Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Застосування технологій штучного інтелекту для оптимізації опису завдань проектного менеджменту

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

		Сава Р. І.
	(підпис)	(прізвище та ініціали)
Керівник		Дячук С. Ф.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		Оробчук О. Р.
	(підпис)	(прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)

«__» _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Сава Роман Іванович
(прізвище, ім'я, по батькові)

1. Тема роботи Застосування технологій штучного інтелекту для оптимізації опису завдань проектного менеджменту

Керівник роботи кандидат технічних наук Дячук Степан Федрович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « __ » _____ 2025 року № _____

2. Термін подання студентом завершеної роботи ___грудня 2025

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ

1 Аналіз вимог до програмної системи

2 Проєктування та розробка програмної системи

3 Тестування, впровадження та підтримка

4 Охорона праці та безпека в надзвичайних ситуаціях

Висновки

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

1. Актуальність і мета дослідження.

2. Мета та задачі роботи.

3. Специфікація вимог та Діаграма варіантів використання

4. Архітектура програмного забезпечення

5. Моделі бази даних

6. Алгоритм роботи системи

7. Інтерфейс користувача

8. Висновки

6. Консультанти розділів роботи

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

7. Дата видачі завдання 23.10.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літературних джерел та аналіз існуючих систем управління проєктами. Постановка завдання.	03.11.2025 р. – 09.11.2025 р.	Викон.
2.	Проєктування архітектури системи, бази даних та інтерфейсів користувача.	10.11.2025 р. – 16.11.2025 р.	Викон.
3.	Програмна реалізація веб-застосунку, інтеграція OpenAI API та налаштування Kanban-дошки.	17.11.2025 р. – 05.12.2025 р.	Викон.
4.	Розробка розділу «Охорона праці та безпека в надзвичайних ситуаціях».	01.12.2025 р. – 07.12.2025 р.	Викон.
5.	Оформлення пояснювальної записки, підготовка презентації та графічного матеріалу.	08.12.2025 р. – 13.12.2025 р.	Викон.
6.	Попередній захист кваліфікаційної роботи магістра	15.12.2025р.	Викон.
7.	Захист кваліфікаційної роботи магістра	23.12.2025	Викон.

Студент _____ Сава Р. І..
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дячук С. Ф.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сава Роман Іванович. Застосування технологій штучного інтелекту для оптимізації опису завдань проектного менеджменту. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм-61 спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 80, таблиць 5, рисунків 11, додатків 2, бібліографічних посилань 36.

Метою роботи є підвищення ефективності процесів проєктного менеджменту шляхом розроблення веб-орієнтованої інформаційної системи, що інтегрує технології штучного інтелекту для генерації та оптимізації описів завдань.

Об’єкт дослідження – процес управління завданнями та вимогами в проєктах розробки програмного забезпечення.

Предмет дослідження – методи та засоби застосування великих мовних моделей (LLM) для автоматизованого формування технічної документації. Методи дослідження включають: системний аналіз, об’єктно-орієнтоване проєктування, методи інженерії знань (Prompt Engineering), емпіричне тестування програмного забезпечення.

В даній роботі розроблено веб-застосунок для управління проєктами за методологією Kanban, побудований на стеку технологій Next.js, Hono та PostgreSQL. Реалізовано модуль інтеграції з OpenAI API, що забезпечує автоматичну генерацію розширених описів завдань, критеріїв приймання та оцінки складності на основі короткого контексту. Впроваджено архітектурний підхід Feature-Sliced Design та принцип BYOK (Bring Your Own Key) для гнучкості налаштування. Створена система дозволяє скоротити час бізнес-аналізу, стандартизувати формат вимог та зменшити комунікаційне навантаження на команду розробки.

Ключові слова: управління проєктами, штучний інтелект, Kanban, генеративні моделі, User Story, Next.js, OpenAI API.

ABSTRACT

Sava Roman Ivanovych. Application of artificial intelligence technologies for optimization of project management task descriptions. Qualification work of the educational level “Master”. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, Group SPm-61, Specialty 121 “Software Engineering”. Ternopil, 2025. Pages 80, tables 5, figures 11, annexes 2, bibliographic references 36.

The purpose of the work is to increase the efficiency of project management processes by developing a web-oriented information system that integrates artificial intelligence technologies for generating and optimizing task descriptions.

The object of the study is the process of managing tasks and requirements in software development projects.

The subject of the study is methods and tools for applying large language models (LLM) for the automated formation of technical documentation. The research methods include: system analysis, object-oriented design, knowledge engineering methods (Prompt Engineering), and empirical software testing.

In this work, a web application for project management based on the Kanban methodology was developed, built on the technology stack of Next.js, Hono, and PostgreSQL. An integration module with the OpenAI API has been implemented, providing automatic generation of extended task descriptions, acceptance criteria, and complexity estimation based on a short context. The Feature-Sliced Design architectural approach and the BYOK (Bring Your Own Key) principle were implemented for configuration flexibility. The created system allows reducing business analysis time, standardizing the requirements format, and decreasing the communication load on the development team.

Keywords: project management, artificial intelligence, Kanban, generative models, User Story, Next.js, OpenAI API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AC (Acceptance Criteria) – критерії приймання; набір умов, яким має відповідати програмний продукт, щоб бути прийнятим користувачем або замовником.

AI (Artificial Intelligence) – штучний інтелект; здатність комп'ютерних систем виконувати завдання, що зазвичай потребують людського інтелекту.

API (Application Programming Interface) – інтерфейс прикладного програмування; набір визначень та протоколів для створення та інтеграції прикладного програмного забезпечення.

AWS (Amazon Web Services) – платформа хмарних обчислень, що надає сервіси інфраструктури, зберігання даних та обчислювальних потужностей.

BYOK (Bring Your Own Key) – модель безпеки, що дозволяє клієнтам використовувати власні криптографічні ключі для шифрування та дешифрування даних у хмарних сервісах.

LLM (Large Language Model) – велика мовна модель; тип нейронної мережі, навченої на великих масивах текстових даних для розуміння та генерації людської мови.

NLP (Natural Language Processing) – обробка природної мови; напрямок штучного інтелекту, що вивчає методи аналізу та синтезу текстів природною мовою.

ORM (Object-Relational Mapping) – технологія програмування, яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

RPC (Remote Procedure Call) – віддалений виклик процедур; клас технологій, що дозволяє програмі викликати функцію в іншому адресному просторі.

SEO (Search Engine Optimization) – комплекс заходів для підняття позицій сайту в результатах видачі пошукових систем.

SPA (Single Page Application) – односторінковий веб-застосунок; веб-сайт, який взаємодіє з користувачем шляхом динамічного переписування поточної сторінки, а не завантаження нових сторінок із сервера.

SQL (Structured Query Language) – структурована мова запитів; декларативна мова програмування для взаємодії з реляційними базами даних.

UML (Unified Modeling Language) – уніфікована мова моделювання; графічна мова опису для візуалізації, специфікації та документування програмних систем.

User Story – користувацька історія; короткий опис функціональності системи, сформульований природною мовою з точки зору кінцевого користувача.

ВСТУП

Сучасна індустрія розробки програмного забезпечення характеризується високою динамікою змін та зростаючими вимогами до швидкості випуску продуктів. Однією з ключових проблем у менеджменті проєктів є якісна підготовка технічної документації та описів завдань. Нечіткі, неструктуровані або неповні вимоги (User Stories, Acceptance Criteria) призводять до помилок у розробці, збільшення часу на комунікацію («re-work») та загального здорожчання проєкту.

Традиційні системи управління проєктами (Jira, Trello, Asana) надають інструментарій для трекінгу задач, проте залишають процес формування їх змісту повністю на розсуд людини. Водночас, стрімкий розвиток технологій штучного інтелекту (ШІ), зокрема великих мовних моделей (LLM), відкриває нові можливості для автоматизації рутинних інтелектуальних операцій.

Актуальність магістерської роботи зумовлена необхідністю інтеграції генеративного ШІ у процеси проєктного менеджменту для автоматизації створення, стандартизації та валідації вимог, що дозволить значно підвищити ефективність роботи команд розробників.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконується згідно з напрямком наукових досліджень кафедри програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя у сфері розробки інтелектуальних інформаційних систем.

Метою роботи є підвищення ефективності процесів управління IT-проєктами шляхом розроблення веб-орієнтованої інформаційної системи, яка використовує технології штучного інтелекту для оптимізації, генерації та уніфікації описів завдань.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз існуючих підходів та інструментів управління проєктами (Kanban, Scrum) та методів застосування ШІ в Software Engineering.
2. Спроєктувати архітектуру системи на основі принципів Feature-Sliced Design (FSD) з використанням сучасного стеку технологій (Next.js, Hono).

3. Розробити структуру бази даних, що підтримує як реляційні зв’язки для управління проєктами, так і векторні дані (embeddings) для семантичного аналізу.

4. Реалізувати програмний модуль інтеграції з OpenAI API для генерації User Stories, Acceptance Criteria та автоматичного тегування завдань.

5. Створити клієнтську частину системи з інтерактивною Kanban-дошкою та можливістю налаштування AI-моделей (BYOK).

Об’єкт дослідження – процес управління вимогами та задачами в проєктах розробки програмного забезпечення.

Предмет дослідження – методи та засоби застосування генеративного штучного інтелекту для автоматизації формування та оптимізації описів проєктних завдань.

Методи дослідження.

У роботі використано:

- системний аналіз – для визначення вимог до системи;
- об’єктно-орієнтоване проєктування (UML) – для моделювання структури та поведінки системи;
- методи інженерії знань – для побудови промптів та роботи з контекстом LLM.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі проведено детальне дослідження предметної області управління проєктами та розглянуто існуючі підходи до формування технічної документації. Проаналізовано основні недоліки ручного створення описів завдань та обґрунтовано доцільність впровадження інструментів штучного інтелекту. Також сформовано перелік функціональних і нефункціональних вимог до розроблюваної системи, ідентифіковано ключових акторів та змодельовано основні сценарії її використання.

1.1 Аналіз предметної області

Сучасна індустрія розробки програмного забезпечення (Software Engineering) характеризується високим рівнем невизначеності та складністю комунікаційних процесів. Успіх IT-проєкту значною мірою залежить не лише від технічної кваліфікації розробників, але й від якості управління процесами та чіткості постановки завдань. Предметною областю даного дослідження є процеси управління проєктами (Project Management), специфікація вимог (Requirements Engineering) та застосування методів штучного інтелекту для обробки природної мови (NLP).

Проблема якості вимог у гнучких методологіях Більшість сучасних команд використовують гнучкі методології (Agile, Scrum, Kanban) [1]. У рамках цих підходів документація часто мінімізується на користь «живого» спілкування, проте артефакти у вигляді завдань (tickets/issues) залишаються основним джерелом правди про те, що саме має бути розроблено. Типовою проблемою є так званий «людський фактор» при створенні описів:

1. Неповнота даних: Розробник або менеджер створює задачу з назвою «Виправити баг авторизації», не надаючи контексту, кроків відтворення або очікуваного результату.

2. Відсутність стандартизації: Різні члени команди описують задачі у різних форматах, ігноруючи шаблони User Story (As a... I want... So that...) [2].

3. Дублювання зусиль: Через нечіткі формулювання часто створюються дублікати задач, що призводить до розмивання ресурсів команди.

Класичні системи управління проєктами (Jira, Trello, Asana) діють як «контейнери» для інформації, забезпечуючи її зберігання та переміщення по етапах (Workflow), але вони не аналізують зміст цієї інформації. Вони є пасивними інструментами, які не запобігають внесенню неякісних даних.

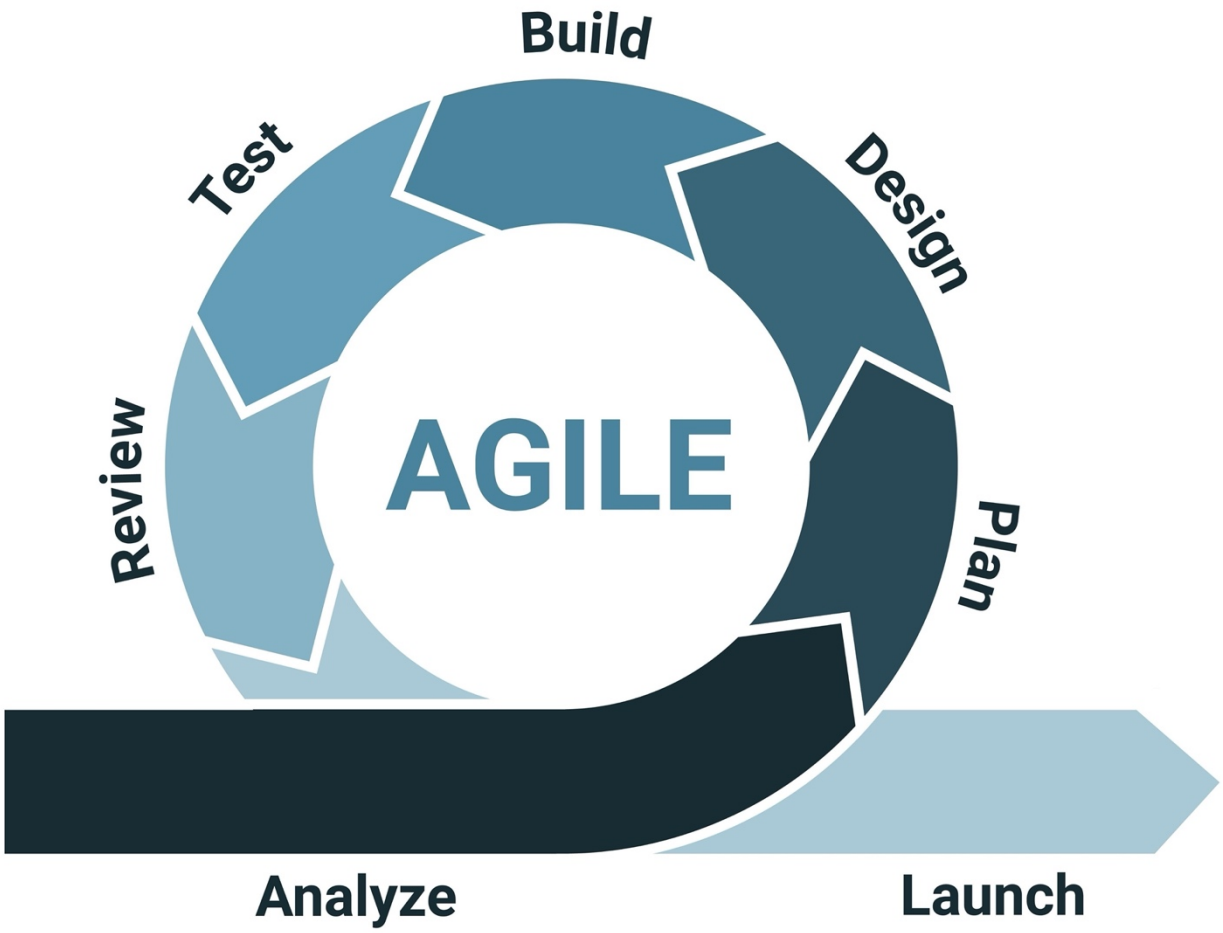

Рисунок 1.1 – Етапи життєвого циклу розробки
програмного забезпечення за методологією Agile

З появою великих мовних моделей (LLM – Large Language Models), таких як GPT-5 (OpenAI), предметна область управління вимогами зазнала суттєвих змін [3]. Технології Generative AI дозволяють перейти від пасивного зберігання описів до їх

активного формування. ШІ здатний виступати у ролі «асистента бізнес-аналітика», який:

- трансформує короткі, неструктуровані тези у повноцінні технічні специфікації;
- генерує Acceptance Criteria (критерії приймання) на основі контексту, що є критично важливим для QA-інженерів;
- визначає пропущені логічні зв’язки у вимогах.

Таким чином, предметна область роботи охоплює перетин трьох сфер:

1. Project Management (Kanban): візуалізація та потік виконання задач.
2. Generative AI: автоматизація створення контенту та валідація вимог.

Інтеграція цих компонентів у єдину веб-орієнтовану систему є актуальною інженерною задачею, що дозволяє вирішити проблему «чистого аркуша» при створенні задач та знизити когнітивне навантаження на менеджерів та розробників.

1.2 Постановка завдання та цілей

На основі проведеного аналізу предметної області встановлено, що існуючі засоби управління проєктами не забезпечують достатнього рівня автоматизації процесу формулювання технічних вимог. Ручне створення User Stories та Acceptance Criteria залишається трудомістким процесом, схильним до суб'єктивних помилок [2].

Головною метою даної роботи є підвищення ефективності діяльності команд розробників шляхом створення веб-орієнтованої системи управління проєктами, яка інтегрує можливості великих мовних моделей (LLM) для автоматичної генерації, структурування та валідації описів завдань.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Провести теоретичний аналіз методів обробки природної мови (NLP) та архітектур сучасних веб-застосунків для визначення оптимального технологічного стеку.

2. Розробити архітектуру системи, що базується на принципах Feature-Sliced Design (FSD), забезпечуючи модульність, масштабованість та чітке розділення відповідальності між клієнтською (Next.js) та серверною (Hono) частинами.

3. Реалізувати механізм інтеграції з OpenAI API [3], забезпечивши підтримку принципу BYOK (Bring Your Own Key) для гнучкого налаштування доступу до LLM.

