МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра інформаційних технологій

«До захисту допущено»

В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Вебдодаток підтримки процесу управління ІТ- проектами»

Здобувача (ки) групи ІТ-02 Серебрянського Руслана В'ячеславовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Руслан СЕРЕБРЯНСЬКИЙ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент кафедри ІТ к.т. н., доцент Володимир НАГОРНИЙ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО
«__» _____ 2024 р.

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

Серебрянському Руслану В'ячеславовичу

1. Тема роботи *Вебдодаток підтримки процесу управління ІТ- проектами*

керівник роботи Володимир НАГОРНИЙ, *к.т.н., доцент*,

затверджені наказом по університету від «*07*» травня 2024 р. №0516-VI

2. Строк подання студентом роботи «26» травня 2024 р.

3. Вхідні дані до роботи технічне завдання на розробку вебдодатку підтримки процесу управління ІТ- проектами

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, моделювання та проектування , розробка програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність, постановка задачі, аналіз продуктів-аналогів, порівняння продуктів-аналогів, функціональні вимоги, функціональне моделювання з точки зору користувача, моделювання варіантів використання вебдодатку, засоби реалізації, архітектура вебдодатку, фізична модель даних, реалізація, демонстрація роботи вебдодатку, тестування, висновки

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання «04» квітня 2024 р

КАЛЕНДАРНИЙ ПЛАН

№	Вид робіт	Термін виконання
1	Огляд досліджень і публікацій у сфері управління IT-проектами	05.04.2024 - 07.04.2024
2	Аналіз аналогів, визначення унікальних функцій, написання технічного завдання на розроблення додатку	08.04.2024 - 11.04.2024
3	Розподіл та планування задач, формування діаграм, дослідження ризиків, розробка архітектури та загального шаблону додатку	12.04.2024 - 18.04.2024
4	Створення вебдодатку підтримки процесу управління IT- проектами	19.04.2024 - 24.04.2024
5	Тестування створеного додатку	25.04.2024 - 30.04.2024
6	Аналіз отриманих результатів	01.05.2024 - 20.05.2024
7	Оформлення пояснювальної записки до кваліфікаційної роботи	21.05.2024 - 26.05.2024

Студент _____ Руслан СЕРЕБРЯНСЬКИЙ
(підпис)

Керівник роботи _____ к.т.н., доц. Володимир НАГОРНИЙ
(підпис)

АНОТАЦІЯ

Тема кваліфікаційної роботи бакалавра «Вебдодаток підтримки процесу управління IT-проектами». Пояснювальна записка складається зі вступу, трьох розділів, висновків, списку використаних джерел із 23 найменувань, додатків. Загальний обсяг роботи – 105 сторінок, у тому числі 55 сторінок основного тексту, 2 сторінки списку використаних джерел, 44 сторінок додатків.

Актуальність роботи полягає в тому, що управління IT-проектами є критично важливим аспектом успішності будь-якої організації у сучасному світі, де швидкість змін у технологічному секторі є надзвичайною. Від ефективного управління залежить успішний запуск нових продуктів та оновлення існуючих систем і сервісів. Використання відповідних інструментів для управління проектами дозволяє забезпечити не тільки технічні потреби проекту, але й гнучкість та адаптивність до змінних умов роботи.

Мета роботи: розробити вебдодаток підтримки процесу управління IT-проектами, що забезпечить зручний, гнучкий інтерфейс та інструменти для ефективної роботи з командами та проектами. У роботі використані методи аналізу предметної області, проектування інформаційних систем, програмування та тестування програмного забезпечення. Результати роботи включають розробку вебдодатку з використанням технологій React, Node.js, PostgreSQL та Tailwind CSS, що забезпечує ефективне управління IT-проектами.

Рекомендації щодо використання розробленого вебдодатку включають його впровадження в організаціях для підвищення ефективності управління IT-проектами. Впровадження додатку сприятиме покращенню комунікації між учасниками проектів, оптимізації робочих процесів та підвищенню загальної продуктивності. Результати досліджень та розробок можуть бути використані для подальшого вдосконалення програмного забезпечення та адаптації його до специфічних потреб різних організацій.

Ключові слова: УПРАВЛІННЯ ПРОЕКТАМИ, ВЕБДОДАТОК, IT-ПРОЕКТИ, REACT, NODE.JS, POSTGRESQL, TAILWIND CSS.

ЗМІСТ

ВСТУП

У сучасному світі, де швидкість змін у технологічному секторі є надзвичайною, ефективне управління IT-проектами стає критично важливим аспектом успішності будь-якої організації. Від успішного запуску нових продуктів до оновлення існуючих систем і сервісів, IT-проекти вимагають чіткого управління, точного планування та ефективної комунікації між усіма учасниками проекту [1]. Значна частина успіху в цих ініціативах залежить від інструментів, які використовуються для управління проектами. Важливо, що ці інструменти не тільки відповідають технічним потребам проекту, але й забезпечують гнучкість і адаптивність для змінних умов роботи.

У цьому контексті, методологія Scrum [2], яка є найпопулярнішою методологією управління проектами, здобула велику популярність завдяки своїй здатності до швидкої адаптації до змінних вимог і непередбачуваних викликів, що є нормою в IT-сфері. Основними перевагами Scrum є його орієнтованість на клієнта, фокус на співпраці міжнародних команд та ітераційний процес розробки, який дозволяє командам швидко реагувати на зміни і вносити корективи у проект в будь-який час [3].

Мета даного проекту – розробити вебдодаток підтримки процесу управління IT- проектами.

В ході кваліфікаційної роботи необхідно вирішити наступні задачі, а саме:

- Провести аналіз предметної області управління IT-проектами, включаючи вивчення методів і підходів;
- Оглянути наявні дослідження та публікації, пов'язані з управлінням IT-проектами;
- Провести аналіз існуючих програмних продуктів-аналогів
- Розробити технічне завдання для вебдодатку, що буде допомогати у процесі управління IT-проектами;
- Спланувати виконання робіт з розробки вебдодатку управління IT-проектами, розробивши детальний план проекту з часовими рамками;
- Реалізувати вебдодаток з використанням сучасних веб-технологій,

таких як React, Node.js, PostgreSQL та Tailwind CSS;

- Забезпечити належне тестування вебдодатку, включаючи функціональні та нефункціональні аспекти.

Важливість розробки такого додатку обумовлена зростаючою потребою в інструментах для ефективного управління IT-проектами, які могли б забезпечувати не тільки високу продуктивність роботи, але й гнучкість у взаємодії з командою та іншими зацікавленими сторонами. Завдяки широкій функціональності та адаптації до специфічних потреб керівників проектів, додаток має потенціал стати незамінним інструментом в арсеналі сучасного IT-менеджера.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасних досліджень і публікацій у галузі управління IT-проектами

В останні роки значну увагу дослідників привертають теми пов'язані з адаптацією традиційних методів управління проектами до сучасних цифрових технологій [4]. Враховуючи зростаючий обсяг даних та потребу в їх обробці, значний акцент робиться на використанні штучного інтелекту та машинного навчання для автоматизації різних аспектів управління проектами [5]. Наприклад, компанії як Atlassian [6] і Microsoft [7] розробляють рішення, що дозволяють не тільки ефективно планувати ресурси, але й прогнозувати потенційні ризики проектів на основі аналізу великих даних. Ці підходи демонструють значне покращення в швидкості та якості прийняття рішень, дозволяючи менеджерам проектів оперативно реагувати на зміни в проектному середовищі.

Також, сучасні дослідження [8] в області управління IT-проектами підкреслюють важливість інтеграції розробки програмного забезпечення з неперервними методами впровадження та доставки продукту, знаними як DevOps [9]. Ці методи, які орієнтовані на співпрацю між розробниками та операційними командами, сприяють більш ефективному виявленню та виправленню помилок, що значно скорочує часові рамки проектів [10] та підвищує якість кінцевого продукту. Підходи DevOps також включають автоматизацію багатьох процесів, які традиційно вимагали значних зусиль та часу, таких як тестування, розгортання та моніторинг програмного забезпечення.

Крім того, значний інтерес в сфері управління проектами викликає впровадження гнучких методологій, таких як Scrum і Kanban [11]. Ці методології покращують гнучкість процесу розробки та здатність команди швидко адаптуватися до змін у вимогах проекту. Наприклад, Scrum використовує короткі цикли розробки, звані спринтами, що дозволяють командам регулярно переоцінювати цілі проекту та пріоритети задач. Kanban ж зосереджує увагу на

візуалізації робочих процесів, що сприяє кращій координації та розподілу роботи між членами команди.

На завершення, останні дослідження [12] також звертають увагу на зростання значення соціальних аспектів управління проектами, особливо на важливість створення ефективних комунікаційних стратегій, які забезпечують високий рівень взаєморозуміння між усіма зацікавленими сторонами проекту. Комунікація вважається ключем до успішної реалізації проектів, особливо в умовах, коли команди працюють віддалено та мають члени з різних культурних контекстів.

1.2 Порівняльний аналіз інструментів для управління IT-проектами

Ринок програмних продуктів для управління проектами є динамічним і різноманітним, пропонуючи рішення, які задовольняють потреби різних галузей і специфікацій проектів. В цьому розділі буде розглянуто основні програмні продукти, які використовуються для управління IT-проектами, а саме: Microsoft Project, Jira, Asana, Trello та Basecamp [13-16]. Мета аналізу полягає у виявленні сильних та слабких сторін кожного продукту з метою визначення ключових характеристик для розробки власного рішення .

1.2.1 Microsoft Project як інструменту для управління IT-проектами

Microsoft Project є одним з найстаріших і найвідоміших інструментів управління проектами. Він забезпечує високий рівень деталізації у плануванні ресурсів, бюджетів та часових рамок. Сильні сторони Microsoft Project включають його здатність інтегруватися з іншими продуктами Microsoft, такими як Office 365 та SharePoint, що дозволяє централізувати документацію та спілкування в рамках проекту. Крім того, Microsoft Project пропонує розширені можливості для управління ресурсами, дозволяючи менеджерам проектів оптимізувати використання ресурсів і виявляти перевантаження робочих навантажень (рис. 1.1). Проте, слабкі сторони включають високу складність і вартість, що може бути бар'єром для малих та середніх підприємств. Також, деякі

користувачі вказують на негнучкість інтерфейсу та керування проектами, що може ускладнювати швидку адаптацію до змін у проекті.

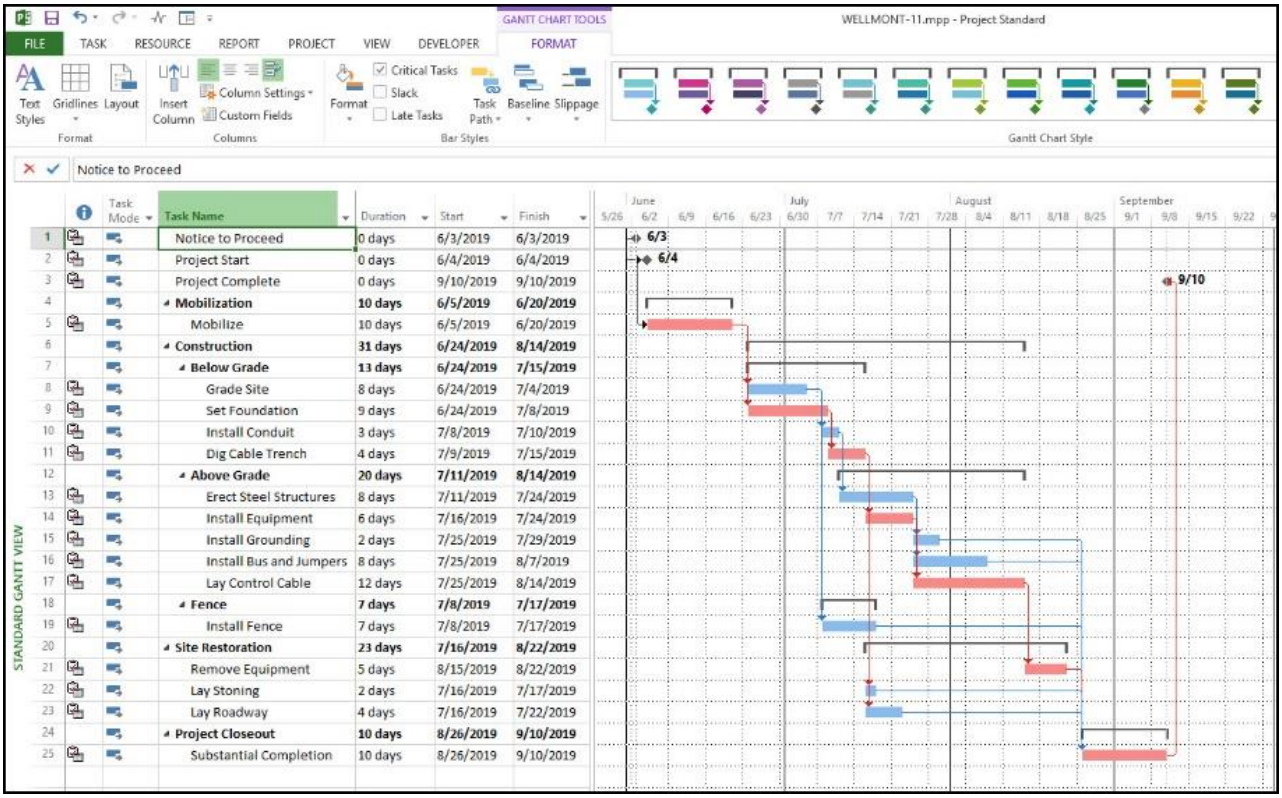

Рисунок 1.1 – Візуальний вигляд Microsoft Project

1.2.2 Оцінка Jira в контексті управління IT-проектами

Jira від Atlassian є вибором, що користується популярністю серед розробників програмного забезпечення, особливо тих, хто застосовує гнучкі методології управління проектами, такі як Agile і Scrum. Її основні переваги включають високу адаптивність до конкретних вимог проекту та можливість інтеграції з різноманітними інструментами розробки, що дозволяє створити дуже налаштоване робоче середовище. Jira дозволяє керувати проектами через систему "карток" (рис. 1.2) для завдань, які користувачі можуть легко перетягувати між колонками станів процесу, візуалізуючи прогрес у реальному часі.

Однак, одним з недоліків Jira є її складність налаштування, яка може вимагати значної кількості часу та зусиль для впровадження, особливо для великих команд. Також, висока вартість підписки може стати бар'єром для малих

команд або індивідуальних користувачів, оскільки вона суттєво зростає зі збільшенням кількості користувачів та додаткових функцій. Крім того, для оптимізації роботи з Jira користувачам часто потрібно використовувати додаткові плагіни, які також можуть збільшити загальні витрати на утримання системи.

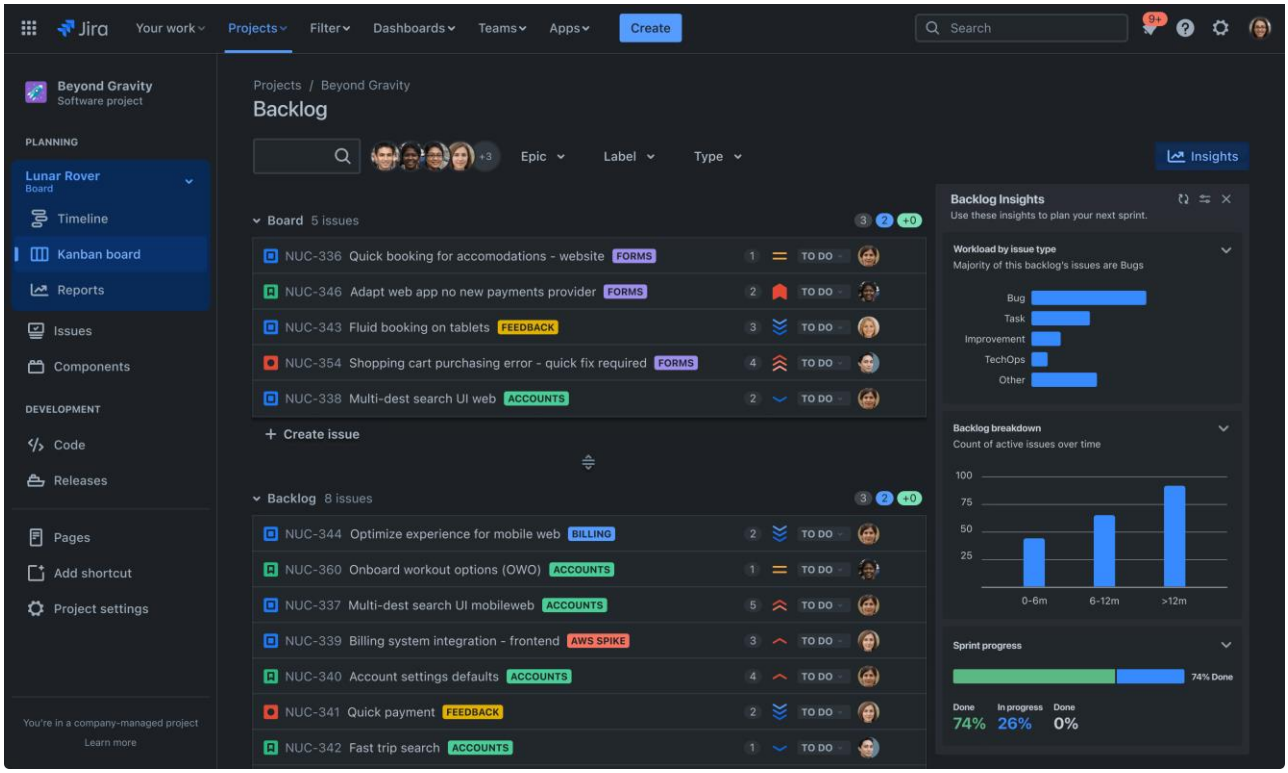

Рисунок 1.2 – Візуальний вигляд Jira

1.2.3 Аналіз Asana спільній роботі в IT-проектах

Asana є відносно новим гравцем на ринку управління проектами, який акцентує увагу на поліпшенні об’єднанні та комунікації в командах. Однією з головних переваг Asana є її інтуїтивно зрозумілий інтерфейс, який разом із великою гнучкістю управління завданнями та проектами робить її ідеальною для міжфункціональних команд. Asana дозволяє користувачам легко створювати та налаштовувати списки завдань, налаштовувати нагадування та ділитися прогресом зі стейкхолдерами, підтримуючи високий рівень видимості робочих процесів(рис. 1.3).

Однак, слабкі сторони Asana включають її обмежені можливості для ведення складних проектів, які вимагають детального планування ресурсів, розкладів і бюджетів. Також відсутні вбудовані інструменти для глибокої звітності проекту, що може створити труднощі для керування більш складними ініціативами. Крім того, хоча Asana забезпечує велику гнучкість, вона може виявитися менш ефективною для команд, які потребують дуже структурованих методологій управління, таких як водоспад або традиційний проектний менеджмент. Це може призвести до необхідності використання додаткових інструментів чи плагінів, що збільшує загальні витрати на IT-інфраструктуру.

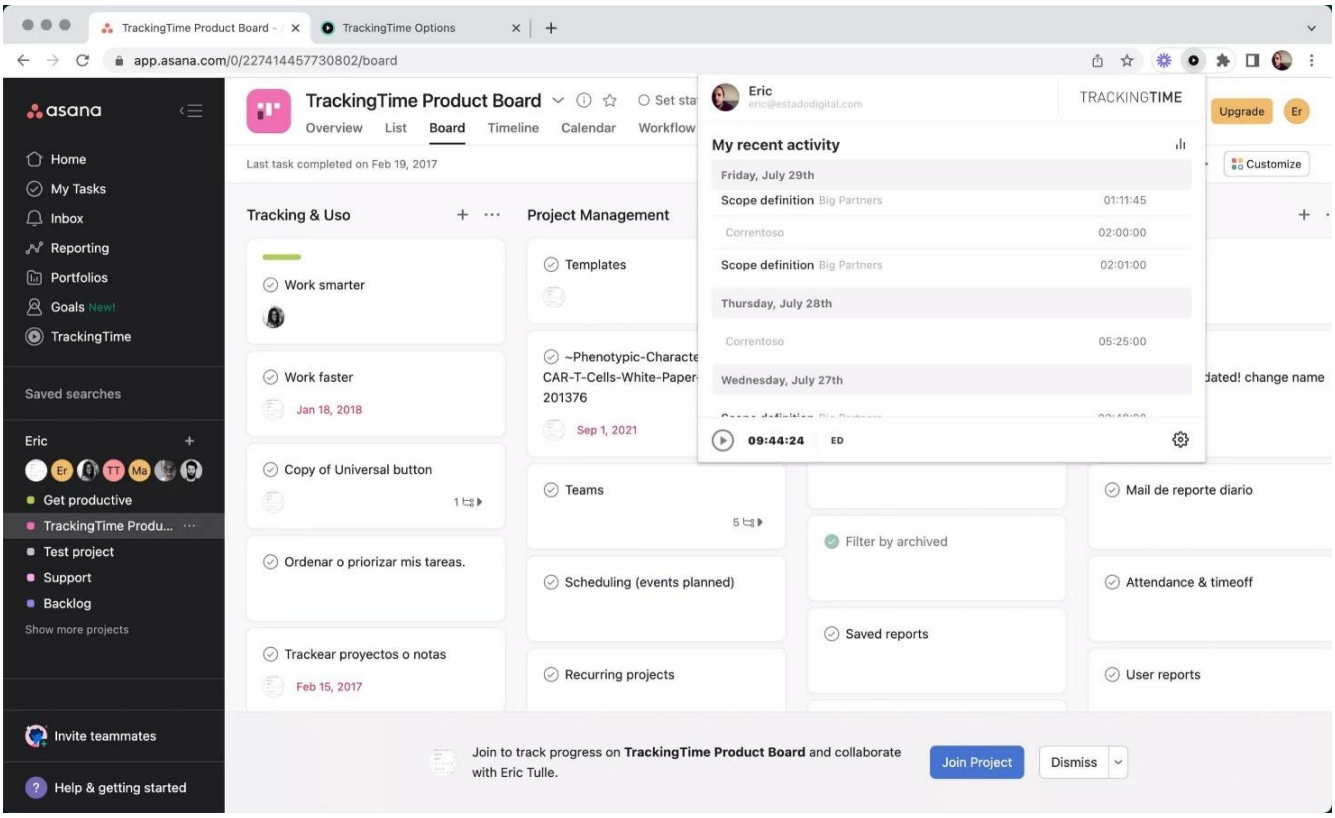

Рисунок 1.3 – Візуальний вигляд Asana

1.2.4 Trello як візуального інструменту для організації IT-проектів

Trello відомий своїм візуальним підходом до управління проектами, використовуючи систему дошок, списків та карток для організації завдань і проектів. Його ключові переваги включають простоту використання, візуальну привабливість і легкість впровадження, особливо у малі команди. Trello ідеально

підходить для проектів з меншою кількістю складності, де головне швидко організувати процес і забезпечити видимість стану завдань для всіх учасників. Це робить Trello чудовим інструментом для швидкого старту і візуального моніторингу ходу робіт (рис. 1.4).

