МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»»
Факультет комп'ютерно-інформаційних технологій
Кафедра комп'ютерних інформаційних систем і технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Тема «Проєктування та розробка системи керування проєктами на основі мікросервісної архітектури з інтеграцією штучного інтелекту»

Виконав студент групи ІК-9-22-Б1ІПЗ (4.0ддс) Жук Р.В.

Керівник: к.т.н., доцент Гордієнко О.М.

Допущено до захисту

Завідувач кафедри _____ д.е.н., професор Кавун С.В.
(підпис)

Київ, 2026

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи: 137 с., 34 рисунки, 43 таблиці, 2 додатки, 96 джерело.

Об’єктом дослідження є процес проєктування і розробки системи для керування проєктами на основі принципів мікросервісної архітектури з інтеграцією штучного інтелекту.

Метою роботи є розробка системи для керування проєктами на основі принципів мікросервісної архітектури і інтеграцією штучного інтелекту. Система повинна дозволяти керування проєктами і завданнями з семантичним пошуком, пошуком дублікатів, інтерпретацією завдань та звітністю.

Методи розробки базуються на використанні системного аналізу, принципів побудови розподілених мікросервісних систем та відопвідного стеку технологій.

Результатом роботи є спроєктована і розроблена мікросервісна система, функціонал якої спрямований на вирішення проблем цільової аудиторії і підвищення ефективності роботи на проєктах.

КЛЮЧОВІ СЛОВА: СИСТЕМА КЕРУВАННЯ ПРОЄКТАМИ, МІКРОСЕРВІСНА АРХІТЕКТУРА, ВЕКТОРИЗАЦІЯ, КОНТЕЙНЕРИЗАЦІЯ, FASTAPI, VUE, DOCKER, KAFKA, POSTGRESQL, QDRANT, OLLAMA, DEBEZIUM, NGINX, СЕМАНТИЧНИЙ ПОШУК

ABSTRACT

The object of the research is the process of designing and developing a project management system based on the principles of microservice architecture with the integration of artificial intelligence.

The aim of the work is to develop a project management system based on the principles of microservice architecture and the integration of artificial intelligence. The system should provide project and task management functionality, including semantic search, duplicate detection, task interpretation, and reporting.

The development methods are based on the application of system analysis, the principles of designing distributed microservice-based systems, and an appropriate technology stack.

The result of the work is a designed and developed microservice-based system whose functionality is aimed at addressing the needs of the target audience and improving the efficiency of project work.

KEYWORDS: PROJECT MANAGEMENT SYSTEM, MICROSERVICE ARCHITECTURE, VECTORIZATION, CONTAINERIZATION, FASTAPI, VUE, DOCKER, KAFKA, POSTGRESQL, QDRANT, OLLAMA, DEBEZIUM, NGINX, SEMANTIC SEARCH

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACID — Набір транзакційних властивостей у базі даних (атомарність, узгодженість, ізольованість, довговічність)

API — Application Programming Interface (програмний інтерфейс застосунку)

CDC — Change Data Capture – механізм відстеження змін в базі даних

MRR — Mean Reciprocal Rank - середній зворотній ранг, показник, який використовується для оцінки якості пошукової моделі

MVP — Minimum Viable Product - продукт який містить основний функціонал для роботи системи для перевірки ідеї, демонстрації продукту та отримання зворотнього зв’язку.

Open-source — Спосіб розробки і розповсюдження програмного забезпечення при якій код є відкритим для перегляду і подальшого використання.

UC — Use case – сценарій використання

Ембедінг — Представлення тексту у векторному вигляді

Квантизація — Метод оптимізації моделі машинного навчання який дозволяє за рахунок зменшення точності досягнути зменшення розміру моделі і знизити вимоги до ресурсів.

Консюмер — Програмний компонент для зчитування повідомлень з топіка

НФВ — Нефункціональна вимога – описує умови при яких система має залишатися ефективною, а також атрибути якості.

Продюсер — Програмний компонент для відправки повідомлень в топік

Топік — Логічний канал для публікації повідомлень і їх зчитування

ФВ — Функціональна вимога – описує можливості системи і поведінку

ШІ — Штучний інтелект - сукупність засобів і методів здатних виконувати завдання так само добре або краще ніж людина.