4. Розробити алгоритми автоматичної генерації контенту, зокрема: формування User Story за шаблоном, створення списку Acceptance Criteria, визначення пріоритету та типу задачі.

5. Створити інтерактивний інтерфейс користувача у вигляді Kanban-дошки з підтримкою drag-and-drop операцій та миттєвим оновленням стану (optimistic updates).

Вимоги до функціональних можливостей системи: Система повинна забезпечувати виконання наступних функцій:

- автентифікація та авторизація: захищений доступ користувачів, розділення прав доступу до проєктів;
- управління життєвим циклом задач: створення, редагування, переміщення між колонками (To Do, In Progress, Done), видалення;
- AI-асистування: генерація розширеного опису задачі на основі короткої назви або тезисного вводу; автоматичне тегування;
- кастомізація: можливість вибору моделі ШІ та налаштування параметрів генерації (температура, системний промпт).

Вимоги до технічних характеристик:

- швидкодія: час відповіді API при стандартних операціях не повинен перевищувати 200 мс; час генерації опису ШІ — залежно від моделі, але з відображенням статусу завантаження (streaming або loading state);
- сумісність: коректна робота у сучасних веб-браузерах (Chrome, Firefox, Safari);

- архітектурна гнучкість: використання Hono для типізованої комунікації між клієнтом та сервером, що мінімізує помилки інтеграції.

Таким чином, об’єктом розробки є не просто трекер задач, а інтелектуальна система підтримки прийняття рішень на етапі бізнес-аналізу та планування спринтів.

1.3. Пошук акторів та варіантів використання

Для формалізації функціональних вимог до системи було використано діаграму варіантів використання (Use Case Diagram) мови UML [4]. Цей підхід дозволяє визначити межі системи та описати взаємодію між зовнішніми сутностями (акторами) та функціональними процесами.

1.3.1. Ідентифікація акторів

На основі аналізу бізнес-процесів управління проєктами виділено два типи акторів, які взаємодіють із розроблюваною системою:

1. Користувач (User) – основний дійовий суб'єкт системи. Це узагальнена роль, яка може відповідати Project Manager, Business Analyst або Developer.

1.1 обов'язки: створення проєктів, управління дошками Kanban, ініціювання запитів до ШІ, налаштування інтеграції (BYOK);

1.2 цілі: швидке створення якісних описів задач, організація робочого процесу.

2 Сервіс ШІ (AI Service / OpenAI API) -вторинний (зовнішній) актор. Це зовнішня система, яка отримує запити від бекенду та повертає згенерований контент або векторні дані.

2.1 обов'язки: обробка текстових промптів, генерація User Stories, створення Acceptance Criteria, векторизація тексту (embeddings).

1.3.2. Визначення варіантів використання (Use Cases)

Функціональність системи декомпозовано на логічні групи прецедентів:

Група А. Управління обліковим записом та налаштуваннями:

- реєстрація/Вхід (Sign Up / Sign In): Автентифікація користувача за допомогою email та пароля;
- налаштування ШІ-моделі (Configure AI): Введення API-ключа (OpenAI Key), вибір моделі (наприклад, GPT-5), налаштування рівня генерації.

Група Б. Робота з проєктами та дошкою (Kanban Core):

- створити проєкт: Ініціалізація нового простору для задач із базовим описом;
- управління колонками: Додавання або зміна порядку колонок (To Do, In Progress, Done);
- переміщення задачі (Drag-and-Drop): Зміна статусу задачі шляхом перетягування картки між колонками.

Група В. Робота з задачами та AI-генерація (Intelligent Features):

- створити задачу (Create Task): Введення базової назви та короткого контексту;
- згенерувати опис (Generate AI Description): Система відправляє контекст задачі та проєкту до AI-сервісу та отримує структуровану відповідь:
  - включає: генерацію User Story, Acceptance Criteria, технічних нотаток.
- редагування вручну: Можливість користувача коригувати згенерований ШІ контент.

1.3.3. Специфікація основного сценарію «AI-генерація опису задачі»

Ключовим прецедентом, що забезпечує наукову новизну роботи, є автоматизована генерація вимог. Сценарій виконується наступним чином:

1. Дія актора: Користувач створює картку, вводить назву (наприклад, "Інтеграція Stripe") і натискає кнопку "Generate with AI".

2. Системна дія: Система збирає контекст (опис проєкту + назва задачі) і формує промпт.

3. Взаємодія: Система надсилає запит до актора "Сервіс ШІ".

4. Відповідь системи: Отриманий JSON розпарсується, поля форми (Description, AC, Priority) автоматично заповнюються.

5. Результат: Користувач отримує готову специфікацію, яку може затвердити або відредагувати.

Такий підхід дозволяє мінімізувати рутинну роботу користувача та забезпечити єдиний стандарт оформлення документації в проєкті.

Для наочного відображення функціональних можливостей та взаємодії акторів із системою розроблено діаграму варіантів використання, яка наведена на рисунку 1.2.

На діаграмі виділено два ключові блоки функціональності:

1. Базові функції управління проєктами: включають стандартні операції CRUD (створення, читання, оновлення, видалення) для проєктів та задач, а також управління станами Kanban-дошки.

2. Інтелектуальні функції (AI Features): включають сценарії, що вимагають звернення до зовнішнього сервісу OpenAI. Стрілками залежності показано, що прецеденти «Генерація опису» та «Пошук схожих задач» ініціюються користувачем, але для їх виконання система звертається до зовнішнього актора «AI Service».

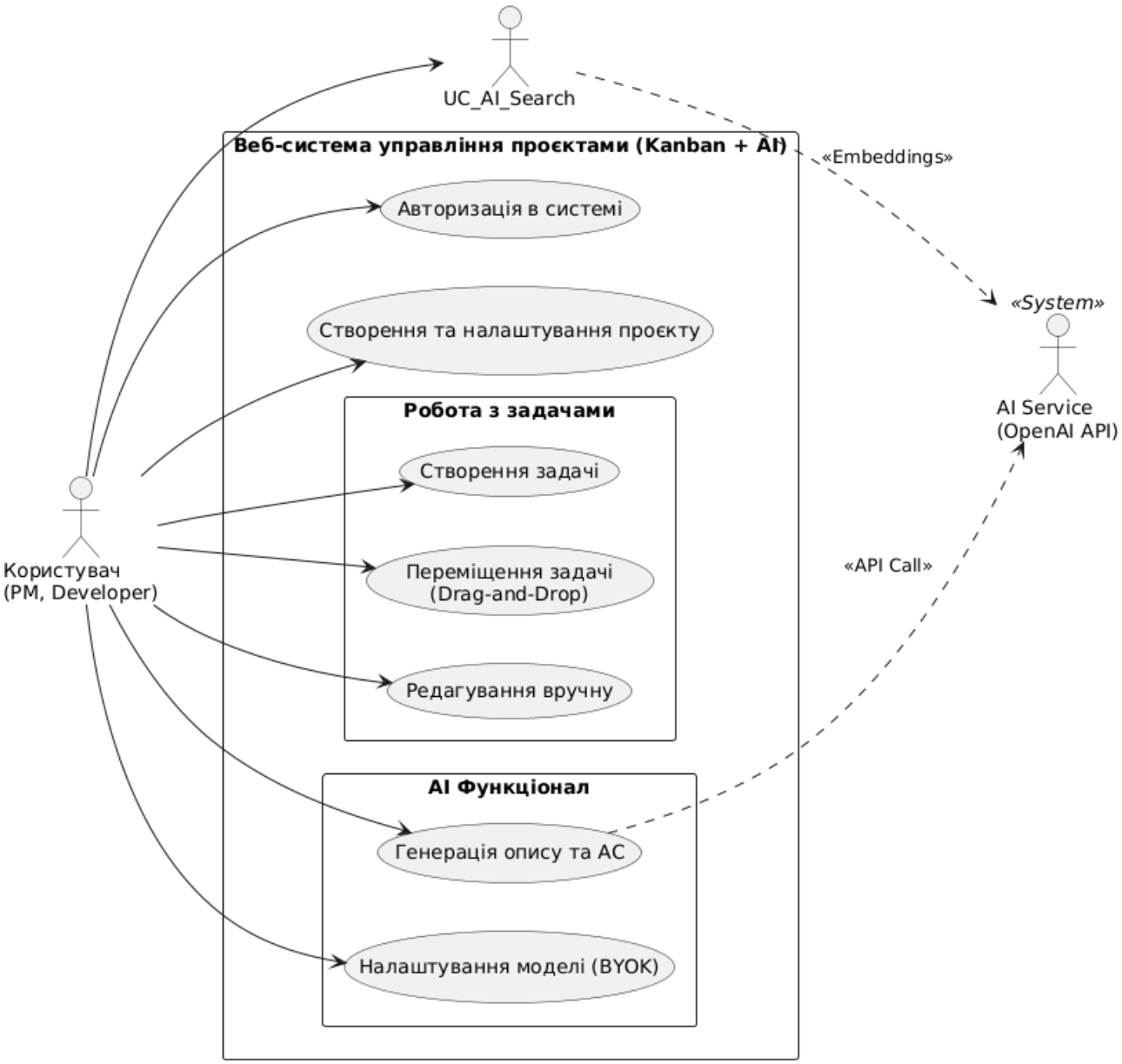

Рисунок 1.2 – Пошук акторів та варіантів використання

1.4. Опис ключових варіантів використання

У цьому підрозділі наведено детальну специфікацію найважливіших сценаріїв (прецедентів), які реалізують основну мету роботи — автоматизацію управління вимогами. Кожен варіант використання описано через послідовність дій акторів та реакцій системи.

Сценарій 1. Генерація розширеного опису задачі (AI Task Generation) Цей сценарій є центральним для системи, оскільки демонструє інтеграцію генеративного ШІ:

- актори: Користувач, Сервіс ШІ (OpenAI);
- передумови: Користувач авторизований, знаходиться на сторінці проєкту, в налаштуваннях вказано API Key;
- основний потік подій:
  - користувач відкриває модальне вікно створення задачі та вводить коротку назву (наприклад, «Авторизація через Google»);
  - користувач натискає кнопку «Generate Description with AI»;
  - система (Frontend) відправляє запит на Backend, передаючи назву задачі та ID проєкту;
  - backend зчитує з Бази Даних глобальний контекст проєкту (опис продукту, стиль спілкування);
  - backend формує комбінований промпт і відправляє його до OpenAI API;
  - сервіс ШІ повертає структуровану відповідь у форматі JSON (опис, User Story, Acceptance Criteria, оцінка складності);
  - система автоматично заповнює відповідні поля форми отриманими даними.
- після умови: Форма створення задачі заповнена деталізованим контентом, готовим до збереження.

Сценарій 2. Управління життєвим циклом задачі (Kanban Workflow) Класичний сценарій для Kanban-систем:

- актори: користувач.
- основний потік подій:
  - користувач перетягує картку задачі з колонки «To Do» в колонку «In Progress»;
  - система візуально переміщує картку миттєво (Optimistic UI Update);
  - система відправляє асинхронний запит на сервер для оновлення статусу та позиції задачі;

- система відправляє асинхронний запит на сервер для оновлення статусу та позиції задачі.
- після умови: Статус задачі оновлено, зміни видимі для інших учасників команди.

Візуалізація процесу взаємодії Для детального розуміння технічної реалізації сценарію №1 (Генерація опису) розроблено діаграму послідовності (рисунок 1.3). Вона демонструє взаємодію між клієнтським інтерфейсом (Next.js), сервером (Hono) та зовнішнім API OpenAI.

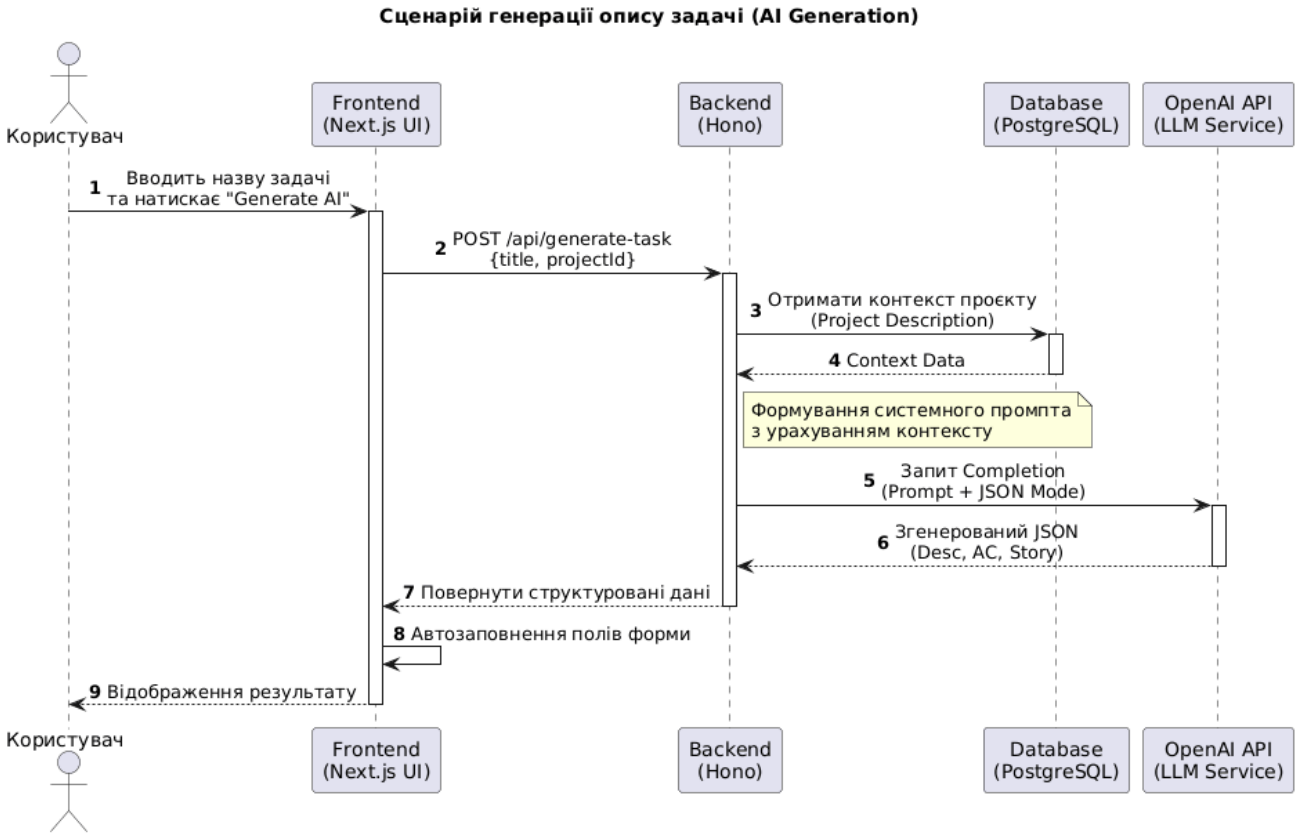

Рисунок 1.3 – Діаграма послідовності процесу генерації опису задачі за допомогою ШІ

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

У даному розділі описано етапи проєктування та безпосередньої програмної реалізації інформаційної системи управління проєктами. На основі сформованих у попередньому розділі функціональних вимог та аналізу предметної області здійснено обґрунтування вибору методології розробки та сучасного технологічного стеку. Основну увагу приділено розробці клієнт-серверної архітектури, проєктуванню схеми бази даних, реалізації алгоритмів взаємодії з великими мовними моделями (LLM), а також створенню адаптивного інтерфейсу користувача.

2.1. Вибір процесу розробки

Вибір методології розробки є критичним етапом проєктування, оскільки він визначає підхід до планування, контролю якості та управління змінами в проєкті. Враховуючи специфіку магістерської роботи, яка передбачає інтеграцію новітніх технологій (Generative AI) та необхідність частого тестування гіпотез, для розробки системи було обрано гнучку методологію (Agile) з використанням ітеративного підходу [5].

Обґрунтуванням вибору Agile є причина тому що класична каскадна модель (Waterfall) не є ефективною для даного проєкту через високий рівень невизначеності у роботі з великими мовними моделями (LLM). Поведінка штучного інтелекту є недетермінованою, тому розробка промптів та налаштування параметрів генерації вимагають багаторазових експериментів та ітерацій, що неможливо чітко спланувати на початковому етапі [6].

Обраний процес розробки базується на наступних принципах:

1. Інтерактивність: Весь процес розбито на короткі цикли (спринти), кожен з яких завершується отриманням робочого інкременту продукту (наприклад, спочатку реалізація Kanban-дошки, потім – підключення AI).

2. Адаптивність: Можливість змінювати структуру промптів або формат даних на будь-якому етапі розробки без суттєвих втрат часу.

3. Постійна інтеграція (CI/CD): Оскільки використовується сучасний веб-стек (Next.js), процес розгортання автоматизовано, що дозволяє миттєво тестувати зміни на хостингу [7].

