносити корективи в процес розробки, щоб кінцевий продукт задовольняв мої вимоги.

Як клієнт, я хочу мати можливість звертатися до служби підтримки та отримувати швидку та якісну відповідь на свої запити.

Історії корисувачів для менеджерів проекту.

Як менеджер проекту, я хочу мати можливість створювати та організовувати проекти у системі, щоб ефективно керувати роботою команди.

Як менеджер проекту, я хочу мати можливість визначати та розподіляти завдання між учасниками проекту, щоб досягти цілей проекту.

Як менеджер проекту, я хочу мати можливість аналізувати результати та прогрес проекту, щоб приймати обґрунтовані управлінські рішення.

2.3 Діаграми взаємодії

Діаграма послідовності (sequence diagram) (рисунок 2.3) відображає взаємодію різних учасників системи та компонентів системи під час виконання різних дій у проектному середовищі.

Діаграма показує основні етапи таких взаємодій:

– вхід користувача: Учасники (менеджер, технічний спеціаліст, клієнт) входять в систему через графічний інтерфейс;

– управління системою адміністратором: Здійснюється управління системою, наприклад, виконання адміністративних функцій;

– створення та управління проектом: Менеджер створює проект, додає завдання та призначає їх технічним спеціалістам;

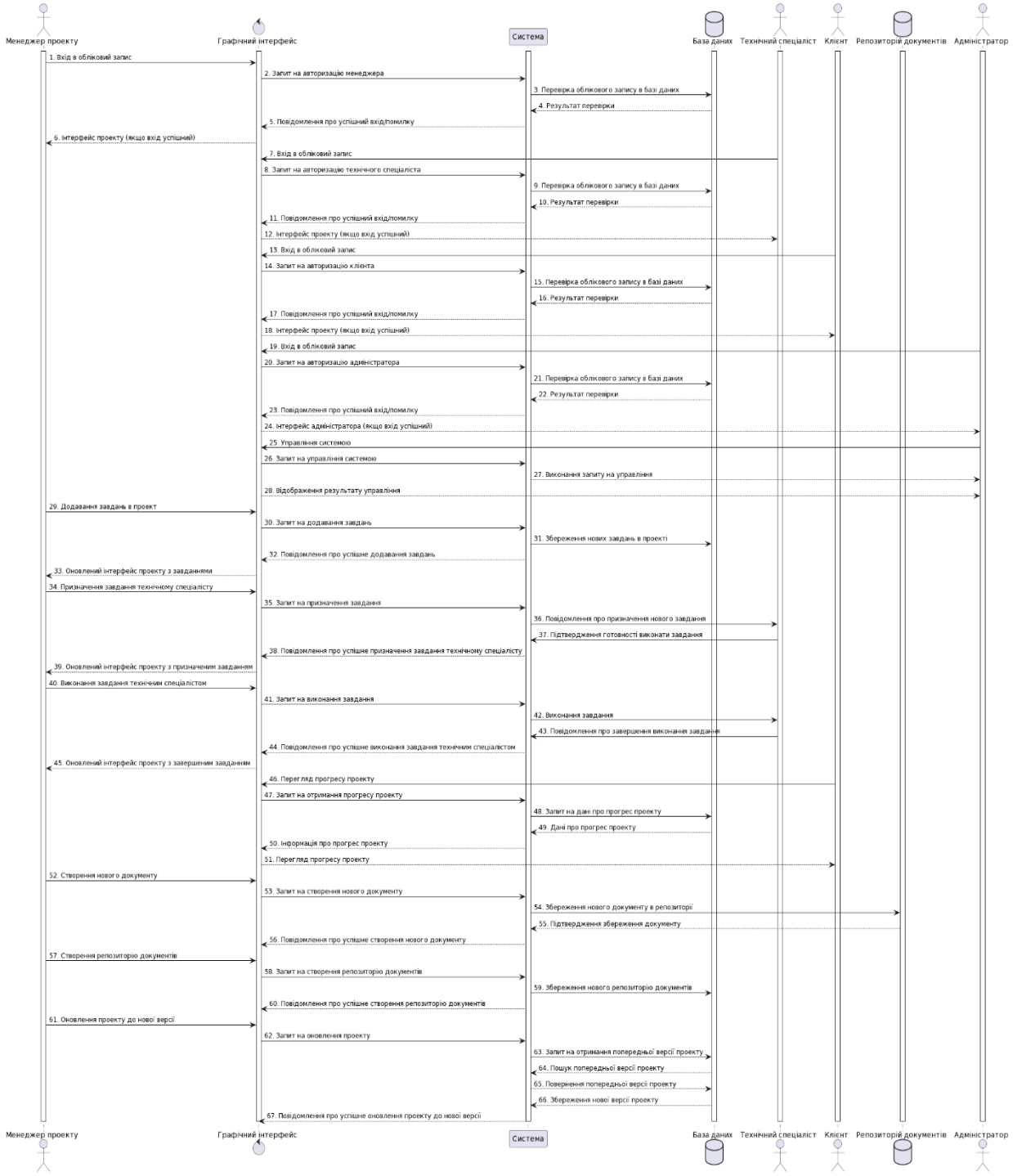


Рисунок 2.3 - Діаграма послідовності

– виконання завдань: Технічний спеціаліст виконує призначені завдання та оновлює їх стан;

– перегляд прогресу проекту: Клієнт переглядає прогрес проекту через графічний інтерфейс;

– створення документів: Менеджер створює нові документи, які зберігаються в репозиторії;

– оновлення проекту до нової версії: Менеджер ініціює оновлення проекту, отримує попередню версію з репозиторію, зберігає нову версію та оновлює проект.

Основні учасники та компоненти включають:

1. Менеджер проекту: Керує проектом, створює та редагує завдання, оновлює проект.

2. Адміністратор: Керує системою, здійснює її адміністування

3. Графічний інтерфейс: Забезпечує взаємодію з користувачами (менеджером, технічним спеціалістом, клієнтом) через графічний інтерфейс.

4. Система: Ядро системи, що обробляє запити та забезпечує виконання операцій з даними та управлінням проектом.

5. База даних: Зберігає інформацію про користувачів, проекти, завдання та інші деталі, необхідні для роботи системи.

6. Технічний спеціаліст: Виконує завдання, призначені менеджером, та оновлює стан завдань.

7. Клієнт: Переглядає прогрес проекту та може взаємодіяти з інтерфейсом для отримання інформації.

8. Репозиторій документів: Зберігає документи та версії проекту.

Діаграма співпраці (рисунок 2.4) відображає узагальнену взаємодію різних учасників системи за допомогою графічного інтерфейсу та іншими механізмами.

Основні елементи діаграми:

Графічний інтерфейс: Контроль, який надає інтерфейс для взаємодії менеджера проекту та інших учасників з системою.

Система: Основний компонент, який обробляє та зберігає дані проектів.

База даних: Відображає місце, де зберігаються дані проекти та інші відомості.

Користувачі: Актори, які взаємодіють із системою через графічний інтерфейс

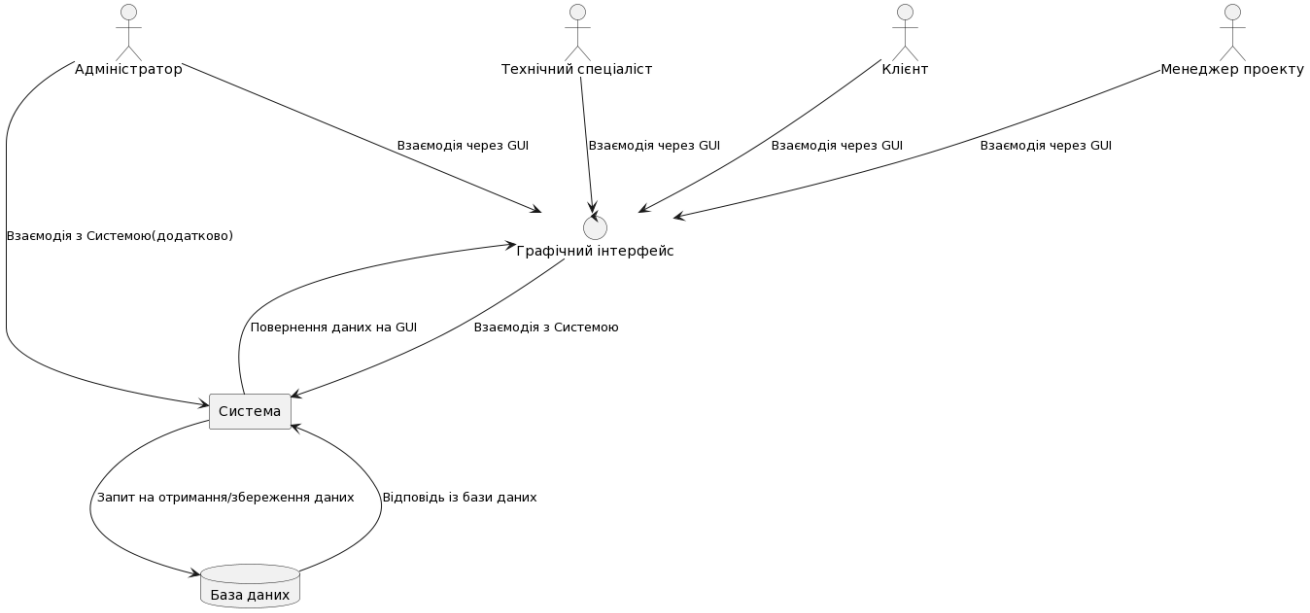


Рисунок 2.4 - Діаграма співпраці

Адміністратор: Користувач який може взаємодіяти із системо безпосередньо.

2.4 Діаграми діяльності

Діаграми діяльності моделюють різні аспекти взаємодії користувача з системою: створення проекту та завдань (рисунок 2.5), оновлення версії проекту (рисунок 2.6), створення документу для проекту (рисунок 2.7). Кожна діаграма має свій власний контекст та процес взаємодії користувача з системою.

На діаграмі (рисунок 2.5) зображено процес створення проєкту у системі управління завданнями. Все починається з авторизації користувача після входу до системи. Далі користувач обирає опцію «Створити новий проєкт». Якщо проєкт створено, система пропонує заповнити інформацію про нього. У випадку, якщо інформація не заповнена, користувач отримує повідомлення з проханням це зробити. Якщо інформацію внесено, користувач переходить до додавання завдань у межах створеного проєкту.