Однак, Trello може виявитися не найкращим варіантом для великих проектів зі складним ресурсним плануванням та строгими вимогами до звітності. Його функціональність обмежена в контексті автоматизації процесів та інтеграції з іншими системами, що може вимагати додаткових інструментів для повного управління проектами. Також, відсутність вбудованих інструментів для глибокої аналітики і звітності може ускладнити відстеження детального прогресу проекту, особливо в умовах, що вимагають точного моніторингу та аналізу результатів.

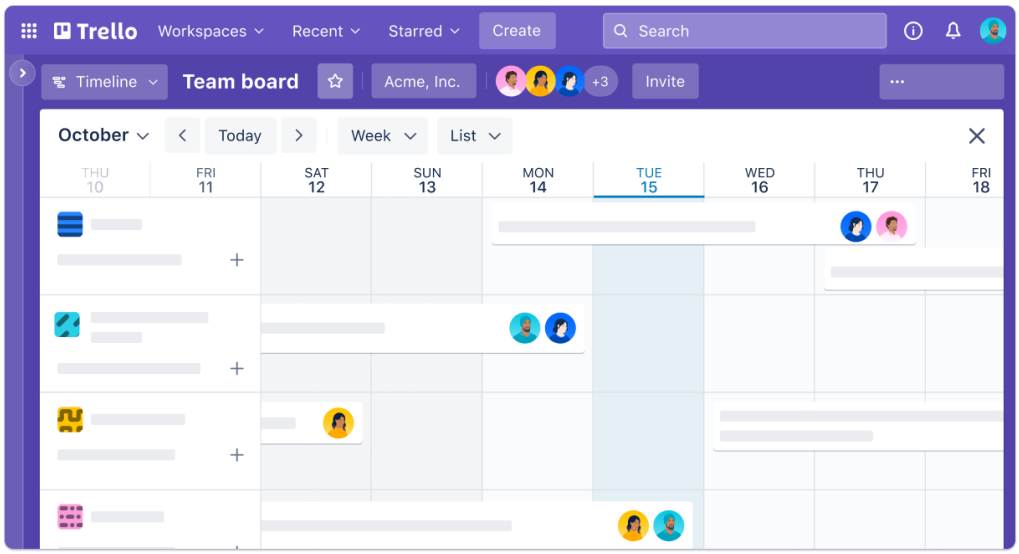

Рисунок 1.4 – Візуальний вигляд Trello

1.2.5 Аналіз програми Basecamp

Basecamp є відомим рішенням, яке спрямоване на спрощення управління проектами та покращення комунікації між учасниками команди. Його інтерфейс є простим і інтуїтивно зрозумілим, що забезпечує легке освоєння навіть для нових користувачів (рис. 1.5). Basecamp надає широкий спектр інструментів, включаючи можливості для дискусій, зберігання файлів, організації завдань і відстеження часу, що робить його відмінним вибором для управління проектами

різного типу.

Однак, Basecamp може не підійти для дуже великих організацій або складних проектів, оскільки він не має розширених аналітичних інструментів та засобів звітності, які могли б допомогти в глибокому аналізі даних або складному ресурсному плануванні. Важливо відзначити, що попри свою універсальність, іноді користувачам може знадобитися інтеграція з іншими системами для забезпечення більш комплексного управління проектами.

Таким чином, попри численні переваги, такі як простота використання та ефективне управління завданнями, Basecamp може мати обмеження в контексті вимог до детальної аналітики та звітності, що варто враховувати при виборі інструменту для керування складними проектами.

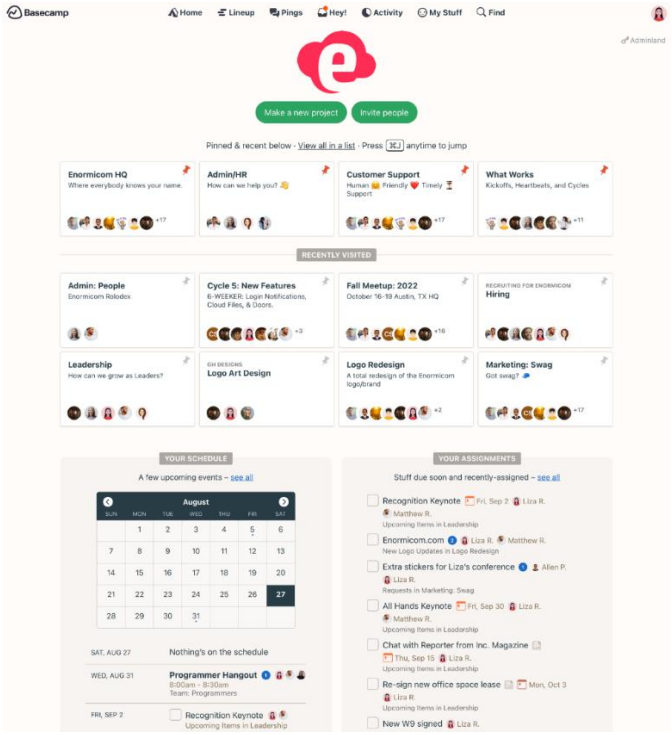

Рисунок 1.5 – Візуальний вигляд Basecamp

Аналізуючи сильні та слабкі сторони існуючих програмних продуктів-аналогів, було визначити ключові характеристики, та була створена порівняльна таблиця характеристик аналогів (табл. 1.1) які повинні бути враховані при розробці нового продукту.

Таблиця 1.1 – Порівняльна таблиця характеристик аналогів

Характеристика	**Microsoft Project**	**Jira**	**Asana**	**Trello**	**Basecamp**	**Власна розробка**
Спрямованість	Комплексне управління проектами	Гнучке управління проектами в IT	Задачі та проекти різних галузей	Візуальне управління проектами	Просте управління проектами	Комплексне управління проектами
Інтерфейс	Складний, багатофункціональний	Конфігуруємий, технічний	Інтуїтивний, дружній	Простий, візуальний	Простий, мінімалістичний	Інтуїтивний, простий
Інтеграція	Висока (Office, SharePoint)	Висока (DevOps інструменти)	Середня (багато інструментів)	Обмежена (плагіни)	Обмежена (декілька інструментів)	Обмежена (декілька інструментів)
Методології	Водоспад, Agile	Agile, Scrum, Kanban	Agile, Scrum	Kanban	Неформальні методи	Agile, Scrum, Kanban
Ресурсне планування	Детальне	Обмежене	Обмежене	Недоступне	Недоступне	Обмежене
Звітність	Розгалужена звітність	Обмежена звітність	Обмежена звітність	Найпростіша звітність	Найпростіша звітність	Найпростіша звітність
Вартість	Висока	Висока	Середня	Низька	Середня	Безкоштовна
Співпраця	Складна конфігурація	Висока	Висока	Висока	Висока	Висока

Важливо забезпечити гнучкість налаштувань, високий рівень інтеграції з іншими інструментами, легкість використання та адаптивність до змінних умов проекту. Також критично важливим є врахування потреб цільової аудиторії, особливо в контексті специфіки управління IT-проектами [17].

Таблиця дозволяє наочно порівняти основні характеристики лідируючих інструментів управління проектами, що допомагає визначити, особливості кожного продукту. Також, вона висвітлює потенційні недоліки кожного продукту, які потрібно врахувати при розробці рішення для уникнення аналогічних проблем. Для майбутнього вебдодатку, який націлений на управління IT-проектами, важливо врахувати наступні ключові аспекти: Scrum, безкоштовне використання, простий інтерфейс, просте управління проектами, інтеграція чату з GPT для допомоги та таск-трекери [18].

Підтримка Scrum вимагає від додатку можливості створення спринтів, управління завданнями в рамках спринтів, проведення щоденних нарад (daily scrums), планування спринтів та ретроспектив. Це забезпечує гнучкість і адаптивність до змін, які є критичними для Scrum-проектів.

Безкоштовний вебдодаток збільшує доступність і привабливість для користувачів.

Проектування чистого, інтуїтивно зрозумілого інтерфейсу, який не перевантажений зайвими елементами, сприятиме кращій орієнтації користувачів у вашому додатку. Особливу увагу слід приділити юзабіліті, щоб нові користувачі могли легко розпочати роботу без додаткових навчань.

Додаток має підтримувати базові функції управління проектами, такі як створення завдань, їх розподіл між членами команди, відстеження прогресу та дедлайнів. Це має бути реалізовано таким чином, щоб користувачі могли легко орієнтуватися між проектами та завданнями.

Інтеграція чатбота на базі GPT може значно покращити досвід користувачів, надаючи миттєву підтримку та відповіді на запитання про використання додатку. Це також може включати автоматизацію деяких завдань, таких як нагадування про терміни завдань або допомогу в організації щоденних нарад.

Функціонал для відстеження завдань є важливим для ефективного управління проектами. Це повинно включати можливості для налаштування статусів завдань, призначення завдань користувачам та визначення пріоритетів.

Ці ключові аспекти формують основу вебдодатку та детально розглянуті в ДОДАТКУ А. з технічним завданням, де детально продумані на етапі проектування, щоб забезпечити високу функціональність та користувацьку зручність.

1.3 Мета та задачі дослідження

Мета даного проекту – розробити вебдодаток підтримки процесу управління IT- проектами Цей додаток повинен забезпечити користувачам зручний, гнучкий інтерфейс та інструменти для ефективної роботи з командами та проектами. Задля досягнення цієї мети, розробка додатку буде включати наступні ключові задачі:

- Провести аналіз предметної області управління IT-проектами, включаючи вивчення методів і підходів - це забезпечує розуміння найкращих практик і визначення вимог додатку;
- Проаналізувати аналоги, та дослідити існуючі рішеня у сфері управління проектами, визначення їхніх сильних та слабких сторін для формулювання унікальних властивостей запропонованого рішення;
- Визначити та сформулювати функціональні вимоги. Визначати основні і додаткові функції додатку, включаючи інтерфейси для управління спринтами, завданнями, роллю користувачів, звітами про продуктивність та інші;
- Проектування вебдодатку [20]. Включає структурно-функціональне моделювання для визначення основних компонентів та їх взаємодії, моделювання варіантів використання для розуміння взаємодії користувачів з системою, та проектування бази даних для ефективного зберігання та доступу до даних;
- Розробити вебдодаток з використанням таких технологій як

PostgreSQL, Node.js, React, Tailwind CSS [20-23] який буде легким і швидким.

- Провести комплексне тестування вебдодатку

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

2.1 Структурно-функціональне моделювання

Важливим кроком у процесі створення програмного продукту є моделювання структури та функціональних процесів. Цей етап дозволяє краще зрозуміти систему та ефективніше її оптимізувати, що допомагає зекономити час і ресурси.

Для представлення функціональних процесів використовується методологія IDEF0. Цей підхід призначений для детального опису взаємопов'язаних систем та процесів. Він складається з блоків, що символізують функції та процеси, і стрілок, які відображають вхідні та вихідні дані.

На рисунку 2.1 зображена функціональна діаграма, яка демонструє процес підтримки роботи вебдодатку для управління IT-проектами.

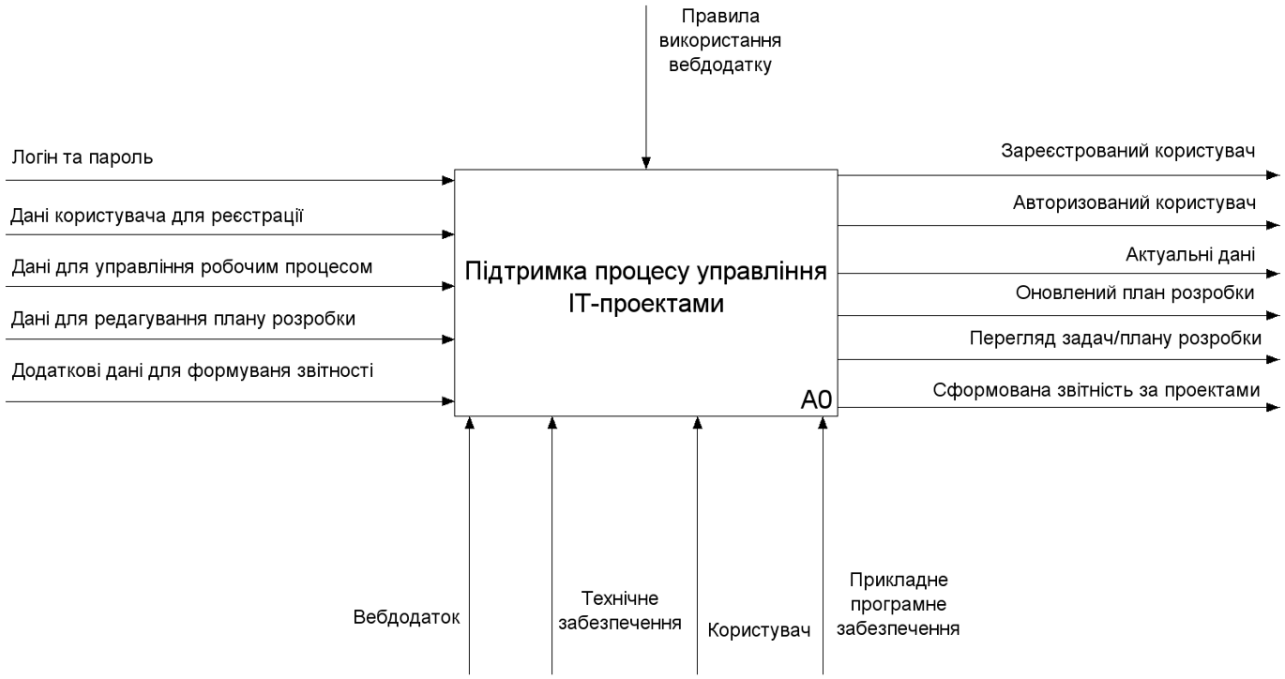

Рисунок 2.1 – Діаграма IDEF0

Як видно з діаграми, визначено наступні елементи IDEF0:

- **Вхідні дані**: логін та пароль, дані користувача для реєстрації, дані для управління робочим процесом, дані для редагування плану розробки,

додаткові дані для формування звітності;

- **Управління**: правила використання вебдодатку;
- **Механізми**: користувач, технічне забезпечення, приклдане програмне забезпечення;
- **Вихідні дані**: авторизований користувачб зареєстрований користувач, актуальні дані, оновлений план розробки, перегляд задач/плану розробки, сформовано звітність за проектом/проектами.

Для детальнішого аналізу процесів використовується декомпозиція функціонального проектування. Це дозволяє розділити функції на більш дрібні та прості елементи, що сприяє кращому розумінню системи. Декомпозиція може включати кілька рівнів і тривати до виділення конкретних підпроцесів. На рисунку 2.2 зображена діаграма декомпозиції першого рівня у нотації IDEF0, яка деталізує процеси підтримки вебдодатку для управління IT-проектами.

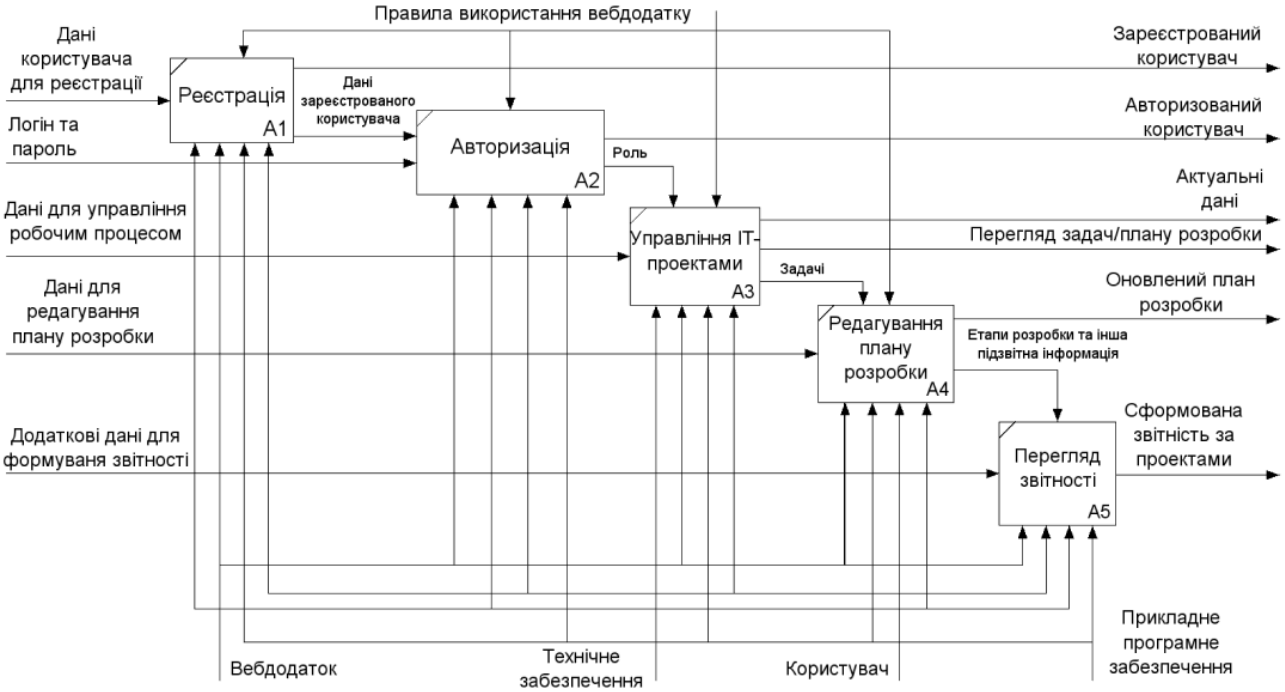

Рисунок 2.2 – Діаграма декомпозиції функціональної моделі

Елементи діаграми IDEF0:

- **Вхідні дані**: логін та пароль, дані користувача для реєстрації, дані для управління робочим процесом, дані для редагування плану розробки, додаткові дані для формування звітності;

- **Управління**: правила використання вебдодатку;
- **Механізми**: користувач, технічне забезпечення, приклдане програмне забезпечення;
- **Вихідні дані**: авторизований користувачб зареєстрований користувач, актуальні дані, оновлений план розробки, перегляд задач/плану розробки, сформовано звітність за проектом/проектами.

Декомпозиція функціональної моделі дозволяє створити більш детальне представлення системи, розділяючи її на конкретні підпроцеси, що забезпечує точність і гнучкість у розробці програмного продукту. У даній діаграмі декомпозиція розбита на такі основні процеси: авторизація, управління процесом, редагування плану розробки, перегляд звітності. Кожен з цих процесів взаємопов’язаний і не може функціонувати без попереднього.

2.2 Проектування вебдодатку

Діаграма варіантів використання є ключовим компонентом при проектуванні інформаційної системи. Вона ілюструє взаємодію між акторами та системою, а також відображає функціональні вимоги з точки зору користувачів. На рисунку 2.3 зображена діаграма варіантів використання для вебдодатку управління IT-проектами зі спільними кейсами для всіх акторів.