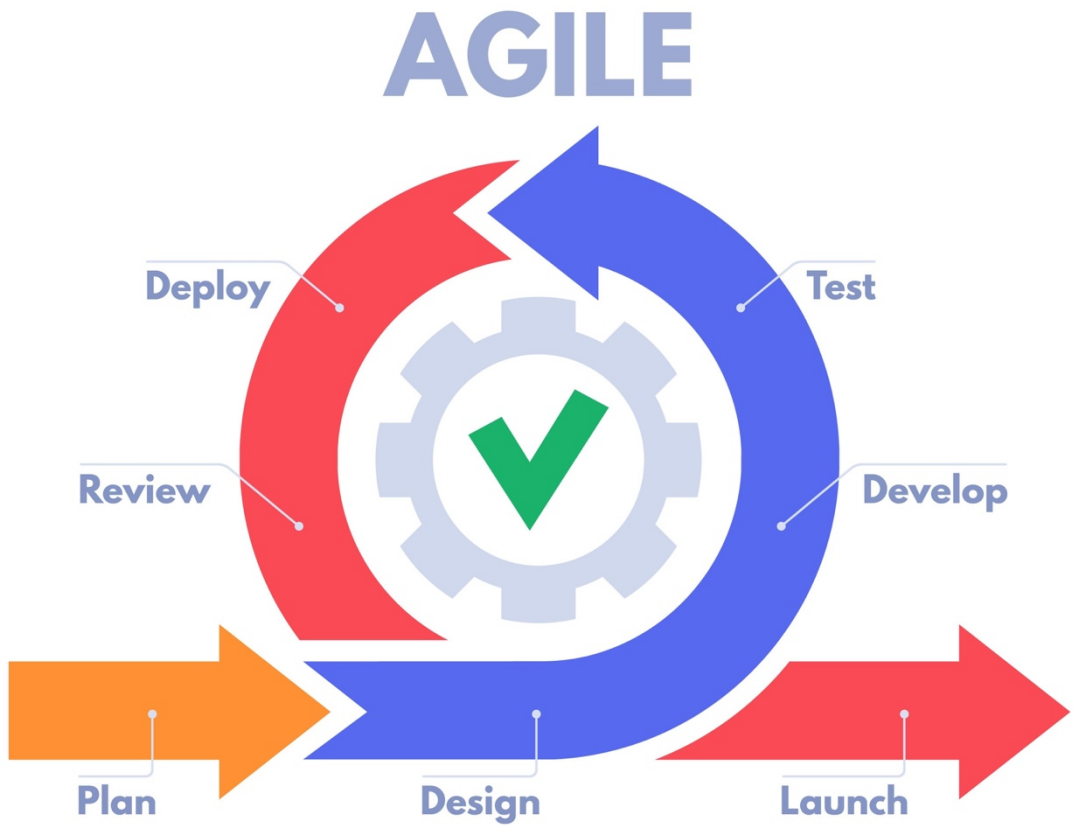

Рисунок 2.1 – Схема ітеративного процесу розробки системи

Етапи життєвого циклу розробки (SDLC) Відповідно до обраного процесу, розробка системи поділяється на такі фази:

1. Аналіз вимог та моделювання: На цьому етапі формуються вимоги до структури даних (User Story, Task types) та визначаються сценарії взаємодії користувача з AI. Результатом є UML-діаграми та технічне завдання.

2. Проєктування архітектури: Вибір технологічного стеку (Hono, Drizzle, React Query) та архітектурного патерну. Для забезпечення масштабованості frontend-частини застосовано методологію Feature-Sliced Design (FSD), яка дозволяє розбити код на незалежні модулі (features), спрощуючи підтримку [7].

3. Реалізація (Implementation): Написання програмного коду. Розробка ведеться за принципом «Code First» для бізнес-логіки та «API First» для взаємодії клієнта з сервером через RPC.

4. Тестування та налагодження (Testing & Tuning): Окрім стандартного функціонального тестування (Unit/Integration tests), цей етап включає специфічне «Prompt Engineering Testing» - перевірку якості відповідей ШІ на різних типах вхідних даних.

5. Розгортання (Deployment): Публікація веб-застосунку на хмарній платформі (Vercel/Railway) для доступу кінцевих користувачів.

Такий підхід дозволяє мінімізувати ризики та забезпечити створення якісного програмного продукту, що відповідає поставленим вимогам.

2.2. Проєктування архітектури системи

Для забезпечення масштабованості, надійності та простоти підтримки програмного забезпечення було розроблено багаторівневу клієнт-серверну архітектуру. В основу проєктування покладено принципи модульності та слабкої зв'язності компонентів (Loose Coupling) [8].

Загальна архітектурна схема Система побудована на базі мета-фреймворку Next.js 16, що дозволяє реалізувати гібридну модель рендерингу (Server-Side Rendering та Client-Side Rendering). Взаємодія між клієнтською частиною та сервером реалізована через технологію RPC (Remote Procedure Call) на базі фреймворку Hono, що дозволяє використовувати єдині типи даних (TypeScript interfaces) на всіх рівнях системи без необхідності генерації коду.

Архітектура системи складається з наступних логічних рівнів:

1. Рівень представлення (Presentation Layer): Відповідає за взаємодію з користувачем. Реалізований за допомогою бібліотеки React та компонентів інтерфейсу shadcn/ui. Цей рівень обробляє події (кліки, drag-and-drop на Kanban-дошці) та ініціює запити до сервера.

2. Рівень API та маршрутизації (Transport Layer): Побудований на базі Hono. Цей шар приймає запити від клієнта, виконує валідацію вхідних даних (за допомогою бібліотеки Zod) та маршрутизує їх до відповідних бізнес-контролерів. Використання Hono RPC забезпечує повну типобезпеку (type-safety) між бекендом та фронтендом [9].

3. Рівень бізнес-логіки (Domain Layer): Містить основні алгоритми системи: управління станами задач, логіку формування промптів для штучного інтелекту, обробку прав доступу. Саме тут відбувається інтеграція з OpenAI API для генерації контенту.

4. Рівень даних (Data Access Layer): Відповідає за збереження та отримання інформації з реляційної бази даних. Реалізований за допомогою Drizzle ORM, що дозволяє взаємодіяти з SQL-базою даних, використовуючи об'єктно-орієнтований підхід.

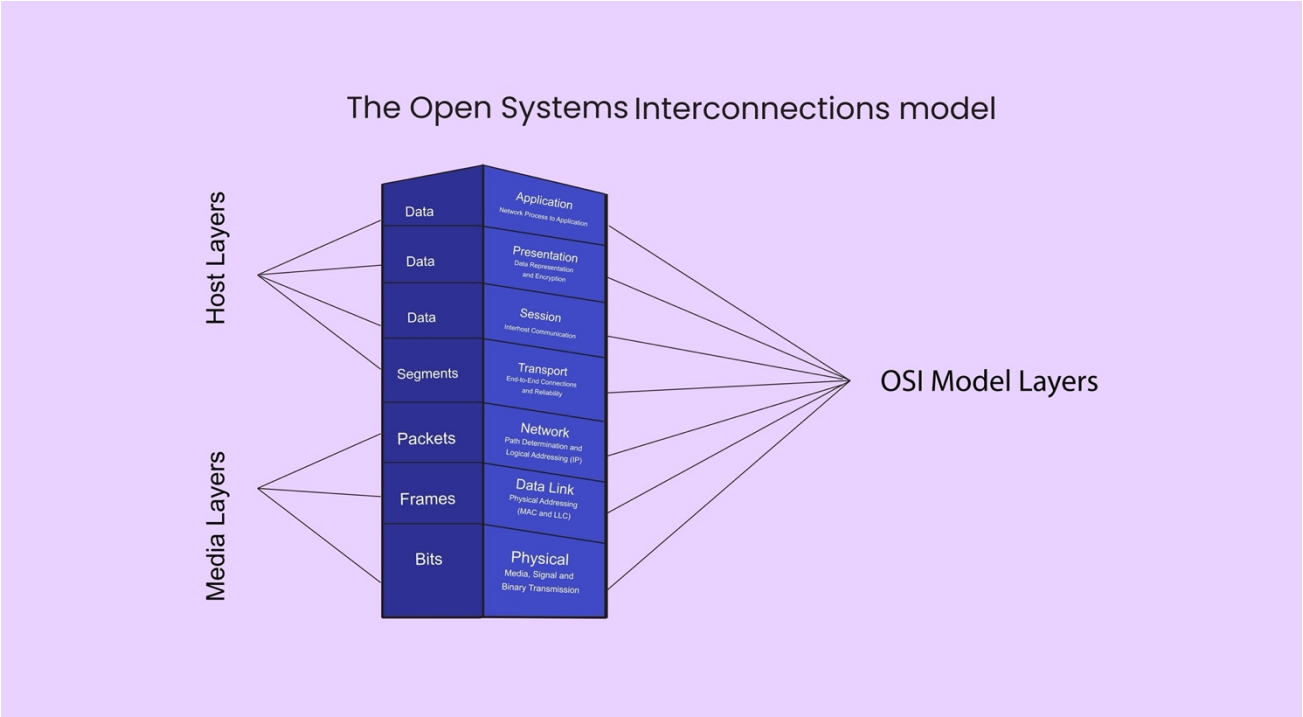

Рисунок 2.2 – Високорівнева архітектура програмної системи

Архітектурна методологія Feature-Sliced Design (FSD) Для структурування кодової бази фронтенду застосовано методологію Feature-Sliced Design. Це дозволяє уникнути проблеми «спагеті-коду» та забезпечує чіткий розподіл відповідальності. Згідно з FSD, проєкт розділено на шари (Layers) [10]:

- App: Глобальні налаштування, стилі, провайдери контексту;
- Pages: Сторінки додатку (наприклад, сторінка входу, сторінка проєкту);
- Widgets: Самостійні блоки інтерфейсу (наприклад, Kanban-дошка, форма створення задачі);
- Features: Користувацькі сценарії (наприклад, «авторизуватися», «згенерувати опис», «перетягнути задачу»);
- Entities: Бізнес-сутності (User, Task, Project), які містять інтерфейси та базові UI-елементи (картка задачі);
- Shared: Перевикористовувані компоненти (кнопки, інпути), утиліти та конфігурації API.

Такий поділ дозволяє ізолювати логіку AI-генерації в окремий модуль (Feature), не змішуючи його з логікою відображення Kanban-дошки.

Інтеграція зі службами штучного інтелекту Архітектура передбачає асинхронну взаємодію з LLM. Оскільки генерація тексту може займати певний час, серверна частина використовує потокову передачу даних або асинхронні відповіді, щоб не блокувати інтерфейс користувача під час формування User Story.

2.2.1. Стратегія управління станом та синхронізація даних

У проєктуванні інтерактивних веб-інтерфейсів критично важливим архітектурним рішенням є вибір стратегії управління станом (State Management). Для розроблюваної системи Kanban-дошки було відмовлено від використання глобальних сторів (як Redux) на користь гібридного підходу, що розділяє стан на дві категорії: Server State (дані з БД) та Client State (інтерфейсні стани).

Управління серверним станом (React Query) Оскільки дані задач, колонок та проєктів зберігаються у віддаленій базі даних, вони по суті є кешем на клієнті. Для

синхронізації використано бібліотеку TanStack Query (React Query). Архітектурні переваги цього рішення:

1. Автоматична дедуплікація запитів: Якщо декілька віджетів на сторінці потребують одних і тих самих даних (наприклад, список тегів), система робить лише один запит до API.

2. Stale-While-Revalidate: Користувач миттєво бачить закешовані дані, поки у фоновому режимі відбувається їх оновлення. Це забезпечує відчуття миттєвої реакції інтерфейсу.

3. Smart Refetching: Автоматичне оновлення даних при поверненні користувача на вкладку браузера (Window Focus Refetching), що гарантує актуальність стану дошки.

Реалізація патерну Optimistic Updates (Оптимістичні оновлення): Для Kanban-дошки критичною є швидкість реакції на дії drag-and-drop. Очікування відповіді від сервера (100–300 мс) при кожному перетягуванні картки робить інтерфейс "в'язким". В архітектуру закладено механізм оптимістичних оновлень:

1. UI Update: При події onDragEnd інтерфейс миттєво оновлює локальний кеш, переміщуючи картку в нову колонку.

2. API Request: Паралельно відправляється асинхронний запит на сервер (PUT /api/tasks/move).

3. Rollback: Якщо сервер повертає помилку (наприклад, втрата з'єднання), система автоматично відкочує зміни в локальному кеші до попереднього стану та показує сповіщення про помилку.

Схема потоку даних (Data Flow): Архітектура потоку даних є односпрямованою (Unidirectional Data Flow):

1. Server Component: Завантажує початковий стан (Hydration) на сервері для SEO та швидкого першого відмальовування (FCP).

2. Client Component: "Гідрує" стан і перехоплює управління для подальшої інтерактивності.

3. Mutation: Дії користувача викликають мутації, які оновлюють серверні дані та інвалідують (invalidate) відповідні частини кешу, викликаючи їх оновлення.

Такий підхід дозволив зменшити кількість коду, пов'язаного з ручним управлінням станами завантаження (isLoading, isError), на 40% порівняно з класичним підходом useEffect, а також забезпечив високу консистентність даних між клієнтом та сервером.

2.3. Побудова схем бази даних

Проєктування схеми бази даних є ключовим етапом розробки, що визначає ефективність зберігання, цілісність та швидкість доступу до даних. Для реалізації системи було обрано реляційну модель даних (RDBMS) під управлінням СУБД PostgreSQL. Цей вибір зумовлений високою надійністю системи, підтримкою транзакцій (ACID) та наявністю потужних засобів для роботи зі структурованими даними [11].

Для взаємодії з базою даних на рівні програмного коду використовується Drizzle ORM. Це дозволяє описувати схему даних безпосередньо мовою TypeScript, забезпечуючи автоматичну генерацію SQL-міграцій та типізацію запитів [12].

2.3.1. Логічна модель даних (ER-діаграма)

На основі аналізу предметної області було виділено основні сутності системи: Users (Користувачі), Projects (Проєкти), Columns (Колонки Kanban), Tasks (Задачі) та UserSettings (Налаштування ШІ).

Схему зв’язків між сутностями зображено на рисунку 2.3.

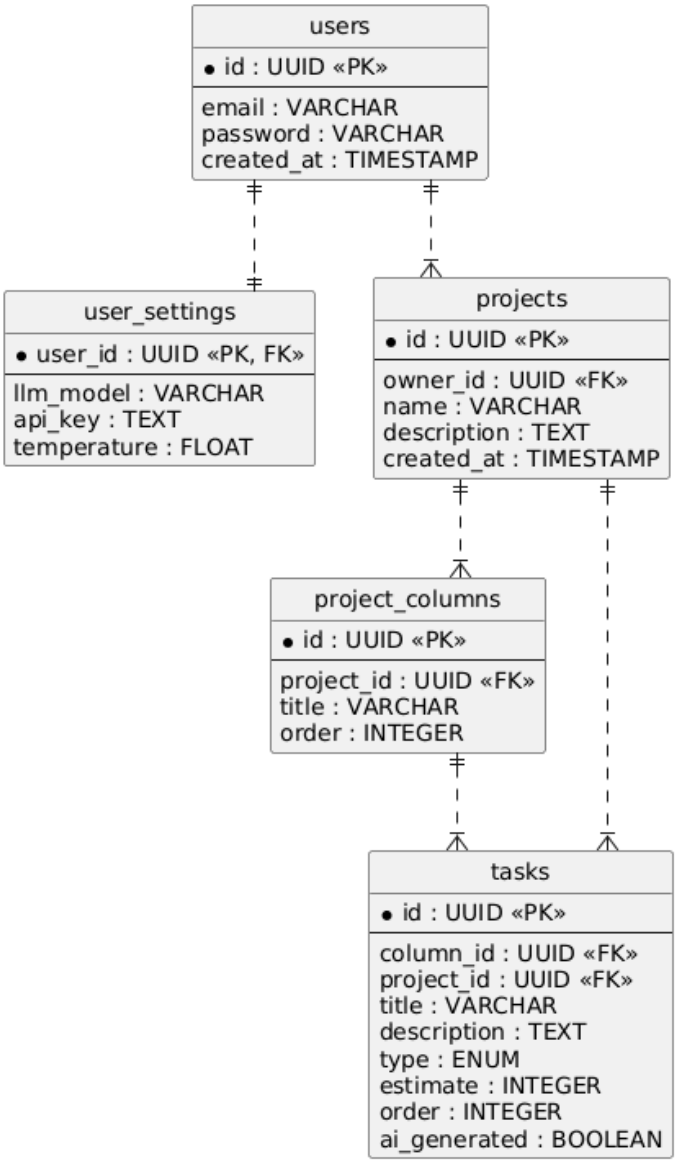

Рисунок 2.3 – ER діаграма бази даних системи

2.3.2. Фізична модель даних

Нижче наведено детальний опис ключових таблиць системи.

Таблиця users Зберігає облікові дані користувачів:

- id (UUID, PK): Унікальний ідентифікатор;
- email (VARCHAR, Unique): Логін користувача;
- password (VARCHAR): Хеш пароля.

Таблиця user_settings Критично важлива таблиця для реалізації концепції BYOK (Bring Your Own Key). Вона має зв'язок 1:1 з таблицею користувачів:

- user_id (UUID, FK): Посилання на користувача;
- llm_model (VARCHAR): Назва обраної моделі (наприклад, "gpt-4o");
- api_key (TEXT): Зашифрований API-ключ для доступу до OpenAI;
- temperature (FLOAT): Параметр варіативності генерації відповідей ШІ.

Таблиця projects Містить інформацію про робочий простір. Важливим полем є description, оскільки його вміст використовується як системний контекст (System Prompt) при генерації задач:

- id (UUID, PK): Ідентифікатор проєкту;
- owner_id (UUID, FK): Власник проєкту;
- name (VARCHAR): Назва;
- description (TEXT): Опис суті проєкту для ШІ.