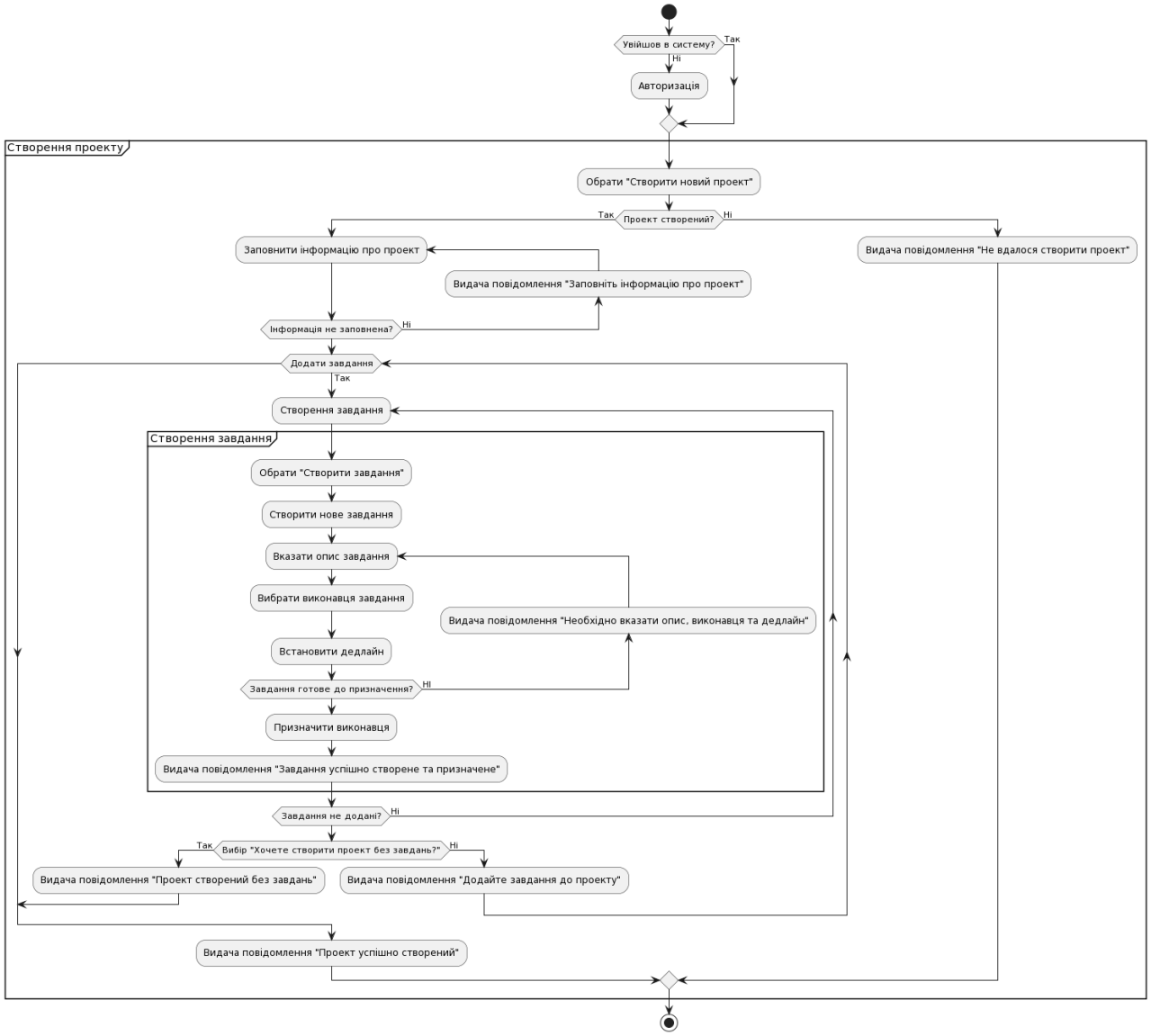


Рисунок 2.5 - Діаграма діяльності «Створення проекту та завдань»

Процес створення завдання передбачає вибір опції «Створити завдання», після чого користувач створює нове завдання, вказує його опис і призначення. Якщо ці дані не зазначено, система повідомляє про необхідність їх заповнення. Після введення деталей, призначення виконавця та збереження, система видає повідомлення про успішне створення завдання.

Після завершення додавання завдань користувач має змогу зберегти проєкт. Якщо завдання не додано, система попереджає, що проєкт створено без завдань і пропонує додати хоча б одне. У підсумку, після виконання всіх кроків, система повідомляє, що проєкт успішно створено.

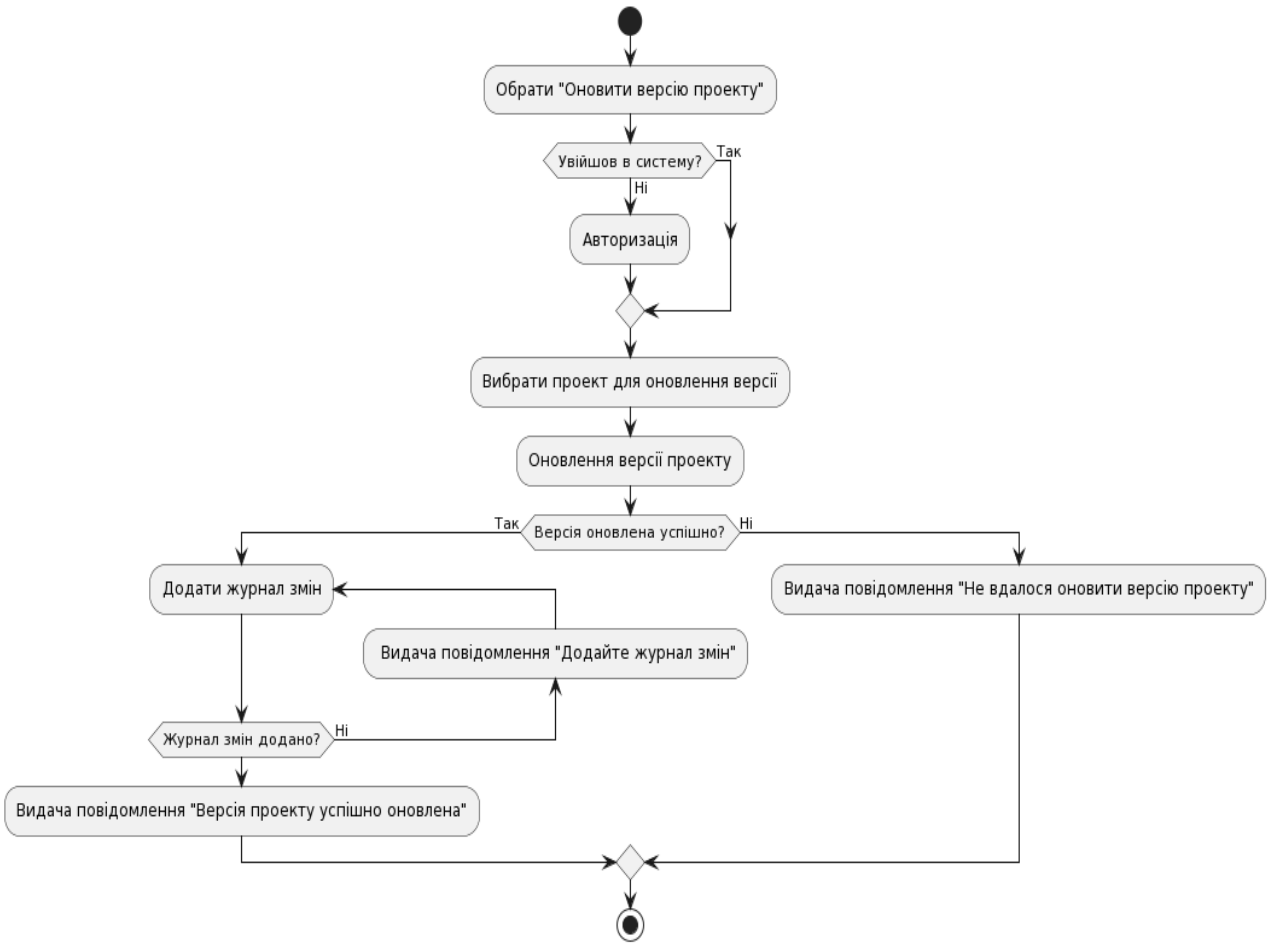


Рисунок 2.6 - Діаграма діяльності «Оновлення версії проекту»

На діаграмі (рисунок 2.6) зображено процес оновлення версії проєкту. Все починається з вибору опції «Оновити версію проєкту». Далі перевіряється, чи користувач увійшов у систему. Якщо ні, відбувається авторизація. Після успішного входу користувач обирає проєкт для оновлення версії. Система намагається оновити версію проєкту.

Якщо оновлення відбулося успішно, користувачеві пропонується додати журнал змін. У випадку, якщо журнал не додано, система видає повідомлення «Додайте журнал змін». Коли журнал змін успішно додано, користувач отримує повідомлення «Версія проєкту успішно оновлена». Якщо ж оновлення не вдалося, система повідомляє про це через повідомлення «Не вдалося оновити версію проєкту».

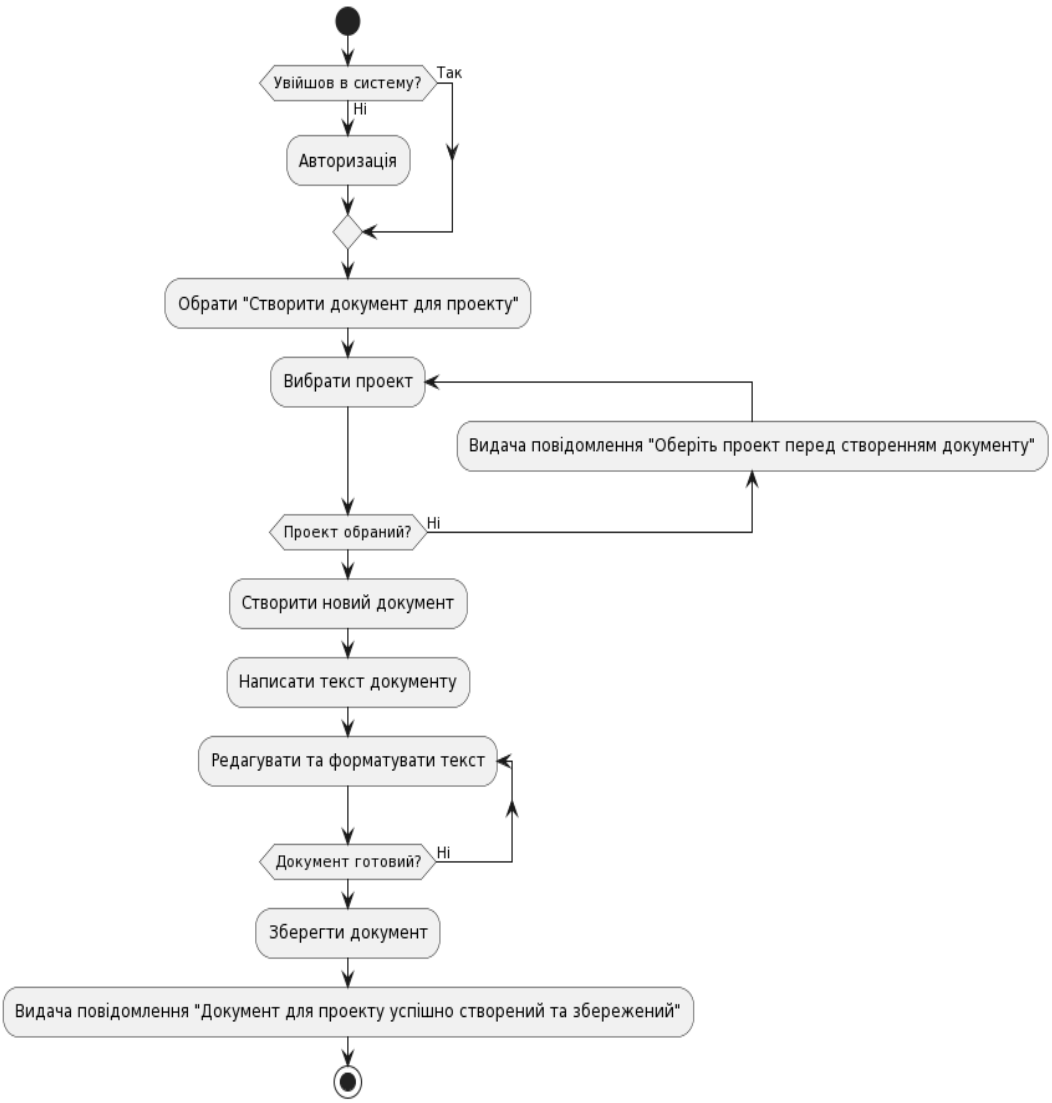


Рисунок 2.7 - Діаграма діяльності «Створення документу для проекту»

На діаграмі (рисунок 2.7) показано процес створення документа для проєкту. Користувач починає з перевірки, чи увійшов він у систему. Якщо ні, виконується авторизація. Після цього обирається опція «Створити документ для проєкту». Далі користувач має вибрати відповідний проєкт. Якщо проєкт не обрано, система видає повідомлення «Оберіть проєкт перед створенням документу».