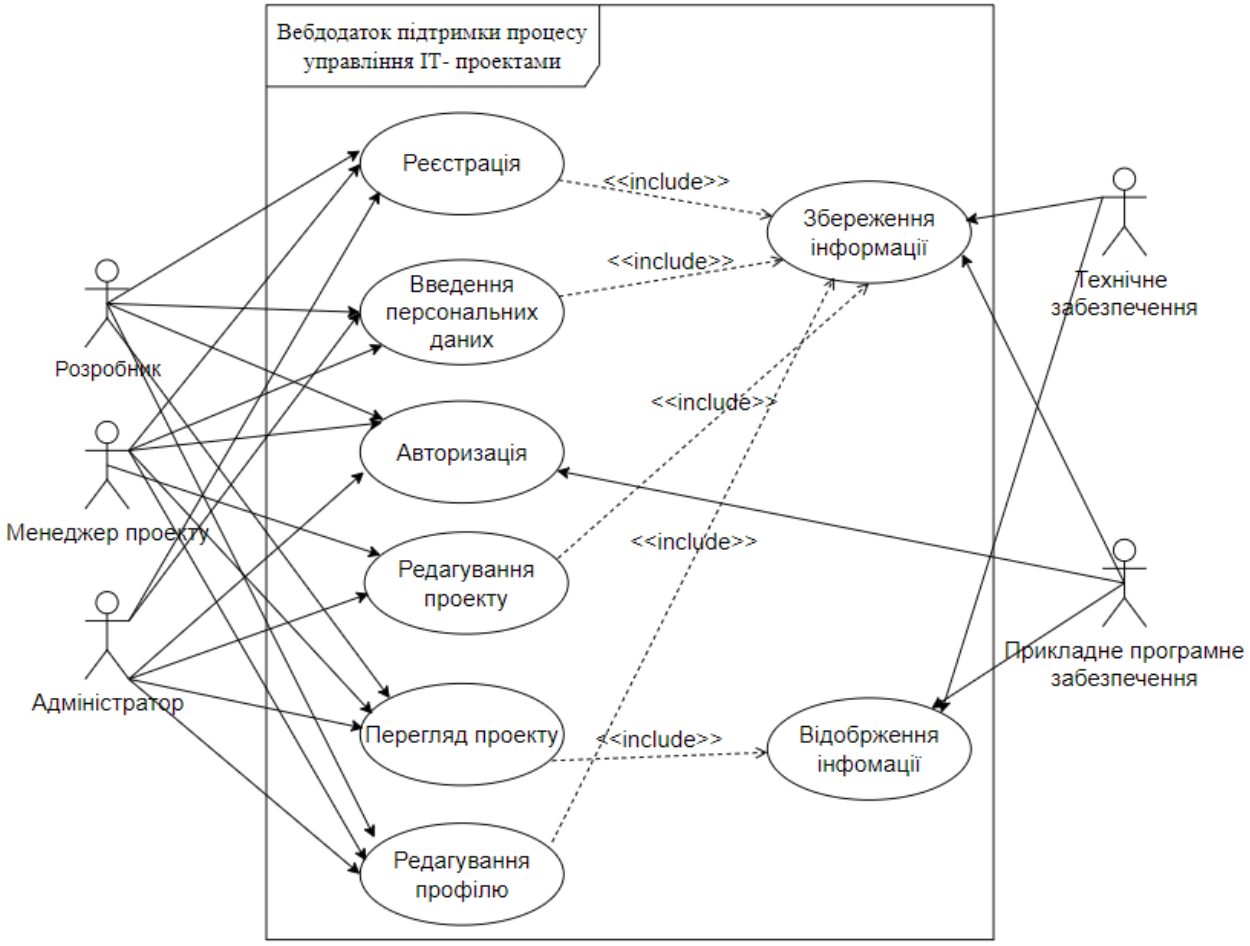

Рисунок 2.3 – Діаграма варіантів використання

З рисунку 2.3 видно, що для вебдодатку були визначені наступні ролі користувачів:

- **Адміністратор**: відповідальний за управління користувачами та проектами (Рисунок 2.4);
- **Менеджер проекту**: керує проектами, створює і редагує задачі (Рисунок 2.5);
- **Розробник**: виконує задачі, призначені менеджером проекту (Рисунок 2.6);.

Рисунок 2.4 – Роль адміністратор

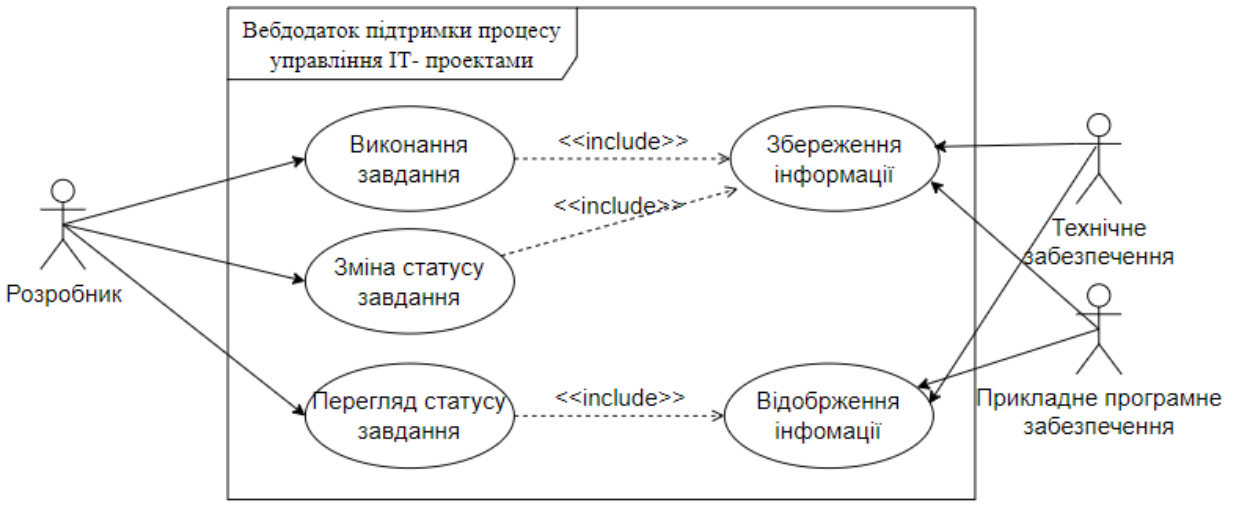

Рисунок 2.5 – Роль розробник

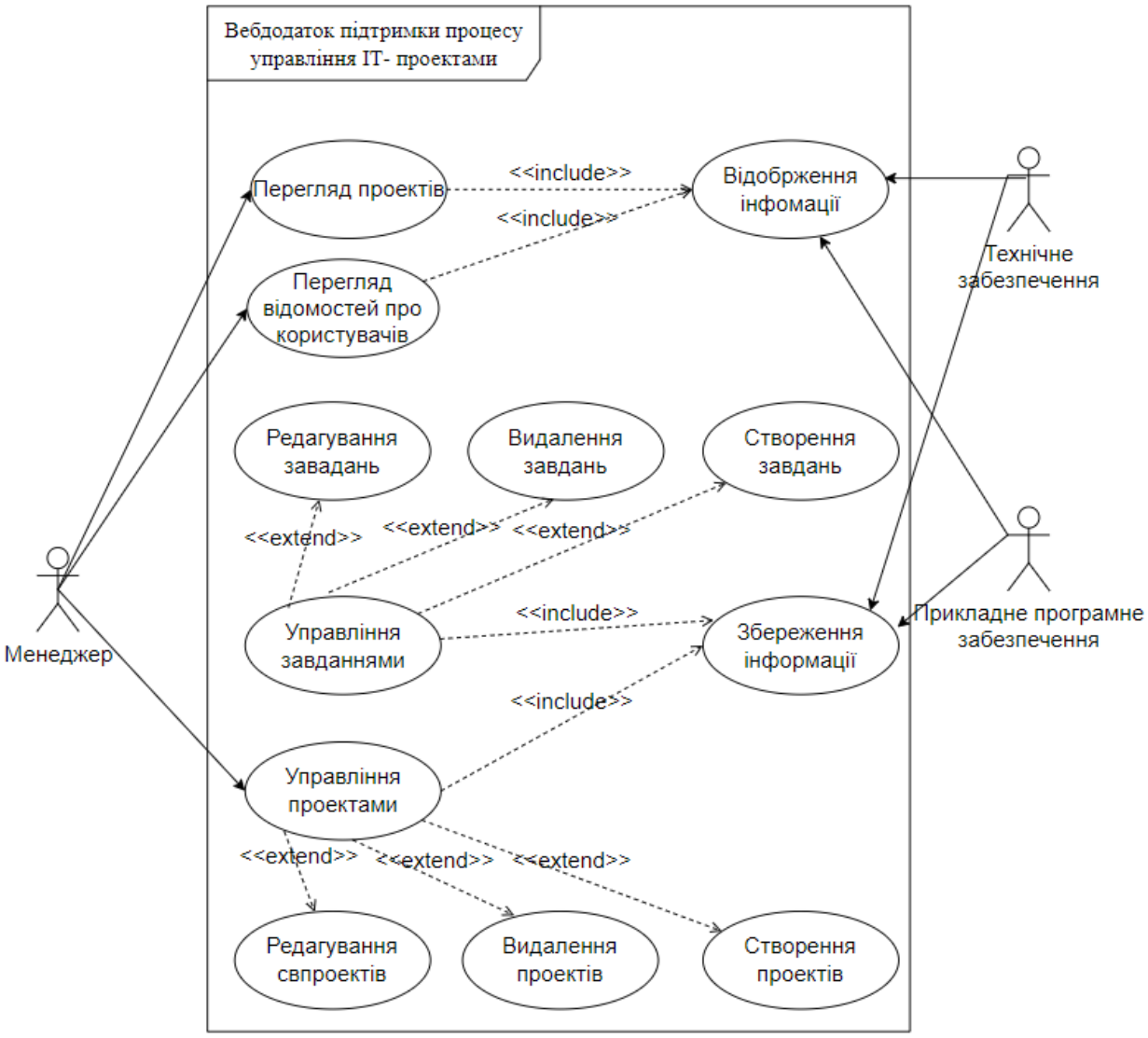

Рисунок 2.6 – Роль менеджер

Варіанти використання для кожної ролі описані нижче:

- **Адміністратор**:
  - Управління користувачами: додавання, редагування та видалення користувачів.
  - Перегляд системних відомостей
  - Перегляд всіх користувачів
- **Менеджер проекту**:
  - Управління проектами
    - Створення проекту: ініціалізація нових проектів.
    - Редагування проекту: внесення змін до існуючих проектів.

        - Перегляд проекту: доступ до інформації про проекти.
    - Управління завданнями
        - Редагування завдань;
        - Створення завдань;
        - Видалення завдань;
    - Перегляд проектів
- **Розробник**:
    - Виконання завдань
    - Зміна статусу завдань: в залежності від стану виконання завдання.
    - Перегляд статусу завдань: моніторинг виконання завдань.

Ця діаграма допомагає чітко визначити ролі користувачів та їх взаємодію з веб-додатком, що є критично важливим для успішного проектування та реалізації системи управління IT-проектами. Створимо діаграми ви

2.3 Проектування моделі бази даних

Логічна модель даних була розроблена для представлення абстрактної структури інформаційної області веблодатку для управління IT-проектами. Модель відображає сутності, їх атрибути та взаємозв’язки між ними.

На рисунку 2.4 показано логічну модель бази даних, для якої була обрана система PostgreSQL. PostgreSQL – це потужна, відкрита система управління реляційними базами даних, що забезпечує високу продуктивність, надійність і підтримку стандартів SQL. Вона підтримує складні запити, транзакції та розширення, що робить її ідеальною для реалізації складних інформаційних систем, таких як наш вебдодаток для управління IT-проектами.

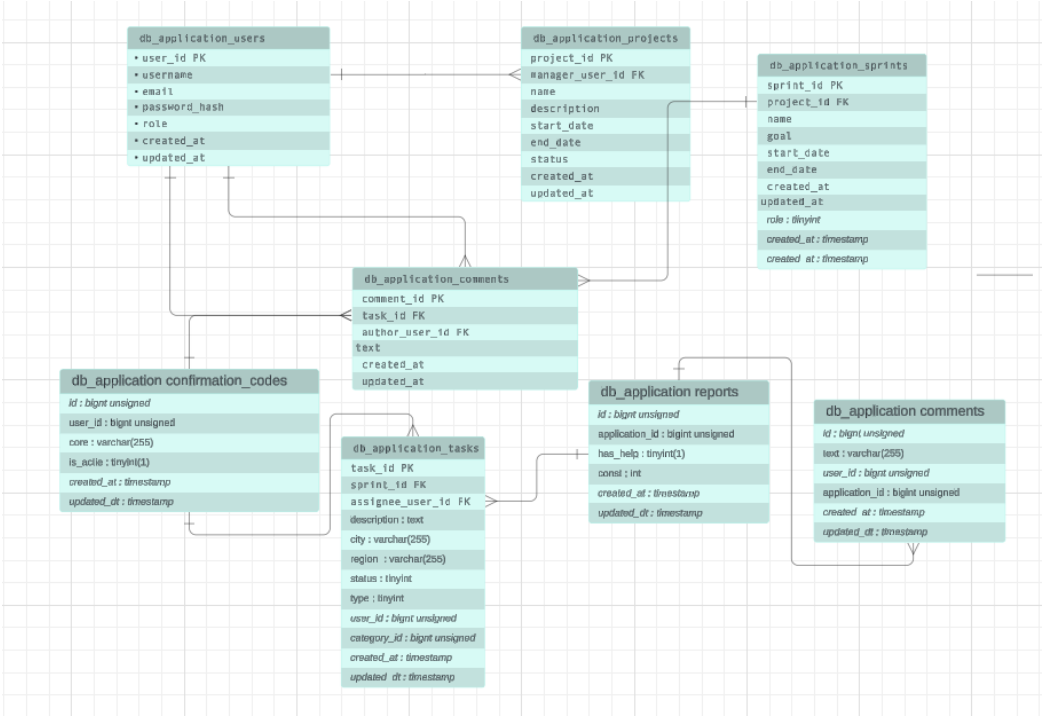

Рисунок 2.7 – Логічна модель бази даних

Далі розглянемо кожну сутність детальніше:

- **db_application_users**: містить інформацію про користувачів системи (user_id, username, email, password_hash, role, created_at, updated_at);
- **db_application_projects**: зберігає інформацію про проекти (project_id, manager_user_id, name, description, start_date, end_date, status, created_at, updated_at);
- **db_application_sprints**: містить дані про спринти в проектах (sprint_id, project_id, name, goal, start_date, end_date, created_at, updated_at);
- **db_application_tasks**: зберігає завдання проектів (task_id, sprint_id, project_id, assignee_user_id, description, start_date, due_date, status, priority, created_at, updated_at);
- **db_application_comments**: містить коментарі до завдань (comment_id, task_id, author_user_id, text, created_at, updated_at);
- **db_application_reports**: зберігає звіти по завданням і проектам (report_id, task_id, report_text, created_at, updated_at);
- **db_application_confirmation_codes**: зберігає коди підтвердження для користувачів (id, user_id, code, is_active, created_at, updated_at).

Кожна сутність бази даних містить необхідні атрибути для забезпечення

зберігання і обробки даних, а також визначає зв’язки між різними сутностями, що забезпечує цілісність даних.

Опис сутностей:

- **db_application_users**:
  - user_id: унікальний ідентифікатор користувача;
  - username: ім'я користувача;
  - email: електронна адреса користувача;
  - password_hash: хеш пароля користувача;
  - role: роль користувача (адміністратор, менеджер проекту, розробник);w
  - created_at: дата створення запису.
  - updated_at: дата останнього оновлення запису.
- **db_application_projects**:
  - project_id: унікальний ідентифікатор проекту.
  - manager_user_id: ідентифікатор користувача, що є менеджером проекту.
  - name: назва проекту.
  - description: опис проекту.
  - start_date: дата початку проекту.
  - end_date: дата завершення проекту.
  - status: статус проекту.
  - created_at: дата створення запису.
  - updated_at: дата останнього оновлення запису.
- **db_application_sprints**:
  - sprint_id: унікальний ідентифікатор спринту.
  - project_id: ідентифікатор проекту, до якого належить спринт.
  - name: назва спринту.
  - goal: ціль спринту.
  - start_date: дата початку спринту.
  - end_date: дата завершення спринту.
  - created_at: дата створення запису.

    - updated_at: дата останнього оновлення запису.
- **db_application_tasks**:
    - task_id: унікальний ідентифікатор завдання.
    - sprint_id: ідентифікатор спринту, до якого належить завдання.
    - project_id: ідентифікатор проекту, до якого належить завдання.
    - assignee_user_id: ідентифікатор користувача, якому призначено завдання.
    - description: опис завдання.
    - start_date: дата початку завдання.
    - due_date: дата завершення завдання.
    - status: статус завдання.
    - priority: пріоритет завдання.
    - created_at: дата створення запису.
    - updated_at: дата останнього оновлення запису.
- **db_application_comments**:
    - comment_id: унікальний ідентифікатор коментаря.
    - task_id: ідентифікатор завдання, до якого належить коментар.
    - author_user_id: ідентифікатор користувача, який залишив коментар.
    - text: текст коментаря.
    - created_at: дата створення запису.
    - updated_at: дата останнього оновлення запису.
- **db_application_reports**:
    - report_id: унікальний ідентифікатор звіту.
    - task_id: ідентифікатор завдання, до якого належить звіт.
    - report_text: текст звіту.
    - created_at: дата створення запису.
    - updated_at: дата останнього оновлення запису.
- **db_application_confirmation_codes**:
    - id: унікальний ідентифікатор коду підтвердження.
    - user_id: ідентифікатор користувача, для якого створено код.
    - code: код підтвердження.

- is_active: статус активності коду.
- created_at: дата створення запису.
- updated_at: дата останнього оновлення запису.

Ця модель забезпечує ефективне зберігання і управління даними, що необхідні для функціонування нашого вебдодатку управління IT-проектами.

2.4 Архітектура програмного продукту

Клієнт-серверна архітектура для вебдодатку було обрано клієнт-серверну архітектуру, яка забезпечує ефективну взаємодію між користувачами та сервером. Клієнт-серверна архітектура дозволяє розподілити обробку даних між клієнтською частиною (фронтенд) та серверною частиною (бекенд), що підвищує продуктивність і надійність системи.

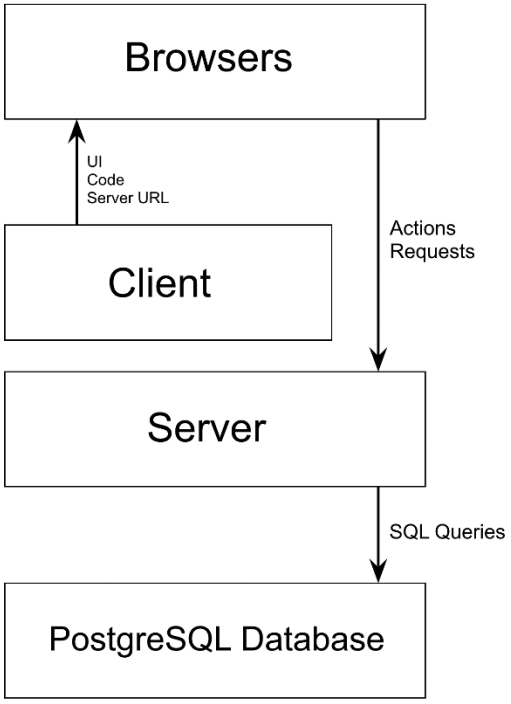

Рисунок 2.8 – Клієнт-серверна архітектура

Принцип роботи клієнт-серверної архітектури полягає в том що клієнтська частина системи, яка зазвичай представлена браузером, взаємодіє з користувачем, надаючи йому інтерфейс для роботи з веб-додатком. Це дозволяє користувачам вводити свої дані і надсилати запити до сервера. У цьому випадку, клієнтська частина написана на JavaScript з використанням фреймворку React.

Цей фреймворк забезпечує швидкий і динамічний користувацький інтерфейс, що значно покращує взаємодію користувачів з системою. React дозволяє створювати інтерактивні елементи інтерфейсу, які швидко реагують на дії користувачів, що забезпечує приємний користувацький досвід.

Серверна частина обробляє запити, що надходять від клієнтів. Вона виконує бізнес-логіку, взаємодіє з базою даних і повертає результати назад до клієнта. Для серверної частини використано Node.js, який дозволяє створювати високопродуктивні сервери з асинхронною обробкою запитів. Це означає, що сервер може обробляти багато запитів одночасно, не блокуючи виконання інших запитів. Node.js є ідеальним вибором для системи, оскільки він підтримує масштабованість і високий рівень продуктивності. Сервер також обробляє аутентифікацію користувачів, управління проектами, завданнями та інші операції, що забезпечує безпеку і надійність системи.

Для зберігання даних використовується PostgreSQL – потужна реляційна база даних, яка забезпечує надійне зберігання даних і підтримку складних запитів. PostgreSQL є відкритою системою управління базами даних, яка відповідає всім сучасним стандартам і забезпечує високу продуктивність, надійність і масштабованість. Сервер взаємодіє з базою даних за допомогою SQL-запитів, виконуючи операції зчитування, запису, оновлення та видалення даних. Це дозволяє забезпечити цілісність і безпеку даних, що зберігаються в системі.

Як працює клієнт-серверна архітектура в додатку, користувачі взаємодіють з веб-додатком через браузер, вводячи свої дані і надсилаючи запити. Наприклад, вони можуть надсилати запити на авторизацію, перегляд проектів або редагування завдань. Браузер відправляє ці запити до клієнтської частини додатку, яка формує HTTP-запити до сервера. Фреймворк React забезпечує динамічний і інтерактивний користувацький інтерфейс, обробляючи введення даних користувачами і відправляючи їх на сервер. Це дозволяє забезпечити швидку і ефективну обробку запитів користувачів і надання їм актуальної інформації.

Сервер на Node.js приймає запити від клієнта, обробляє їх, виконує

необхідну бізнес-логіку і взаємодіє з базою даних для отримання або збереження даних. Сервер обробляє запити асинхронно, що дозволяє одночасно обробляти багато запитів без блокування. Це забезпечує високу продуктивність і масштабованість системи. Сервер надсилає SQL-запити до бази даних PostgreSQL, щоб отримати необхідні дані або зберегти нову інформацію. База даних повертає результати запитів серверу, що дозволяє серверу формувати відповіді на основі результатів запитів до бази даних і надсилати їх назад до клієнта.