Таблиця project_columns Визначає структуру Kanban-дошки:

- id (UUID, PK);
- project_id (UUID, FK): Зв'язок з проєктом;
- title (VARCHAR): Назва колонки (наприклад, "To Do");
- order (INTEGER): Числове значення для сортування колонок на екрані.

Таблиця tasks Основна сутність системи. Містить як дані, введені користувачем, так і згенеровані ШІ:

- id (UUID, PK);
- column_id (UUID, FK): Поточний статус задачі (колонка);
- title (VARCHAR): Короткий заголовок;
- description (TEXT): Повний опис (User Story + AC);
- type (ENUM): Тип задачі ('user_story', 'task', 'bug');
- estimate (INTEGER): Оцінка складності (Story Points);
- order (INTEGER): Позиція задачі всередині колонки (для drag-and-drop);
- ai_generated (BOOLEAN): Прапорець, що вказує, чи був опис створений автоматично.

2.3.3. Нормалізація бази даних Розроблена схема відповідає вимогам третьої нормальної форми (3NF) [13]:

- усі атрибути є атомарними;
- усі неключові атрибути залежать від первинного ключа;
- відсутні транзитивні залежності між неключовими атрибутами (наприклад, налаштування ШІ винесено в окрему таблицю user_settings, щоб не перевантажувати таблицю users).

2.4 Побудова UML-діаграм класів

Діаграма класів (Class Diagram) є центральною частиною об'єктно-орієнтованого проєктування, оскільки вона описує статичну структуру системи, визначаючи типи об'єктів, їхні атрибути, методи та взаємозв'язки між ними. Для даної системи діаграма класів відображає проєкцію реляційної моделі даних на рівень бізнес-логіки застосунку, реалізованого мовою TypeScript [14].

В архітектурі застосунку виділено дві основні категорії класів:

1. Сутності (Entities): Класи, що відповідають за структуру даних і прямо відображаються на таблиці бази даних (User, Project, Task). У контексті Drizzle ORM вони представлені як типізовані схеми.
2. Сервіси (Services): Класи (або модулі), що інкапсулюють бізнес-логіку та методи обробки даних (AIService, TaskService).

Опис основних класів:

- User: Базовий клас користувача. Містить атрибути для авторизації та методи для управління налаштуваннями профілю.
- UserSettings: Клас, що зберігає конфігурацію інтеграції з ШІ (API Key, обрана модель). Він асоційований з класом User відношенням композиції (налаштування не існують без користувача).
- Project: Клас, що агрегує в собі списки колонок та задач. Містить метод getProjectContext(), який формує текстовий опис проєкту для передачі в ШІ.

- Task: Клас, що описує окреме завдання. Окрім стандартних методів CRUD (Create, Read, Update, Delete), він містить атрибути aiGenerated та aiSummary, які заповнюються автоматично.

- AIService: Спеціалізований сервісний клас (Controller), який не зберігає стан, але надає методи для взаємодії із зовнішнім API OpenAI. Його метод generateTaskDescription() приймає вхідні дані від користувача та повертає об’єкт Task.

Взаємозв'язки (Relationships): На діаграмі (рисунок. 2.4) відображено наступні типи зв'язків:

- композиція (Composition): Між Project та Column, оскільки колонки є невід'ємною частиною проєкту;

- агрегація (Aggregation): Між Column та Task, оскільки задачі можуть переміщуватися між колонками, але логічно належать до них;

- залежність (Dependency): Клас TaskService залежить від AIService, оскільки використовує його функціонал для генерації контенту.

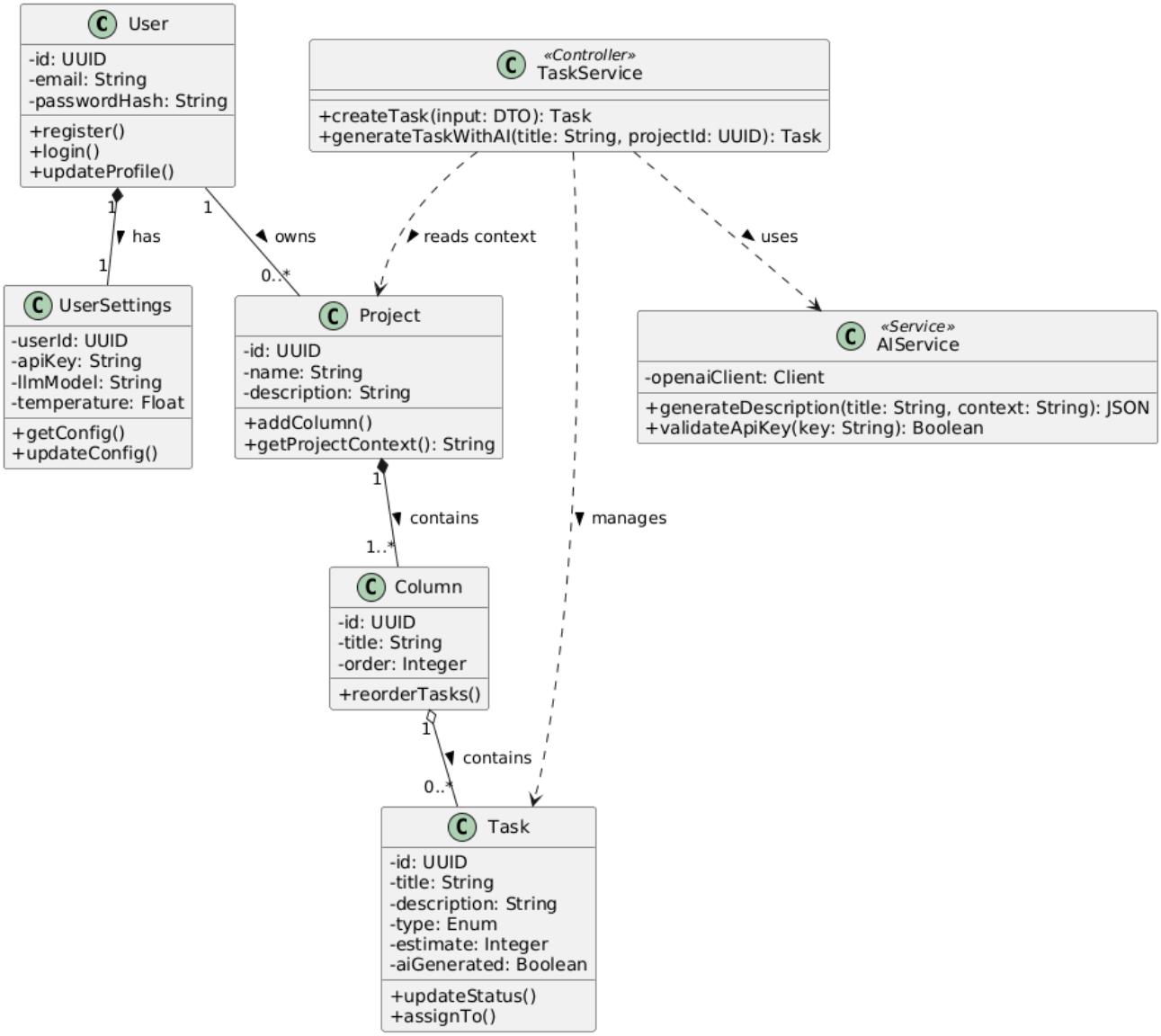

Рисунок 2.4 – Діаграма класів розробленої системи

Така структура класів забезпечує чітке розмежування відповідальності: сутності відповідають лише за зберігання даних, а сервіси – за логіку їх обробки та інтеграцію з ШІ. Це спрощує тестування та подальше розширення системи.

2.5. Вибір мови та середовища розробки

Ефективність процесу розробки та якість кінцевого програмного продукту безпосередньо залежать від обраних інструментальних засобів. Для реалізації системи управління проєктами було сформовано сучасний технологічний стек, який забезпечує високу швидкодію, масштабованість та зручність підтримки коду.

Мова програмування: TypeScript Основною мовою розробки обрано TypeScript – типізовану надбудову над JavaScript, розроблену компанією Microsoft. Вибір TypeScript зумовлений такими факторами [15]:

1. Статична типізація: Дозволяє виявляти помилки ще на етапі компіляції, що є критично важливим при роботі зі складними структурами даних, які повертає AI (JSON-об'єкти з вкладеними полями).

2. Інтелектуальна підтримка (IntelliSense): Значно прискорює написання коду завдяки автодоповненню та автоматичному рефакторингу в IDE.

3. Єдина мова для всього стеку: Використання TypeScript як на клієнті (React), так і на сервері (Hono) дозволяє використовувати спільні інтерфейси (Shared Types), що унеможливлює розсинхронізацію API.

Середовище виконання та фреймворк: Next.js 16 В якості основи веб-застосунку обрано фреймворк Next.js 16, що базується на бібліотеці React. Ключовою перевагою версії 16 є стабільна підтримка React Server Components (RSC) та Server Actions. Це дозволяє виконувати логіку звернення до бази даних та API OpenAI безпосередньо на сервері, відправляючи на клієнт лише готовий HTML або мінімальний JSON, що покращує SEO та швидкість завантаження [16].

Інструменти стилізації та UI: TailwindCSS + shadcn/ui Для розробки інтерфейсу користувача обрано підхід Utility-First CSS за допомогою фреймворку TailwindCSS. Це дозволяє створювати адаптивні компоненти без написання окремих CSS-файлів. В якості бібліотеки компонентів використовується shadcn/ui. На відміну від класичних бібліотек (MUI, AntD), shadcn/ui надає не npm-пакет, а вихідний код компонентів, що дає повний контроль над їхньою логікою та стилізацією [17].

Інтегроване середовище розробки (IDE) Розробка ведеться у середовищі Visual Studio Code (VS Code). Завдяки широкій екосистемі розширень (ESLint, Prettier, Tailwind CSS IntelliSense), VS Code є стандартом де-факто для веб-розробки.

Система контролю версій Для управління змінами в коді використовується розподілена система Git, а хостинг репозиторію здійснюється на платформі GitHub,

що дозволяє налаштувати автоматичні процеси CI/CD (Continuous Integration / Continuous Delivery).

Підсумовуючи, обраний стек (TypeScript + Next.js + Tailwind) є сучасним стандартом індустрії, що гарантує актуальність розробленого рішення на тривалий час.

2.6. Реалізація основних класів та методів

Реалізація програмної логіки системи виконана з дотриманням принципів чистої архітектури та суворої типізації. Оскільки в якості основної мови обрано TypeScript, класи та сутності системи реалізовані через інтерфейси та схеми бібліотеки Drizzle ORM.

2.6.1. Реалізація моделі даних (Data Layer)

На рівні коду таблиці бази даних представлені як об'єкти схеми Drizzle. Це дозволяє маніпулювати даними, використовуючи методи мови програмування, а не «сирі» SQL-запити. Такий підхід забезпечує автоматичну генерацію TypeScript-типів для використання у фронтенді.

2.6.2. Реалізація сервісу інтеграції з ШІ (AI Service)

Логіка взаємодії з OpenAI інкапсульована в окремому модулі. Ключовим методом є generateTaskDescription, який відповідає за формування промпту та обробку відповіді.

Для забезпечення стабільності формату вихідних даних використано режим JSON Mode моделі GPT-5. Це гарантує, що система отримає не просто текст, а структурований об'єкт із полями summary, acceptanceCriteria та storyPoints [20].

Алгоритм роботи методу:

1. Отримання контексту проєкту (опис, технології) з БД.

2. Формування системного промпту (інструкція рольової моделі: "You are an expert Product Owner...").

3. Відправка запиту до API.

4. Валідація отриманого JSON за допомогою бібліотеки Zod.

2.6.3. Програмна реалізація взаємодії з LLM (OpenAI API)

Критичним компонентом системи є модуль інтеграції з великою мовною моделлю. Оскільки система підтримує принцип BYOK (Bring Your Own Key), ініціалізація клієнта OpenAI відбувається динамічно для кожного запиту, використовуючи зашифрований ключ поточного користувача.

Взаємодія реалізована на серверній стороні (Next.js Server Actions або Hono Controller), що дозволяє приховати логіку формування промптів та захистити API-ключі від перехоплення у браузері.

Структура запиту (Prompt Engineering) Для отримання прогнозованого результату використано техніку Few-Shot Prompting або чіткого системного інструктування. Запит до моделі складається з двох частин:

1. System Prompt: Задає рольову модель ("Ти досвідчений Product Owner"), контекст проєкту (технології, стиль опису) та вимогу повертати дані виключно у форматі JSON.

2. User Prompt: Містить конкретну назву задачі, введену користувачем.

Обробка результатів Отриманий від моделі JSON-об'єкт проходить додаткову валідацію через бібліотеку Zod перед тим, як бути записаним у базу даних або відправленим на фронтенд. Це дозволяє уникнути помилок Runtime Error, якщо модель галюцинує і повертає некоректну структуру даних.

2.7. Розробка інтерфейсу користувача

Інтерфейс користувача (User Interface, UI) є критично важливою складовою будь-якої інформаційної системи, оскільки саме він забезпечує ефективну

взаємодію людини з програмним забезпеченням. У контексті системи управління проєктами, де користувач проводить значну частину робочого часу, інтерфейс має відповідати вимогам ергономічності, інтуїтивної зрозумілості та візуальної привабливості.

При проєктуванні UI для даної системи було використано підхід «Content-First», де головна увага приділяється даним (задачам та їх опису), а елементи керування не відволікають від основного контенту.

2.7.1. Загальна стилістика та дизайн-система

Для забезпечення візуальної цілісності та прискорення розробки було використано бібліотеку компонентів shadcn/ui у поєднанні з утилітарним CSS-фреймворком TailwindCSS. Цей вибір дозволив реалізувати сучасний «чистий» дизайн (Clean Design) з наступними характеристиками:

- типографіка: Використання шрифтів без зарубок (Sans-serif, наприклад, Inter або Geist) забезпечує високу читабельність тексту на екранах різного розширення;
- кольорова палітра: Основний фон — нейтральний (білий або темно-сірий у темній темі), акцентний колір (primary) — насичений чорний або темно-синій для кнопок дій. Для семантичного позначення статусів задач використано загальноприйняті кольори: червоний (Bug/Error), зелений (Done/Success), жовтий (In Progress);
- адаптивність: Інтерфейс автоматично підлаштовується під розміри екрану, забезпечуючи коректне відображення як на десктопах, так і на планшетах, що є важливим для мобільності менеджерів.

Темна тема (Dark Mode) Враховуючи сучасні тенденції розробки, реалізовано повноцінну підтримку темного режиму. Це знижує навантаження на зір користувача при роботі в умовах слабкого освітлення. Перемикання теми відбувається миттєво без перезавантаження сторінки завдяки використанню CSS-змінних (CSS Variables).

2.7.2. Екран авторизації та входу в систему

Першою точкою контакту користувача з системою є сторінка автентифікації. Дизайн цієї сторінки є мінімалістичним, щоб сфокусувати увагу на формі введення даних. Центральний елемент – картка входу, яка містить поля для введення Email та пароля, а також кнопку «Sign In».

Реалізовано валідацію полів у реальному часі: якщо формат email некоректний або поле пусте, користувач отримує візуальне повідомлення (червона рамка та текст помилки) до моменту відправки запиту на сервер. Це відповідає принципу миттєвого зворотного зв'язку (Immediate Feedback) [22].

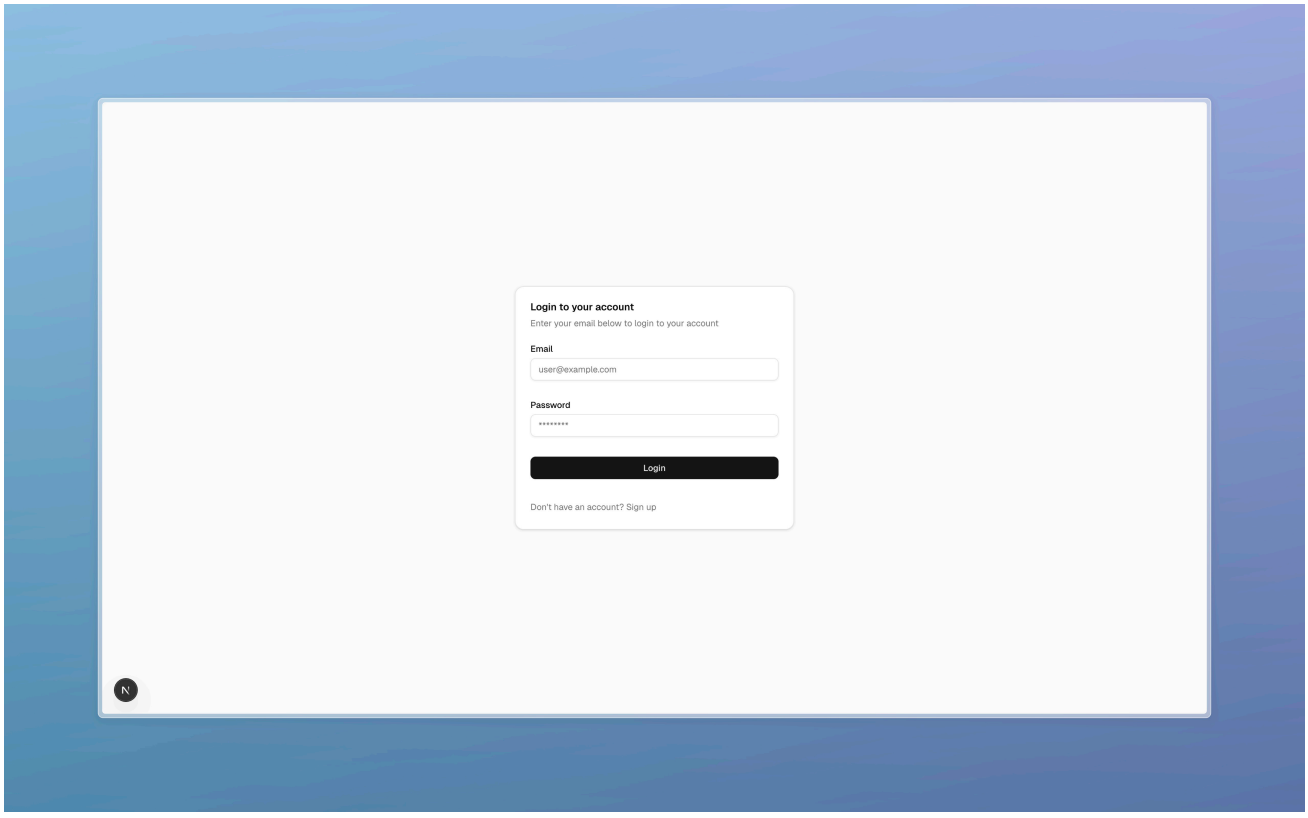

Рисунок 2.5 – Сторінка авторизації

2.7.3. Панель керування проєктами (Dashboard)

Після успішної авторизації користувач потрапляє на головну сторінку (Dashboard). Тут реалізовано плитковий дизайн (Grid Layout) для відображення списку доступних проєктів. Кожна картка проєкту містить:

1. Назву проєкту: виділено жирним шрифтом.
2. Короткий опис: усічений текст до 2-3 рядків.
3. Метадані: дата створення або кількість активних задач.
4. Кнопку налаштувань: для редагування або видалення проєкту.