Після вибору проєкту створюється новий документ. Користувач вводить текст документа, після чого може його редагувати та форматувати. Якщо документ вважається готовим, його зберігають. Після збереження система видає повідомлення «Документ для проєкту успішно створений та збережений».

2.5 Діаграма класів

Діаграма класів (рисунок 2.8) моделює структуру системи, ілюструючи класи, їх атрибути, методи та взаємозв'язки між класами.

Перелік класів системи:

1. Project (Проект). Керує інформацією про проект, включаючи ID, назву, опис, дати початку та закінчення, пов'язані задачі та управління документацією.

2. Task (Задача). Представляє задачу проекту з атрибутами, такими як ID, назва, опис, статус, термін виконання та відповідальний.

3. User (Користувач). Представляє користувача системи з ID, ім'ям користувача, паролем, роллю та можливістю обміну повідомленнями.

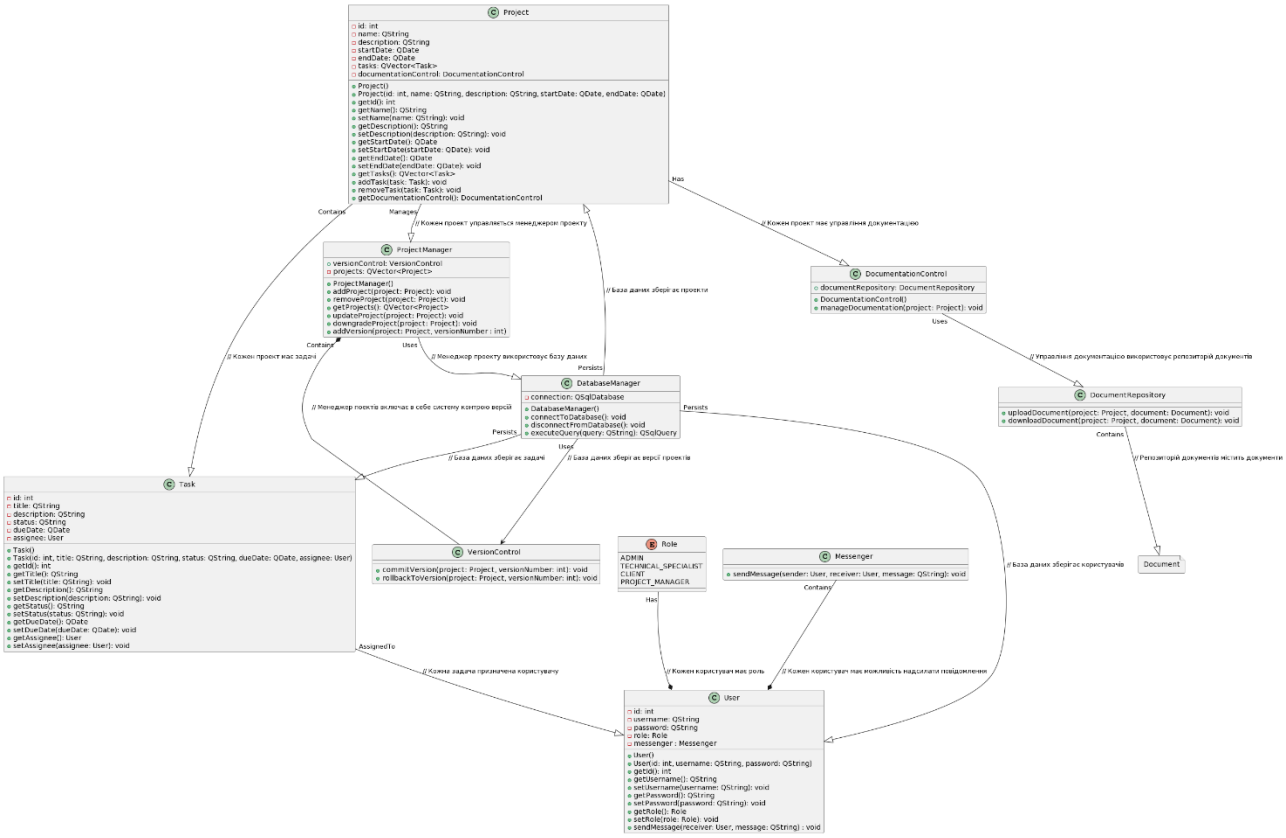


Рисунок 2.8 - Діаграма класів

4. ProjectManager (Менеджер проекту). Керує проектами, включаючи контроль версій, додавання, видалення, отримання, оновлення, зниження версії та додавання версії.

5. Database Manager (Менеджер бази даних). Обробляє операції бази даних, включаючи підключення, відключення та виконання запитів.

6. Documentation Control (Управління документацією). Керує документацією проекту, використовуючи репозитарій документів та полегшуючи оновлення документації.

7. VersionControl (Контроль версій). Керує версіями проекту, дозволяючи фіксацію версій та їх відновлення.

8. DocumentRepository (Репозитарій документів). Зберігає та отримує документи проекту, полегшуючи завантаження та вивантаження документів.

9. Document (Документ). Представлення певного файлу.

10. Role (Роль). Вибраний перелік ролей користувача, таких як адміністратори, технічні спеціалісти, клієнти, менеджери проекту

11. Messenger (Повідомлення). Забезпечує обмін повідомленнями користувачів, дозволяючи їм надсилати повідомлення одне одному.

2.6 Діаграма станів та переходів

Діаграма станів та переходів (рисунок 2.9) визначає основні етапи роботи з проектом, включаючи створення проекту, додавання завдань, документації та оновлення версії.

На діаграмі зображено загальний процес керування проєктом, який охоплює створення проєкту, додавання завдань, оновлення версії та роботу з документацією. Спочатку користувач може вибрати створення нового проєкту, після чого відбувається заповнення інформації. Якщо інформація заповнена правильно, проєкт створюється, і стає можливим додавання завдань.

При створенні завдань виконується кілька кроків: введення опису, призначення виконавця, визначення дедлайну та введення деталей. Якщо всі поля заповнено, завдання зберігається та успішно додається до проєкту.

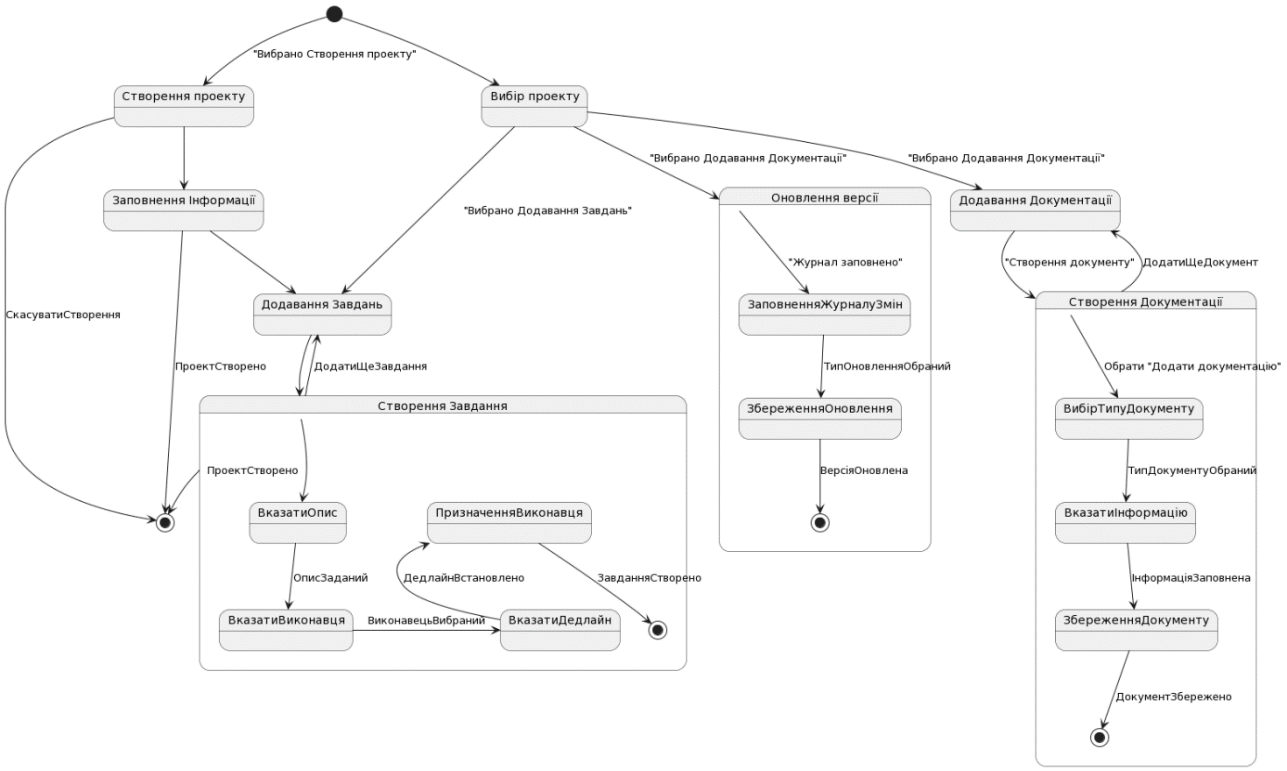


Рисунок 2.9 - Діаграма станів та переходів

Після вибору проєкту користувач має змогу оновити його версію. Для цього додається журнал змін, який зберігається після заповнення, і версія вважається оновленою.

Також користувач може додавати документацію до проєкту. У процесі створення документа обирається його тип, вводиться відповідна інформація, після чого документ зберігається. Якщо всі дії виконано успішно, система підтверджує збереження документа. Таким чином, діаграма демонструє взаємозв’язані етапи повного життєвого циклу управління проєктом.

2.7 Діаграма розгортання

Діаграма розгортання (рисунок 2.10) зображує фізичні компоненти системи та їх взаємодію.