Клієнтська частина отримує ці відповіді і оновлює користувацький інтерфейс відповідно до отриманих даних. Це дозволяє забезпечити актуальну інформацію для користувачів про стан проектів, завдань і інші функції вебдодатку. Клієнтська частина відображає отримані дані у браузері, надаючи користувачам можливість взаємодіяти з системою в режимі реального часу.

Такий підхід дозволяє розподілити навантаження між клієнтом і сервером, забезпечуючи високу продуктивність і масштабованість системи. Використання сучасних технологій, таких як React, Node.js і PostgreSQL, забезпечує надійність і ефективність вебдодатку для управління IT-проектами. Ця архітектура дозволяє створювати інтерактивні, масштабовані і надійні вебдодатки, які можуть задовольнити потреби сучасних користувачів і забезпечити високу якість обслуговування.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Програмна реалізація

Розробка вебдодатку для управління IT-проектами є складним процесом, який включає кілька ключових етапів і модулів. У цьому розділі буде розглянений процес розробки додатку, описано основні модулі, представдено приклади коду, а також детально розглянемо поєднання бекенду і фронтенду, роботу API та процес реєстрації користувачів.

3.1.1 Клієнтська частина

Клієнтська частина була розроблена з використанням фреймворку React. Використання цього фреймворку дозволило створити динамічні та інтерактивні інтерфейси користувача, що значно покращує взаємодію користувачів з системою. Клієнтська частина реалізована за допомогою окремих компонентів, кожен з яких відповідає за певну функціональність і може бути легко повторно використаний або змінений без впливу на інші частини системи. Компоненти на React забезпечують високу гнучкість та модульність додатку, що дозволяє швидко адаптувати його до змін вимог або розширювати функціональність.

Було створено кілька ключових компонентів, які забезпечують основні функції системи. Серед них – форми авторизації, панелі управління проектами, сторінки завдань, сторінки звітів, і профілі користувачів. Всі ці компоненти забезпечують зручний інтерфейс для користувачів і сприяють ефективному виконанню їхніх завдань.

Форми авторизації дозволяють користувачам вводити свої електронну пошту та пароль для входу в систему. Для стилізації компонентів використовувався Tailwind CSS, що дозволило швидко і легко створювати адаптивний дизайн. Приклад коду компонента на React для форми авторизації виглядає наступним чином:

```
import React, { useState } from 'react';
import axios from 'axios';
```

```
const LoginForm = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleLogin = async (event) => {
    event.preventDefault();
    try {
      const response = await axios.post('/api/login', { email, password });
      localStorage.setItem('token', response.data.token);
      window.location.href = '/dashboard';
    } catch (err) {
      setError('Invalid credentials');
    }
  };

  return (
    <form onSubmit={handleLogin} className="max-w-sm mx-auto p-4 bg-
white shadow-md rounded">
      <div className="mb-4">
        <label className="block text-gray-700">Email:</label>
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
          className="w-full px-3 py-2 border rounded"
        />
      </div>
      <div className="mb-4">
```

```
                <label className="block text-gray-700">Password:</label>
                <input
                  type="password"
                  value={password}
                  onChange={(e) => setPassword(e.target.value)}
                  required
                  className="w-full px-3 py-2 border rounded"
                />
              </div>
              {error && <div className="mb-4 text-red-500">{error}</div>}
              <button type="submit" className="w-full bg-blue-500 text-white py-2
rounded">Login</button>
            </form>
        );
    };

    export default LoginForm;
```

Цей компонент дозволяє користувачам вводити свої електронну пошту та пароль, і надсилає ці дані на сервер для перевірки. У разі успішної авторизації користувач перенаправляється на панель управління проектами, де він може переглядати та керувати своїми проектами та завданнями. У випадку помилки користувач отримає повідомлення про невірні дані, що дозволяє уникнути неправильної авторизації.

Панель управління проектами надає зручний інтерфейс для перегляду та управління проектами. Вона дозволяє користувачам легко додавати нові проекти, редагувати існуючі та видаляти непотрібні. Для створення цієї панелі був використаний React разом з Tailwind CSS для забезпечення адаптивного та стильного інтерфейсу.

```
    import React, { useState, useEffect } from 'react';
    import axios from 'axios';
```

```
const ProjectList = () => {
  const [projects, setProjects] = useState([]);

  useEffect(() => {
    const fetchProjects = async () => {
      try {
        const response = await axios.get('/api/projects', {
          headers: {
            Authorization: `Bearer ${localStorage.getItem('token')}`
          }
        });
        setProjects(response.data);
      } catch (err) {
        console.error('Error fetching projects', err);
      }
    };

    fetchProjects();
  }, []);

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4">Projects</h1>
      <ul>
        {projects.map(project => (
          <li key={project._id} className="mb-2 p-2 border rounded bg-
gray-100">
            {project.name}
          </li>
        ))}
```

```
            </ul>
        </div>
    );
};

export default ProjectList;
```

Сторінки завдань дозволяють користувачам створювати, редагувати та переглядати завдання. Кожне завдання має опис, дату початку і кінця, статус та пріоритет. Це забезпечує ефективне управління робочим процесом і допомагає користувачам тримати все під контролем.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const TaskList = ({ projectId }) => {
    const [tasks, setTasks] = useState([]);

    useEffect(() => {
        const fetchTasks = async () => {
            try {
                const response = await axios.get(`/api/projects/${projectId}/tasks`, {
                    headers: {
                        Authorization: `Bearer ${localStorage.getItem('token')}`
                    }
                });
                setTasks(response.data);
            } catch (err) {
                console.error('Error fetching tasks', err);
            }
        };
```

```
        fetchTasks();
    }, [projectId]);

    return (
        <div className="container mx-auto p-4">
            <h1 className="text-2xl font-bold mb-4">Tasks</h1>
            <ul>
                {tasks.map(task => (
                    <li key={task._id} className="mb-2 p-2 border rounded bg-gray-
100">
                        {task.description}
                    </li>
                ))}
            </ul>
        </div>
    );
};

export default TaskList;
```

Сторінки звітів надають доступ до аналітичних даних і статистики по проектам. Це допомагає користувачам аналізувати ефективність виконання завдань і приймати обґрунтовані рішення.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const ReportList = ({ projectId }) => {
    const [reports, setReports] = useState([]);

    useEffect(() => {
        const fetchReports = async () => {
```

```
            try {
                const response = await axios.get(`/api/projects/${projectId}/reports`,
{
                    headers: {
                        Authorization: `Bearer ${localStorage.getItem('token')}`
                    }
                });
                setReports(response.data);
            } catch (err) {
                console.error('Error fetching reports', err);
            }
        };

        fetchReports();
    }, [projectId]);

    return (
        <div className="container mx-auto p-4">
            <h1 className="text-2xl font-bold mb-4">Reports</h1>
            <ul>
                {reports.map(report => (
                    <li key={report._id} className="mb-2 p-2 border rounded bg-
gray-100">
                        {report.text}
                    </li>
                ))}
            </ul>
        </div>
    );
};
```

```
export default ReportList;
```

Цей підхід до розробки клієнтської частини дозволив створити інтерактивний, зручний і надійний інтерфейс, що забезпечує всі необхідні функції для ефективного управління IT-проектами. Завдяки використанню React і Tailwind CSS вдалося досягти високого рівня продуктивності і зручності використання, що є ключовими факторами для успішної реалізації проекту.

3.1.2 Серверна частина (Backend)

Серверна частина нашої системи була розроблена з використанням Node.js та фреймворку Express. Це забезпечило нам можливість створювати високопродуктивні сервери з асинхронною обробкою запитів, що дозволяє одночасно обробляти багато запитів без блокування. Серверна частина відповідає за обробку запитів від клієнтів, виконання бізнес-логіки, взаємодію з базою даних і повернення результатів назад до клієнта. У цьому розділі ми детально розглянемо основні модулі серверної частини, представимо приклади коду та опишемо процес поєднання бекенду з фронтендом.

Основні модулі серверної частини

- Аутентифікація та авторизація;
- Управління проектами;
- Управління завданнями;
- Обробка коментарів;
- Генерація звітів.

Для забезпечення безпеки додатку були реалізовані механізми аутентифікації та авторизації за допомогою JWT (JSON Web Tokens). Користувачі проходять процес авторизації, отримують токен, який використовується для доступу до захищених ресурсів.

Приклад коду для реєстрації користувачів:

```
const express = require('express');
const bcrypt = require('bcrypt');
```

```
const jwt = require('jsonwebtoken');
const User = require('./models/User');
const router = express.Router();

router.post('/register', async (req, res) => {
  const { username, email, password } = req.body;
  try {
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, email, password: hashedPassword
});
    await newUser.save();

    const token = jwt.sign({ userId: newUser._id }, 'secret_key', { expiresIn:
'1h' });
    res.status(201).json({ token });
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
});

module.exports = router;
```

Приклад коду для аутентифікації користувачів включає створення маршруту для обробки запитів на вхід до системи. Цей маршрут приймає електронну пошту і пароль користувача, перевіряє їх коректність і повертає JWT токен у випадку успішної аутентифікації.

```
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: 'User not found' });
    }

    const isMatch = await bcrypt.compare(password, user.password_hash);
    if (!isMatch) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }

    const token = jwt.sign({ userId: user._id }, 'secret_key', { expiresIn: '1h' });
    res.json({ token });
  } catch (err) {
    res.status(500).json({ message: 'Server error' });
  }
});
```

Серверна частина також включає модуль для управління проектами. Цей модуль дозволяє створювати нові проекти, редагувати існуючі і видаляти завершені або непотрібні проекти. Він також забезпечує доступ до списку проектів, які належать певному користувачу.

Приклад коду маршруту для створення проекту:

```
const Project = require('./models/Project');
const authenticateToken = require('./middleware/authenticateToken');

router.post('/projects', authenticateToken, async (req, res) => {
  const { name, description, startDate, endDate } = req.body;
  try {
```

```
        const newProject = new Project({
            manager_user_id: req.user.userId,
            name,
            description,
            start_date: startDate,
            end_date: endDate,
        });
        await newProject.save();
        res.status(201).json(newProject);
    } catch (err) {
        res.status(500).json({ message: 'Server error' });
    }
});

module.exports = router;
```

Модуль управління завданнями дозволяє користувачам створювати, редагувати та видаляти завдання в рамках проектів. Завдання мають опис, дату початку і кінця, статус та пріоритет.

Приклад коду маршруту для створення завдання:

```
const Task = require('./models/Task');

router.post('/projects/:projectId/tasks', authenticateToken, async (req, res) => {
    const { description, startDate, dueDate, priority } = req.body;
    try {
        const newTask = new Task({
            project_id: req.params.projectId,
            description,
            start_date: startDate,
            due_date: dueDate,
            priority,
```

```
            status: 'Pending',
            assignee_user_id: req.user.userId,
        });
        await newTask.save();
        res.status(201).json(newTask);
    } catch (err) {
        res.status(500).json({ message: 'Server error' });
    }
});

module.exports = router;
```

Модуль обробки коментарів дозволяє користувачам додавати коментарі до завдань. Це забезпечує можливість обговорення та уточнення деталей завдань в рамках проектів.

Приклад коду маршруту для додавання коментаря:

```
const Comment = require('./models/Comment');
router.post('/tasks/:taskId/comments', authenticateToken, async (req, res) => {
    const { text } = req.body;
    try {
        const newComment = new Comment({
            task_id: req.params.taskId,
            author_user_id: req.user.userId,
            text,
        });
        await newComment.save();
        res.status(201).json(newComment);
    } catch (err) {
        res.status(500).json({ message: 'Server error' });
    }
});
```

```
module.exports = router;
```

Модуль генерації звітів надає можливість створювати звіти по завданням і проектам. Це допомагає користувачам аналізувати ефективність виконання завдань і приймати обґрунтовані рішення.

Приклад коду маршруту для створення звіту:

```
const Report = require('./models/Report');

router.post('/projects/:projectId/reports', authenticateToken, async (req, res) =>
{
    const { text } = req.body;
    try {
        const newReport = new Report({
            project_id: req.params.projectId,
            report_text: text,
        });
        await newReport.save();
        res.status(201).json(newReport);
    } catch (err) {
        res.status(500).json({ message: 'Server error' });
    }
});

module.exports = router;
```

Для забезпечення взаємодії між клієнтською та серверною частинами були реалізовані методи для виконання запитів до API з клієнтської частини. Використання бібліотеки Axios дозволяє відправляти HTTP-запити з React-компонентів. Це забезпечує передачу даних між клієнтом і сервером, а також отримання необхідної інформації для відображення на клієнтській частині.

3.1.3 Взаємодія API

Для забезпечення ефективної взаємодії між клієнтською та серверною частинами вебдодатку була розроблена API (Application Programming Interface). API дозволяє клієнтським додаткам (зазвичай браузеру) надсилати запити до серверу для отримання або надсилання даних, що забезпечує необхідну функціональність додатку. Вебдодаток використовує RESTful API, що дозволяє створювати, читати, оновлювати та видаляти ресурси, пов'язані з користувачами, проектами, завданнями, коментарями та звітами.

Основні концепції RESTful API базується на використанні стандартних HTTP методів для взаємодії з ресурсами:

- **GET**: отримання даних з сервера;
- **POST**: створення нових ресурсів на сервері;
- **PUT**: оновлення існуючих ресурсів на сервері;
- **DELETE**: видалення ресурсів з сервера;

Кожен ресурс має свій унікальний URL (Uniform Resource Locator), що дозволяє легко взаємодіяти з ним за допомогою відповідних HTTP методів.

Клієнтська частина використовує бібліотеку Axios для виконання HTTP-запитів до API. Наприклад, для отримання списку проектів клієнт надсилає GET-запит до відповідного кінцевого пункту API.

Приклад коду для отримання списку проектів у React-компоненті:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const ProjectList = () => {
  const [projects, setProjects] = useState([]);

  useEffect(() => {
    const fetchProjects = async () => {
      try {
        const response = await axios.get('/api/projects', {
```

```
                headers: {
                    Authorization: `Bearer ${localStorage.getItem('token')}`
                }
            });
            setProjects(response.data);
        } catch (err) {
            console.error('Error fetching projects', err);
        }
    };

    fetchProjects();
}, []);

return (

    <div className="container mx-auto p-4">
        <h1 className="text-2xl font-bold mb-4">Projects</h1>
        <ul>
            {projects.map(project => (
                <li key={project._id} className="mb-2 p-2 border rounded bg-
gray-100">
                    {project.name}
                </li>
            ))}
        </ul>
    </div>
);
};

export default ProjectList;
```

У цьому прикладі, після рендерингу компонента, викликається функція **fetchProjects**, яка надсилає GET-запит до **/api/projects**. Запит включає заголовок Authorization з токеном користувача для аутентифікації. Після отримання даних від сервера, вони зберігаються у стані компонента і відображаються у вигляді списку.

Для створення нового проекту клієнтська частина надсилає POST-запит до відповідного кінцевого пункту API з необхідними даними про проект.

Приклад коду для створення нового проекту:

```
import React, { useState } from 'react';
import axios from 'axios';

const CreateProject = () => {
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');

  const handleSubmit = async (event) => {
    event.preventDefault();
    try {
      const response = await axios.post('/api/projects', { name, description }, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem('token')}`
        }
      });
      // Handle successful project creation
      console.log('Project created:', response.data);
    } catch (err) {
      console.error('Error creating project', err);
    }
  };
```

```
    return (
      <form onSubmit={handleSubmit} className="max-w-sm mx-auto p-4 bg-white shadow-md rounded">
        <div className="mb-4">
          <label className="block text-gray-700">Project Name:</label>
          <input
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)}
            required
            className="w-full px-3 py-2 border rounded"
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700">Description:</label>
          <textarea
            value={description}
            onChange={(e) => setDescription(e.target.value)}
            required
            className="w-full px-3 py-2 border rounded"
          />
        </div>
        <button type="submit" className="w-full bg-blue-500 text-white py-2 rounded">Create Project</button>
      </form>
    );
  };

  export default CreateProject;
```

У цьому прикладі, після заповнення форми, викликається функція

handleSubmit, яка надсилає POST-запит до /api/projects з даними про новий проект. Запит включає заголовок Authorization з токеном користувача для аутентифікації. Для оновлення завдання клієнтська частина надсилає PUT-запит до відповідного кінцевого пункту API з оновленими даними завдання.

Приклад коду для оновлення завдання:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const UpdateTask = ({ taskId }) => {
    const [description, setDescription] = useState('');
    const [status, setStatus] = useState('');

    useEffect(() => {
        const fetchTask = async () => {
            try {
                const response = await axios.get(`/api/tasks/${taskId}`, {
                    headers: {
                        Authorization: `Bearer ${localStorage.getItem('token')}`
                    }
                });
                setDescription(response.data.description);
                setStatus(response.data.status);
            } catch (err) {
                console.error('Error fetching task', err);
            }
        };

        fetchTask();
    }, [taskId]);

    const handleSubmit = async (event) => {
```

```
        event.preventDefault();
        try {
          const response = await axios.put(`/api/tasks/${taskId}`, { description, status }, {
            headers: {
              Authorization: `Bearer ${localStorage.getItem('token')}`
            }
          });
          // Handle successful task update
          console.log('Task updated:', response.data);
        } catch (err) {
          console.error('Error updating task', err);
        }
      };

      return (
        <form onSubmit={handleSubmit} className="max-w-sm mx-auto p-4 bg-white shadow-md rounded">
          <div className="mb-4">
            <label className="block text-gray-700">Description:</label>
            <textarea
              value={description}
              onChange={(e) => setDescription(e.target.value)}
              required
              className="w-full px-3 py-2 border rounded"
            />
          </div>
          <div className="mb-4">
            <label className="block text-gray-700">Status:</label>
            <input
              type="text"
```

```
                    value={status}
                    onChange={(e) => setStatus(e.target.value)}
                    required
                    className="w-full px-3 py-2 border rounded"
                />
            </div>
            <button type="submit" className="w-full bg-blue-500 text-white py-2 rounded">Update Task</button>
        </form>
    );
};

export default UpdateTask;
```

Для видалення завдання клієнтська частина надсилає DELETE-запит до відповідного кінцевого пункту API.

Приклад коду для видалення завдання:

```
import React from 'react';
import axios from 'axios';

const DeleteTask = ({ taskId }) => {
    const handleDelete = async () => {
        try {
            await axios.delete(`/api/tasks/${taskId}`, {
                headers: {
                    Authorization: `Bearer ${localStorage.getItem('token')}`
                }
            });
            // Handle successful task deletion
            console.log('Task deleted');
        } catch (err) {
```

```
            console.error('Error deleting task', err);
        }
    };

    return (
        <button onClick={handleDelete} className="bg-red-500 text-white py-2 px-4 rounded">
            Delete Task
        </button>
    );
};

export default DeleteTask;
```

Для забезпечення інтеграції між клієнтською та серверною частинами вебдодатку, API було розроблено таким чином, щоб забезпечити легке і зрозуміле взаємодію між компонентами. Використання бібліотеки Axios в клієнтській частині дозволило відправляти HTTP-запити до сервера і обробляти відповіді, що надходять від нього. Це забезпечило плавну інтеграцію між клієнтською і серверною частинами, дозволяючи створювати, читати, оновлювати і видаляти дані у реальному часі.

Цей підхід до розробки серверної частини дозволив створити надійну, масштабовану і продуктивну систему, яка відповідає всім вимогам користувачів. Використання Node.js, Express і PostgreSQL забезпечило високу продуктивність і надійність серверної частини, що дозволяє ефективно обробляти запити користувачів і забезпечувати безперебійну роботу вебдодатку для управління IT-проектами.

3.2 Робота вебдодатку

Форма авторизації є першою сторінкою, яку бачить користувач при взаємодії з додатком. Вона забезпечує критичну функцію захисту даних і

контролю доступу до системи. На цій сторінці користувачі можуть увійти до системи за допомогою свого облікового запису Google або через введення корпоративної електронної пошти та пароля(рис.3.1). Додатково є опція "Forgot Password?", яка дозволяє відновити доступ у разі втрати пароля. Кнопка "Log In" запускає процес аутентифікації користувача. Після введення даних і натискання кнопки "Log In" формується HTTP-запит до серверу, який перевіряє введені дані. У разі успішної авторизації користувач отримує JWT токен, який зберігається в локальному сховищі браузера і використовується для подальших запитів до захищених ресурсів системи.