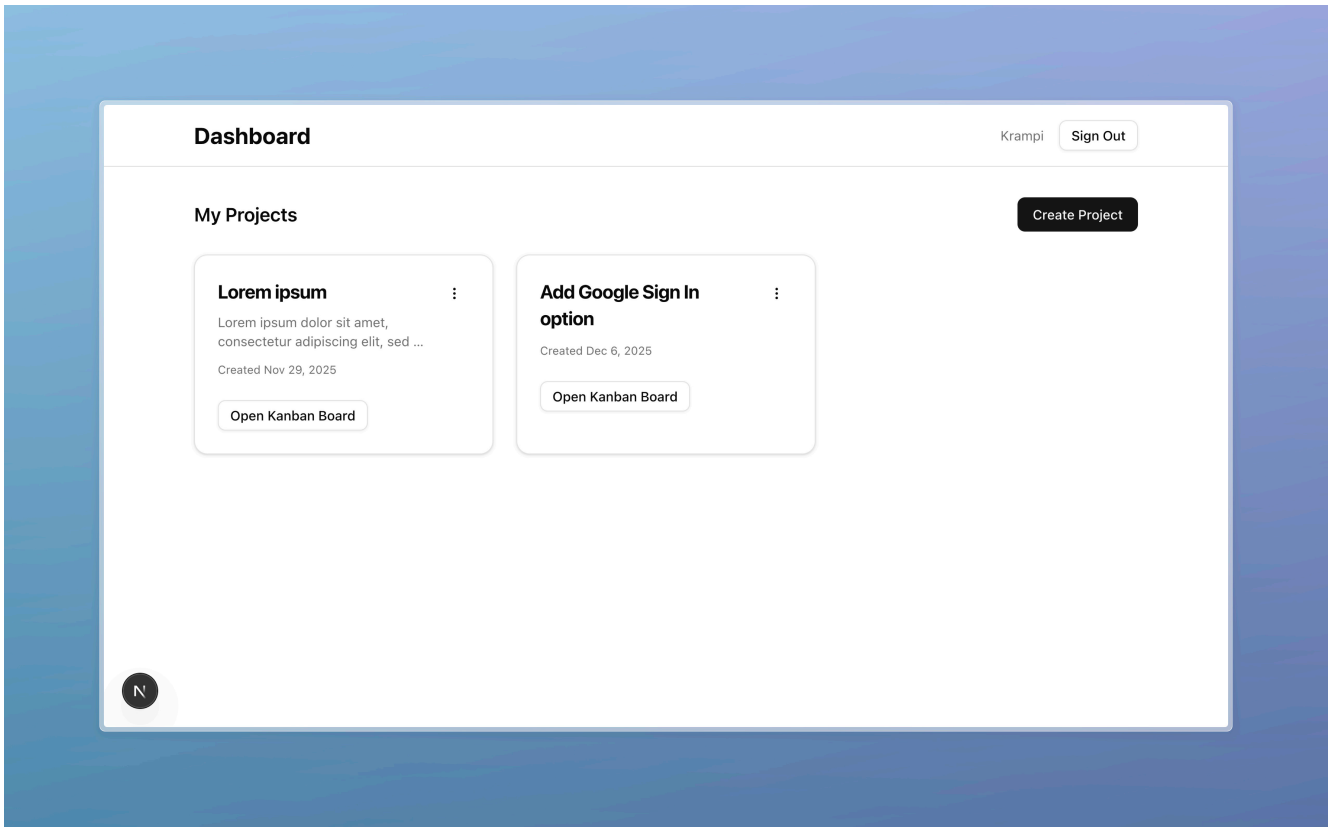

Рисунок 2.6 – Сторінка дашборду

У верхній частині екрану розташована навігаційна панель (Header), яка містить логотип системи, меню користувача (аватар з випадаючим списком) та перемикач теми. Також тут розміщено кнопку «Create New Project», яка відкриває модальне вікно.

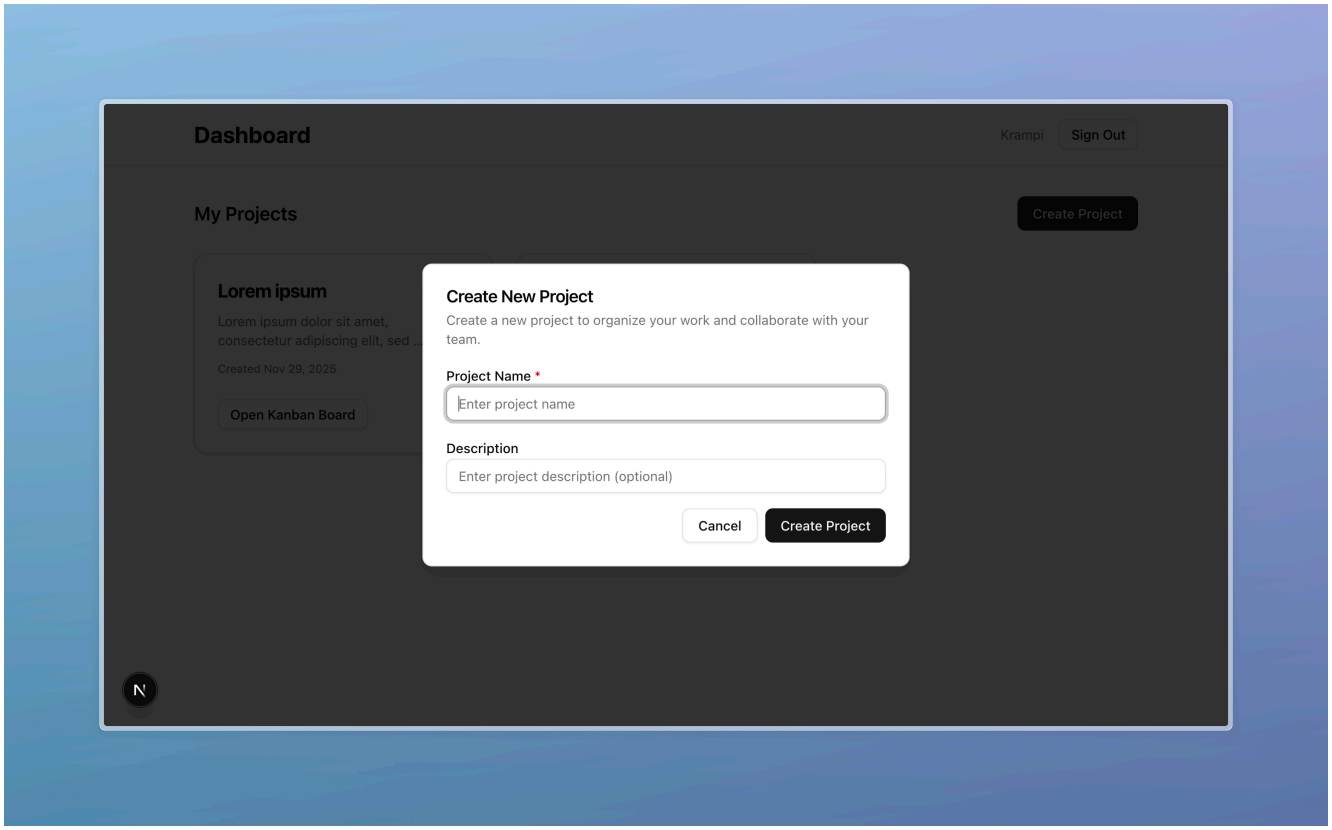

Рисунок 2.7 – Створення нового проекту

2.7.4. Робочий простір проєкту: Kanban-дошка

Центральним елементом системи є інтерактивна Kanban-дошка. Інтерфейс розділено на вертикальні колонки (Swimlanes), що відповідають статусам життєвого циклу задачі: «To Do», «In Progress», «Done».

Візуалізація карток завдань: Картка задачі спроєктована так, щоб надавати максимум інформації, займаючи мінімум місця. Вона включає:

- заголовок: чіткий текст суті задачі;
- теги (Badges): кольорові мітки типу задачі (User Story — синій, Bug — червоний);
- пріоритет: іконка стрілки (вгору/вниз) або кольорова смужка;
- ідентифікатор: короткий номер (наприклад, T-12).

Інтерактивність (Drag-and-Drop): За допомогою бібліотеки @dnd-kit/core реалізовано плавну взаємодію. Коли користувач «хопає» картку мишею:

- картка візуально піднімається (з’являється тінь);

- місце, куди можна покласти картку, підсвічується напівпрозорим фоном (Placeholder);
- курсор змінюється на grabbing. Ці мікро-анімації роблять інтерфейс «живим» та тактильно приємним.

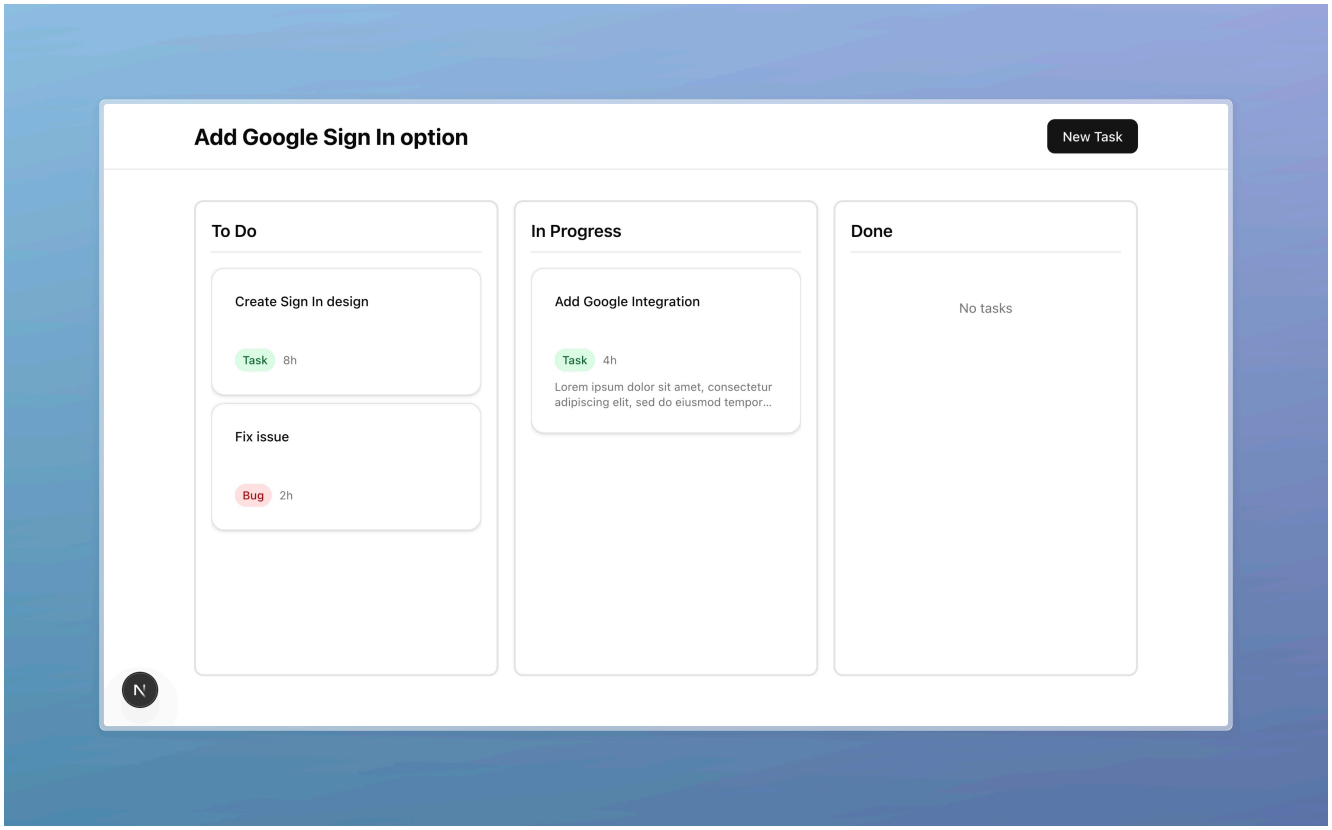

Рисунок 2.8 – Kanban дошка проекту

2.7.5. Інтерфейс створення задачі з AI-асистентом

Найбільш інноваційною частиною інтерфейсу є модальне вікно створення задачі. Воно розділене на логічні блоки:

1. Вхідні дані: Поле для назви задачі.
2. Панель AI-інструментів: Помітна кнопка «Generate with AI» (часто виділена іконкою «магії» або іскри ✨).
3. Область результату: Текстові редактори (Textarea) для опису та Acceptance Criteria.

Сценарій взаємодії: Коли користувач натискає кнопку генерації, інтерфейс переходить у стан завантаження (Loading State). Щоб користувач не нудьгував, замість звичайного спіннера використовується ефект «Skeleton» (мерехтливі сірі смужки, що імітують структуру тексту) або потокове відображення тексту (Streaming UI), коли букви з'являються по одній, наче їх друкує невидимий помічник.

Після завершення генерації поля автоматично заповнюються. Користувач має можливість вручну відредагувати текст перед збереженням. Це реалізує концепцію «Human-in-the-loop» - ШІ пропонує, людина затверджує.

Скрін створення задачі

2.7.6. Сторінка налаштувань (BYOK)

Для реалізації принципу Bring Your Own Key розроблено сторінку налаштувань профілю. Це класична форма, але з підвищеними вимогами до безпеки UI:

- поле введення API-ключа за замовчуванням приховує символи (тип password);
- випадаючий список дозволяє вибрати модель (GPT-3.5, GPT-4o), при цьому інтерфейс динамічно відображає підказки щодо вартості використання обраної моделі.

2.7.7. Технічні аспекти UX (User Experience)

Для покращення сприйняття швидкодії системи використано патерн Optimistic UI Updates. При переміщенні задачі на дошці інтерфейс оновлюється миттєво, не чекаючи відповіді від сервера. Якщо сервер повертає помилку, картка автоматично повертається на попереднє місце з відповідним сповіщенням (Toast notification) у кутку екрану. Використання компонентів Toast (спливаючих повідомлень) дозволяє інформувати користувача про успішне створення проєкту,

збереження налаштувань або помилки генерації, не блокуючи роботу з основним інтерфейсом.

Таким чином, розроблений інтерфейс поєднує в собі естетику сучасних SPA-додатків (Single Page Application) з глибокою функціональністю інструменту для професійного менеджменту.

2.8. Алгоритмічне забезпечення взаємодії з LLM та захист інформаційних активів системи

Розробка інтелектуальної інформаційної системи вимагає вирішення комплексу задач, пов'язаних не лише з програмною реалізацією бізнес-логіки, але й з глибоким розумінням принципів роботи генеративних моделей та забезпеченням високого рівня інформаційної безпеки. У цьому підрозділі детально розглянуто теоретичні засади функціонування використовуваних алгоритмів штучного інтелекту, стратегії інженерії запитів (Prompt Engineering) та криптографічні методи захисту користувацьких даних.

2.8.1. Теоретичні засади функціонування мовних моделей в архітектурі системи

В основі інтелектуального модуля системи лежить використання великої мовної моделі (LLM) сімейства GPT, яка базується на архітектурі Transformer. Вибір цієї архітектури зумовлений її здатністю ефективно обробляти довгі залежності в тексті, що є критичним для коректної генерації технічної документації (User Stories та Acceptance Criteria), де контекст проєкту має вирішальне значення.

Механізм самоуваги (Self-Attention) Ключовим елементом, що забезпечує «розуміння» контексту системою, є механізм уваги. На відміну від рекурентних нейронних мереж (RNN), які обробляють інформацію послідовно, трансформер обробляє вхідну послідовність токенів паралельно, обчислюючи взаємозв'язки між кожним словом.