Основні компоненти цієї діаграми:

– Database (Хмара бази даних): Це фізичний сервер або хмарний сервіс, який використовується для збереження даних системи;

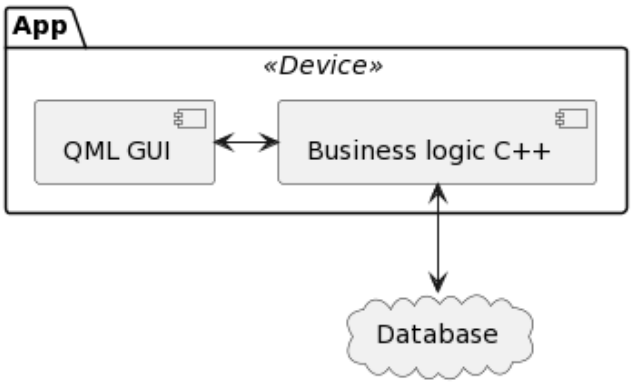


Рисунок 2.10 - Діаграма розгортання

– Business logic C++ (Бізнес-логіка на C++): Це компонент, де знаходиться бізнес-логіка програми, написана на мові програмування C++. Цей компонент взаємодіє з базою даних для збереження та отримання даних;

– QML GUI (Графічний інтерфейс QML): Це компонент, який відповідає за графічний інтерфейс користувача, реалізований з використанням QML (Qt Meta-Object Language). Він взаємодіє з бізнес-логікою для відображення та оновлення даних.

2.8 Опис структури бази даних

База даних необхідна для ефективного управління проєктами, зокрема для контролю версій програмного забезпечення, розподілу завдань між користувачами та ведення історії змін. Вона також забезпечує збереження структури ролей, внутрішню комунікацію та прозорість у роботі команди. Завдяки цій системі можна централізовано керувати всіма етапами життєвого циклу проєкту.

На рисунку 2.11 зображена структура бази даних, яка стосується управління проєктами, версіями програмного забезпечення, завданнями, користувачами та комунікацією між ними. Нижче наведено детальний опис кожної таблиці та їхні взаємозв’язки.

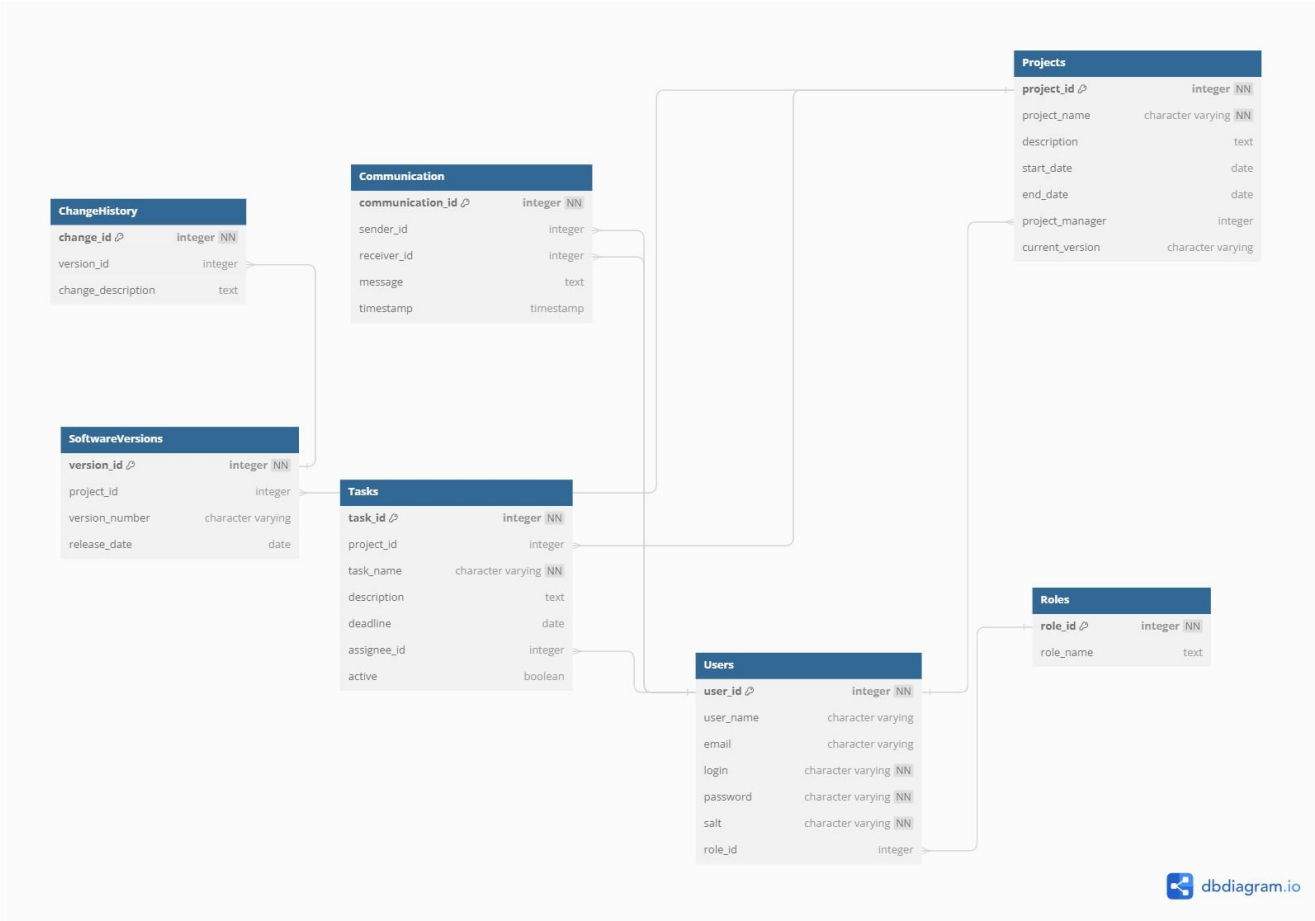


Рисунок 2.11 - Структура бази даних Postgresql

База даних складається з таких основних таблиць

- проєкти;
- версії програмного забезпечення;
- історія змін;
- завдання;
- користувачі;
- ролі;
- комунікація.

Взаємозв’язки між таблицями:

1. Projects ↔ Users. Поле project_manager вказує на Users.user_id — менеджер проєкту.

2. Projects ↔ Tasks. Одному проєкту можуть відповідати кілька завдань (project_id в Tasks).

3. Tasks ↔ Users. Поле assignee_id вказує, хто призначений на виконання завдання.

4. Projects ↔ SoftwareVersions. Один проєкт може мати багато версій ПЗ (project_id).

5. SoftwareVersions ↔ ChangeHistory. Кожна версія може мати багато змін (version_id).

6. Users ↔ Roles. Кожному користувачу призначена одна роль (role_id).

7. Communication ↔ Users. Поля sender_id і receiver_id — внутрішня комунікація між користувачами.

Таблиця 2.1 Проєкти зберігає основну інформацію про проєкти, включаючи назву, опис, дати початку та завершення. Кожен проєкт має унікальний ідентифікатор (project_id). Менеджер проєкту вказується через зовнішній ключ (project_manager), який пов'язаний з користувачами. Також зазначається поточна версія програмного забезпечення (current_version). Ця таблиця є центральною та зв'язана з багатьма іншими, зокрема Tasks та SoftwareVersions.

Таблиця 2.1 – Опис таблиці Проєкти

Назва поля	Тип даних	Значення
project_id	integer	Унікальний ідентифікатор проєкту
project_name	character varying	Назва проєкту
description	text	Опис проєкту
start_date	date	Дата початку
end_date	date	Дата завершення
project_manager	integer	Посилання на Users.user_id — менеджер проєкту
current_version	character varying	Поточна версія ПЗ

Таблиця 2.2 Версії проєкту містить інформацію про всі версії програмного забезпечення, пов'язані з певним проєктом. Кожна версія має унікальний ідентифікатор (version_id), номер версії (version_number) та дату релізу (release_date). Зв'язок із таблицею Projects реалізується через поле project_id. Вона також слугує джерелом для історії змін у таблиці ChangeHistory. Ця таблиця дозволяє відслідковувати розвиток проєктів у часі.

Таблиця 2.2 – Опис таблиці Версії Проєкту

Назва поля	Тип даних	Значення
version_id	integer	Унікальний ідентифікатор версії
project_id	integer	Посилання на Projects.project_id
version_number	character varying	Номер версії
release_date	date	Дата релізу

Таблиця 2.3 Історія змін містить деталізовану інформацію про зміни, внесені у певну версію ПЗ. Кожна зміна має унікальний ідентифікатор (change_id) та опис (change_description). Поле version_id вказує, до якої версії ПЗ належить зміна, утворюючи зовнішній зв'язок із таблицею SoftwareVersions. Ця таблиця допомагає відстежувати, що саме було оновлено або виправлено у кожному релізі. Вона є корисною для аналізу історії розробки та підтримки прозорості.

Таблиця 2.3 – Опис таблиці Історія Змін

Назва поля	Тип даних	Значення
change_id	integer	Унікальний ідентифікатор зміни
version_id	integer	Посилання на SoftwareVersions.version_id
change_description	text	Опис зміни

У таблиці 2.4 описується завдання, що належать до певного проєкту. Кожне завдання має унікальний ідентифікатор (task_id), назву, опис, дедлайн та призначеного виконавця (assignee_id). Поле project_id вказує на проєкт, до якого належить завдання, створюючи зовнішній зв'язок із таблицею Projects. Крім того, є булеве поле active, яке вказує на актуальність завдання. Таблиця дозволяє ефективно розподіляти та контролювати робочі задачі між учасниками проєкту.

Таблиця 2.4 – Опис таблиці Завдання

Назва поля	Тип даних	Значення
task_id	integer	Унікальний ідентифікатор завдання
project_id	integer	Посилання на Projects.project_id
task_name	character varying	Назва завдання
description	text	Опис завдання
deadline	date	Термін виконання
assignee_id	integer	Посилання на Users.user_id — хто виконує
active	boolean	Активне чи ні

Таблиця 2.5 описує користувачів та містить інформацію про всіх користувачів системи, включаючи логін, email, ім’я та пароль (разом із сіллю). Кожен користувач має унікальний ідентифікатор (user_id) і належить до певної ролі (role_id). Користувачі можуть бути менеджерами проєктів, виконавцями завдань або учасниками комунікацій. Дані про логін і пароль захищено відповідно до стандартів безпеки. Таблиця активно використовується в багатьох інших, таких як Projects, Tasks, Communication.

Таблиця 2.5 – Опис таблиці Користувачі

Назва поля	Тип даних	Значення
user_id	integer	Унікальний ідентифікатор користувача
user_name	character varying	Ім'я користувача
email	character varying	Email
login	character varying	Логін
password	character varying	Хеш пароля
salt	character varying	Сіль для пароля
role_id	integer	Посилання на Roles.role_id

Таблиця 2.6 Ролі містить перелік ролей, які можуть бути призначені користувачам, таких як "адміністратор", "розробник", "менеджер" тощо. Кожна роль має унікальний ідентифікатор (role_id) і назву (role_name). Таблиця дозволяє встановити рольову модель доступу в системі. Зв’язана з таблицею Users, що дозволяє фільтрувати користувачів за функціоналом. Вона є важливою частиною системи контролю доступу.