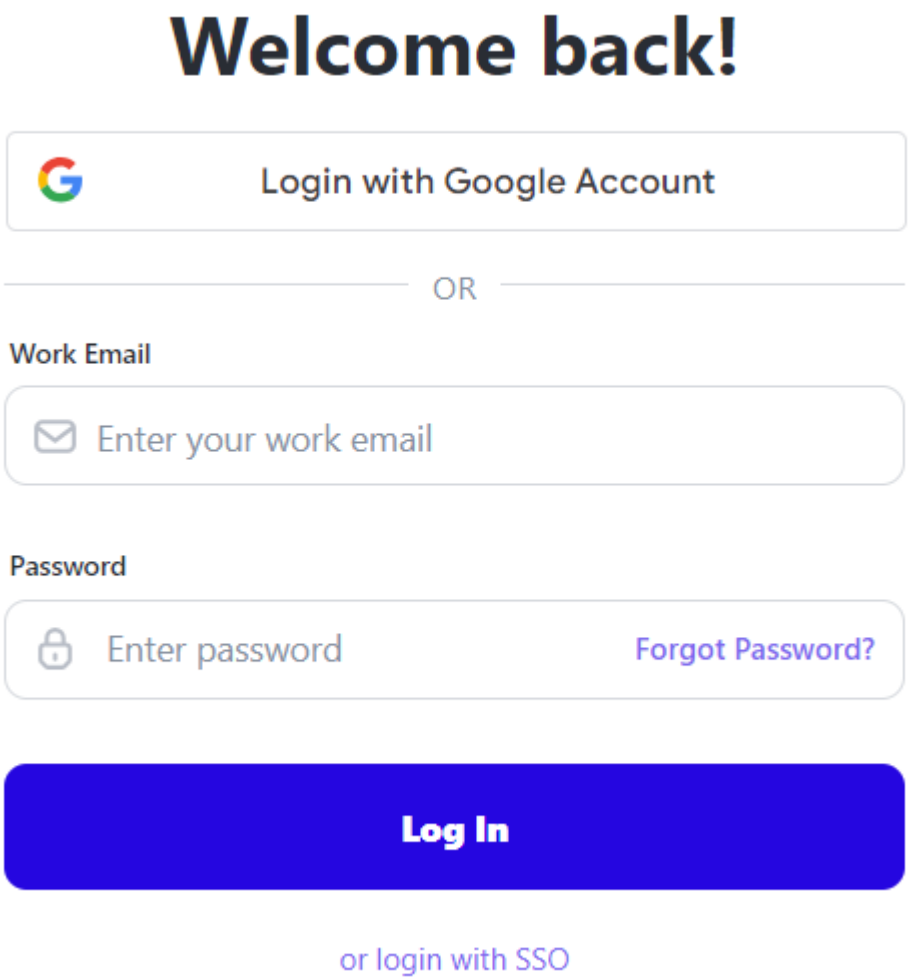

Рисунок 3.1 – Форма авторизації

Сторінка реєстрації дозволяє новим користувачам створити обліковий запис для доступу до системи. Вона включає поля для введення імені

користувача, електронної пошти, пароля та підтвердження пароля(рис.3.2). Після заповнення форми користувач надсилає свої дані на сервер для створення нового облікового запису. У разі успішної реєстрації користувач автоматично отримує доступ до системи. Це дозволяє новим користувачам легко створити обліковий запис і почати користуватися функціоналом додатку.

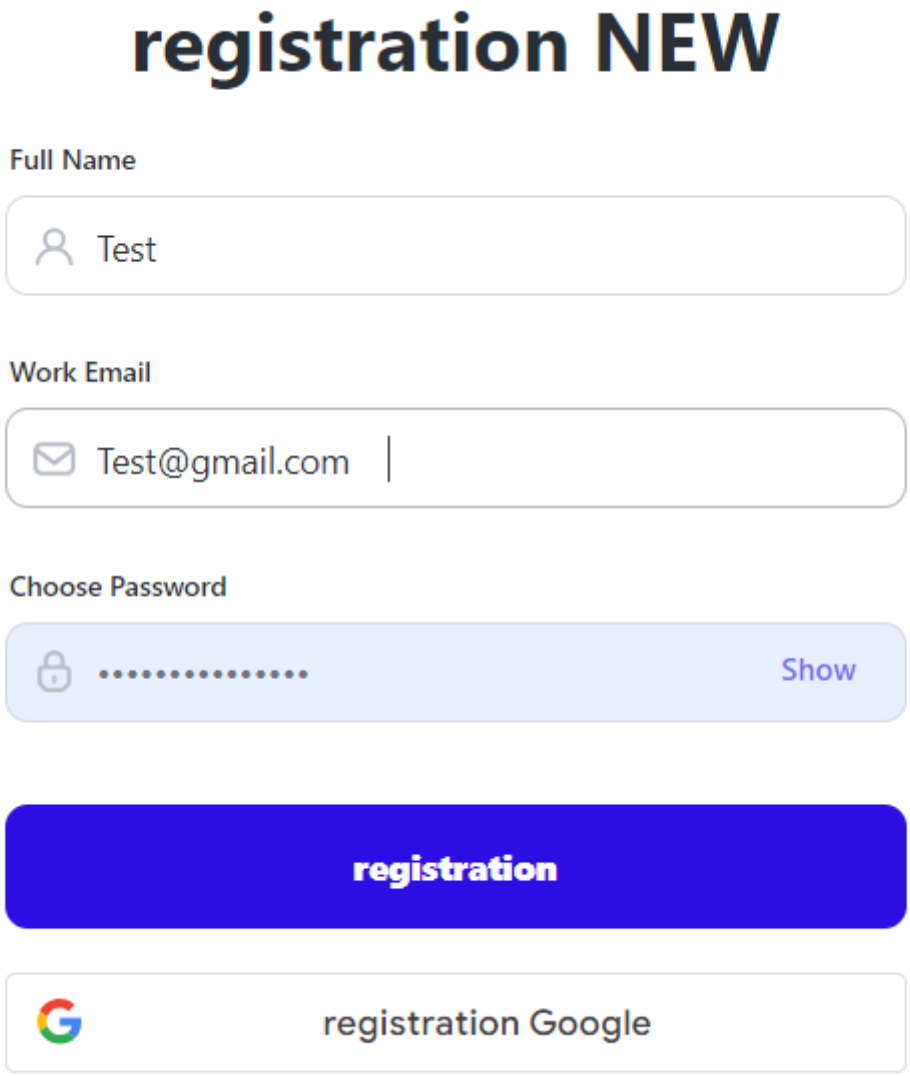

Рисунок 3.2 – Форма реєстрації

Сторінка створення проекту надає користувачам інтерфейс для додавання нових проектів. Вона включає поля для введення назви проекту, опису, дати початку та кінця(рис.3.3). Користувачі можуть додати всі необхідні деталі і створити проект, який буде збережений в базі даних і доступний для подальшого управління. Це забезпечує зручний і зрозумілий спосіб для користувачів

розпочати нові проекти і планувати роботу.

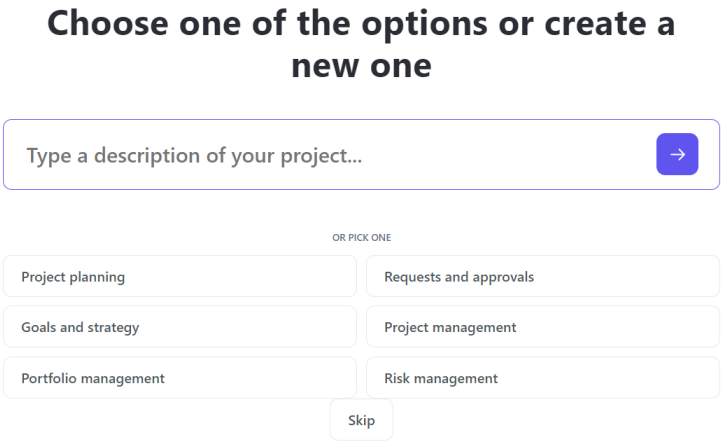

Рисунок 3.3 – Початок створення проекту

На початковій сторінці створення завдань (рис.3.4) користувачі можуть створювати нові завдання в рамках існуючих проектів. Сторінка включає поля для введення опису завдання, вибору відповідальної особи, встановлення дати початку та кінця, а також пріоритету. Ця сторінка забезпечує ефективне управління завданнями і дозволяє тримати всі робочі процеси під контролем.

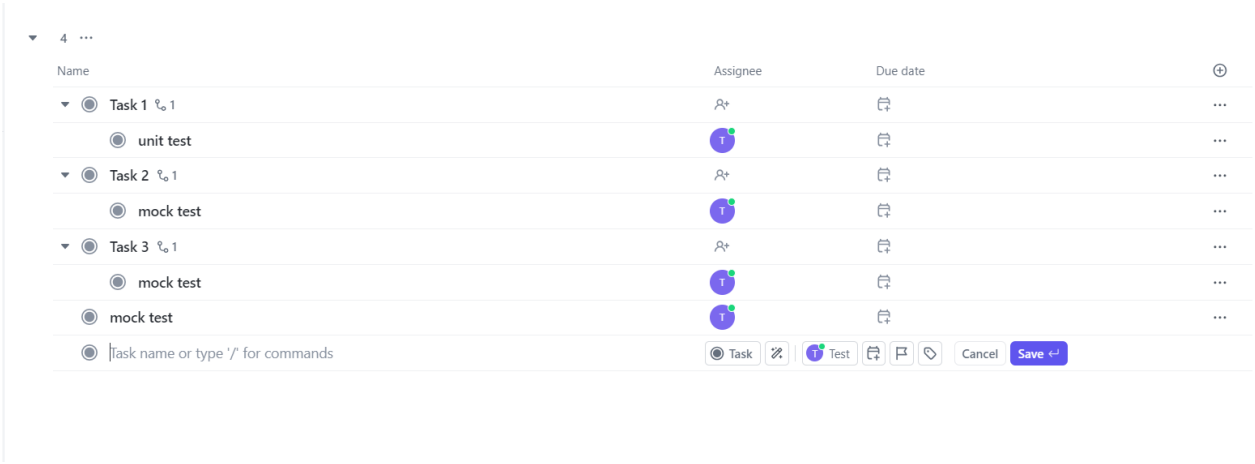

Рисунок 3.4 – Початкова сторінка створення завдань

Сторінка з особистими завданнями (рис.3.5) надає користувачам можливість переглядати всі завдання, які їм призначені. Вона включає список завдань з деталями про кожне завдання, такими як опис, статус, дата початку та кінця. Користувачі можуть швидко переглядати свої завдання і відстежувати їх

виконання.

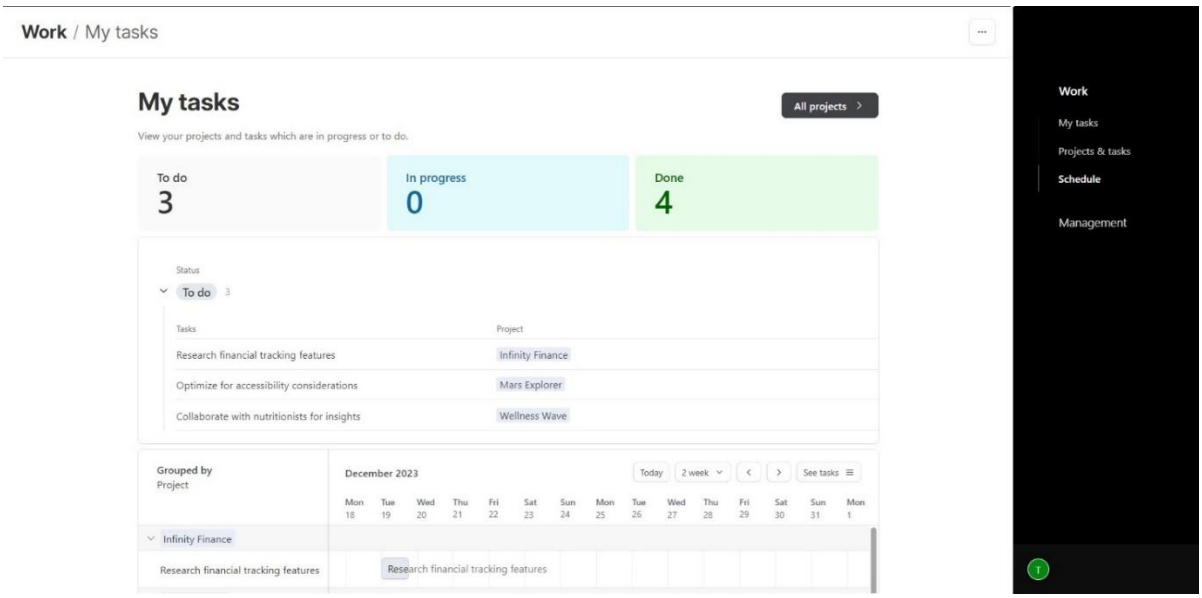

Рисунок 3.5 – Сторінка з особистими завданнями

На сторінці з контролем завданням адміністратори мають доступ до спеціальної сторінки (рис.3.6) для контролю завдань. Ця сторінка дозволяє переглядати всі завдання, призначені всім користувачам, і відстежувати їх виконання. Адміністратори можуть редагувати завдання, змінювати їх статус та пріоритет, а також призначати нових відповідальних осіб. Це забезпечує централізований контроль і координацію роботи в рамках проектів.

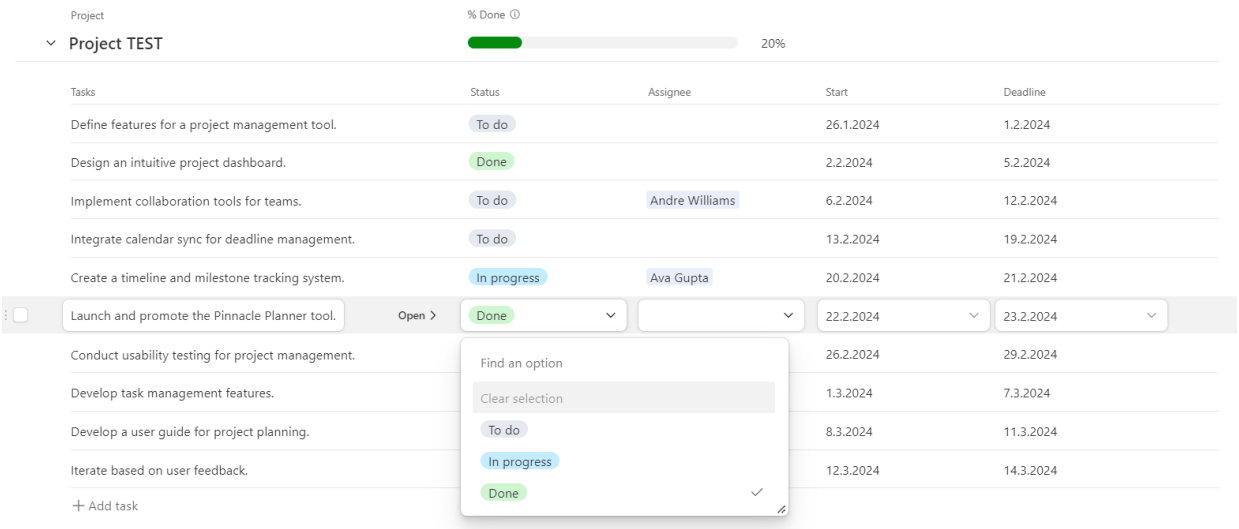

Рисунок 3.6 – Сторінка з контролю завданнями(адміністратор)

На сторінці з контролю користувачів адміністратори можуть переглядати

список усіх користувачів системи (рис.3.7), їхні ролі та активність. Вони можуть додавати нових користувачів, редагувати інформацію про існуючих користувачів, а також видаляти користувачів, якщо це необхідно. Сторінка забезпечує повний контроль над управлінням користувачами в системі і дозволяє адмініструвати доступ до різних функцій.

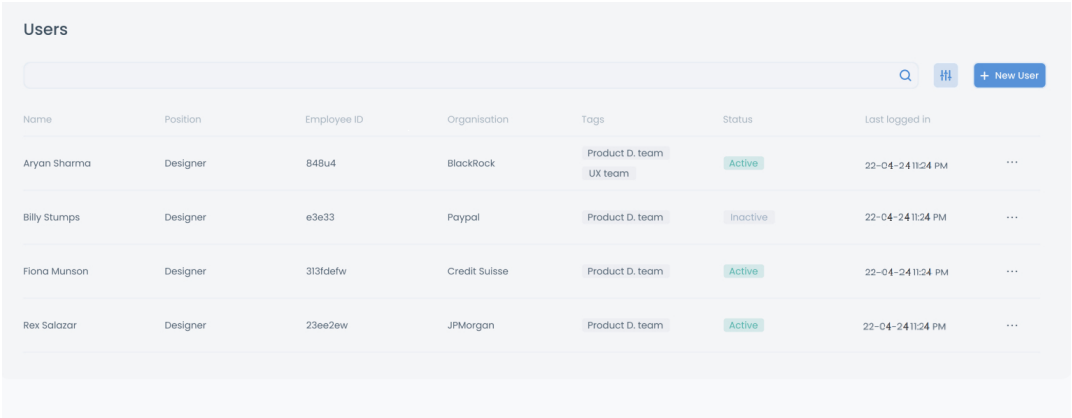

Рисунок 3.7 – Сторінка з контролю користувачів(адміністратор)

Сторінка з календарним планом надає користувачам зручний інтерфейс для перегляду (рис.3.8) всіх проектів і завдань у форматі календаря. Це дозволяє візуалізувати всі поточні і майбутні завдання, відстежувати дедлайни і планувати робочий процес. Користувачі можуть легко переглядати, які завдання потрібно виконати і коли, що сприяє кращій організації роботи.

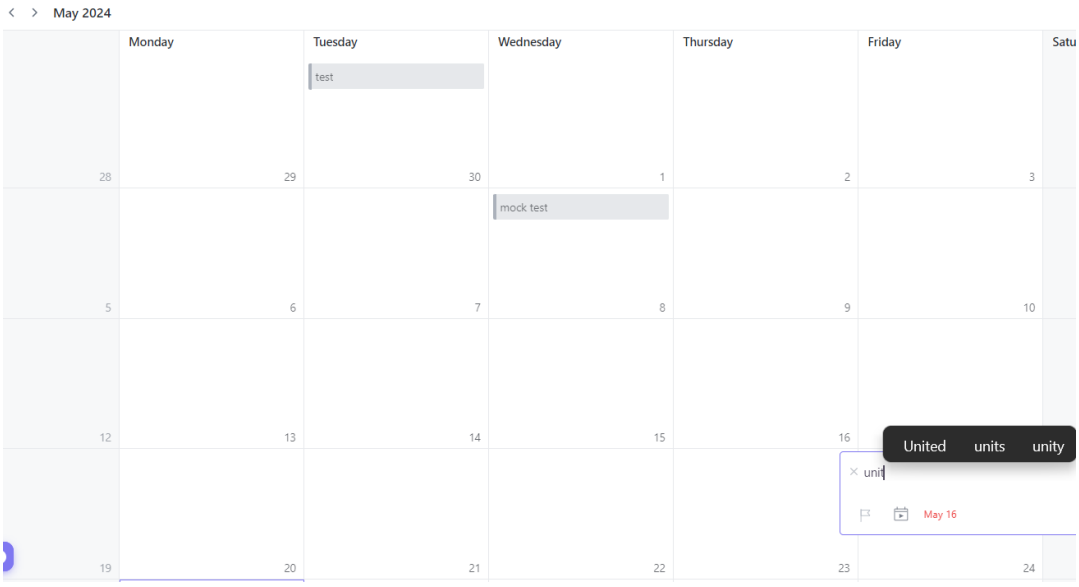

Рисунок 3.8 – Сторінка з календарним планом

Ці сторінки є важливими компонентами додатку, що забезпечують ефективне управління проектами та завданнями, зручний доступ до необхідної інформації і контроль за виконанням робіт.

3.3 Тестування вебдодатку

Тестування є критично важливим етапом у процесі розробки програмного забезпечення. Воно дозволяє виявляти помилки та недоліки на ранніх стадіях розробки, забезпечуючи тим самим високу якість і стабільність коду. Тестування допомагає гарантувати, що всі функції працюють належним чином, а зміни в коді не призводять до неочікуваних наслідків.

Для проекту використали Jest, котрий забезпечує ефективне і зручне написання та виконання тестів. Завдяки Jest покрили код тестами і переконались, що всі модулі функціонують коректно.

На рисунку 3.9 зображено процес запуску тестів за допомогою Jest. Після налаштування тестового середовища і написання тестових випадків, виконуємо всі тести, щоб переконатися в правильності роботи додатку.

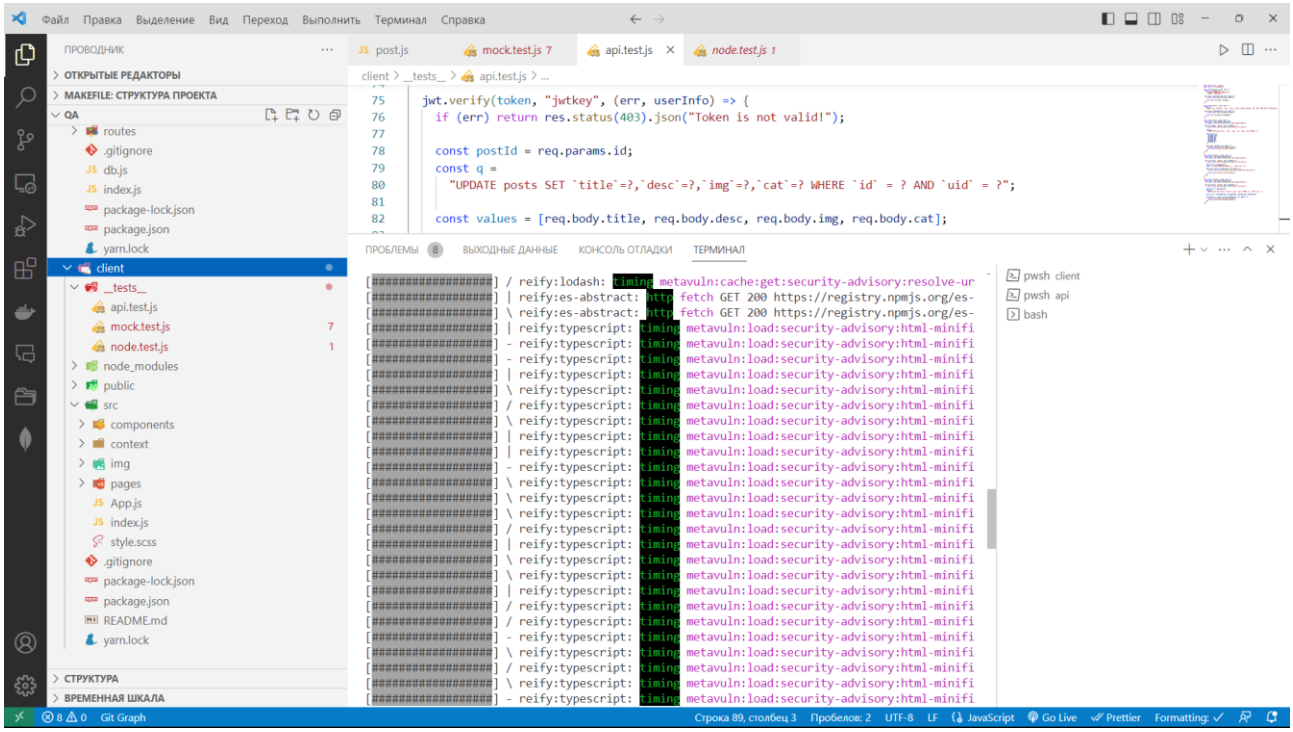

Рисунок 3.9 – Запуск тестів

На рисунку 3.10 показані результати успішного проходження тестів. Це свідчить про те, що всі функції та модулі додатку працюють належним чином, без помилок. Успішне проходження тестів підтверджує, що код відповідає всім вимогам і забезпечує надійність та стабільність системи.