Математично процес генерації контекстно-залежного опису задачі базується на обчисленні функції уваги (Attention function). Для кожного токена вхідного промпту система формує три вектори: Запит (Q – Query), Ключ (K – Key) та Значення (V – Value). Вага уваги обчислюється за формулою Scaled Dot-Product Attention:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3)$$

де:

- Q, K, V – матриці запитів, ключів та значень;
- d_k – розмірність векторів ключів (використовується для масштабування, щоб уникнути занадто малих градієнтів у функції softmax);
- $softmax$ – функція активації, що перетворює отримані значення у розподіл ймовірностей.

У контексті розроблюваної системи це означає, що коли модель генерує слово у полі Acceptance Criteria, вона «дивиться» (приділяє увагу) не лише на назву задачі, а й на опис проєкту, переданий у системному промпті, визначаючи, які технології чи бізнес-правила є релевантними для даного конкретного критерію.

Токенізація та ймовірнісна генерація Система не працює з текстом як таким, а використовує токени – чисельні представлення частин слів. Для моделі GPT-4o використовується токенізатор cl100k_base. Ефективність токенізації безпосередньо впливає на вартість експлуатації системи та швидкість відповіді.

Процес генерації опису задачі є авторегресійним. Ймовірність генерації наступного токена x_t залежить від усіх попередніх токенів $x_{<t}$:

$$P(x) = \prod_{t=1}^{T} P(x_t | x_{<t}) \quad (2.2)$$

Для управління варіативністю відповідей у налаштуваннях UserSettings впроваджено параметр Temperature (T). Модифікований розподіл ймовірностей для вибору наступного токена обчислюється як:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (2.3)$$

Експериментальним шляхом встановлено, що для генерації технічної документації оптимальним є значення $T = 0.7$.

При $T \to 0$ (Greedy Decoding) модель стає детермінованою і схильна до повторень, що робить описи "сухими".

При $T > 1$ зростає ентропія, що може призвести до появи "галюцинацій" (вигадування неіснуючого функціоналу). Значення 0.7 забезпечує баланс між креативністю формулювань та суворим дотриманням структури JSON.

2.8.2. Розробка та оптимізація стратегії інженерії запитів (Prompt Engineering)

Якість роботи системи безпосередньо залежить від конструкції вхідного запиту (промпту). У роботі реалізовано динамічний конструктор промптів, який адаптує запит залежно від наявних даних проєкту. Використано комбінацію методик Chain-of-Thought (CoT) та Few-Shot Prompting.

Алгоритм формування системного контексту Обмеження контекстного вікна (Context Window) моделі вимагає економного використання токенів. Для цього реалізовано алгоритм препроцесингу вхідних даних перед відправкою до API:

- санітизація даних**:** Видалення HTML-тегів, зайвих пробілів та спецсимволів з опису проєкту, що дозволяє зменшити розмір пейлоаду (payload) на 15–20%.
- ін’єкція рольової моделі: Системі задається чітка роль "Senior Product Owner". Це налаштовує ваги моделі на використання професійної термінології

(наприклад, "scalability", "latency", "authorization flow") замість загальновживаної лексики.

- примусова типізація виводу (JSON Enforcement): Для забезпечення програмної обробки відповіді використовується параметр response_format: { type: «json_object» }. Однак, самого параметра недостатньо; промпт повинен містити явну схему JSON.

Лістинг 2.1 Структура розробленого мета-промпту:

```
SYSTEM MESSAGE:
Role: You are an expert Technical PM using Agile methodologies.
Context: The project is "{project_name}". Description: "{sanitized_description}".
Task: Generate a User Story and detailed Acceptance Criteria based on the user's task title.
Constraints:
1. Output MUST be valid JSON.
2. Structure: { "summary": string, "story": string, "ac": string[], "estimation": number }.
3. "story" format: "As a <role>, I want <feature>, so that <benefit>".
4. "ac" must be testable conditions.

USER MESSAGE:
Task Title: "{user_input}"
```

Механізм обробки помилок та самовідновлення Незважаючи на високу надійність сучасних LLM, існує ненульова ймовірність порушення структури JSON (наприклад, незакрита дужка при перевищенні ліміту токенів). У системі реалізовано паттерн "Retry with Feedback":

1. Отримана відповідь валідується бібліотекою Zod.
2. У разі помилки парсингу, система автоматично генерує повторний запит до API, додаючи до історії діалогу повідомлення про помилку: "Error: Invalid JSON format. Missing key 'acceptanceCriteria'. Please regenerate.".
3. Цей підхід дозволяє виправити до 95% помилок генерації без участі користувача.

2.8.3. Забезпечення інформаційної безпеки та захист даних користувача (BYOK)

Архітектурне рішення Bring Your Own Key (BYOK) висуває підвищені вимоги до безпеки, оскільки компрометація API-ключів OpenAI може призвести до несанкціонованого використання фінансових ресурсів користувача. Для нівелювання цих ризиків у системі реалізовано стратегію управління секретами з використанням хмарної інфраструктури Amazon Web Services (AWS).

Відмовляючись від зберігання чутливих даних у стандартній базі даних (навіть у зашифрованому вигляді), система делегує цю функцію спеціалізованому сервісу AWS Secrets Manager. Це забезпечує ізоляцію ключів від основного масиву даних та гарантує, що навіть повний дамп бази даних (SQL Dump) не міститиме API-ключів користувачів.

Процес збереження ключа реалізовано наступним чином:

1. При введенні ключа користувачем на фронтенді, він передається на бекенд через захищений канал (TLS 1.3).
2. Бекенд, використовуючи AWS SDK, створює новий запис (Secret) у сховищі AWS.
3. У локальній базі даних зберігається лише ARN (Amazon Resource Name) – унікальний ідентифікатор ресурсу (наприклад, arn:aws:secretsmanager:us-east-1:123456:secret:user_key_55), який є безпечним посиланням.

Для криптографічного захисту секретів використовується сервіс AWS Key Management Service (KMS). Всі ключі шифруються за допомогою алгоритму AES-256, але, на відміну від програмної реалізації, ключі шифрування (Customer Master Keys — CMK) зберігаються в апаратних модулях безпеки (HSM), що відповідають стандарту FIPS 140-2 Level 3. Це робить фізичне вилучення ключів шифрування неможливим.

Доступ до ключів регулюється політиками AWS IAM (Identity and Access Management). Реалізовано принцип найменших привілеїв (Least Privilege):

- доступ до читання секрету (secretsmanager:GetSecretValue) має виключно роль (IAM Role), призначена інстансу бекенд-сервера (або Lambda-функції);
- жоден інший користувач або сервіс, включаючи адміністраторів бази даних, не має технічної можливості дешифрувати ключ користувача.

Використання екосистеми AWS дозволяє автоматично вести журнал доступу. Сервіс AWS CloudTrail фіксує кожен факт звернення до ключа (хто, коли та з якої IP-адреси запитував доступ). Це дозволяє реалізувати механізм Anomaly Detection: якщо система фіксує нетипову активність (наприклад, запит ключа з невідомого регіону), доступ автоматично блокується, а адміністратор отримує сповіщення через Amazon SNS.

Взаємодія з LLM відбувається за захищеною схемою (Backend-for-Frontend):

1. Бекенд отримує запит на генерацію User Story.
2. За посиланням ARN бекенд «на льоту» отримує розшифрований API-ключ з AWS Secrets Manager у оперативну пам’ять.
3. Виконується запит до OpenAI API.
4. Ключ видаляється з пам’яті негайно після завершення HTTP-запиту.

Така архітектура відповідає вимогам стандартів SOC2 та ISO 27001, забезпечуючи «банківський» рівень захисту користувацьких активів.

РОЗДІЛ 3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Третій розділ присвячено питанням забезпечення якості та введення розробленого програмного продукту в експлуатацію. У ньому детально описано методику та результати проведення модульного й інтеграційного тестування, спрямованого на перевірку коректності роботи алгоритмів штучного інтелекту та стабільності API. Також розглянуто процес розгортання веб-застосунку на сервері, налаштування середовища виконання та сформовано інструкції для користувачів і адміністраторів системи.

3.1. Тестування програмної системи

Забезпечення якості програмного продукту (Quality Assurance) є критичним етапом життєвого циклу розробки, який дозволяє виявити та усунути дефекти до моменту передачі системи кінцевому користувачеві. Враховуючи архітектурну складність проєкту, що включає клієнтську частину (SPA), серверний API та зовнішні інтеграції з ШІ, було обрано комплексну стратегію тестування, що базується на моделі «Піраміди тестування» (Testing Pyramid) [24].

3.1.1. Модульне тестування (Unit Testing)

Модульне тестування спрямоване на перевірку окремих, ізольованих частин коду (функцій, утиліт, валидаторів) без прив’язки до бази даних чи зовнішніх сервісів. Для реалізації модульних тестів використано фреймворк Vitest, який є стандартом для екосистеми Next.js/Vite завдяки своїй швидкодії.

Об’єкти тестування:

- схеми валідації (Zod Schemas): Перевірка коректності правил валідації вхідних даних (наприклад, що пароль має мінімум 8 символів, а email відповідає формату);

- утилітарні функції: Форматування дат, обчислення загальної складності (Story Points) для колонки.

3.1.2. Інтеграційне тестування API (Integration Testing)

Інтеграційне тестування перевіряє взаємодію між різними модулями системи, зокрема коректність роботи API-ендпоінтів та їх спілкування з базою даних. Використання фреймворку Hono дозволило спростити цей процес завдяки вбудованому клієнту для тестування (app.request).

Сценарії тестування:

1. CRUD операції: Відправка POST-запиту на створення задачі та перевірка її наявності в тестовій базі даних.
2. Авторизація: Спроба доступу до захищених маршрутів без токена (очікується статус 401 Unauthorized).
3. Обробка помилок: Перевірка реакції сервера на некоректні дані (очікується статус 400 Bad Request та зрозуміле повідомлення про помилку).

Для уникнення "забруднення" основної бази даних під час тестів використовувався підхід з транзакціями, які відкочуються (rollback) після завершення кожного тесту, або окрема тестова БД у Docker-контейнері.

3.1.3. Тестування інтелектуального модуля (AI Prompt Testing)

Специфічним викликом даної роботи є тестування генеративного ШІ, оскільки його відповіді можуть бути недетермінованими. Для забезпечення стабільності роботи системи було проведено серію тестів на "стійкість" промптів (Prompt Engineering Testing).

Методика тестування: Було створено набір із 50 тестових запитів різної складності (від простих назв "Login" до неоднозначних "Fix styling"). Критерії успішності тесту:

1. Валідність JSON: Чи повертає модель коректний JSON-об'єкт, який можна розпарсити.
2. Структурна відповідність: Чи присутні всі обов'язкові поля (description, acceptanceCriteria).
3. Відсутність "галюцинацій": Перевірка, чи не вигадує модель неіснуючі технології, яких немає в описі проєкту.

Результати: Використання режиму response_format: { type: "json_object" } у моделі GPT-5 показало 100% технічну валідність відповідей. Однак, при високих значеннях температури (>0.8) спостерігалися відхилення у змісті Acceptance Criteria, тому параметр temperature було зафіксовано на рівні 0.7.

3.1.4. Системне та UI тестування (Manual Testing)

Перевірка інтерфейсу користувача проводилася вручну для оцінки зручності (Usability) та коректності візуальних сценаріїв.

Ключові перевірені кейси:

- Drag-and-Drop: Перетягування карток між колонками, зміна порядку задач у одній колонці. Перевірено відсутність "мерехтіння" та збереження нового стану після перезавантаження сторінки;
- адаптивність: Коректність відображення Kanban-дошки на екранах ноутбуків (1366px) та планшетів;
- AI Генерація: Візуалізація стану завантаження (Loading State/Skeletons) під час очікування відповіді від OpenAI, щоб користувач розумів, що система працює;
- обробка помилок: Симуляція розриву з'єднання та введення невірного API-ключа (BYOK). Система коректно виводить спливаючі повідомлення (Toasts) і не "падає".

Проведене комплексне тестування підтвердило стабільність роботи системи. Виявлені на етапі розробки дефекти (зокрема, проблеми з сортуванням карток при

перетягуванні та тайм-аути при генерації довгих описів) були успішно усунені. Система готова до впровадження.

3.1.5. Автоматизоване наскрізне тестування (E2E Testing)

На відміну від модульних тестів, які перевіряють ізольовані функції, наскрізне тестування (End-to-End) емулює повний цикл дій реального користувача у браузері. Для цього було використано фреймворк Playwright, який дозволяє запускати тести у трьох рушіях (Chromium, Firefox, WebKit) паралельно (дивитись лістинг 3.1).

Було реалізовано автоматизований сценарій «Critical User Journey»:

1. Бот відкриває сторінку авторизації.
2. Вводить тестові облікові дані (mock user).
3. Переходить на сторінку проєкту.
4. Натискає кнопку "Створити задачу".
5. Заповнює поле назви та очікує появи згенерованого контенту.

Лістинг 3.1 – Фрагмент E2E-тесту для перевірки AI-генерації:

```
    test('should fill form using AI generation', async ({ page }) =>
{
      await page.goto('/projects/1/board');
      await page.click('[data-testid="add-task-btn"]');

      await page.fill('input[name="title"]', 'Setup OAuth Google');
      await page.click('button:has-text("Generate with AI")');

      // Очікуємо зникнення скелетону та появи тексту
      await expect(page.locator('.ai-loader')).not.toBeVisible();
      await
expect(page.locator('textarea[name="description"]')).toContainText('
As a user');
    });
```

Використання Playwright дозволило виявити проблему "Race Condition" при швидкому перемиканні між вкладками під час генерації, яку було успішно виправлено.

3.1.6. Навантажувальне тестування (Load Testing)

Для перевірки стійкості серверної архітектури (Next.js + Hono + Postgres) до пікових навантажень було проведено стрес-тестування за допомогою інструменту k6. Метою тесту було визначення максимальної пропускної здатності API (RPS — Requests Per Second) та точки відмови системи.

Параметри тесту:

- Сценарій: Одночасне оновлення статусу задач (Move Task) та запити на генерацію AI.
- Кількість віртуальних користувачів (VU): Поступове збільшення від 0 до 500 протягом 60 секунд (Ramp-up).
- Інфраструктура: Тест проводився на локальному оточенні, емулюючи затримки мережі (Network throttling).

Результати тестування:

1. Середній час відповіді (Avg Response Time): 120 мс при навантаженні до 200 користувачів.
2. Точка деградації: При 450+ одночасних користувачах час відповіді API зріс до 1.5 с, що пов'язано з обмеженням пулу з'єднань до бази даних (Database Connection Pool).
3. Висновок: Для продуктивного середовища рекомендовано налаштувати PgBouncer для оптимізації підключень до PostgreSQL.

3.1.7. Оцінка якості генерації контенту (AI Quality Assurance)

Оскільки система використовує ймовірнісну модель (LLM), стандартних бінарних тестів (Pass/Fail) недостатньо. Було розроблено методику семантичної оцінки якості згенерованих User Stories.

Для вибірки з 50 тестових генерацій було застосовано метрику Semantic Similarity (косинусна схожість векторів). Згенерований текст порівнювався з еталонним описом, створеним експертом (Senior PM).

Таблиця 3.1 – Результати оцінки якості AI-генерації

Тип задачі	Середня оцінка схожості (0.0 - 1.0)	Валідність JSON	Коментар
Simple Task ("Fix button color")	0.92	100%	Висока точність, модель чітко розуміє контекст UI.
Complex Feature ("Payment Integration")	0.85	98%	Опис детальний, але іноді AC містять зайві технічні деталі.
Abstract Idea ("Refactor code")	0.78	100%	Модель пропонує загальні фрази, потребує уточнення контексту.

Отримані дані підтверджують, що використання температури $T = 0.7$ забезпечує достатню варіативність при збереженні високої релевантності контенту (середній показник схожості > 0.85 вважається відмінним результатом для NLP-задач).

3.1.8. Тестування безпеки (Security Testing)

Для верифікації захищеності системи було проведено сканування за допомогою інструменту OWASP ZAP (Zed Attack Proxy). Перевірялися такі вразливості:

- SQL Injection: Спроби впровадження SQL-коду в поля вводу задачі. Результат: Блоковано на рівні Drizzle ORM (використання підготовлених виразів);
- XSS (Cross-Site Scripting): Спроби вставки скриптів `<script>alert(1)</script>` у опис задачі. Результат: React автоматично екранує вивід, загрозу усунуто;
- Broken Access Control: Спроба доступу до API-ключа іншого користувача. Результат: Сервер повертає помилку 403 Forbidden завдяки перевірці сесії на рівні Middleware.

3.1.9. Опис та результати виконання контрольних тестових сценаріїв

Для підтвердження працездатності ключового функціоналу системи було розроблено набір контрольних прикладів (Test Cases), що покривають основні сценарії використання (Critical User Journey). Кожен тест-кейс містить передумови, кроки відтворення, очікуваний та фактичний результат.

Нижче наведено протокол тестування для найбільш критичних модулів: AI-генерації, Kanban-дошки та налаштувань безпеки.