Таблиця 2.6 – Опис таблиці Ролі

Назва поля	Тип даних	Значення
role_id	integer	Унікальний ідентифікатор ролі
role_name	text	Назва ролі (наприклад, “адміністратор”, “розробник”, “менеджер”).

Таблиця 2.7 Комунікація зберігає повідомлення між користувачами системи. Кожне повідомлення має унікальний ідентифікатор (communication_id), текст (message), дату й час відправлення (timestamp). Поля sender_id і receiver_id є зовнішніми ключами до таблиці Users, що вказують, хто відправив і хто отримав повідомлення. Ця таблиця дозволяє реалізувати внутрішню систему обміну

інформацією. Вона може бути використана для аудиту або моніторингу комунікацій.

Таблиця 2.7 – Опис таблиці Комунікація

Назва поля	Тип даних	Значення
communication_id	integer	Унікальний ідентифікатор повідомлення
sender_id	integer	Посилання на Users.user_id — хто відправив
receiver_id	integer	Посилання на Users.user_id — хто отримав
message	text	Текст повідомлення
timestamp	timestamp	Час відправлення

У 2-му розділі було створено повний набір UML-діаграм, які відображають логіку системи, її структуру та сценарії використання. Це дозволило візуалізувати взаємодії між модулями та учасниками, а також сформувати чітку архітектуру програмного продукту. Кожен компонент має своє функціональне призначення, що забезпечує масштабованість і гнучкість системи. А також проведений детальний опис розробленої бази даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-СЕРВІСУ ДЛЯ КОНТРОЛЮ ТА УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура коду

Архітектура коду організована шаблоном проектування Модель-Вид-Контролер (MVC) (рисунок 3.1).

Модель (Model). Класи Admission, ProjectManagement і UserManagement відповідають за роботу з даними та бізнес-логікою. Admission включає функціонал для автентифікації та створення нових користувачів. ProjectManagement відповідає за управління проектами та завданнями. UserManagement дозволяє управляти користувачами та їх ролями.

Вид (View). Графічний інтерфейс створений за допомогою QML (Qt Meta-Object Language). Для зв'язку QML-інтерфейсу з бізнес-логікою використовуються об'єкти QQuickWidget, а також мовна конструкція сигнали та слоти, яка є варіацією паттерна Спостерігач(Observer), сигнал призначений для представлення певної дії, що відбувається, слот призначений для представлення певної функції, яка має виконатись, коли відбувається сигнал. Взаємодія користувача з QML-інтерфейсом спричиняє сигнали, які в свою чергу викликають слот.

Контролер (Controller). Клас MainWindow виступає контролером, який взаємодіє з моделлю та оновлює представлення. Контролер приймає сигнали від QML-інтерфейсу, та викликає відповідні слоти. Слоти взаємодіють з об'єктами моделі для вибірки додавання чи оновлення даних Postgresql.

Бізнес-логіка в коді реалізує функціональність модуля для управління проектами та завданнями.

1. Автентифікація користувачів:

- метод logIn класу Admission відповідає за автентифікацію користувачів за їх логінами та паролями. Методи encryptPassword та getSalt призначені для хешування паролю за допомогою «солі» даних, та її створення.

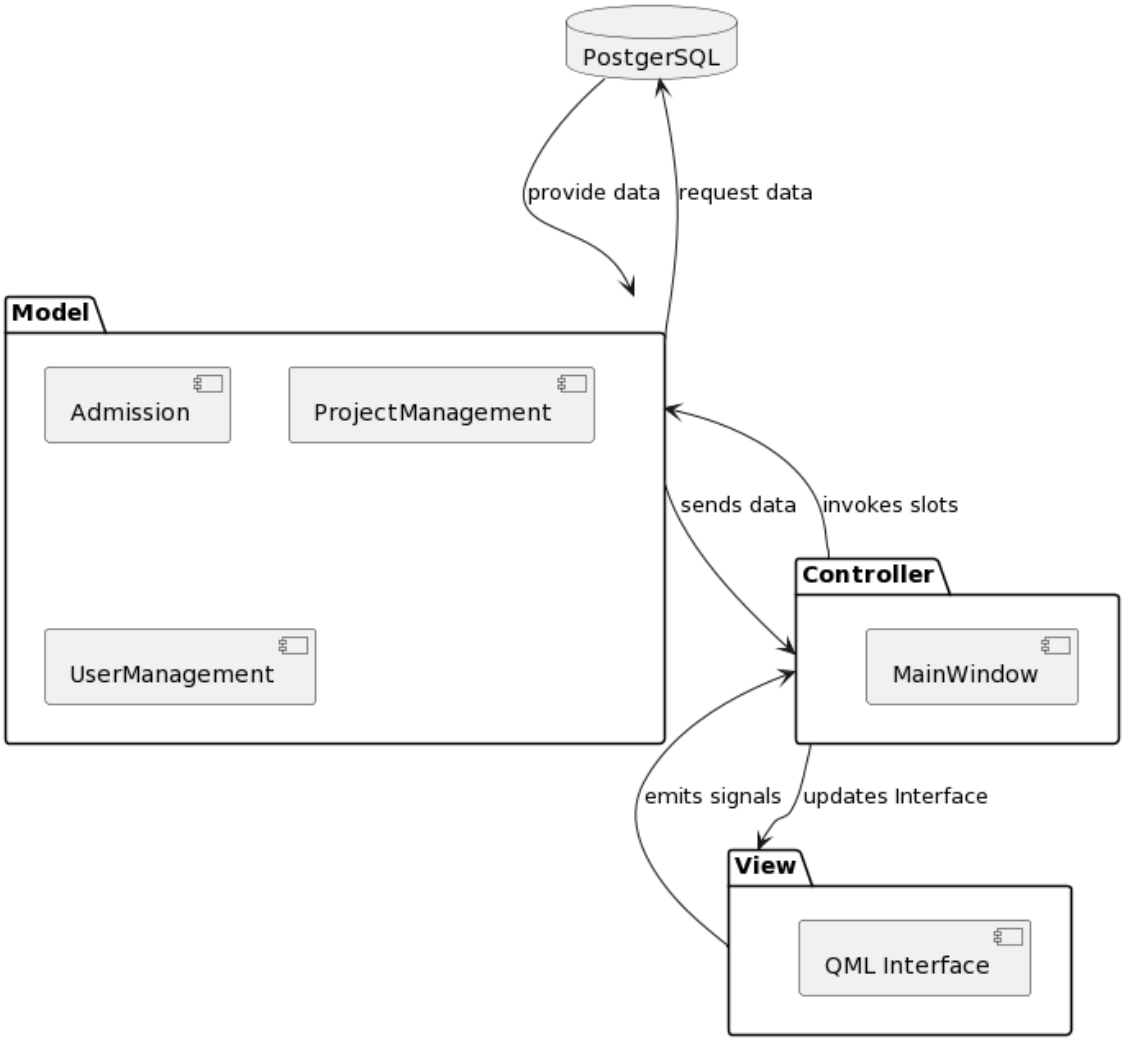


Рисунок 3.1 - Візуалізація MVC

2. Створення нових користувачів:

- метод signUp класу Admission відповідає за реєстрацію нових користувачів.

3. Управління проектами:

- методи addProject, removeProject класу ProjectManagement додають та видаляють проекти відповідно;

- метод getProjectsList повертає список проектів для відображення у QListView.

4. Управління завданнями:

- методи addTask, removeTask, changeTaskStatus класу ProjectManagement додають, видаляють та змінюють статус завдань відповідно;

- метод getProjectTasks повертає список завдань для конкретного проекту;

- метод filterModel слугує для фільтрації моделей даних.

5. Оновлення інформації у графічному інтерфейсі:

- ряд методів, таких як onCurrentProjectChanged, onGoToProject, onGoToTask, оновлюють інформацію у графічному інтерфейсі і відображають деталі проектів та завдань.

6. Робота із користувачами:

- методи getUsersByRole, getUserName, getProjectUsers, getUserId класу UserManagement надають інформацію про користувачів та їх ролі у системі.

7. Робота із моделями даних:

- Методи convertToModel та filterModel використовуються для роботи з моделями даних, а саме для перетворення об'єкта pqxx::result (результат вибірки з бази даних PostgreSQL) у модель даних та стовпцем.

У коді використовуються конектори для встановлення зв'язків між сигналами та слотами у фреймворку Qt (рисунок 3.2).

```
QObject::connect(ui->entry->rootObject(), SIGNAL(logIn(QString, QString)), this, SLOT(onLogIn(QString, QString)));
QObject::connect(ui->entry->rootObject(), SIGNAL(signUp(QString, QString)), this, SLOT(onSignUp(QString, QString)));
QObject::connect(ui->projects->rootObject(), SIGNAL(currentProjectChanged(QString)), this, SLOT(onCurrentProjectChanged(QString)));
QObject::connect(ui->projects->rootObject(), SIGNAL(goToProject(QString)), this, SLOT(onGoToProject(QString)));
QObject::connect(ui->projects->rootObject(), SIGNAL(goToTask(QString)), this, SLOT(onGoToTask(QString)));
QObject::connect(ui->projects->rootObject(), SIGNAL(addProject(QString)), this, SLOT(onAddProject(QString)));
QObject::connect(ui->projects->rootObject(), SIGNAL(removeProject(QString)), this, SLOT(onRemoveProject(QString)));
QObject::connect(ui->project->rootObject(), SIGNAL(addTask(QString, QString)), this, SLOT(onAddTask(QString, QString)));
QObject::connect(ui->project->rootObject(), SIGNAL(completeTask(QString, QString)), this, SLOT(onCompleteTask(QString, QString)));
QObject::connect(ui->project->rootObject(), SIGNAL(restoreTask(QString, QString)), this, SLOT(onRestoreTask(QString, QString)));
QObject::connect(ui->project->rootObject(), SIGNAL(removeTask(QString, QString)), this, SLOT(onRemoveTask(QString, QString)));
QObject::connect(ui->task->rootObject(), SIGNAL(saveTask(int, QString, QString, QString, QString, QString)), this, SLOT(onSaveTask(int, QString, QString, QString, QString, QString)));
QObject::connect(ui->project->rootObject(), SIGNAL(saveProject(int, QString, QString, QString, QString)), this, SLOT(onSaveProject(int, QString, QString, QString, QString)));
QObject::connect(ui->task->rootObject(), SIGNAL(back()), this, SLOT(onGoToProjects()));
QObject::connect(ui->project->rootObject(), SIGNAL(back()), this, SLOT(onGoToProjects()));
QObject::connect(ui->projects->rootObject(), SIGNAL(back()), this, SLOT(onGoToEntry()));
```

Рисунок 3.2 - З’єднання між всіма сигналами та слотами

Створення фільтру, який дозволяє фільтрувати дані в моделі за певним значенням та Конектори реалізуються через функцію QObject::connect. Ця функція дозволяє зв'язувати сигнали, зі слотами.