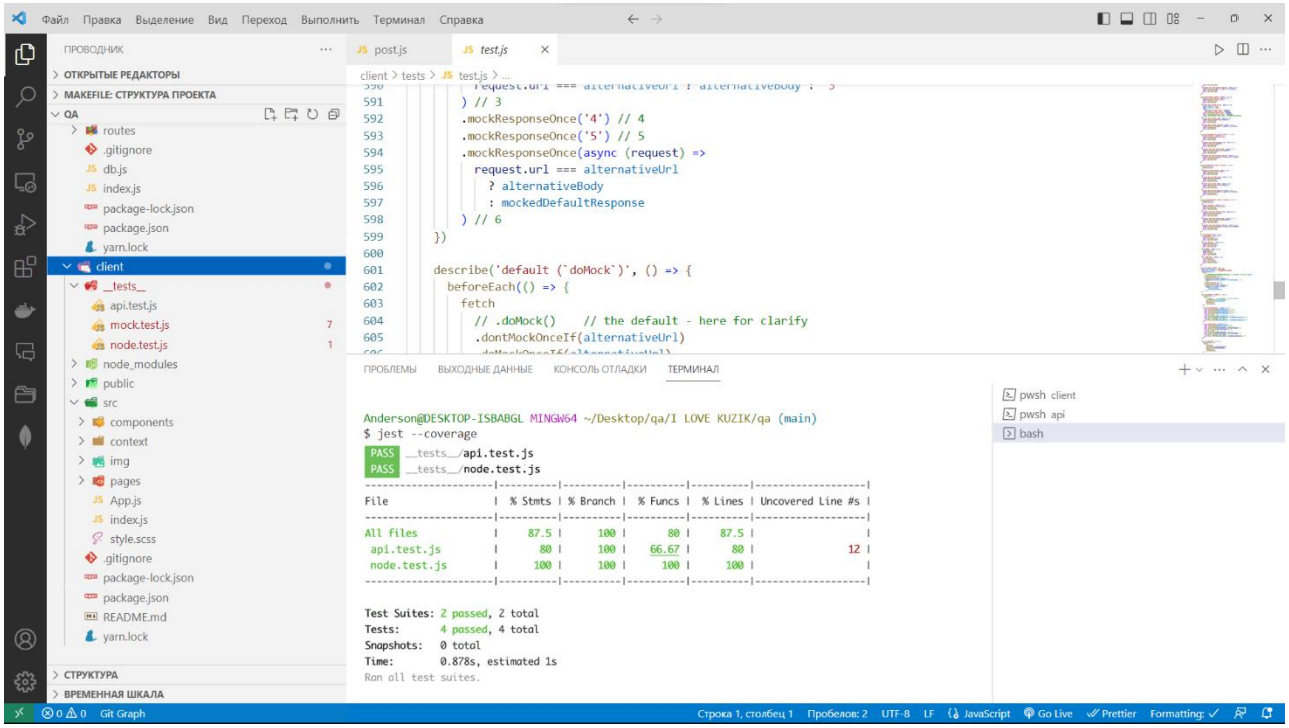

Рисунок 3.10 – Успішно пройдені тести

Тестування допомогло впевнитися у високій якості додатку і готовності його до використання в реальних умовах.

ВИСНОВОК

У ході виконання кваліфікаційної роботи бакалавра було реалізовано вебдодаток підтримки процесу управління IT-проектами. Для успішної його реалізації було виконано низку ключових завдань: аналіз предметної області, створення технічного завдання та планування виконання робіт для розробки вебдодатку підтримки управління IT-проектами. Виконання цих завдань було спрямовано на підготовку та створення ефективного інструменту управління, що може відповідати вимогам сучасних IT-проектів.

Аналіз предметної області включав в себе огляд останніх досліджень та аналіз програмних продуктів-аналогів (Microsoft Project, Jira, Asana, Trello, Basecamp). Цей крок допоміг визначити сильні та слабкі сторони існуючих рішень, що, в свою чергу, дало можливість сформувати уявлення про необхідні функції та можливості для нового додатку. Такий аналіз забезпечив фундамент для розробки технічного завдання.

Створення технічного завдання було важливим етапом, який оформлював вимоги до додатку та окреслював мету та цілі проекту. Це завдання було необхідно для забезпечення чіткого розуміння кінцевих продуктів розробки та їх відповідності вимогам користувачів.

Планування виконання робіт включало деталізацію кроків процесу розробки, визначення ресурсів та таймлайну проекту. Цей процес є критично важливим для забезпечення своєчасного виконання всіх етапів проекту.

Окрім цих основних завдань, було виконано ряд додаткових завдань, включаючи розробку архітектури системи, реалізацію основних функціональних модулів, тестування вебдодатку та впровадження системи безпеки для захисту даних користувачів. Виконання цих завдань забезпечило створення високоякісного інструменту для управління IT-проектами, який відповідає сучасним вимогам та стандартам.

Таким чином, виконана робота не лише забезпечила створення ефективного вебдодатку для управління IT-проектами, але й дозволила отримати

практичні навички та досвід, необхідні для подальшої професійної діяльності у сфері інформаційних технологій.

Подальший розвиток вебдодатку для управління IT-проектами може включати розширення функціональності та інтеграцію з популярними сервісами, такими як Slack, Google Drive і GitHub. Важливо також розробити мобільний додаток, щоб забезпечити зручний доступ з популярного типу пристроїв.

Покращення користувацького досвіду можна досягти за допомогою нових інтерфейсів та функцій персоналізації, які дозволять користувачам налаштовувати додаток під свої потреби. Вдосконалення системи безпеки включає запровадження двофакторної аутентифікації, регулярні аудити безпеки та шифрування даних. Підтримка користувачів може бути покращена через розширену базу знань, вебінари, тренінги та багатомовну підтримку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK® Guide). – [Електронний ресурс] – URL: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok.
2. Schwaber, K. Agile Project Management with Scrum. – [Електронний ресурс] – URL: https://www.scrumguides.org/scrum-guide.html .
3. Scrum.org. The Scrum Guide. – [Електронний ресурс] – URL: https://www.scrumguides.org/scrum-guide.html.
4. Edbrick. AI & ML in Project Management: Exploring the Potential (2023). – [Електронний ресурс] – URL: https://edbrick.com/ai-ml-in-project-management-exploring-the-potential-2023.
5. Project Management Institute (PMI). Artificial Intelligence in Project Management. – [Електронний ресурс] – URL: https://www.pmi.org.
6. Atlassian . – [Електронний ресурс] – https://www.atlassian.com/.
7. Microsof. – [Електронний ресурс] – https://www.microsoft.com/uk-ua/.
8. DevOps Adoption Guidelines, Challenges, and Benefits: A Systematic Review. – [Електронний ресурс] – URL: https://icrrd.com/public/media/27-03-2023-153832DevOps%20Adoption.pdf.
9. A guide to DevOps project management: Essential tools and best practices. – [Електронний ресурс] – URL: https://www.teamwork.com/blog/devops-project-management/.
10. Schwaber, K., & Sutherland, J. (2020). Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust. Wiley.
11. Integration, Application and Importance of Collaboration in Sustainable Project Management . – [Електронний ресурс] – URL: https://www.mdpi.com/2071-1050/12/2/585.
12. Kanban . – [Електронний ресурс] – URL: https://monday.com
13. Microsoft Project. – [Електронний ресурс] – URL:

https://www.microsoft.com/uk-ua/microsoft-365/project/project-management-software

14. Jira by Atlassian. – [Електронний ресурс] – URL: https://www.atlassian.com/software/jira
15. Asana. – [Електронний ресурс] – URL: https://asana.com/
16. Trello. – [Електронний ресурс] – URL: https://trello.com/
17. How to Identify Your Target Audience in 2023: A 5-Step Guide . – [Електронний ресурс] – URL: https://www.simpletiger.com/
18. Artificial Intelligence Enabled Project Management: A Systematic Literature Review . – [Електронний ресурс] – URL:https://www.mdpi.com/2076-3417/13/8/5014
19. PostgreSQL. – [Електронний ресурс] – URL: https://www.postgresql.org/docs/.
20. Що таке MVP – як правильно створити мінімально життєздатний продукт. – [Електронний ресурс] – URL: https://brainlab.com.ua/uk/blog-uk/shho-take-mvp-yak-pravylno-stvoryty-minimalno-zhyttyezdatnyj-produkt
21. Node.js. – [Електронний ресурс] – URL: https://nodejs.org/en/docs/.
22. React. – [Електронний ресурс] – URL: https://reactjs.org/docs/getting-started.html.
23. Tailwind CSS. – [Електронний ресурс] – URL: https://tailwindcss.com/docs.

ДОДАТОК А.

ТЕХНІЧНЕ ЗАВДАННЯ

на створення

«Вебдодаток підтримки процесу управління ІТ- проектами»

ПОГОДЖЕНО:

Доцент кафедри інформаційних технологій

_____ Нагорний В.В

Студент групи ІТ-02____

_____ Серебрянський Р. В.

Суми 2024

1 Призначення й мета створення вебдодатку

1.1 Призначення вебдодатку

Вебдодаток призначений для управління IT-проектами, використовуючи методологію Scrum. Цей додаток надає користувачам інструменти для ефективного планування, виконання та моніторингу проектних завдань, сприяючи збільшенню продуктивності та спрощенню процесів комунікації в командах.

1.2 Мета створення вебдодатку

Головна мета проекту – це надання комплексного рішення для управління проектами, яке інтегрує всі аспекти Scrum-процесу, включаючи спринти, ролі, задачі та відстеження прогресу. Додаток дозволить знизити час на адміністрування та забезпечити більшу прозорість у веденні проектів. Цей інструмент спрямований на забезпечення керівників проектів, команд розробників та інших зацікавлених сторін потрібними функціями для адаптації до змінних умов роботи та виконання проектів на високому рівні ефективності.

1.3 Цільова аудиторія

Цільовою аудиторією даного проекту є IT-компанії та відділи IT у різноманітних організаціях, зокрема керівники проектів, агільні команди розробників та QA-інженери, а також всі інші учасники проектів, які залучені до процесу розробки програмного забезпечення. Додаток також корисний для консультантів та тренерів з управління агільними проектами, які можуть використовувати цей інструмент для навчальних цілей або консультацій.

2 Вимоги до вебдодатку

2.1 Загальні вимоги до вебдодатку

2.1.1 Структурні та функціональні вимоги

Вебдодаток повинен бути розроблений з використанням сучасних вебтехнологій та забезпечувати визначений набір функціональних можливостей, що сприяють ефективному управлінню IT-проектами з використанням методології Scrum. Кінцевим продуктом має бути Вебдодаток, який включає якісну інформаційну структуру, здатну ефективно вирішувати завдання управління проектами.

2.1.2 Вимоги до персоналу

Основною вимогою до користувачів вебдодатку є наявність доступу до ПК з веббраузером та підключенням до Інтернету. Це гарантує, що всі учасники, залучені до процесу управління IT-проектами, включаючи менеджерів проектів, розробників та зацікавлені сторони, можуть взаємодіяти з додатком з будь-якого місця.

2.1.3 Вимоги до зберігання даних

Вся інформація, пов'язана з адмініструванням проекту, така як деталі проекту, дані про спринт, історії користувачів, призначення завдань і звіти про хід виконання, повинна зберігатися в базі даних. База даних повинна бути реалізована з використанням надійної системи управління, такої як PostgreSQL, яка підтримує складні запити і великі обсяги даних, необхідні для управління проектом.

2.1.4 Вимоги до контролю доступу

Розроблений Вебдодаток повинен бути доступним лише для

аутентифікованих користувачів з визначеними ролями та дозволами. Права доступу повинні бути розподілені наступним чином:

- Адміністратори - повний доступ до всієї інформації; можуть створювати проекти, додавати членів команди, налаштовувати налаштування системи та контролювати весь життєвий цикл проекту;
- Менеджери проектів - можливість планувати спринти, призначати завдання, керувати завданнями команди та відстежувати прогрес;
- Члени команди (розробники, тестувальники тощо) - можливість переглядати та оновлювати статуси завдань, реєструвати роботу та отримувати доступ до проектної документації, що відповідає їхнім ролям;
- Зацікавлені сторони/клієнти - доступ до звітів про хід виконання проекту, інформаційних панелей та окремих розділів, які дають уявлення про стан проекту без можливості змінювати дані проекту.

Такий контроль доступу гарантує, що конфіденційна інформація проекту залишається захищеною і що підтримується цілісність процесу управління проектом. Кожен користувач буде взаємодіяти з вебдодатком відповідно до своєї ролі, забезпечуючи впорядкований процес і ефективну комунікацію.

2.2 Структура вебдодатку

2.2.1 Загальна інформація про структуру вебдодатку

Вебдодаток складається з наступних п'яти вебсторінок, кожна з яких призначена для різних ролей в системі:

- Форма авторизації: Спільна сторінка для всіх користувачів, де вони можуть увійти в систему для доступу до вебдодатку. Це точка входу для всіх типів користувачів, яка гарантує, що кожен користувач має доступ лише до інформації та функцій, що відповідають його ролі;
- Сторінка адміністратора: На цій сторінці адміністратор може керувати налаштуваннями проекту та ролями користувачів. Основні функції включають можливість додавати нових користувачів, створювати і призначати

ролі, а також налаштовувати параметри проекту. Адміністратори також можуть переглядати всі дані, що зберігаються в базі даних, що робить її центральним вузлом для управління всією програмою;

- Сторінка менеджера проекту: Менеджери проектів мають доступ до цієї сторінки, де вони можуть планувати спринти, призначати завдання і відстежувати хід виконання проектів. Вони можуть створювати та оновлювати бэклог, керувати спринт-активностями та отримувати доступ до вичерпних звітів, щоб приймати обґрунтовані рішення щодо напрямків проекту;
- Сторінка члена команди: Ця сторінка доступна для членів команди, таких як розробники, тестувальники та інші учасники проекту. Вона дозволяє членам команди переглядати свої завдання, оновлювати прогрес і отримувати доступ до необхідних документів проекту. Функції включають можливість реєструвати робочий час, обговорювати завдання з членами команди і переглядати майбутні дедлайни;
- Сторінка зацікавлених сторін: Подібна за функціоналом до сторінки члена команди, але адаптована для зацікавлених сторін, ця сторінка дозволяє їм переглядати прогрес проекту, фінансові звіти та інші результати проекту на високому рівні. Зацікавлені сторони можуть отримати доступ до інформаційних панелей, які надають швидкий огляд стану проекту та основних етапів.

Слід зазначити, що на початковому етапі не буде сторінки реєстрації. Під час розгортання вебдодатку буде створено єдиний обліковий запис адміністратора, який матиме можливість створювати додаткові облікові записи адміністратора, а також облікові записи для всіх інших типів користувачів. Такий підхід гарантує, що налаштування системи контролюється і що тільки уповноважений персонал може створювати нові облікові записи користувачів, що підвищує безпеку вебдодатку.

2.2.2 Навігація

У заголовку вебдодатку буде реалізовано меню для полегшення навігації, що дозволить користувачам швидко переходити між сторінками, до яких вони

мають доступ. Меню буде відрізнятися для кожного типу користувачів, оскільки кожен з них матиме різні можливості та права доступу до вебдодатку. Така індивідуальна навігація гарантує, що кожен користувач отримає спрощений інтерфейс, який відповідає його конкретним ролям та обов'язкам, що підвищує зручність та ефективність використання.

2.2.3 Управління контентом

Адміністратор буде керувати вмістом вебдодатку. Він відповідатиме за введення та оновлення всієї необхідної інформації, пов'язаної з проектами, включаючи ролі користувачів, налаштування проекту та завдання в базі даних. Таке централізоване управління контентом гарантує, що всі дані є послідовними, актуальними і правильно узгоджені з потребами проекту та дозволами користувачів.

2.2.4 Дизайн та структура додатку

Дизайн вебдодатку для управління IT-проектами через Scrum повинен бути чистим та сучасним, з мінімальним контентом для зручності користувачів. Сторінки мають бути "легкими", інтуїтивно зрозумілими, з візуально виділеними елементами управління, що дозволяють швидко виконувати необхідні дії. Різні види користувачів, такі як адміністратори та розробники, отримують індивідуалізовані меню для ефективної роботи. Авторизація має бути простою і зрозумілою, з ясним визначенням елементів на сторінці.

Розташування елементів при авторизації схема розташована на рисунку А.1

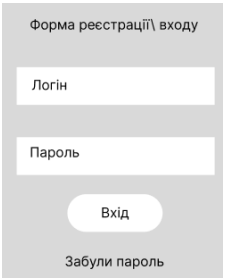

Рисунок А.1 – Схема сторінки авторизації

Розташування елементів при вході в акаунт адміністратора схема розташована на рисунку А.2

Рисунок А.2 – Схема сторінки адміністратора

Розташування елементів при вході в акаунт розробника схема розташована на рисунку А.3

Рисунок А.3 – Схема сторінки розробника

2.2.5 Система навігації (карта вебдодатку)

Карта вебдодатку зображена на рисунку А.4

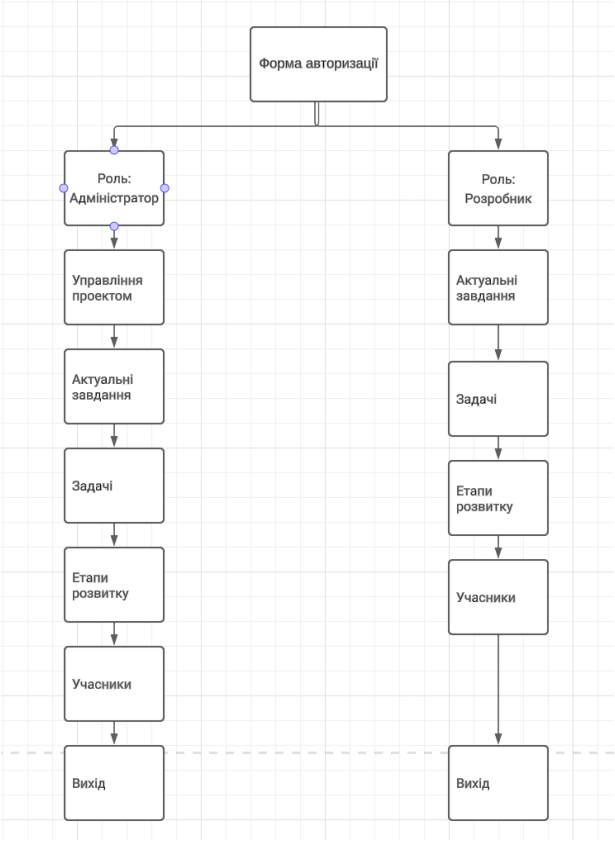

Рисунок А.4 – Карта вебдодатку

2.3 Вимоги до функціональності системи

2.3.1 Потреби користувачів

Потреби користувачів, визначені на основі вимог проекту, представлені в Таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	**Потреба користувача**	**Джерело**
UN-01	Панель управління проектами	Адміністратор, Розробник
UN-02	Система відстеження завдань	Розробники, Скрам-майстер
UN-03	Панель показників продуктивності	Розробники, Скрам-майстер
UN-04	Панель адміністратора	Адміністратор
UN-05	Управління користувачами	Адміністратор
UN-06	Інтеграція з системою контролю версій	Адміністратор, Розробники

У цій таблиці наведено основні функціональні можливості, які очікують різні типи користувачів від програми для управління IT-проектами. Кожна функція узгоджується з певними ролями, щоб забезпечити індивідуальну та ефективну взаємодію користувачів.

2.3.2 Функціональні вимоги

На основі потреб користувачів були визначені наступні функціональні вимоги до вебдодатку для управління IT-проектами з використанням методології Scrum:

- Авторизація;
- Керування інформаційною панеллю проекту;
- Інтерфейс відстеження завдань;
- Управління всією інформацією на панелі адміністратора;
- Інформаційна панель показників ефективності;
- Керування користувачами адміністраторами;
- Видимість для членів команди;
- Можливість адміністратора створювати нових користувачів.