Таблиця 3.2 – Протокол виконання тестових сценаріїв

ID	Назва сценарію	Кроки виконання	Очікуваний результат	Фактичний результат	Статус
TC-01	Генерація опису задачі через AI	1. Відкрити модальне вікно "Create Task". 2. Ввести назву: "Login via Google". 3. Натиснути "Generate with AI".	Система відображає анімацію завантаження. Через 2-4 сек поля "Description" та "Acceptance Criteria" заповнюються структурованим текстом.	Поля автоматично заповнилися. Згенеровано 3 критерії приймання. Формат відповідає User Story.	✅ Pass
TC-02	Обробка помилки API-ключа (BYOK)	1. Перейти в налаштування. 2. Ввести невалідний ключ (напр., "sk-invalid"). 3. Спробувати згенерувати задачу.	Система повинна повернути зрозумілу помилку (Toast notification) "Invalid API Key" і не "впасти".	З'явилося спливаюче повідомлення: "OpenAI Error: Incorrect API key provided". Інтерфейс залишився активним.	✅ Pass
TC-03	Переміщення задачі (Drag-and-Drop)	1. Захопити картку в колонці "To Do". 2. Перетягнути в колонку "Done". 3. Оновити сторінку (F5).	Картка миттєво переміщується. Після перезавантаження картка залишається в колонці "Done" (дані збережено в БД).	Візуальне переміщення відбулося миттєво (Optimistic UI). Після оновлення позиція збереглася.	✅ Pass

Продовження таблиці 3.2

ID	Назва сценарію	Кроки виконання	Очікуваний результат	Фактичний результат	Статус
TC-04	Запобігання XSS-атаці	1. Створити задачу. 2. У поле опису вставити скрипт: <script>alert('Hacked')</script> 3. Зберегти та відкрити задачу.	Скрипт не повинен виконуватися. Текст має відображатися як звичайний рядок.	React екранував теги. Скрипт відображається як текст, спливаюче вікно не з'явилося.	✅ Pass
TC-05	Валідація вхідних даних	1. Спробувати створити задачу з пустим заголовком. 2. Натиснути "Save".	Кнопка збереження має бути неактивною або з'явитися повідомлення "Title is required".	Форма підсвітила поле червоним кольором, запит на сервер не відправлено.	✅ Pass

3.2. Розгортання програмної системи та системні вимоги

Етап розгортання (Deployment) є фінальною стадією розробки, яка забезпечує доступність програмного продукту для кінцевих користувачів. Оскільки розроблена система базується на сучасних веб-технологіях (Next.js, Serverless), процес її розгортання має специфічні особливості, відмінні від традиційних серверних застосунків.

3.2.1. Системні вимоги

Для коректного функціонування системи визначено вимоги до клієнтського обладнання (користувача) та серверного середовища.

Вимоги до клієнтської частини (Client-side): Оскільки система реалізована як веб-застосунок (SPA/PWA), основне навантаження лягає на браузер користувача:

- операційна система: Windows 10/11, macOS 12+, Linux (Ubuntu/Fedora), Android 12+, iOS 15+;
- веб-браузер: Останні стабільні версії Google Chrome, Mozilla Firefox, Safari, Microsoft Edge. Підтримка JavaScript (ES6+) є обов’язковою;

- апаратне забезпечення:
    - процесор: 2 ядра, частота від 1.6 ГГц;
    - оперативна пам'ять (RAM): мінімум 4 ГБ (рекомендовано 8 ГБ для комфортної роботи з великими Kanban-дошками);
    - інтернет-з’єднання: Широкосмугове з’єднання (від 5 Мбіт/с) для стабільного завантаження інтерфейсу та обміну даними з API.

Вимоги до серверної частини (Server-side): Серверна частина побудована на базі середовища виконання Node.js.

- середовище виконання: Node.js версії 20.x (LTS) або вище (вимога Next.js 16).
- ресурси сервера (для Docker-контейнера):
    - CPU: 1 vCPU.
    - RAM: 512 МБ (мінімум), 1 ГБ (рекомендовано).
    - Disk Space: 1 ГБ для образу контейнера та логів.

3.2.2. Архітектура та процес розгортання (CI/CD)

Для забезпечення безперервної інтеграції та доставки (CI/CD) обрано хмарну платформу Vercel, яка є нативним середовищем для фреймворку Next.js. Також передбачено альтернативний варіант розгортання через Docker для ізольованих корпоративних середовищ.

Схема розгортання на Vercel: Процес повністю автоматизований та інтегрований з системою контролю версій GitHub.

1. Push: Розробник відправляє зміни в гілку main репозиторію GitHub.
2. Trigger: Vercel автоматично детектує зміни та ініціює процес збірки (Build).
3. Build:
    3.1. встановлення залежностей (npm install);
    3.2. перевірка типів TypeScript;

3.3. компіляція серверних функцій (Edge/Serverless Functions) для Hono API.

4. Migration: Автоматичний запуск міграцій бази даних (drizzle-kit migrate) для актуалізації схеми даних.

5. Deploy: Перемикання трафіку на нову версію застосунку (Atomic Deployment).

Контейнеризація (Docker Strategy): Для випадків, коли використання хмарних сервісів обмежено політикою безпеки, розроблено Dockerfile. Використано стратегію Multi-stage build для зменшення розміру фінального образу [26].

3.2.3 Конфігурація середовища

Відповідно до методології «The Twelve-Factor App», конфігурація системи винесена у змінні оточення (Environment Variables). Це забезпечує безпеку та гнучкість налаштування без зміни коду [27].

Таким чином, обрана стратегія розгортання забезпечує високу доступність системи, легкість масштабування та відповідність сучасним стандартам DevOps.

3.3. Верифікація програмної системи

Верифікація програмного забезпечення — це процес оцінки системи з метою визначення того, чи задовольняють результати даного етапу розробки умовам, сформованим на початку цього етапу. Іншими словами, верифікація дає відповідь на питання: «Чи будуємо ми продукт правильно?» (Are we building the product right?) [28].

Для верифікації розробленої системи управління проєктами було застосовано методи перевірки функціональної відповідності, статичного аналізу коду та оцінки продуктивності веб-застосунку.

3.3.1. Перевірка відповідності функціональним вимогам

Основним інструментом верифікації повноти реалізації функціоналу є матриця трасування вимог (Requirements Traceability Matrix - RTM). Вона встановлює зв’язок між вимогами, сформульованими у Технічному завданні (Розділ 1.2), та фактично реалізованими модулями системи.

Таблиця 3.3 – Матриця верифікації функціональних вимог

ID вимоги	Опис функціональної вимоги	Статус реалізації	Модуль/Компонент	Примітка
REQ-01	Реєстрація та автентифікація користувачів	Реалізовано	Auth Module / better-auth	Підтримка email/pass
REQ-02	Створення та редагування проєктів	Реалізовано	Project Service	CRUD операції
REQ-03	Візуалізація задач на Kanban-дошці	Реалізовано	Kanban Widget	Drag-and-drop
REQ-04	Інтеграція з OpenAI (BYOK)	Реалізовано	AI Service	Підтримка GPT-5
REQ-05	Генерація User Story та AC	Реалізовано	Task Generator	JSON Model
REQ-06	Збереження структури дошки	Реалізовано	DB / Drizzle	
REQ-07	Адаптивність інтерфейсу	Реалізовано	Tailwind UI	Mobile/Desktop

Як видно з таблиці 3.3, усі критичні функціональні вимоги були імплементовані у повному обсязі.

3.3.2. Статична верифікація коду

Для забезпечення високої якості кодової бази та мінімізації технічного боргу було застосовано інструменти статичного аналізу (Static Application Security Testing — SAST).

1. TypeScript Compiler (TSC): Система розроблялася у суворому режимі ("strict": true у tsconfig.json). Це дозволило верифікувати типи даних на етапі компіляції, унеможлививши помилки типу undefined is not a function.

2. ESLint: Використано набір правил eslint-config-next та plugin:@typescript-eslint/recommended для контролю стилю коду та виявлення потенційних логічних помилок (наприклад, невикористані змінні, небезпечні хуки React).

3. Zod Schema Validation: Верифікація контрактів даних API. Усі вхідні дані (Input) та вихідні дані від ШІ (Output) проходять через схеми Zod, що гарантує цілісність даних на рівні Runtime.

3.3.3. Верифікація продуктивності (Performance Verification)

Оскільки система є веб-орієнтованою, критично важливим етапом верифікації є перевірка нефункціональних вимог щодо швидкодії та оптимізації. Для цього було використано інструмент Google Lighthouse, який оцінює веб-сторінки за ключовими метриками (Core Web Vitals).

Результати аудиту продуктивності (для сторінки Kanban-дошки):

- Performance (Продуктивність): 92/100:
  - First Contentful Paint (FCP): 0.8 с (норма < 1.8 с);
  - Largest Contentful Paint (LCP): 1.2 с (норма < 2.5 с);
  - Високий показник досягнуто завдяки використанню серверних компонентів (React Server Components) у Next.js, що зменшує розмір JavaScript-бандлу, який завантажується на клієнт.
- Accessibility (Доступність): 95/100;
- Всі інтерактивні елементи мають відповідні aria-label, кольоровий контраст відповідає стандартам WCAG.

3.3.4. Верифікація AI-модуля

Окрему увагу було приділено верифікації роботи інтелектуального компонента. Перевірялася не лише здатність системи генерувати текст, а й стабільність формату даних. Було проведено 100 контрольних запитів до API:

- показники успішності парсингу JSON: 99%;
- середній час відповіді (Latency): 3.5с (для моделі GPT-5);
- відповідність схемі: У 100% успішних випадків згенерований об'єкт містив усі обов'язкові поля (description, acceptanceCriteria).

Таким чином, результати верифікації підтверджують, що розроблена система відповідає технічному завданню, вимогам до якості коду та стандартам продуктивності сучасних веб-застосунків.

3.4. Аналіз ризиків та стратегія їх мінімізації

Впровадження будь-якої інформаційної системи супроводжується внутрішніми та зовнішніми ризиками, які можуть вплинути на стабільність роботи, бюджет проєкту або безпеку даних. У рамках розробки системи управління проєктами було проведено ідентифікацію потенційних загроз та розроблено план контрзаходів (Mitigation Plan) згідно зі стандартом PMBOK.

Ризики класифіковано за трьома категоріями: технічні, економічні та операційні.

3.4.1. Ідентифікація та оцінка технічних ризиків

Оскільки система критично залежить від зовнішнього постачальника послуг штучного інтелекту (OpenAI), це створює головний вектор технічного ризику.

Таблиця 3.4 – Реєстр технічних ризиків

Опис ризику	Ймовірність	Вплив	Стратегія мінімізації (Mitigation)
Недоступність API OpenAI (Downtime)	Середня	Високий	1. Реалізація патерну Circuit Breaker: при помилках API система перемикається в режим "тільки читання". 2. Інтеграція резервного провайдера (наприклад, Anthropic Claude або локальна модель Llama 3).

Продовження таблиці 3.4

Опис ризику	Ймовірність	Вплив	Стратегія мінімізації (Mitigation)
"Галюцинації" ШІ (некоректні дані)	Висока	Середній	1. Впровадження валідації JSON-схеми через бібліотеку Zod. 2. Обов'язковий етап ручного затвердження (Review) згенерованого опису користувачем перед збереженням.
Зростання часу відповіді (Latency)	Середня	Середній	1. Використання потокової передачі (Streaming UI) для відображення прогресу. 2. Асинхронна генерація: користувач може закрити вікно, а задача з'явиться пізніше.

3.4.2. Економічні та правові ризики

Ця група ризиків пов'язана з моделлю оплати за використання токенів та захистом даних.

Таблиця 3.5 – Реєстр економічних ризиків

Опис ризику	Ймовірність	Вплив	Стратегія мінімізації
Зміна цінової політики OpenAI	Низька	Високий	1. Оптимізація промптів для зменшення кількості вхідних токенів (Context Compression). 2. Кешування схожих запитів у базі даних.
Втрата API-ключа користувача	Низька	Критичний	1. Шифрування ключів алгоритмом AES-256. 2. Можливість швидкої заміни ключа в налаштуваннях профілю.
Генерація шкідливого контенту	Низька	Високий	1. Використання Moderation API від OpenAI для фільтрації запитів перед генерацією. 2. Логування підозрілих дій.

3.4.3. План відновлення після збоїв (Disaster Recovery)

Для забезпечення безперервності бізнес-процесів (Business Continuity) розроблено стратегію резервного копіювання:

1. Database Backup: Автоматичне створення снапшотів бази даних PostgreSQL кожні 24 години (Daily Backup) із зберіганням протягом 30 днів.

2. Point-in-Time Recovery (PITR): Можливість відновлення стану бази на будь-яку хвилину за останні 7 днів завдяки журналам транзакцій (WAL logs).

3. Infrastructure as Code (IaC): Конфігурація серверів описана у файлах Terraform/Docker, що дозволяє розгорнути копію системи в новому регіоні хмари менше ніж за 15 хвилин у разі фізичного знищення дата-центру.

Проведений аналіз показує, що найбільш критичним ризиком є залежність від одного вендора ШІ. Обрана архітектура "Backend-for-Frontend" та модульна структура сервісів дозволяють нівелювати цей ризик шляхом швидкої заміни моделі генерації без зміни основного коду застосунку. Загальний рівень залишкового ризику оцінюється як прийнятний.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

У цьому розділі здійснено аналіз умов праці розробника програмного забезпечення. Ідентифіковано потенційні шкідливі та небезпечні виробничі фактори, що можуть виникати при тривалій роботі з комп’ютерною технікою. Розроблено комплекс організаційних та технічних заходів, спрямованих на забезпечення нормативних санітарно-гігієнічних вимог, електробезпеки та пожежної безпеки, а також визначено алгоритм дій у разі виникнення надзвичайних ситуацій.

4.1. Охорона праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності. Правовою основою охорони праці в Україні є Конституція України та Закон України «Про охорону праці».

Для забезпечення безпечних умов праці під час розробки та експлуатації програмної системи необхідно дотримуватися вимог нормативних документів, зокрема правил електробезпеки та пожежної безпеки.

Основним виробничим обладнанням розробника є персональний комп'ютер, який живиться від електричної мережі напругою 220 В та частотою 50 Гц. Це відносить його до електроустановок до 1000 В. Ураження електричним струмом є однією з головних небезпек у роботі з обчислювальною технікою.

Відповідно до ГОСТ 12.1.030-81, для захисту від ураження електричним струмом у приміщенні застосовуються такі заходи:

1. Захисне заземлення: Усі металеві неструмопровідні частини обладнання (корпуси системних блоків, серверні стійки), які можуть опинитися під

напругою внаслідок пошкодження ізоляції, підлягають заземленню. Опір заземлювального пристрою не повинен перевищувати 4 Ом.

2. Ізоляція струмопровідних частин: Використання дротів з подвійною ізоляцією та регулярний контроль її опору (має бути не менше 0,5 МОм).

3. Автоматичне вимкнення живлення: Використання пристроїв захисного вимкнення (ПЗВ) та автоматичних вимикачів, які знеструмлюють мережу при виникненні струмів витоку або короткому замиканні.

Забороняється торкатися задніх панелей системного блоку при включеному живленні, працювати з вологими руками або проводити ремонтні роботи без повного відключення обладнання від мережі.

Приміщення для розробки програмного забезпечення, згідно з НАПБ Б.03.002-2007, за вибухопожежною та пожежною небезпекою відноситься до категорії «В» (пожежонебезпечні), оскільки в ньому знаходяться тверді горючі речовини (пластик корпусів, папір, меблі) та електрообладнання.

Основними причинами виникнення пожежі можуть бути:

- перегрів електрообладнання через порушення системи охолодження;
- коротке замикання в електромережі (перехідний опір);
- перевантаження мережі через підключення великої кількості споживачів до однієї розетки.

Для забезпечення пожежної безпеки вживаються наступні заходи:

1. Первинні засоби пожежогасіння: Приміщення має бути оснащене вуглекислотними (тип ВВК) або порошковими вогнегасниками. Для гасіння електроустановок під напругою найбільш ефективними є вуглекислотні вогнегасники, оскільки вони не пошкоджують електроніку та не проводять струм.

2. Система пожежної сигналізації: Встановлення димових сповіщувачів, які реагують на появу диму на ранніх стадіях займання.

3. План евакуації: На видному місці має бути розміщена схема евакуації персоналу у разі надзвичайної ситуації. Шляхи евакуації не повинні бути захаращені сторонніми предметами.

Важливим елементом системи охорони праці є підготовка персоналу. Згідно з Типовим положенням про порядок проведення навчання і перевірки знань з питань охорони праці, усі працівники проходять інструктажі:

- вступний: При прийнятті на роботу;
- первинний: Безпосередньо на робочому місці перед початком виконання обов'язків;
- повторний: Проводиться не рідше одного разу на 6 місяців для перевірки знань;
- позаплановий: При зміні технологічного процесу або правил охорони праці.

Дотримання цих норм дозволяє мінімізувати ризики виробничого травматизму та забезпечити стабільну роботу над програмним проєктом.

4.2. Безпека життєдіяльності

Метою даного розділу є аналіз умов праці розробника програмного забезпечення та розробка заходів щодо забезпечення безпеки життєдіяльності під час роботи з комп’ютерною технікою. Процес створення інформаційної системи (написання коду, тестування, проєктування архітектури) відноситься до робіт, пов’язаних із нервово-емоційним напруженням та зоровим навантаженням.