ВСТУП

На сьогоднішній день керування проєктами базується на ефективному управлінні ресурсами, часом та бюджетом. Правильний розподіл відповідальності щодо завдань та дотримання термінів є ключовими факторами які визначають успішність проєктів. Керування проєктами вимагає значних зусиль в умовах роботи з контекстом який змінюється в процесі роботи над проєктом, моніторингу прогресу та прогнозуванню результатів. Учасники та керівники таких проєктів мають різні рівні технічних знань, що в свою чергу призводить до непорозумінь при визначенні пріоритетів завдань і під час прийняття важливих рішень.

Системи керування проєктами на сьогоднішній день представлені такими лідерами як Jira, Asana, Trello, monday.com та Azure DevOps. Ці продукти надають необхідний для роботи функціонал, але мають свої недоліки такі як довга крива навчання, складні налаштування, перевантажений інтерфейс, вартість. Також присутньою є проблема пошуку завдань, розуміння контексту завдання і необхідність пам’ятати завдання для того щоб повторно не виконувати роботу яка вже в якомусь об’ємі була виконана раніше. Продуктивність самої системи і її здатність працювати в складних умовах також є важливим критерієм вибору серед користувачів.

Для вирішення цих проблем доцільно застосувати такі рішення як інтеграція методів штучного інтелекту для роботи з завданнями, інтуїтивний інтерфейс та мікросервісна архітектура системи.

Об’єктом дослідження є процес проєктування і розробки системи для керування завданнями на основі принципів мікросервісної архітектури з інтеграцією штучного інтелекту.

Предметом дослідження є визначення методів і засобів побудови системи керування завданнями з мікросервісною архітектурою з інтеграцією штучного інтелекту (обробка, пошук, аналіз запитів).

Метою кваліфікаційної роботи є проєктування та розробка системи керування проєктами на основі мікросервісної архітектури з інтеграцією штучного інтелекту.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- виконати аналіз предметної області керування проєктами;
- дослідити існуючі програмні рішення;
- визначити проблеми користувачів та цільову аудиторію продукту;
- визначити сценарії використання, функціональні та нефункціональні вимоги до системи;
- спроєктувати архітектуру системи;
- обґрунтувати вибір технологічних рішень;
- виконати розробку програмного продукту.

Методи розробки базуються на використанні системного аналізу, принципів побудови розподілених мікросервісних систем та відповідного стеку технологій.

Результатом роботи є спроєктована і розроблена мікросервісна система, функціонал якої спрямований на вирішення проблем цільової аудиторії і підвищення ефективності роботи на проєктах.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область керування проєктами включає в себе широкий спектр методологій, процесів, інструментів, та артефактів. Каскадна модель (Waterfall), Скрам (Scrum), Канбан (Kanban) та інші підходи до організації проєкту і роботи над ним вимагають забезпечення певних вимог для успішності запуску і завершення проєктів.

Планування роботи є ключовим фактором який впливає на успішність проєктів. Системи для керування проєктами та завданнями продовжують розвиватися і покриваються основні запити такі як централізоване зберігання завдань, планування релізів, колаборацію між багатьма користувачами, координацію роботи та візуалізацію прогресу.

До ключових процесів предметної області відносяться наступні [1]:

1. процес керування межами проєкту. Робота з вимогами і визначення об’єму проєкту;
2. процес планування та декомпозиції завдань. Сюди входить формування списку завдань, пріоритезація, оцінка тривалості виконання або складності, планування релізів;
3. процес роботи над виконанням завдань. Сюди входить призначення виконавців, зміна станів завдань, контроль взаємозалежностей між завданнями і послідовністю виконання завдань;
4. процес керування і моніторингу ресурсами, комунікацією, ризиками, якістю.

Ринок програмних рішень для керування проєктами завданнями демонструє стабільний ріст кожного року. Часткоо це зумовлено появою дистанційної роботи та цифровізацією бізнс процесів. У 2025 році оцінка

вартості ринку складала 4 475,96 млн. доларів і за прогнозами аналітиків зросте до 12 млрд. додарів до 2035 року.

Ключовим регіоном є США (понад 72 млн корпоративних користувачів). Відзначається, що продуктивність роботи після впровадження програмного рішення зросла на 34%. Близько 79 % всіх рішень є хмарними, оскільки дозволяє швидко масштабуватися в залежності від попиту. Близько 48% користувачів працюють над завданнями через мобільні пристрої.