3.2 Підключені бібліотеки та фреймворки

pqxx/pqxx: Це бібліотека C++ для взаємодії з PostgreSQL базами даних. Основні функції бібліотеки дозволяють створювати та виконувати SQL-запити до бази даних, обробляти результати запитів та керувати з'єднаннями з базою даних. Клас DataBaseConnection використовує її для встановлення та управління підключенням до бази даних. У класі DataBaseConnection використовується паттерн Singleton, який гарантує, що клас має тільки один екземпляр і надає глобальний спосіб отримання цього екземпляра через метод connectDataBase.

libconfig.h++: Це бібліотека C++ для роботи з конфігураційними файлами. У цьому випадку, вона використовується для зчитування параметрів підключення до бази даних з файлу "databaseConfig.cfg" (рисунок 3.3). З цими параметрами потім встановлюється підключення до PostgreSQL бази даних за допомогою бібліотеки pqxx.

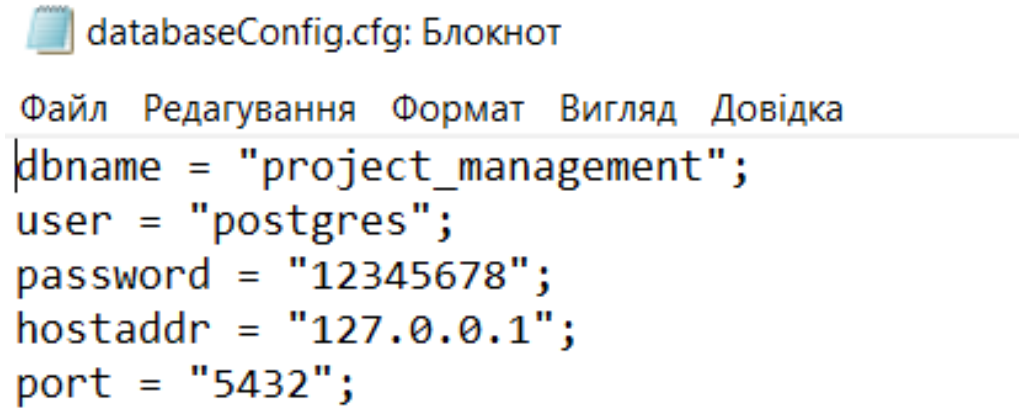


Рисунок 3.3 - Конфігураційний файл для підключення до бази даних

chrono: Це бібліотека C++ для роботи з часом. Використовується для отримання часового мітки у мілісекундах для генерації "солі" даних для паролю користувача.

Фреймворк Qt. Це набір інструментів та бібліотек для розробки крос-платформених програм на мові програмування C++.

Інструменти, що належать до Qt. У коді використовується клас QMessageBox з фреймворку Qt для виведення вікон повідомлень під час обробки помилок.

Клас QCryptographicHash з бібліотеки Qt використовується для хешування паролю та генерації "солі" даних (рисунок 3.4). Він надає можливість використовувати різні алгоритми хешування, але в роботі використовується Sha512.

```
}

std::string UserAdmission::getSalt()
{
    std::chrono::milliseconds timestamp = std::chrono::duration_cast<std::chrono::milliseconds>(
        std::chrono::system_clock::now().time_since_epoch()
        );

    hash.reset();
    hash.addData(QByteArray(std::to_string(timestamp.count()).c_str()).constData());

    return hash.result().toHex().toStdString();
}

std::string UserAdmission::encryptPassword(std::string password, std::string salt)
{
    hash.reset();

    hash.addData(password + salt);

    return hash.result().toHex().toStdString();
}
```

Рисунок 3.4 - Хешування паролю та створенні «солі» даних для нього

У коді використовуються класи QStandardItemModel та QSortFilterProxyModel (рисунок 3.5), які належать до фреймворку Qt і призначені для роботи з моделями даних у графічному інтерфейсі користувача. QStandardItemModel використовується для конвертації формату даних бібліотеки pqxx/pqxx у формат, прийнятний для фреймворку Qt, а саме у модель даних, яка може бути представлена у різних виглядах (список, таблиця, дерево тощо). QSortFilterProxyModel використовується для сортування та фільтрації моделей даних фреймворку Qt.

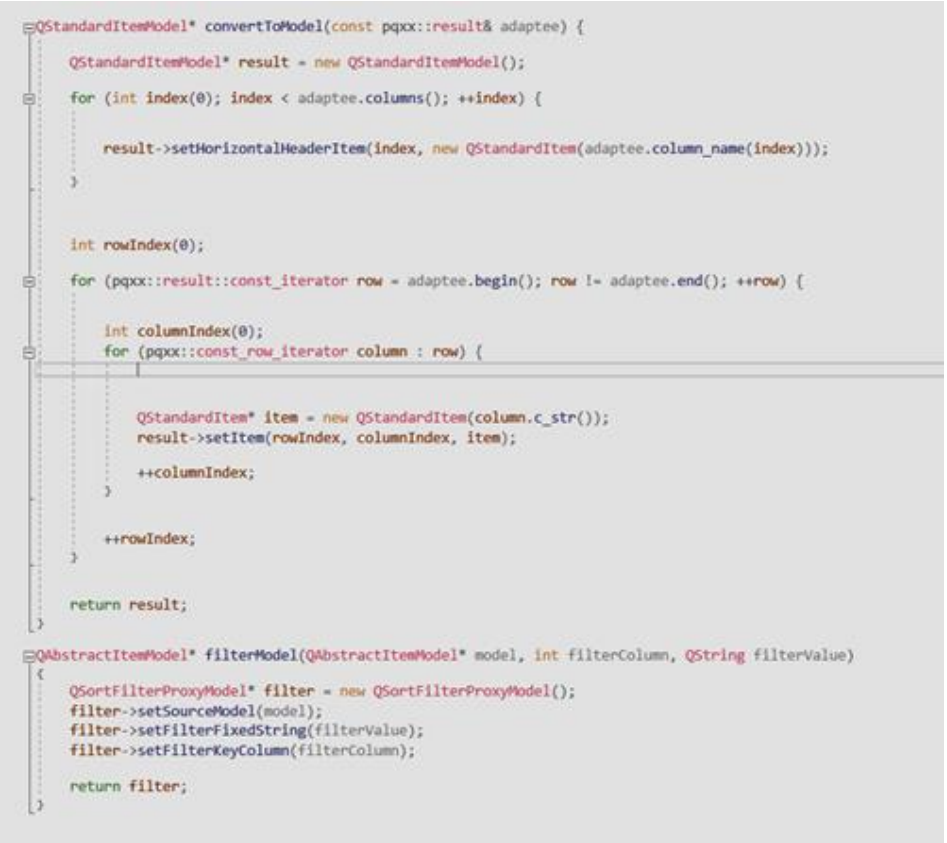


Рисунок 3.5 - Методи для конвертації результату SQL-запиту в модель даних, та фільтрації моделі

В коді використовуються елементи QML для взаємодії з C++ кодом за допомогою QQuickWidget, QQmlContext та QQuickItem. QQuickWidget вбудовує QML-інтерфейс в Qt-додаток. QQmlContext використовується для взаємодії між C++ і QML кодом, передачі даних з бази даних в графічний інтерфейс. QQuickItem використовується для представлення окремого елемента QML та взаємодії з ним в C++ коді за допомогою сигналів та слотів, а саме з'єднання сигналів QML-інтерфейс зі слотами C++.

Також у коді використовують типи даних з фреймворку Qt, такі як QString або QByteArray, вони являють собою типи даних із звичайного C++ з додатковим функціоналом, також більшість інструментів Qt працюють саме з такими типами даних.

Декларативна мова для опису користувацького інтерфейсу QML (Qt Meta-Object Language) використовується в проекті для опису структури і вигляду

інтерфейсу. Кожна сторінка додатку реалізована в окремому файлі. В користувацькому інтерфейсі використовуються такі об’єкти QML як:

Rectangle: Основний контейнер для інтерфейсу.

Text: Призначений для відображення тексту в QML-інтерфейсах

AddDialog, RemoveDialog, SaveDialog: Кастомні діалогові вікна, на базі елемента Dialog

MyTextField: Кастомне текстове поле на базі елемента TextField

MyComboBox: Кастомний випадаючий список на базі елемента ComboBox

MyButton1: Кастомна кнопка на базі елемента Button

MyList: Кастомний список на базі елементів ListView та MyButton1

3.3 Налаштування для тестового розгортання (середовища, параметри)

Тестове розгортання проведено в середовищі Microsoft Visual Studio на локальному комп’ютері. База даних PostgreSQL зберігається локально.

Параметри проекту:

Тип проекту: додаток (.exe), Проект призначений для створення виконуваного файлу (.exe) (рисунок 3.6), що є звичайним для самостійних додатків у середовищі Windows.

Конфігурація: Release, Налаштування проекту для компіляції в режимі Release, означає, до розгортання оптимізованої та виробничої версії додатка.

Цільова платформа: x64, Проект призначений для роботи на 64-бітній платформі, що дозволяє використовувати більше пам'яті та оптимізує продуктивність для 64-бітних систем.

Мова програмування: C++

Версія компілятора: Visual Studio 2019 (v142)

Стандарт мови C++: стандарт ISO C++17 (/std:c++17), Стандарт C++17 - це версія мови програмування C++, яка включає в себе ряд нововведень та покращень порівняно з попередніми стандартами. Деякі з ключових особливостей та змін в C++17 включають:

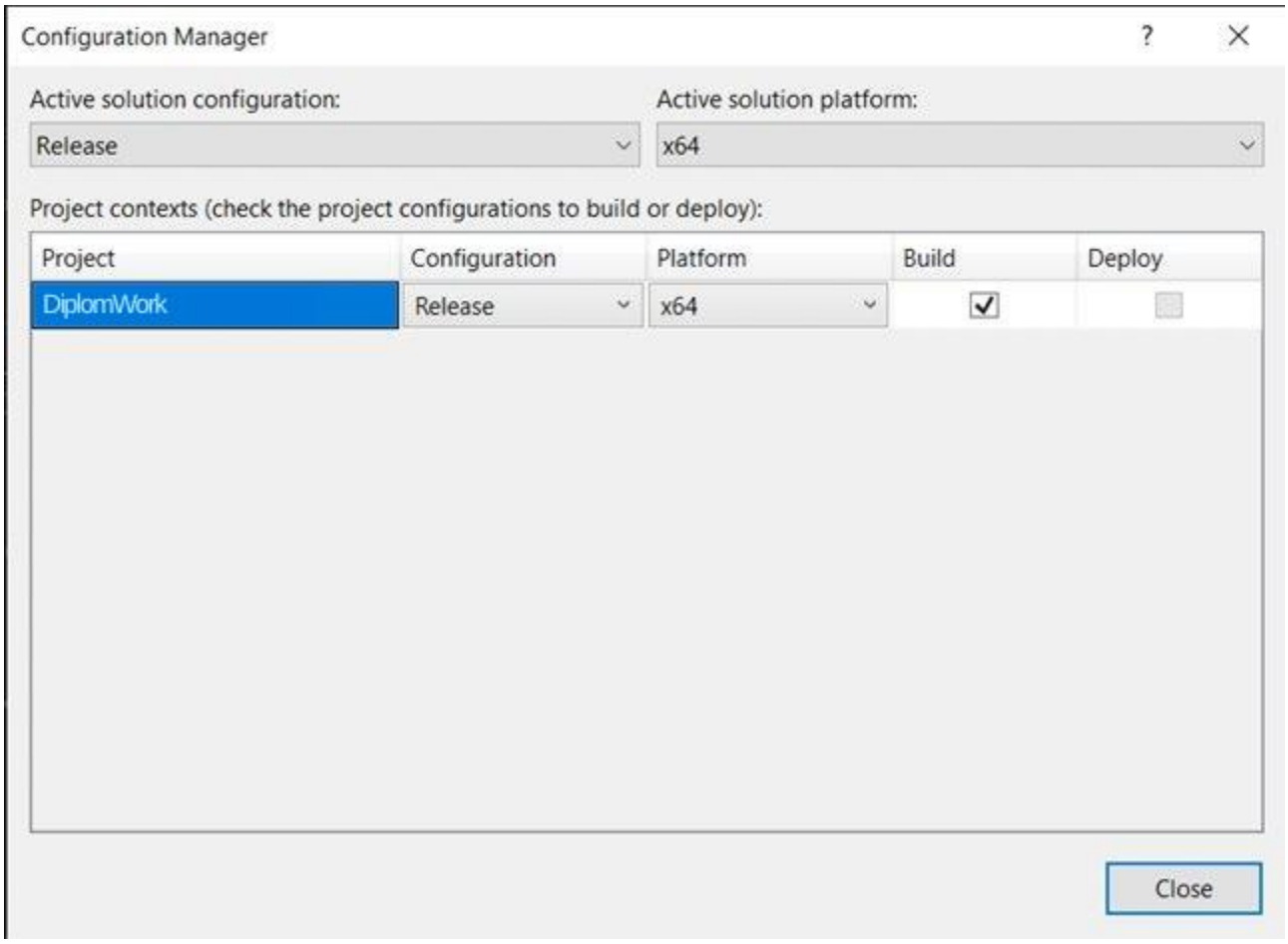


Рисунок 3.6 - Менеджер конфігурацій

– Structured Bindings (Структуровані зв'язування): Дозволяє зручний доступ до компонентів складених об'єктів, таких як кортежі чи масиви.

– Parallel Algorithms (Паралельні алгоритми): Додає можливість використання паралельних обчислень для деяких стандартних алгоритмів.

– If and Switch with Initializers (If та Switch з ініціалізаторами): Дозволяє ініціалізувати змінні прямо в умовних операторах if та switch.

– constexpr if (constexpr if): Новий механізм компіляції для умовної компіляції на етапі компіляції.

– std::optional та std::variant (std::optional та std::variant): Нові типи даних для роботи з відсутністю значень та варіативними типами.

Платформа: Windows

Версія Windows SDK: 10.0.19041.0, Windows Software Development Kit, включає в себе необхідні бібліотеки та інструменти для розробки в середовищі Windows.

Версія фреймворку Qt (рисунок 3.7): 6.4.3_msvc2019_64, Версія Qt фреймворку 6.4.3, яка скомпільована для використання з MSVC 2019 на 64-бітній платформі.

Версія розширення Microsoft Visual Studio для роботи з Qt : 3.4

Сторонні бібліотеки додано за допомогою менеджера пакетів vspkg.

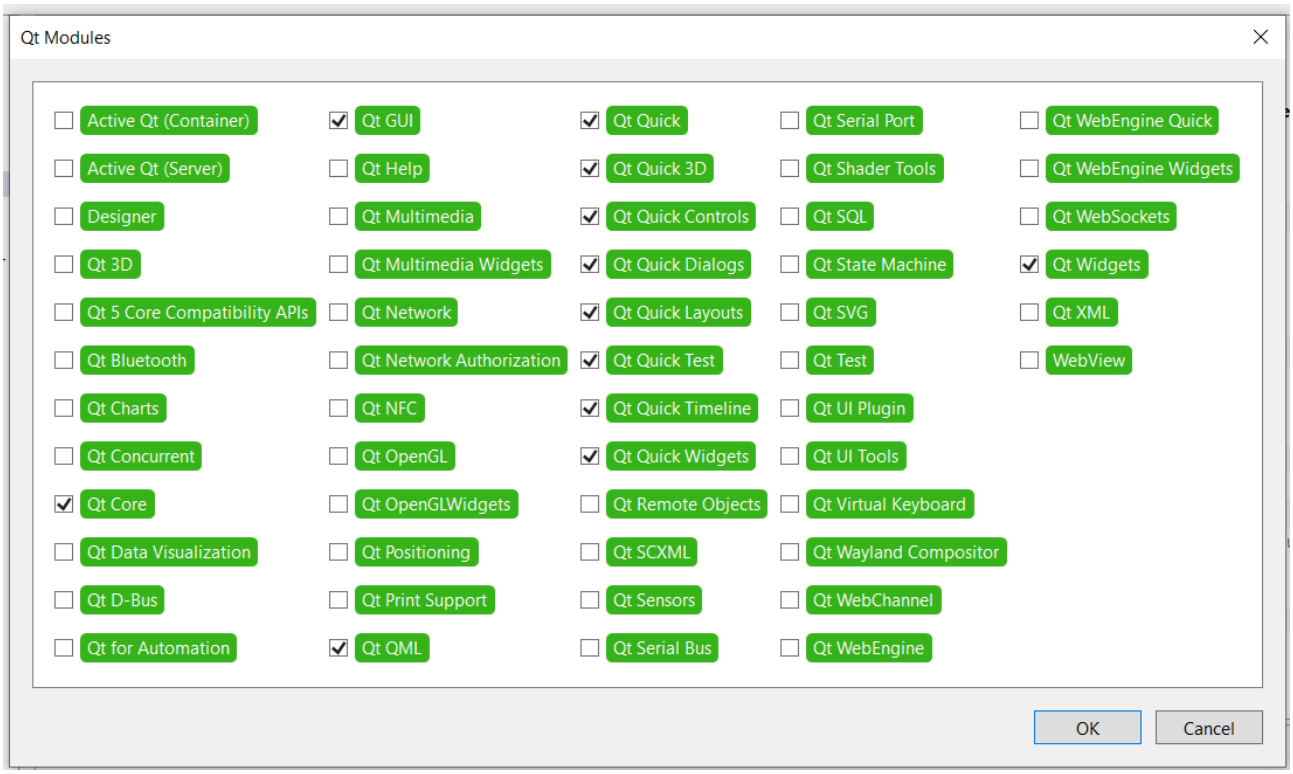


Рисунок 3.7 - Модулі Qt, використані в проекті

На рисунку 3.7 показано вікно вибору модулів Qt у середовищі розробки. Доступні модулі поділені на категорії та можуть бути увімкнені або вимкнені відповідно до потреб розробника. Зеленим кольором виділені модулі, які вже встановлено або увімкнено. Серед обраних модулів є Qt Core, Qt GUI, Qt QML, Qt Quick, Qt Quick Controls, Qt Quick Dialogs, Qt Quick Layouts, Qt Quick Test, Qt Quick Timeline, Qt Widgets, Qt State Machine, Qt SVG, Qt SQL, Qt Serial Port, Qt 3D, Qt WebSockets, Qt WebEngine Quick та кілька інших.

Ці модулі забезпечують базову функціональність (наприклад, Qt Core, Qt GUI, Qt Widgets), підтримку інтерфейсів користувача (Qt Quick, Qt QML, Qt Quick Controls), роботу з базами даних (Qt SQL), мережевими з'єднаннями (Qt WebSockets), графікою (Qt SVG, Qt 3D), та взаємодію з WebEngine.

Інші модулі, як-от Qt Charts, Qt Multimedia, Qt Bluetooth, Qt Sensors, Qt NFC — не позначені, отже не будуть включені до поточного проєкту. Користувач може додатково активувати їх перед натисканням кнопки «OK», щоб включити необхідну функціональність у свій додаток.

3.4 План перевірки та тестування

Цілі тестування

1. Функціональність:

- Перевірка коректності реєстрації та автентифікації користувачів.
- Тестування додавання, видалення та редагування проектів.
- Перевірка додавання, видалення та редагування завдань в межах проекту.
- Тестування взаємодії інтерфейсу з користувачем.

2. Зручність використання:

- Оцінка зручності використання графічного інтерфейсу за допомогою тестування.

Ручний підхід до тестування: Цей підхід використовується для перевірки функціональностей, взаємодії інтерфейсу, зручності використання та інших аспектів програмного забезпечення. Тестування включає:

1. Запис результатів:

- Фіксація всіх виявлених дефектів, а також успішно пройдених тестів. Детальне описання помилок.

2. Регресійне тестування:

- Перевірка, чи впливають внесені зміни на існуючий функціонал.

3. Тестування виправлень:

– Перевірка виправлень помилок після їх виправлення розробниками.

4. Тестування нових функцій:

– Перевірка нових або змінених функцій відповідно до оновлених тест-кейсів.

Обсяг тестування: Ручне тестування орієнтоване на основні функціональності системи та сценарії використання.

Результати тестування: буде створено звіт з результатами тестування

Середовище тестування : відповідає середовищу тестового розгортання

3.5 Звіт з тестування із аналізом відповідності функціональним вимогам

Огляд модуля: Модуль розроблений для здійснення управління проектами та завданнями в межах проекту

Обсяг тестування: Тестування охоплювало функціональні вимоги модуля, тести проводилися вручну.

Типи проведених тестів: Проводилося функціональне тестування для перевірки коректності реєстрації та автентифікації користувачів; додавання, видалення та редагування проектів; додавання, видалення та редагування завдань в межах проекту; тестування взаємодії інтерфейсу з користувачем.

Список проблем і рішень:

При введенні некоректних даних автентифікації користувачу все одно надавався доступ. Вирішено за допомогою додавання перевірки на те чи ідентифікатор користувача не дорівнює -1.

Фільтрована модель даних не відображалася в графічному інтерфейсі. Вирішено розділенням декларації і ініціалізації моделі.

Випадаючий список не відображав коректні значення елементів. Вирішено явним визначення ролі даних.

Метод removeProject класу ProjectManagement при виклику викидав виняток pqxx_usage_error. Вирішено зміною структури метода так, що не виконувалось дві транзакції до бази даних одночасно.

Метод updateProject класу ProjectManagement порушував обмеження таблиці бази даних NOT NULL. Вирішено зміною SQL-запиту, а саме додаванням функції COALESCE, яка повертає перше значення яке не є NULL, з тих що їй передали.

Методи, що витягують інформацію з бази даних викидали виняток pqxx_conversion_error. Вирішено додаванням перевірки на існування значення.

Хешування паролю записувало в базу даних некоректне значення. Вирішено очищенням об'єкту класу QCryptographicHash, перед кожним використанням

Критерії виходу: Завершення етапу тестування визначено повним покриттям тестових випадків та усуненням дефектів.

Підсумок тестування: Загальний підсумок етапу тестування вказує на готовність програми до випуску.