Ці функціональні вимоги покликані гарантувати, що Вебдодаток ефективно підтримує управління IT-проектами, сприяючи ефективним робочим процесам і надійному відстеженню проектів відповідно до практик Scrum.

2.3.3 Системні вимоги

Для задоволення потреб користувачів і виконання функціональних вимог визначено системні вимоги. Вони представлені в таблиці А.2. Ця таблиця детально описує необхідне апаратне та програмне забезпечення, а також інші технічні параметри, критичні для успішної реалізації проекту.

Таблиця А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR-01	Використання Node.js	M	Використання Node.js для розробки серверної частини дозволяє ефективно обробляти запити і забезпечувати швидку відповідь.
SR-02	Використання бази даних PostgreSQL	B	База даних необхідна для зберігання даних про проекти, користувачів, задачі та їх статуси.
SR-03	Використання React для front-end	M	React дозволяє створювати інтерактивні користувацькі інтерфейси, оптимізувати користувацький досвід і швидко реагувати на взаємодії користувача.
SR-04	Використання HTTPS протоколу	B	Забезпечення безпеки передачі даних, валідація введених даних, обмеження доступу для різних видів користувачів.
SR-05	Використання Tailwind CSS	S	Tailwind CSS буде використовуватися для створення сучасного, адаптивного інтерфейсу, який є зручним та візуально привабливим для користувачів.
SR-06	Використання JWT для аутентифікації	M	JWT дозволяє безпечно управляти сесіями користувачів, забезпечувати авторизацію і контроль доступу в додатку.

Ці системні вимоги задумані для забезпечення надійності, ефективності та безпеки вебдодатку, з врахуванням потреб і ролей користувачів у сфері управління IT-проектами. Вони покривають як технічні аспекти розробки, так і забезпечення зручності і доступності для кінцевих користувачів.

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Вебдодаток для управління IT-проектами за методологією Scrum буде реалізований з використанням наступних інструментів та технологій:

- **Node.js;**
- **React**;

- **PostgreSQL;**
- **Tailwind CSS;**
- **JWT.**

2.4.2 Вимоги до лінгвістичного забезпечення

Інтерфейс вебдодатку для управління IT-проектами має бути доступний українською мовою для забезпечення зручності та доступності для україномовних користувачів.

2.4.3 Вимоги до програмного забезпечення

Для забезпечення стабільної роботи вебдодатку, користувачам необхідно мати ПК зі стабільним інтернет з'єднанням та наступними системними вимогами:

- Операційна система: Підтримка Windows, Linux або macOS;
- Оперативна пам’ять: 2 Гб і більше;
- Процесор: Двоядерний з частотою 1,8 ГГц або вище;
- Веббраузер: Сучасний веббраузер такий як Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari. Рекомендується використовувати останню доступну версію для забезпечення оптимальної сумісності та безпеки.

Ці вимоги забезпечують, що користувачі матимуть змогу ефективно використовувати Вебдодаток без технічних збоїв, використовуючи типове обладнання та програмне забезпечення.

3 Склад і зміст робіт зі створення вебдодатку

Детальний опис етапів створення вебдодатку для управління IT-проектами з використанням методології Scrum наведено в таблиці А.3.

Таблиця А.3 – Етапи створення вебдодатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Дослідження предметної області	3 дні
2	Аналіз аналогів та визначення унікальних функцій	1 день
3	Створення технічного завдання	2 дні
4	Розробка архітектури системи	2 дні
5	Розробка загального шаблону вебдодатку	4 дні
6	Верстка сторінок вебдодатку	5 днів
7	Створення бази даних	3 дні
8	Інтеграція API та зовнішніх сервісів	3 дні
9	Розробка панелі адміністратора	4 дні
10	Розробка модулю для керівників проектів	3 дні
11	Розробка модулю для розробників	4 дні
12	Розробка модулю звітності та аналітики	3 дні
13	Тестування всіх модулів і звітність	4 дні
14	Обирання хостингу та розгортання	1 день
15	Реліз вебдодатку та публікація	1 день
	Загальна тривалість робіт:	43 дні

4 Вимоги до складу й змісту робіт із введення вебдодатку в експлуатацію

Введення вебдодатку для управління IT-проектами в експлуатацію починається з тестування, щоб забезпечити його відповідність технічним та функціональним вимогам. Після успішного тестування обирається хостинг для

розміщення додатку. Завершальний етап — розгортання додатку на сервері та його офіційний запуск. Кожен етап супроводжується перевірками забезпечення якості, налаштуваннями безпеки та підготовкою середовища для забезпечення стабільної роботи додатку.

ДОДАТОК Б

Планування робіт

У сучасному світі інформаційних технологій, де велика увага приділяється ефективності управління проектами, розробка вебдодатку для управління IT-проектами за методологією Scrum є актуальним і значущим завданням. Такі додатки представляють собою інноваційні рішення, які спрямовані на покращення управління проектами, оптимізацію робочих процесів та підвищення продуктивності команд.

Цінність розробки такого вебдодатку полягає в тому, що він дозволяє IT-компаніям, керівникам проектів та розробникам використовувати інноваційний підхід до управління проектами. Додаток може надавати доступ до інструментів планування, відстежування прогресу, ведення документації проектів, а також підтримувати комунікацію між учасниками.

Мета розробки вебдодатку полягає в покращенні процесу управління IT-проектами та спрощенні доступу до інструментів проектного управління. Головною метою є забезпечення зручного, ефективного і адаптивного досвіду в управлінні проектами, підвищення мотивації команди та покращення результатів роботи.

Розробити такий Вебдодаток, призначений для автоматизації рутинних задач управління проектами з можливістю інтеграції в існуючу систему управління проектами компанії. Мета - зменшити час на управління проектами, тим самим підвищуючи продуктивність управління. Крім того, прагнемо підвищити задоволеність команди досвідом роботи з управлінськими системами на 20%, що вимірюється за допомогою внутрішніх опитувань. Завдяки наявності ресурсів, включаючи команду кваліфікованих розробників та існуючу інфраструктуру, цей проект є здійсненним і відповідає стратегічним цілям компанії щодо покращення ефективності управління. виконання з цільовою датою завершення 1 травня 2024 року.

Таблиця Б.1 SMART-цілей для вебдодатку управління IT-проектами

Категорія	**Опис**
Specific (Специфічність)	Розробити Вебдодаток для управління IT-проектами з використанням методології Scrum, що дозволить керівникам проектів та розробникам ефективно відстежувати прогрес та взаємодіяти в реальному часі.
Measurable (Вимірюваність)	Зменшити час потрібний на планування та оновлення статусів проектів з 30 хвилин до 10 хвилин. Підвищити загальну продуктивність команди на 25%, як це вимірюється через звітність про продуктивність та задоволеність команди.
Achievable (Досяжність)	Ресурси для розробки додатку доступні, включаючи кваліфікованих розробників та інтеграцію з сучасними технологічними стеками. Сучасні інструменти управління проектами підтвердили свою ефективність у веденні IT-проектів.
Relevant (Актуальність)	Вебдодаток відповідає стратегічним цілям IT-компаній, спрямованим на оптимізацію процесів управління проектами, зменшення помилок і покращення комунікації між учасниками проекту.
Time-framed (Обмеженість у часі)	Планується розробка та введення в експлуатацію вебдодатку протягом наступних 6 місяців з чіткими етапами розробки, тестування, впровадження та проміжними термінами для кожного етапу.

Планування змісту робіт: WBS є ключовим інструментом у плануванні і управлінні вебдодатком для управління IT-проектами за методологією Scrum. Ця діаграма, яка показана на Рисунок Б.1, відіграє важливу роль у успіху проекту. Вона дозволяє менеджеру проекту планувати, виконувати, моніторити та контролювати всі процеси. На верхньому рівні WBS розміщується назва проекту,

яка служить основою для подальшої розбивки на більш деталізовані завдання та елементарні роботи з конкретними результатами.

Планування структури виконавців: Діаграма OBS використовується для візуального представлення структури виконавців проекту, як показано на Рисунок Б.2 Цей інструмент дозволяє чітко визначити ролі та відповідальності кожного учасника проекту, забезпечуючи ефективне розподілення завдань та забезпечення відповідальності на всіх рівнях проектної команди. OBS є невід'ємною частиною управління проектами, яка сприяє кращій координації та зв'язності між різними групами та індивідуумами, залученими до проекту.

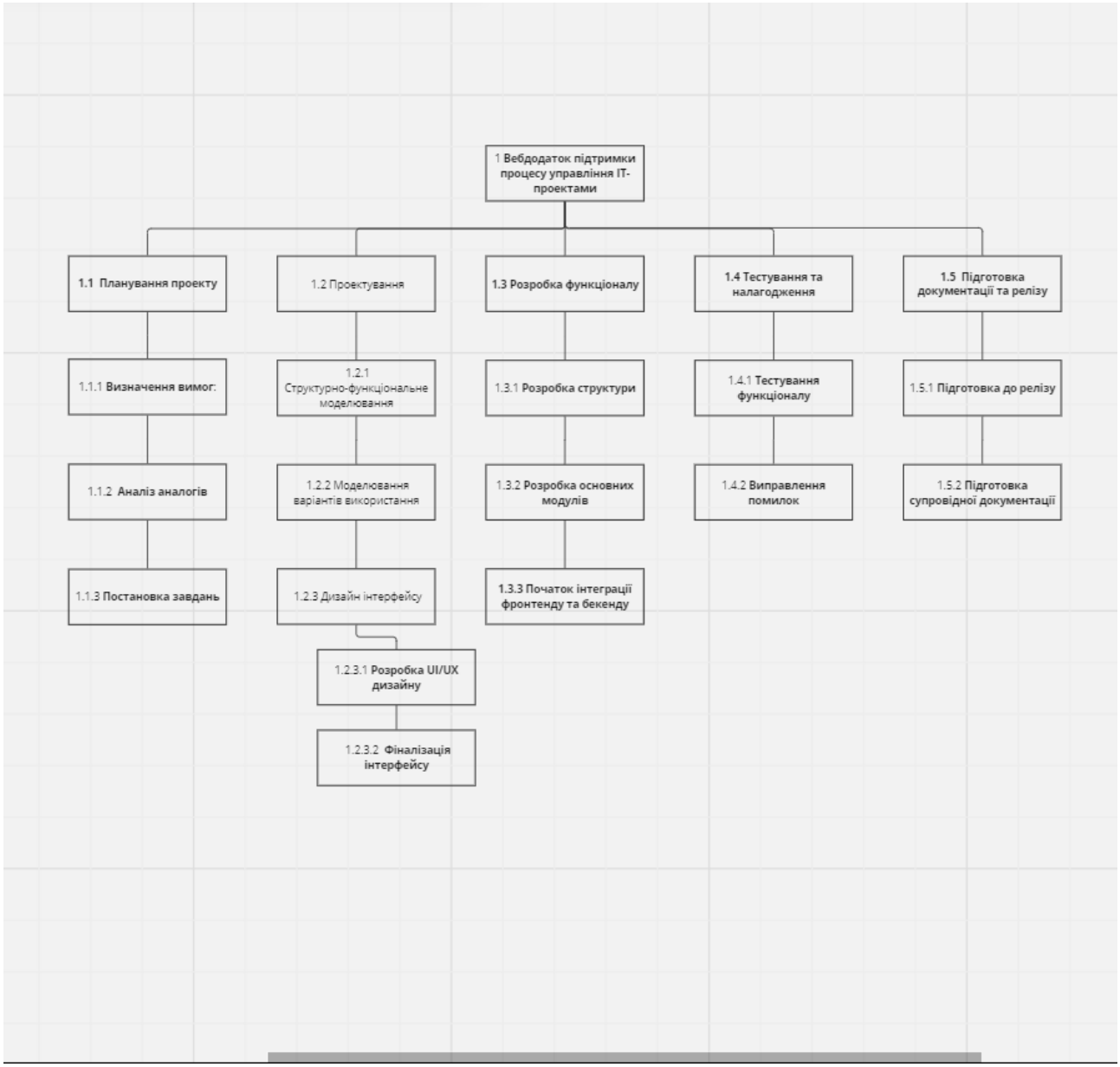

Рисунок Б.1 – WBS-структура робіт проекту

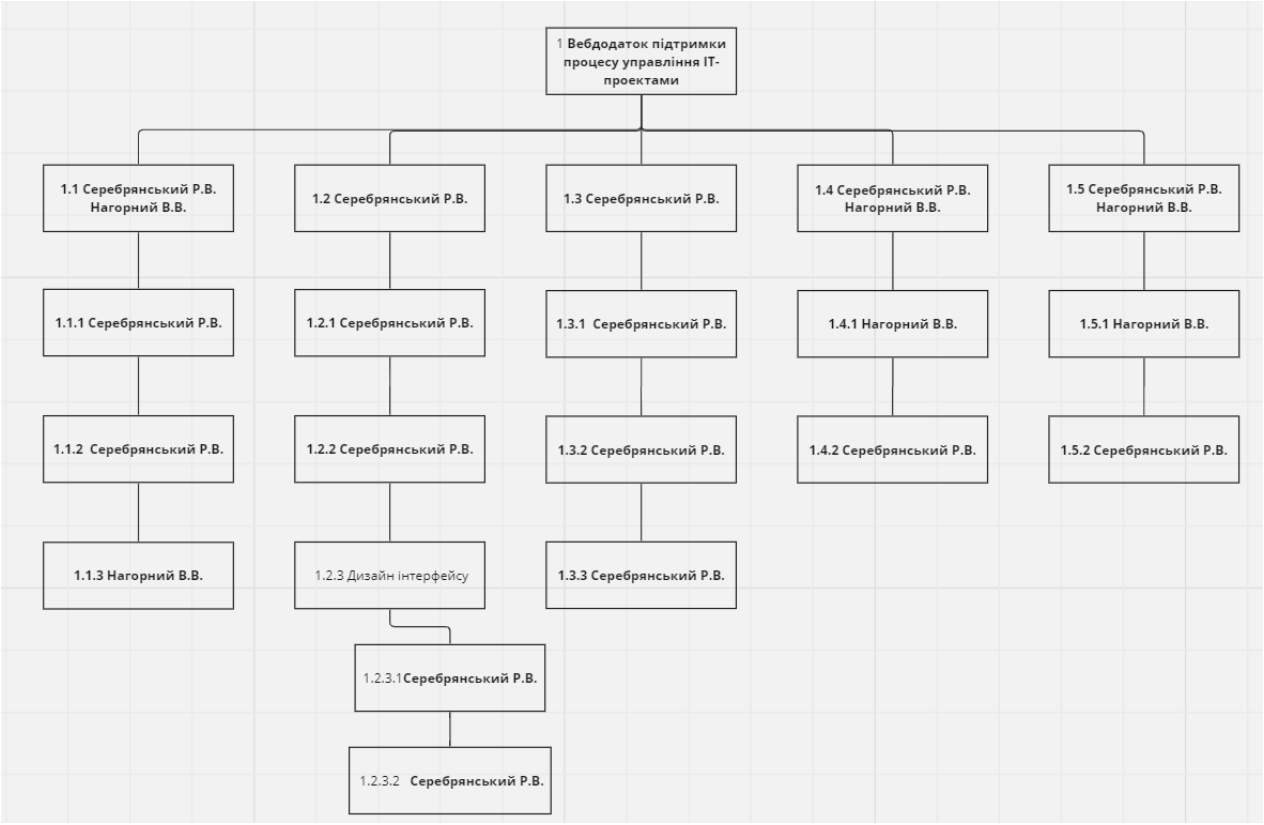

Рисунок Б.2 – OBS-структура робіт проекту

Список виконавців проекту знаходиться в Таблиця Б.2

Таблиця Б.2 – Виконавці проєкту

Роль	ПІБ	Проєктна роль
Розробник	Серебрянський Р.В.	Займається розробкою проекту.
Проєктувальник	Серебрянський Р.В.	Проектує архітектуру бази даних та розробляє структуру мобільного додатку.
Тестувальник	Нагорний В.В.	Відповідає за функціональне тестування.
Керівник проєкту	Нагорний В.В.	Ставить завдання на розробку проекту.
Роль	ПІБ	Проєктна роль
Менеджер проєкту	Серебрянський Р.В.	Керує розподілом ресурсів, завданнями та дедлайнами

Під час планування робіт по розробці вебдодатку для управління IT-

проектами важливим кроком є створення діаграми Ганта, яка відображена на Рисунок Б. Цей інструмент дозволяє візуалізувати роботи проекту у часовій послідовності, показуючи детальний календарний план завдань, їх взаємозалежності та загальний графік виконання. Кожна смуга на діаграмі відповідає окремому завданню, а її довжина вказує на тривалість цього завдання. Це допомагає всій команді розуміти часові рамки проекту та координувати свої зусилля ефективно.

Аналіз ризиків є критично важливою частиною планування розробки вебдодатку. Важливо систематично ідентифікувати потенційні ризики, оцінити їх можливий вплив на проект і розробити стратегії для їх мінімізації або управління. Це включає аналіз ймовірних проблем, які можуть порушити запланований хід робіт або збільшити витрати часу та ресурсів. Ефективний аналіз ризиків дозволяє команді передбачити можливі перешкоди та підготуватися до них, забезпечуючи більшу впевненість у досягненні цілей проекту в установлені строки.

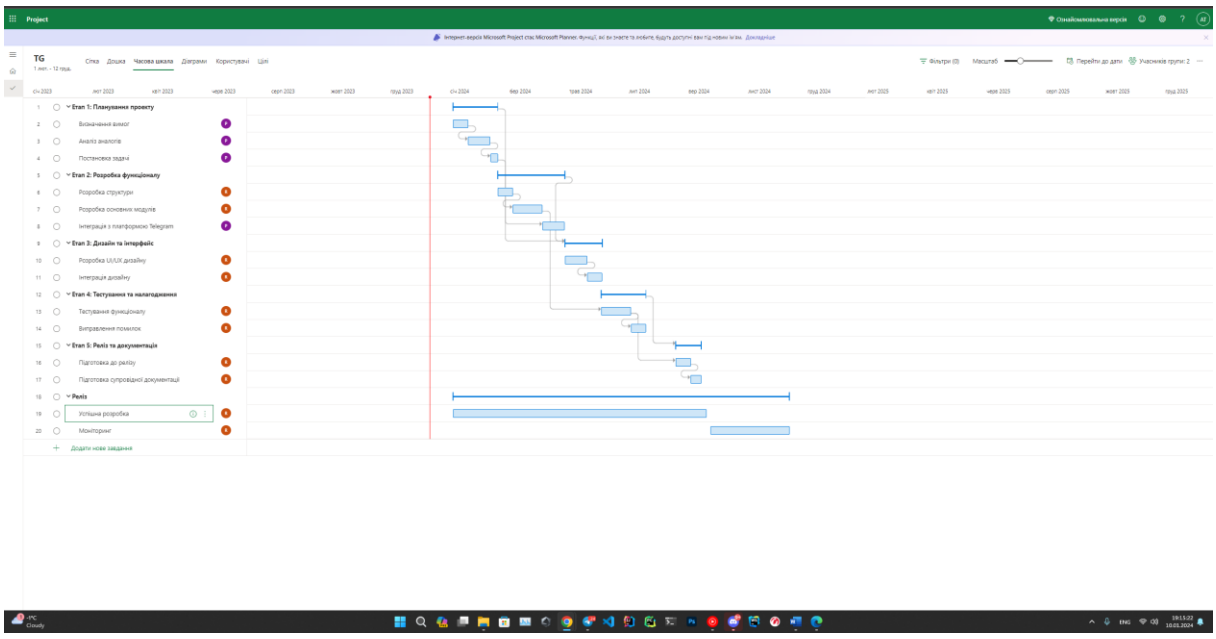

Рисунок Б.3 – Календарний графік проєкту

Управління ризиками в проекті передбачає критичний аналіз та оцінку потенційних ризиків. На початковому етапі важливо провести якісну оцінку ризиків, щоб виявити ті з них, які потребують негайної уваги. Реакція на кожен ризик залежить від його значущості та потенційного впливу на проект. Згодом

важливо провести кількісну оцінку ризиків, яка дозволяє більш точно визначити ймовірність і вплив кожного ризику. Ці оцінки можуть проводитися одночасно або окремо, залежно від стадії готовності проекту та його потреб.

У переліку ризиків проекту, представленому в Таблиці Б.3, визначено потенційні загрози та виклики для розробки додатку. Результати оцінки цих ризиків наведено в таблиці Б.4. Для класифікації ризиків використовується спеціальна шкала, наведена в таблиці Б.5. Ця шкала допомагає класифікувати ризики за рівнем їх впливу на проект та ймовірністю настання, що є ключовим для ефективного управління ризиками.

Таблиця Б.3 – Ризики проєкту

№	Ризик	Опис ризику
1	Технічні ризики	Ризики, пов'язані з можливими технічними збоями, включно з програмними помилками, проблемами сумісності та відмовою обладнання.
2	Відключення електроенергії	Ризики, пов'язані з можливими відключеннями електроенергії та інтернету, що можуть призвести до зупинки роботи додатку.
3	Невідповідність технічних вимог	Ризик, що технічні рішення не зможуть відповідати всім потребам управління проектами.
4	Порушення конфіденційності даних	Ризик несанкціонованого доступу до даних проектів та користувачів або їх витоку.
5	Залежність від сторонніх API	Ризик, пов'язаний зі змінами або відмовою зовнішніх сервісів, які використовуються для інтеграції в додаток.
6	Проблеми з масштабуванням	Труднощі зі збільшенням кількості користувачів або обробки даних без втрати продуктивності.
7	Відповідність законодавству	Ризик невідповідності регулятивним вимогам, що може призвести до юридичних наслідків.
8	Зміни в технологіях	Ризик швидких змін у технологічному середовищі, що може вимагати частих оновлень додатку.
9	Недостатність ресурсів для розробки	Ризик виникнення затримок у проекті через нестачу фінансових або людських ресурсів.
10	Відсутність доступу до інтернету	Ризики, пов'язані з можливими труднощами в доступі до онлайн ресурсів, важливих для розробки та експлуатації додатку.