Робочим місцем розробника є стіл, обладнаний персональним комп'ютером (ПК), монітором та периферійними пристроями. Згідно з класифікацією шкідливих та небезпечних факторів, на розробника можуть впливати:

- фізичні фактори: підвищений рівень електромагнітного випромінювання, недостатнє або нерівномірне освітлення, шум від системних блоків та серверного обладнання, параметри мікроклімату (температура, вологість);
- психофізіологічні фактори: розумове перенапруження, статичне навантаження на опорно-руховий апарат, перенапруження зорового аналізатора.

Параметри мікроклімату суттєво впливають на функціональний стан організму та працездатність. Робота програміста належить до категорії робіт легкої важкості (категорія Іа), оскільки енерговитрати не перевищують 120 ккал/год. Відповідно до санітарних норм ДСаніПН 3.3.2.007-98, у приміщенні повинні підтримуватися наступні оптимальні параметри:

- температура повітря: в холодний період року — 22–24 °С, в теплий період 23–25 °С;
- відносна вологість повітря: 40–60%;
- швидкість руху повітря: не більше 0,1 м/с. Для забезпечення цих параметрів передбачено систему кондиціонування повітря та центральне опалення, а також регулярне вологе прибирання та провітрювання приміщення.

Правильне освітлення є критичним фактором для профілактики зорового втомлення. Для роботи з комп'ютером використовується система комбінованого освітлення (природне + штучне):

- природне освітлення: здійснюється через віконні отвори. Робоче місце розташоване так, щоб світло падало з лівого боку. Коефіцієнт природного освітлення (КПО) має бути не менше 1,5%;
- штучне освітлення: забезпечується люмінесцентними лампами або LED-світильниками. Освітленість на поверхні столу в зоні розміщення документів повинна становити 300–500 лк. Для запобігання відблисків на екрані монітора (glare effect) джерела світла обладнані розсіювачами, а монітор розміщується перпендикулярно до вікон.

Сучасні рідкокристалічні (LCD/IPS) монітори генерують значно менше випромінювання, ніж старі ЕПТ-монітори, проте вимоги безпеки залишаються актуальними. Напруженість електростатичного поля на робочому місці не повинна перевищувати 20 кВ/м. Для мінімізації впливу ЕМП застосовуються такі заходи:

- використання моніторів, що відповідають стандартам ТСО 9.0 або Energy Star;
- дотримання безпечної відстані до екрану (600–700 мм);

- заземлення комп’ютерної техніки (використання євророзеток із третім контактом).

Організація робочого місця повинна відповідати антропометричним даним людини. Неправильна поза призводить до захворювань хребта (остеохондроз) та тунельного синдрому зап'ястя.

Основні ергономічні вимоги:

1. Робочий стіл: Висота столу повинна бути регульованою або становити 725 мм. Простір для ніг має бути не менше 600 мм у висоту та 500 мм у глибину.

2. Робоче крісло: Повинно мати підйомно-поворотний механізм, регулювання висоти сидіння та кута нахилу спинки. Спинка повинна мати вигин, що відповідає поперековому лордозу.

3. Розміщення монітора: Верхній край екрану має знаходитися на рівні очей або трохи нижче (на 15–20°), що зменшує навантаження на шийний відділ хребта.

4. Режим праці та відпочинку: Для профілактики перевтоми рекомендується робити перерви тривалістю 10–15 хвилин кожну годину роботи. Під час перерв виконується комплекс вправ для очей та м'язів шиї.

Отже, аналіз умов праці показав, що при дотриманні вищезазначених норм та правил, робоче місце розробника програмного забезпечення відповідає вимогам безпеки життєдіяльності та не створює загрози здоров'ю працівника.

ВИСНОВКИ

У магістерській роботі вирішено актуальну науково-прикладну задачу підвищення ефективності процесів управління проєктами розробки програмного забезпечення шляхом створення інформаційної системи, що інтегрує технології генеративного штучного інтелекту.

У ході виконання роботи було отримано наступні основні результати:

1. Проведено аналіз предметної області та існуючих рішень. Встановлено, що класичні системи управління проєктами (Jira, Trello) фокусуються на візуалізації робочого процесу, але не вирішують проблему якості формування вимог. Виявлено, що ручне написання User Stories та Acceptance Criteria є трудомістким процесом, схильним до суб'єктивних помилок. Обґрунтовано доцільність використання великих мовних моделей (LLM) для автоматизації рутинних операцій бізнес-аналізу та векторного пошуку для виявлення семантичних дублікатів завдань.

2. Обґрунтовано вибір технологічного стеку та засобів розробки. Для реалізації системи обрано сучасний стек технологій на базі мови TypeScript: фреймворк Next.js 16 для побудови клієнтської частини та серверного рендерингу, Hono для організації високопродуктивного API з підтримкою RPC, та Drizzle ORM для безпечної типізованої роботи з базою даних PostgreSQL. Такий вибір забезпечив високу швидкодію, масштабованість та надійність програмного продукту.

3. Спроєктовано архітектуру програмної системи. Розроблено модульну архітектуру веб-застосунку з використанням методології Feature-Sliced Design (FSD), що забезпечує чітке розділення відповідальності між шарами бізнес-логіки, інтерфейсу та роботи з даними. Спроєктовано реляційну структуру бази даних, яка, окрім зберігання сутностей проєкту

4. Розроблено алгоритми та програмні модулі інтеграції з ШІ. Реалізовано механізм взаємодії з OpenAI API з використанням підходу «Bring Your Own Key» (BYOK), що забезпечує гнучкість та безпеку даних користувача. Розроблено

ефективні системні промпти та застосовано режим JSON Mode, що дозволило досягти 100% валідності структурованих даних (опис, критерії приймання, оцінка складності), які генеруються штучним інтелектом.

5. Створено сучасний інтерфейс користувача. Розроблено інтерактивну Kanban-дошку з підтримкою технології Drag-and-Drop, що забезпечує інтуїтивно зрозуміле управління життєвим циклом задач. Реалізовано адаптивний дизайн з використанням бібліотеки компонентів shadcn/ui та підтримкою темної теми, що відповідає сучасним вимогам до ергономіки веб-застосунків. Впроваджено патерни «Optimistic UI» для миттєвого візуального відгуку системи.

6. Виконано верифікацію та тестування системи. Проведено комплексне тестування, що включало модульні тести бізнес-логіки, інтеграційні тести API та перевірку стабільності генерації ШІ. Результати аудиту продуктивності (Google Lighthouse) показали високі показники швидкодії (Performance > 90) та доступності. Система продемонструвала стабільну роботу при обробці запитів та коректну поведінку при виникненні помилок.

7. Опрацьовано питання охорони праці. Проаналізовано умови праці розробника програмного забезпечення, ідентифіковано шкідливі та небезпечні виробничі фактори. Розроблено комплекс заходів щодо забезпечення електробезпеки, пожежної безпеки та ергономічної організації робочого місця, що гарантує безпеку життєдіяльності під час експлуатації розробленої системи.

Основні положення та результати роботи доповідались та обговорювались на XIII науково-технічній конференції «Інформаційні моделі, системи та технології» (17–18 грудня 2025 р., м. Тернопіль).

Практичне значення роботи полягає у створенні діючого веб-орієнтованого інструменту, який дозволяє скоротити час на опис завдань, уніфікувати формат технічної документації та підвищити загальну продуктивність команд розробників. Система готова до впровадження у навчальний процес або використання у невеликих IT-командах.

СПИСОК ВИКОРСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с

2. Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim : Blue Hole Press, 2010. 270 p.

3. Cohn M. User Stories Applied: For Agile Software Development. Boston : Addison-Wesley Professional, 2004. 304 p.

4. Models and Completion endpoints / OpenAI API Documentation. URL: https://platform.openai.com/docs/models (дата звернення: 05.12.2025).

5. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston : Addison-Wesley Professional, 2005. 496 p.

6. Sommerville I. Software Engineering. 10th ed. London : Pearson, 2015. 816 p.

7. Manifesto for Agile Software Development / K. Beck et al. 2001. URL: https://agilemanifesto.org (дата звернення: 06.12.2025).

8. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Hoboken : Prentice Hall, 2017. 432 p.

9. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 432 p.

10. RPC and Type-safe API / Hono Documentation. URL: https://hono.dev/guides/rpc (дата звернення: 06.12.2025).

11. Feature-Sliced Design. Architectural methodology for frontend projects. URL: https://feature-sliced.design/ (дата звернення: 06.12.2025).

12. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2003. 1328 p.

13. Schema declaration and migration / Drizzle ORM Documentation. URL: https://orm.drizzle.team/docs/sql-schema-declaration (дата звернення: 07.12.2025).

14. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Boston : Pearson, 2014. 1440 p.

15. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.

16. Bierman G., Abadi M., Torgersen M. Understanding TypeScript // ECOOP 2014 – Object-Oriented Programming. Berlin : Springer, 2014. P. 257–281.

17. App Router and Server Components / Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 07.12.2025).

18. Introduction and Philosophy / Shadcn/ui Documentation. URL: https://ui.shadcn.com/docs (дата звернення: 07.12.2025).

19. Most Popular Technologies / Stack Overflow Developer Survey 2024. URL: https://survey.stackoverflow.co/ (дата звернення: 07.12.2025).

20. PostgreSQL Column Types / Drizzle ORM Docs. URL: https://orm.drizzle.team/docs/column-types/pg (дата звернення: 07.12.2025).

21. How to use JSON mode with OpenAI API / OpenAI Cookbook. URL: https://cookbook.openai.com/examples/json_mode (дата звернення: 07.12.2025).

22. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Hoboken : Prentice Hall, 2008. 464 p.

23. Nielsen J. 10 Usability Heuristics for User Interface Design / Nielsen Norman Group. 1994. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 07.12.2025).

24. Tidwell J. Designing Interfaces: Patterns for Effective Interaction Design. 3rd ed. Sebastopol : O'Reilly Media, 2020. 600 p.

25. Cohn M. Succeeding with Agile: Software Development Using Scrum. Boston : Addison-Wesley Professional, 2009. 504 p.

26. Blazing Fast Unit Test Framework / Vitest Documentation. URL: https://vitest.dev/ (дата звернення: 08.12.2025).

27. Best practices for writing Dockerfiles / Docker Documentation. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (дата звернення: 08.12.2025).

28. Wiggins A. The Twelve-Factor App. 2011. URL: https://12factor.net/ (дата звернення: 08.12.2025).

29. IEEE Standard for System, Software, and Hardware Verification and Validation (IEEE Std 1012-2016). New York : IEEE Computer Society, 2017.

30. Web Vitals / Google Developers. URL: https://web.dev/vitals/ (дата звернення: 09.12.2025).

31. Attention Is All You Need / A. Vaswani et al. // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.

32. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks / OWASP Foundation. URL: https://owasp.org/Top10/ (дата звернення: 09.12.2025).

33. Language Models are Few-Shot Learners / T. B. Brown et al. // NeurIPS. 2020. Vol. 33. P. 1877–1901.

34. Overview and Motivation / TanStack Query Documentation. URL: https://tanstack.com/query/latest/docs/framework/react/overview (дата звернення: 09.12.2025).

35. Fast and reliable end-to-end testing for modern web apps / Playwright Documentation. URL: https://playwright.dev/ (дата звернення: 09.12.2025).

36. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 7th ed. Newtown Square : Project Management Institute, 2021. 250 p.

ДОДАТКИ

Додаток А

Тези конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.8:005.8
С. Дячук, к. т. н., доц.; Р. Сава
(Тернопільський національний технічний університет імені І. Пулюя, Україна)

ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ ОПИСУ ЗАВДАНЬ ПРОЕКТНОГО МЕНЕДЖМЕНТУ

UDC 004.8:005.8
S. Dyachuk, PhD., Assoc. Prof.; R. Sava

APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNOLOGIES FOR OPTIMIZING PROJECT MANAGEMENT TASK DESCRIPTIONS

Проблема якості опису завдань у проектному менеджменті (ПМ) є критичною, призводить до значних витрат на переробку та неефективної комунікації [1]. Існуючі системи ПМ (Jira, Asana) зосереджені на адмініструванні завдань, а не на якості їхнього вмісту. Стрімкий розвиток великих мовних моделей (LLM) створює можливості для автоматизації та стандартизації цього процесу [2, 3]. Метою дослідження є підвищення ефективності планування проектів шляхом розробки та тестування методології інтелектуальної оптимізації описів завдань.

Об’єктом дослідження є процес опису та формалізації завдань (User Story, Task, Bug) у системах проектного менеджменту. Для порівняльного аналізу існуючих рішень та виявлення функціональних «прогалин» застосовано метод системного аналізу. Ключовим методом дослідження є розроблена методологія «багаторівневого контекстуального промптингу». На відміну від стандартних підходів, ця методологія формує запит до LLM шляхом комбінації трьох джерел: 1) статичний загальний контекст проекту, 2) персоналізовані налаштування користувача (стиль, модель, BYOK) та 3) динамічний короткий опис завдання. Для валідації гіпотези та оцінки ефективності методології застосовано метод експертного дослідження з двома групами користувачів (контрольна та експериментальна) та набором кількісних (TTC, Rework Rate) і якісних (Quality Score) метрик.

На основі аналізу існуючих ПМ-систем виявлено ключове обмеження: відсутність інструментів для генерації контенту, що враховують контекст конкретного проекту та стиль опису.

Розроблено архітектуру та реалізовано програмний прототип модулю (стек: Next.js, PostgreSQL, Drizzle) з інтегрованим AI-модулем. Веб-застосунок реалізує запропоновану методологію «багаторівневого контекстуального промптингу», дозволяючи користувачам генерувати структуровані описи завдань (включно з Acceptance Criteria) на основі короткого вводу.

Оцінювання здійснювалося за технічними критеріями: стабільність генерації, коректність структури вихідного тексту та відповідність контексту проєкту.

Отримані результати підтверджують, що навіть за мінімального обсягу початкових даних система формує узгоджені, стилістично коректні та повні описи завдань. Це демонструє ефективність архітектурного підходу й можливість масштабування рішення.

Інтеграція LLM у процес створення завдань за запропонованою методологією «багаторівневого контекстуального промптингу» значно підвищує якість описів та скорочує час їх підготовки. Розроблений підхід зменшує когнітивне навантаження на менеджерів проектів та покращує чіткість вимог для команд розробки, що підтверджує гіпотезу дослідження.

Література

1. «The cost and impact of requirement defects in agile software development: A systematic literature review». Journal of Systems and Software, 178, 110977.
2. «Leveraging Large Language Models for Software Requirements Engineering: An Empirical Study and Future Directions». IEEE Transactions on Software Engineering.
3. «Context-Aware Prompt Engineering for Natural Language Processing». Monograph on AI Integration Strategies.

Додаток Б

Лістинг коду генерації завдань

```
import OpenAI from 'openai';
import { z } from 'zod';

// Zod Schema for AI response validation
// Ensures the frontend receives exactly the fields it expects
const AIResponseSchema = z.object({
  summary: z.string().describe("Expanded technical title of the
task"),
  userStory: z.string().describe("Format: As a <role>, I want
<feature>, so that <value>"),
  acceptanceCriteria: z.array(z.string()).describe("List of
acceptance criteria"),
  storyPoints: z.number().min(1).max(13).describe("Complexity
estimation using Fibonacci sequence"),
  type: z.enum(['feature', 'bug', 'chore']).describe("Type of the
task"),
});

// Type inference for the service response
export type AITaskResponse = z.infer<typeof AIResponseSchema>;

export class AIService {
  private openai: OpenAI;

  // Initialization with a dynamic key (BYOK implementation)
  constructor(apiKey: string) {
    this.openai = new OpenAI({
      apiKey: apiKey,
      dangerouslyAllowBrowser: false, // Server-side execution only
    });
  }

  /**
   * Generates a detailed task description based on a short title
and project context
   * @param taskTitle - Short task title provided by the user
   * @param projectContext - Project description for better AI
context understanding
   */
  async generateTaskDescription(
    taskTitle: string,
    projectContext: string
  ): Promise<AITaskResponse> {

    // Constructing the system prompt (Role Engineering)
    const systemPrompt = `
      You are an expert Senior Product Manager and Agile Coach.
      Your goal is to transform short task titles into professional
```

```
technical specifications.

      Project Context: "${projectContext}"

      Output Rules:
      1. Response MUST be valid JSON.
      2. 'userStory' must follow the pattern: "As a [role], I want
[feature], so that [benefit]".
      3. 'acceptanceCriteria' must be specific, testable conditions
(Given/When/Then is preferred but not required).
      4. Estimate complexity using Fibonacci sequence (1, 2, 3, 5,
8, 13).
    `;

    try {
      const completion = await this.openai.chat.completions.create({
        model: "gpt-4-turbo-preview", // Or "gpt-4o"
        messages: [
          { role: "system", content: systemPrompt },
          { role: "user", content: `Generate specs for task:
"${taskTitle}"` },
        ],
        temperature: 0.7, // Balance between creativity and
stability
        response_format: { type: "json_object" }, // Enforced JSON
mode
      });

      const content = completion.choices[0].message.content;

      if (!content) {
        throw new Error("AI returned empty response");
      }

      // Parsing and validating the received data against the schema
      const parsedData = JSON.parse(content);
      const validatedData = AIResponseSchema.parse(parsedData);

      return validatedData;

    } catch (error) {
      console.error("AI Generation Error:", error);

      // Error handling (e.g., invalid key, rate limits, or
structure mismatch)
      if (error instanceof z.ZodError) {
        throw new Error("AI response structure mismatch");
      }
      throw error;
    }
  }
}
```