3.6 Рев'ю коду на відповідність нефункціональним вимогам та постулатам «чистого коду»

Код є досить чистим та організованим, але його можна покращити. Завдяки застосуванню кількох підходів «чистого коду» можна покращити його читабельність та обслуговуваність.

Можна розділити функціонал класу MainWindow на менші підкласи або окремі модулі. Клас зараз досить великий, і це може призвести до складнощів у майбутньому.

Можна додати більше обробки помилок у функціях, які взаємодіють із зовнішніми ресурсами, наприклад, базою даних. Це допоможе уникнути неочікуваного поведінки програми.

Було би кращим рішенням збільшити використання абстракцій та інтерфейсів для зменшення залежностей між класами.

Є деяке повторення коду в методах onCompleteTask, onRestoreTask, onRemoveTask. Можливо об'єднання цих методів в один з загальною логікою, аргументом якого буде булеве значення або який-небудь інший індикатор є хорошим рішенням, але в реалізації цього будуть складності через те, що ці методи є слотами, тому їхня сигнатура має повністю відповідати пов’язаним з ними сигналам.

Використання констант для уникнення магічних чисел або рядків у коді, може покращити зрозумілість коду, це в основному стосується методу filterModel

Метод convertToModel можна замінити та використання шаблону пректування «Адаптер».

Модуль забезпечує безпеку даних. Для зберігання паролю використовується алгоритм хешування SHA512, а також «сіль» даних, створена на основі часової мітки в мілісекундах, що також хешується за допомогою алгоритму SHA512, це забезпечує унікальне значення для кожного користувача, яке також змінюється при кожній зміні пароля.

Для захисту від SQL-ін’єкцій використовується метод завчасно заготованих запитів. Замість вбудовування значень напряму в SQL-запит, використовуються параметри, які передаються окремо в запиті. Система бази даних трактує ці параметри як дані, а не як частину SQL-запиту, що робить неможливим використання їх для впровадження зловмисного коду.

```
connection->getDataBase()->prepare("updateProject", "UPDATE \"Projects\"\
    SET project_name = COALESCE($2, project_name), description = COALESCE($3, description), \
    current_version = COALESCE($4, current_version), project_manager = CAST(COALESCE($5, project_id) AS int)\
    WHERE project_id = $1");
```

Рисунок 3.8 - Приклад заготованого SQL-запиту

На рисунку 3.8 показано SQL-запит, який готується у коді для оновлення інформації про проєкт у таблиці Projects. Це підготовлений SQL-запит (prepared statement), що оновлює запис у таблиці Projects з урахуванням того, що деякі значення можуть бути NULL.

Використовується функція COALESCE, яка повертає перше ненульове значення зі списку аргументів. Це означає, що якщо нове значення (наприклад $2) є NULL, то використовується поточне значення з бази (project_name). Таким чином, можна частково оновлювати поля без втрати вже наявних даних.

Опис полів у запиті:

– project_name = COALESCE($2, project_name) — оновлює назву проєкту, якщо $2 не є NULL;

– description = COALESCE($3, description) — оновлює опис;

– current_version = COALESCE($4, current_version) — оновлює версію проєкту;

– project_manager = CAST(COALESCE($5, project_id) AS int) — оновлює ID менеджера проєкту, за потреби приводячи до типу int.

Умова WHERE project_id = $1 вказує, що зміни стосуються тільки того проєкту, ID якого дорівнює значенню $1.

Цей підхід з COALESCE зручний у випадках, коли не всі поля мають оновлюватися кожного разу.

3.7 Точки та план рефакторингу

Рефакторинг коду є важливим етапом для поліпшення якості, зрозумілості та підтримки кодової бази.

Основні точки рефакторингу для покращення коду модуля:

1. Розділення на класи та модулі. Розділення коду на класи або модулі відповідно до принципів єдиної відповідальності та розділення інтерфейсу та реалізації. Клас MainWindow досить великий, його можна розділити на декілька менших одиниць

2. Виділення в окремі методи. Розділення великих функцій або блоків коду на менші методи для полегшення розуміння та тестування окремих частин. Наприклад як було зроблено з методами encryptPassword та getSalt.

3. Видалення дублікатів. Перегляд коду для виявлення дублікатів та видалення їх шляхом винесення загальних частин в окремі функції або класи. Наприклад об'єднання методів onCompleteTask, onRestoreTask та onRemoveTask в один метод з параметром, який буде вказувати тип дії

4. Використання констант та перелічень. Заміна магічних чисел або рядків на константи або перелічення для полегшення змін та підтримки коду. Наприклад, у методі filterModel можна використовувати константи для значень фільтрації.

5. Організація обробки помилок. Додавання обробки помилок для забезпечення безпечності та адекватності реагування на непередбачувані ситуації. Наприклад для методів, які взаємодіють з базою даних, для забезпечення коректного виконання програми.

6. Використання імен, що відображають призначення. Перегляд імен змінних, методів та класів, забезпечуючи їхню зрозумілість та відповідність призначенню.

7. Використання стандартів коду. Дотримання стандартів коду мови програмування C++, таких як використання стилю camel case, використання префіксів для змінних, використання просторів імен, та впровадження кращих практик, таких як використання локальних змінних, відділення концепцій та інші.

8. Виправлення архітектурних проблем. Виправлення архітектурних недоліків, таких як занадто сильна залежність, низька зчепленість та висока залежність. Клас MainWindow досить сильно зв'язаний із UI-елементами (наприклад, ui->entry, ui->projects).

9. Оптимізація продуктивності. Виділення можливостей для оптимізації продуктивності, таких як уникнення зайвих операцій, покращення алгоритмів та зменшення надмірного споживання пам'яті.

10. Покриття коду тестами. Перехід на автоматизований підхід тестування, створення нових тестових випадків для існуючого коду та впроваджуйте керований тестуванням розвиток для нового коду.

Отже, програмна реалізація відповідає усім поставленим функціональним вимогам. Реалізовано ключові функції: автентифікація користувачів, створення та

редагування проєктів і завдань, керування версіями, робота з документацією. Проведене тестування підтвердило стабільність і готовність ПЗ до використання.

ВИСНОВКИ

У межах даної роботи було розроблено програмний web-сервіс для контролю та управління життєвим циклом програмного забезпечення.

У другому розділі проаналізовано логічну та програмну архітектуру web-системи, представлено діаграми варіантів використання, класів, станів і діяльності. На основі діаграм деталізовано процеси створення проєктів, задач, документації та оновлення версій. Візуалізація бізнес-логіки дала змогу формалізувати ролі користувачів, визначити точки контролю доступу та забезпечити послідовність дій у системі.

У третьому розділі реалізовано архітектуру коду відповідно до патерну MVC, описано використані бібліотеки, мови та фреймворки. Проведено тестування функціональності, підтверджено відповідність системи вимогам, проаналізовано та запропоновано рефакторинг. Система демонструє стабільну роботу, високу продуктивність і захист даних, з можливістю подальшого розширення.

Реалізація охопила всі ключові етапи – від аналізу предметної області до створення повноцінної архітектури коду, що базується на патерні MVC. Особливу увагу приділено захисту даних, багаторівневому управлінню ролями, модульності архітектури та підтримці гнучкого інтерфейсу користувача.

Проведене тестування підтвердило відповідність системи заявленим вимогам. Результати проєкту мають значну практичну цінність, зокрема для команд розробників, які потребують інструменту для керування складними програмними проектами.

Подальший розвиток системи можливий шляхом впровадження CI/CD, інтеграції з системами контролю версій, додавання розширеного API та мобільної версії. Розроблений продукт є конкурентоспроможним рішенням для автоматизації процесів керування ПЗ у сучасному IT-середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Basic Design Patterns in C++. medium.com: веб-сайт. URL: https://medium.com/must-know-computer-science/basic-design-patterns-in-c-39bd3d477a5c (Дата звернення: 25.03.2025)
2. C++ Design Patterns. github.com: веб-сайт. URL: https://github.com/Junzhuodu/design-patterns (Дата звернення: 30.03.2025)
3. Code Quality Metrics. sealights: веб-сайт. URL: https://www.sealights.io/code-quality/code-quality-metrics-is-your-code-any-good/ (Дата звернення: 22.03.2025)
4. Conceptual Design. alltimedesign: веб-сайт. URL: https://alltimedesign.com/conceptual-design/ (Дата звернення: 06.04.2025)
5. High-Level Solution Design Documents. softwaredominos: веб-сайт. URL: https://softwaredominos.com/home/software-design-development-articles/high-level-solution-design-documents-what-is-it-and-when-do-you-need-one/ (Дата звернення: 12.04.2025)
6. Modeling the interactions between objects in UML. ibm: веб-сайт. URL: https://www.ibm.com/docs/en/rsm/7.5.0?topic=model-modeling-interactions-between-objects-in-uml (Дата звернення: 15.03.2025)
7. Object Oriented Design. Geeksforgeeks: веб-сайт. URL: https://www.geeksforgeeks.org/oops-object-oriented-design/ (Дата звернення: 17.03.2025)
8. Object-oriented Design in Software Engineering. artoftesting: веб-сайт. URL: https://artoftesting.com/object-oriented-design-in-software-engineering (Дата звернення: 19.03.2025)
9. Qt Documentation. doc.qt : веб-сайт. URL: https://doc.qt.io/ (Дата звернення: 28.04.2025)
10. Software Testing Reporting. appsero: веб-сайт. URL: https://appsero.com/tips-tricks/software-testing-reporting/ (Дата звернення: 12.04.2025)

11. Solution Domain. wiki.c2: веб-сайт. URL: https://wiki.c2.com/?SolutionDomain (Дата звернення: 23.04.2025)

12. Structured Design Methodology. faadooengineers: веб-сайт. URL: http://www.faadooengineers.com/online-study/post/cse/software-engineering/216/structured-design-methodology (Дата звернення: 17.04.2025)

13. Synthesis-Based Software Architecture Design. link.springer: веб-сайт. URL: https://link.springer.com/chapter/10.1007/978-1-4615-0883-0_5 (Дата звернення: 23.04.2025)

14. Test Report. strongqa: веб-сайт. URL: https://strongqa.com/qa-portal/testing-docs-templates/test-report (Дата звернення: 22.04.2025)

15. Understanding High-Level Design (HLD) and Low-Level Design (LLD) in Software Development. linkedin: веб-сайт. URL: https://www.linkedin.com/pulse/understanding-high-level-design-hld-low-level-lld-fady-mohammed-deeb (Дата звернення: 12.05.2025)

16. What is a Test Plan? Test Plan vs. Test Strategy (Ultimate Guide). katalon: веб-сайт. URL: https://katalon.com/resources-center/blog/test-plan (Дата звернення: 20.05.2025)

17. What Is TDD (Test Driven Development)? Process, Importance, and Limitations. spiceworks: веб-сайт. URL: https://www.spiceworks.com/tech/devops/articles/what-is-tdd/ (Дата звернення: 02.05.2025)

18. What is Test-Driven Development?. Testdriven : веб-сайт. URL: https://testdriven.io/test-driven-development/ (Дата звернення: 07.05.2025)

19. PostgreSQL 16.1 Documentation. postgresql: веб-сайт. URL: https://www.postgresql.org/docs/current/ (Дата звернення: 15.05.2025)