Таблиця Б.4 – Результати визначення ймовірності, впливу та рангу ризиків проєкту

№	**Назва (опис) ризику**	**Ймовірність**	**Вплив**	**Ранг**
1	Технічні проблеми	0.4	0.4	0.49
2	Відключення світла	0.4	0.5	0.20
3	Невідповідність технічних вимог	0.4	0.5	0.20
4	Порушення конфіденційності даних	0.3	0.8	0.24
5	Залежність від сторонніх API	0.4	0.6	0.24
6	Відсутність доступу до інтернету	0.5	0.5	0.26
7	Проблеми з масштабуванням	0.5	0.7	0.35
8	Зміни в технологіях	0.6	0.6	0.36
9	Відповідність законодавству	0.5	0.5	0.25
10	Зміни в технологіях	0.6	0.7	0.42

Таблиця Б.5 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	Ймовірність виникнення	Вплив ризику	Тип ризику
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Щоб мінімізувати негативний вплив ризиків на проект розробки необхідно ретельно планувати стратегії реагування на них. Це передбачає аналіз потенційних наслідків кожного ризику для проекту та розробку відповідних планів дій. Оцінка ризиків здійснюється на основі критеріїв, зазначених у Таблиці Б.4. В результаті цього планування була створена матриця ймовірності та впливу ризиків, як показано на Рисунку Б.4.

У цій матриці ризики позначені кольором: зеленим кольором позначені прийнятні ризики, якими можна легко управляти або які мають низький вплив на проект, жовтим - ризики, які потребують уваги і можуть бути виправдані в контексті проекту, а червоним - неприйнятні ризики з високим рівнем впливу, що вимагають негайного втручання. Належне управління ризиками за допомогою цієї матриці допомагає визначити пріоритети та розробити ефективні стратегії для пом'якшення потенційного негативного впливу на проект.

3.Оцінка власного проекту

Ймовірність ризику (Й)	Вплив загрози (ризику)				
	Дуже малий	Малий	Середній	Великий	Дуже велики
	0,05	0,1	0,2	0,4	0,8
0,9	0,045	0,09	0,18	0,36	0,72
0,7	0,035	0,07 R10	0,14	0,28	0,56
0,5	0,025 R3, R8	0,05 R7	0,10 R1, R6	0,20	0,40
0,3	0,015 R9	0,03 R4	0,06 R2, R5	0,12	0,24
0,1	0,005	0,01	0,02	0,04	0,08

Рисунок Б.4. – Матриця ймовірності та впливу

Класифікація ризиків проєкту за рівнями, відповідно до значення отриманого індексу ризиків, представлена в таблиці Б.5. У таблиці Б.6 детально розглянуто ризик №2 та стратегії реагування на нього в контексті розробки вебдодатку для управління IT-проектами.

Таблиця Б.6 – Шкала оцінювання ризику за рівне

№	Назва	Межі	Ризики, які входять (номера)
1	Прийнятні	$0,005 \leq R \leq 0,05$	2,3,4,5,7,8,9
2	Виправдані	$0,05 < R \leq 0,14$	1, 6, 10
3	Недопустимі	$0,14 < R \leq 0,72$	

Таблиця Б.7 – Ризики та стратегії реагування на них

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
1	Новий	Технічні проблеми	Середня	Середній	Середній	Ухилення	Регулярні технічні аудити і рецензії коду	Розробка плану виправлення помилок
2	Новий	Відключення світла	Середня	Середній	Середній	Прийняття	Передбачення альтернативних варіантів доступу	Адаптація та розробка в університеті
3	Новий	Невідповідність технічних вимог	Низька	Низький	Низький	Зменшення	Розробка на резервних платформах	Імплементація адаптивних технологічних рішень
4	Новий	Порушення конфіденційності даних	Низька	Низький	Низький	Передача	Розробка та імплементація політик безпеки даних	Не використовувати дані користувачів
5	Новий	Залежність від сторонніх API	Низька	Низький	Низький	Ухилення	Створення модульної архітектури для гнучкої інтеграції	Пошук альтернативних рішень
6	Новий	Відсутність доступу до інтернету	Середня	Середній	Середній	Зменшення	Ефективний розподіл часу при доступі в інтернет	Пошук альтернативних джерел інформації
7	Новий	Проблеми з масштабуванням	Низька	Низький	Низький	Передача	Співпраця з хмарними провайдерами для масштабування	Збільшення технічних ресурсів

Продовження таблиці Б.7

ID ризику	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	Тип стратегії реагування	План А	План Б
8	Новий	Зміни в технологіях	Низька	Низький	Низький	Передача	Постійне оновлення технологічних навичок.	Впровадження адаптивної архітектури системи.
9	Новий	Відповідність законодавству	Низька	Низький	Низький	Ухилення	Моніторинг нових законопроектів	Аналіз діючих законопроектів
10	Новий	Зміни в технологіях	Середня	Середній	Середній	Зменшення	Постійне оновлення технічних знань команди	Гнучке планування для інтеграції нових технологій

ДОДАТОК В. Лістинг програмного коду основних модулів вебдодатку

Main.js

```
import Notification from "../models/notification.model.js";
import Post from "../models/post.model.js";
import User from "../models/user.model.js";
import { v2 as cloudinary } from "cloudinary";
import bcrypt from "bcryptjs";

export const createPost = async (req, res) => {
  try {
    const { text } = req.body;
    let { img } = req.body;
    const userId = req.user._id.toString();

    const user = await User.findById(userId);
    if (!user) return res.status(404).json({ message: "User not found" });

    if (!text && !img) {
      return res.status(400).json({ error: "Post must have text or image" });
    }

    if (img) {
      const uploadedResponse = await cloudinary.uploader.upload(img);
      img = uploadedResponse.secure_url;
    }

    const newPost = new Post({
      user: userId,
      text,
      img,
    });

    await newPost.save();
    res.status(201).json(newPost);
```

```
    } catch (error) {
        res.status(500).json({ error: "Internal server error" });
        console.log("Error in createPost controller: ", error);
    }
};

export const deletePost = async (req, res) => {
    try {
        const post = await Post.findById(req.params.id);
        if (!post) {
            return res.status(404).json({ error: "Post not found" });
        }

        if (post.user.toString() !== req.user._id.toString()) {
            return res.status(401).json({ error: "You are not authorized to delete this post" });
        }

        if (post.img) {
            const imgId = post.img.split("/").pop().split(".")[0];
            await cloudinary.uploader.destroy(imgId);
        }

        await Post.findByIdAndDelete(req.params.id);

        res.status(200).json({ message: "Post deleted successfully" });
    } catch (error) {
        console.log("Error in deletePost controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const commentOnPost = async (req, res) => {
    try {
        const { text } = req.body;
        const postId = req.params.id;
        const userId = req.user._id;
```

```
        if (!text) {
            return res.status(400).json({ error: "Text field is required" });
        }
        const post = await Post.findById(postId);

        if (!post) {
            return res.status(404).json({ error: "Post not found" });
        }

        const comment = { user: userId, text };

        post.comments.push(comment);
        await post.save();

        res.status(200).json(post);
    } catch (error) {
        console.log("Error in commentOnPost controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const likeUnlikePost = async (req, res) => {
    try {
        const userId = req.user._id;
        const { id: postId } = req.params;

        const post = await Post.findById(postId);

        if (!post) {
            return res.status(404).json({ error: "Post not found" });
        }

        const userLikedPost = post.likes.includes(userId);

        if (userLikedPost) {
```

```
            // Unlike post
            await Post.updateOne({ _id: postId }, { $pull: { likes: userId } });
            await User.updateOne({ _id: userId }, { $pull: { likedPosts: postId } });

            const updatedLikes = post.likes.filter((id) => id.toString() !== userId.toString());
            res.status(200).json(updatedLikes);
        } else {
            // Like post
            post.likes.push(userId);
            await User.updateOne({ _id: userId }, { $push: { likedPosts: postId } });
            await post.save();

            const notification = new Notification({
                from: userId,
                to: post.user,
                type: "like",
            });
            await notification.save();

            const updatedLikes = post.likes;
            res.status(200).json(updatedLikes);
        }
    } catch (error) {
        console.log("Error in likeUnlikePost controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const getAllPosts = async (req, res) => {
    try {
        const posts = await Post.find()
            .sort({ createdAt: -1 })
            .populate({
                path: "user",
                select: "-password",
            })
```

```
            .populate({
                path: "comments.user",
                select: "-password",
            });

        if (posts.length === 0) {
            return res.status(200).json([]);
        }

        res.status(200).json(posts);
    } catch (error) {
        console.log("Error in getAllPosts controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const getLikedPosts = async (req, res) => {
    const userId = req.params.id;

    try {
        const user = await User.findById(userId);
        if (!user) return res.status(404).json({ error: "User not found" });

        const likedPosts = await Post.find({ _id: { $in: user.likedPosts } })
            .populate({
                path: "user",
                select: "-password",
            })
            .populate({
                path: "comments.user",
                select: "-password",
            });

        res.status(200).json(likedPosts);
    } catch (error) {
        console.log("Error in getLikedPosts controller: ", error);
```

```
        res.status(500).json({ error: "Internal server error" });
    }
};

export const getFollowingPosts = async (req, res) => {
    try {
        const userId = req.user._id;
        const user = await User.findById(userId);
        if (!user) return res.status(404).json({ error: "User not found" });

        const following = user.following;

        const feedPosts = await Post.find({ user: { $in: following } })
            .sort({ createdAt: -1 })
            .populate({
                path: "user",
                select: "-password",
            })
            .populate({
                path: "comments.user",
                select: "-password",
            });

        res.status(200).json(feedPosts);
    } catch (error) {
        console.log("Error in getFollowingPosts controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const getUserPosts = async (req, res) => {
    try {
        const { username } = req.params;

        const user = await User.findOne({ username });
        if (!user) return res.status(404).json({ error: "User not found" });
```

```
        const posts = await Post.find({ user: user._id })
            .sort({ createdAt: -1 })
            .populate({
                path: "user",
                select: "-password",
            })
            .populate({
                path: "comments.user",
                select: "-password",
            });

        res.status(200).json(posts);
    } catch (error) {
        console.log("Error in getUserPosts controller: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

export const getUserProfile = async (req, res) => {
    const { username } = req.params;

    try {
        const user = await User.findOne({ username }).select("-password");
        if (!user) return res.status(404).json({ message: "User not found" });

        res.status(200).json(user);
    } catch (error) {
        console.log("Error in getUserProfile: ", error.message);
        res.status(500).json({ error: error.message });
    }
};

export const followUnfollowUser = async (req, res) => {
    try {
        const { id } = req.params;
```

```
        const userToModify = await User.findById(id);
        const currentUser = await User.findById(req.user._id);

        if (id === req.user._id.toString()) {
            return res.status(400).json({ error: "You can't follow/unfollow yourself" });
        }

        if (!userToModify || !currentUser) return res.status(400).json({ error: "User not found" });

        const isFollowing = currentUser.following.includes(id);

        if (isFollowing) {
            // Unfollow the user
            await User.findByIdAndUpdate(id, { $pull: { followers: req.user._id } });
            await User.findByIdAndUpdate(req.user._id, { $pull: { following: id } });

            res.status(200).json({ message: "User unfollowed successfully" });
        } else {
            // Follow the user
            await User.findByIdAndUpdate(id, { $push: { followers: req.user._id } });
            await User.findByIdAndUpdate(req.user._id, { $push: { following: id } });
            // Send notification to the user
            const newNotification = new Notification({
                type: "follow",
                from: req.user._id,
                to: userToModify._id,
            });

            await newNotification.save();

            res.status(

200).json({ message: "User followed successfully" });
        }
    } catch (error) {
        console.log("Error in followUnfollowUser: ", error.message);
```

```
        res.status(500).json({ error: error.message });
    }
};

export const getSuggestedUsers = async (req, res) => {
    try {
        const userId = req.user._id;

        const usersFollowedByMe = await User.findById(userId).select("following");

        const users = await User.aggregate([
            {
                $match: {
                    _id: { $ne: userId },
                },
            },
            { $sample: { size: 10 } },
        ]);

        const filteredUsers = users.filter((user) => !usersFollowedByMe.following.includes(user._id));
        const suggestedUsers = filteredUsers.slice(0, 4);

        suggestedUsers.forEach((user) => (user.password = null));

        res.status(200).json(suggestedUsers);
    } catch (error) {
        console.log("Error in getSuggestedUsers: ", error.message);
        res.status(500).json({ error: error.message });
    }
};

export const updateUser = async (req, res) => {
    const { fullName, email, username, currentPassword, newPassword, bio, link } = req.body;
    let { profileImg, coverImg } = req.body;

    const userId = req.user._id;
```

```
    try {
        let user = await User.findById(userId);
        if (!user) return res.status(404).json({ message: "User not found" });

        if ((!newPassword && currentPassword) || (!currentPassword && newPassword)) {
            return res.status(400).json({ error: "Please provide both current password and new password"
});
        }

        if (currentPassword && newPassword) {
            const isMatch = await bcrypt.compare(currentPassword, user.password);
            if (!isMatch) return res.status(400).json({ error: "Current password is incorrect" });
            if (newPassword.length < 6) {
                return res.status(400).json({ error: "Password must be at least 6 characters long" });
            }

            const salt = await bcrypt.genSalt(10);
            user.password = await bcrypt.hash(newPassword, salt);
        }

        if (profileImg) {
            if (user.profileImg) {
                //
https://res.cloudinary.com/dyfqon1v6/image/upload/v1712997552/zmxorcxexpdbh8r0bkjb.png
                await cloudinary.uploader.destroy(user.profileImg.split("/").pop().split(".")[0]);
            }

            const uploadedResponse = await cloudinary.uploader.upload(profileImg);
            profileImg = uploadedResponse.secure_url;
        }

        if (coverImg) {
            if (user.coverImg) {
                await cloudinary.uploader.destroy(user.coverImg.split("/").pop().split(".")[0]);
            }
```

```
            const uploadedResponse = await cloudinary.uploader.upload(coverImg);
            coverImg = uploadedResponse.secure_url;
        }

        user.fullName = fullName || user.fullName;
        user.email = email || user.email;
        user.username = username || user.username;
        user.bio = bio || user.bio;
        user.link = link || user.link;
        user.profileImg = profileImg || user.profileImg;
        user.coverImg = coverImg || user.coverImg;

        user = await user.save();

        // password should be null in response
        user.password = null;

        return res.status(200).json(user);
    } catch (error) {
        console.log("Error in updateUser: ", error.message);
        res.status(500).json({ error: error.message });
    }
};
```

Test.js

```
import { createPost, deletePost, commentOnPost, likeUnlikePost, getAllPosts, getLikedPosts, getFollowingPosts, getUserPosts, getUserProfile, followUnfollowUser, updateUser, getSuggestedUsers } from '../controllers/postController.js';
import Notification from '../models/notification.model.js';
import Post from '../models/post.model.js';
import User from '../models/user.model.js';
import { v2 as cloudinary } from 'cloudinary';
import bcrypt from 'bcryptjs';
```

```
// Mock the dependencies
jest.mock('../models/notification.model.js');
jest.mock('../models/post.model.js');
jest.mock('../models/user.model.js');
jest.mock('cloudinary');
jest.mock('bcryptjs');

describe('Post Controller Tests', () => {

  afterEach(() => {
    jest.clearAllMocks();
  });

  describe('createPost', () => {
    it('should create a new post with text', async () => {
      const req = {
        body: { text: 'test text' },
        user: { _id: 'userId' }
      };
      const res = {
        status: jest.fn().mockReturnThis(),
        json: jest.fn()
      };

      User.findById.mockResolvedValue({ _id: 'userId' });
      Post.prototype.save = jest.fn().mockResolvedValue({ _id: 'postId', user: 'userId', text: 'test text' });

      await createPost(req, res);

      expect(User.findById).toHaveBeenCalledWith('userId');
      expect(Post.prototype.save).toHaveBeenCalled();
      expect(res.status).toHaveBeenCalledWith(201);
      expect(res.json).toHaveBeenCalledWith({ _id: 'postId', user: 'userId', text: 'test text' });
    });
```

```
    it('should handle errors during post creation', async () => {
        const req = {
            body: { text: 'test text' },
            user: { _id: 'userId' }
        };
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        User.findById.mockRejectedValue(new Error('Database error'));

        await createPost(req, res);

        expect(User.findById).toHaveBeenCalledWith('userId');
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.json).toHaveBeenCalledWith({ error: 'Internal server error' });
    });
});

describe('deletePost', () => {
    it('should delete a post successfully', async () => {
        const req = {
            params: { id: 'postId' },
            user: { _id: 'userId' }
        };
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        Post.findById.mockResolvedValue({ _id: 'postId', user: 'userId' });
        Post.findByIdAndDelete = jest.fn().mockResolvedValue(true);
        cloudinary.uploader.destroy = jest.fn().mockResolvedValue(true);
```

```
        await deletePost(req, res);

        expect(Post.findById).toHaveBeenCalledWith('postId');
        expect(Post.findByIdAndDelete).toHaveBeenCalledWith('postId');
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.json).toHaveBeenCalledWith({ message: 'Post deleted successfully' });
    });

    it('should handle errors during post deletion', async () => {
        const req = {
            params: { id: 'postId' },
            user: { _id: 'userId' }
        };
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        Post.findById.mockRejectedValue(new Error('Database error'));

        await deletePost(req, res);

        expect(Post.findById).toHaveBeenCalledWith('postId');
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.json).toHaveBeenCalledWith({ error: 'Internal server error' });
    });
});

describe('commentOnPost', () => {
    it('should add a comment to a post', async () => {
        const req = {
            body: { text: 'test comment' },
            params: { id: 'postId' },
            user: { _id: 'userId' }
        };
        const res = {
```

```
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        Post.findById.mockResolvedValue({ _id: 'postId', comments: [] });
        Post.prototype.save = jest.fn().mockResolvedValue({ _id: 'postId', comments: [{ user: 'userId', text: 'test comment' }] });

        await commentOnPost(req, res);

        expect(Post.findById).toHaveBeenCalledWith('postId');
        expect(Post.prototype.save).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.json).toHaveBeenCalledWith({ _id: 'postId', comments: [{ user: 'userId', text: 'test comment' }] });
    });

    it('should handle errors during adding a comment', async () => {
        const req = {
            body: { text: 'test comment' },
            params: { id: 'postId' },
            user: { _id: 'userId' }
        };
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        Post.findById.mockRejectedValue(new Error('Database error'));

        await commentOnPost(req, res);

        expect(Post.findById).toHaveBeenCalledWith('postId');
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.json).toHaveBeenCalledWith({ error: 'Internal server error' });
    });
```

```
  });

  describe('likeUnlikePost', () => {
    it('should like a post', async () => {
      const req = {
        params: { id: 'postId' },
        user: { _id: 'userId' }
      };
      const res = {
        status: jest.fn().mockReturnThis(),
        json: jest.fn()
      };

      Post.findById.mockResolvedValue({ _id: 'postId', likes: [] });
      Post.prototype.save = jest.fn().mockResolvedValue({ _id: 'postId', likes: ['userId'] });
      User.updateOne = jest.fn().mockResolvedValue(true);
      Notification.prototype.save = jest.fn().mockResolvedValue(true);

      await likeUnlikePost(req, res);

      expect(Post.findById).toHaveBeenCalledWith('postId');
      expect(Post.prototype.save).toHaveBeenCalled();
      expect(User.updateOne).toHaveBeenCalledWith({ _id: 'userId' }, { $push: { likedPosts:
'postId' } });
      expect(res.status).toHaveBeenCalledWith(200);
      expect(res.json).toHaveBeenCalledWith(['userId']);
    });

    it('should handle errors during liking/unliking a post', async () => {
      const req = {
        params: { id: 'postId' },
        user: { _id: 'userId' }
      };
      const res = {
        status: jest.fn().mockReturnThis(),
        json: jest.fn()
```

```
        };

        Post.findById.mockRejectedValue(new Error('Database error'));

        await likeUnlikePost(req, res);

        expect(Post.findById).toHaveBeenCalledWith('postId');
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.json).toHaveBeenCalledWith({ error: 'Internal server error' });
    });
});

describe('getAllPosts', () => {
    it('should retrieve all posts', async () => {
        const req = {};
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };

        Post.find.mockResolvedValue([{ _id: 'postId', text: 'test post' }]);

        await getAllPosts(req, res);

        expect(Post.find).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(200);
        expect(res.json).toHaveBeenCalledWith([{ _id: 'postId', text: 'test post' }]);
    });

    it('should handle errors during retrieving all posts', async () => {
        const req = {};
        const res = {
            status: jest.fn().mockReturnThis(),
            json: jest.fn()
        };
```

```
        Post.find.mockRejectedValue(new Error('Database error'));

        await getAllPosts(req, res);

        expect(Post.find).toHaveBeenCalled();
        expect(res.status).toHaveBeenCalledWith(500);
        expect(res.json).toHaveBeenCalledWith({ error: 'Internal server error' });
      });
    });

    // Additional tests for the other functions can be written in a similar manner

});
```