Відзначено, що стимулом до зростання ринку є потреба в координації крос-функціональних команд в умовах віддаленої роботи та сприяння розвитку прогнозувальної аналітики. Ще одним важливим фактором є проблематика впровадження складних систем для працівників певної категорії [2].

Частку користувачів відповідно до сфери використання систем для керування проєктами та завданнями зображено на рисунку 1.1.

Рисунок 1.1 – Кількість активних користувачів відповідно до сфери діяльності у 2024 році, млн. осіб

У таблиці 1.1. відображено порівняння між статистичними даних по керівниках проєктів у США за 2021 рік наступні згідно з дослідженнями компанії Zippa [3-5].

Таблиця 1.1 – Статистика по користувачах які працюють у якості керівників проєктів у США

Характеристика користувача	**Глобальні керівники проєктів**	**Керівники IT проєктів**	**Власники бізнесу (керівники)**
К-сть осіб	240475	22008	5582
Науковий ступінь бакалавра	68%	70%	65%
Науковий ступінь магістра	22%	19%	11%
Середній вік	71% - 40+ років 24% - 30-40 років 5% - 20-30 років	45% - 40+ років 36% - 30-40 років 18% - 20-30 років	69% - 40+ років 25% - 30-40 років 6% - 20-30 років
Розмір компанії, Ксть осіб	58% - > 10,000 21% - 1000-10000 4% - 500-1,000 8% - 100-500 9% < 100	33% - > 10,000 26% - 1000-10000 9% - 500-1,000 18% - 100-500 13% < 100	21% - > 10,000 18% - 1000-10000 8% - 500-1,000 21% - 100-500 32% < 100
Стать	56,7% - чол 43,5% - жін	67,2% - чол 32,8% - жін	80,8% - чол 19,2% - жін

Типові проблеми з якими стикаються користувачі в процесі роботи над проєктами включають:

1. складний і не ефективний пошук за допомогою ключових слів, фраз, повного або частково співпадіння, що в результаті спричиняє труднощі з запам’ятовуванням конкретних назв та появу нерелевантної інформації серед результатів пошуку;
2. дублювання інформації та дослідження на виявлення факту дублікації частини робіт, що виникає у відносно великих проєктах внаслідок високого рівня грануляції завдань, зміни підходу з каскадного на гнучкий або навпаки, а також через обмеження людської пам’яті щодо сформульованого завдання або виконаної роботи;

3. труднощі інтерпретації технічних нюансів завдань бізнес користувачами;
4. обмежені можливості аналітики проєкту, де більш глибокий аналіз вимагає додаткової інвестиції часу і ручної роботи, аналіз використаних ресурсів і стан проєкту;
5. відсутність систематизованого підходу до розподілу, контролю і моніторингу роботи.

Актуальність розробки нових рішень підтверджується незалежним аналізом різноманітних компаній. MarketGrowthReports дає оцінку зростання ринку рішень для керування проєктами в 3 рази до 2035 року. На сьогодні к-сть активних користувачів таких систем тільки у США становить близько 312 млн. [2]. Astute Analytica у своєму досліджедженні ринку ПЗ також наводить схожі цифри у 11,5 млрд доларів вартості ринку до 2033 року [6]. Fortune Business Insights також наводить результати аналізу які підтверджують тренд зростання ринку з 2018 до 2026 року [7]. GrandViewResearch наводить в досліджені кількісні показники які відрізняються від конкурентів, але підкреслюють тенденцію розвитку рішень для керування проєктами [8].

Доцільність створення нового рішення ґрунтується на наявності ринкової потреби у простому та ефективному інструменті. Копанія Capterra у своїх дослідженнях зазначає, що на ринку США у 2025 році лише 23% компаній використовують ПЗ для керування проєктами, 77% в свою чергу використовують Microsoft Excel, поштову скриньку за інші засоби. [9] Це є свідченням того, що значна частина малого та середнього бізнесу все ще має проблему з якісним структуруванням і керуванням проєктом (дублювання завдання, складний пошук, занадто складні налаштування та своєрідна звітність).

Ще одне дослідження Capterra вказує на те, що тенденція у 2025 році мати ШІ-функціонал є ключовим фактором для покупки ПЗ для керування проєктами для 55% опитаної аудиторії і безпека даних - для 71%.

Отже сучасні вимоги включать простоту в використанні, максимальный контроль над безпекою даних, здатність до масштабування, адаптація під мобільні пристрої, можливість використання інтелектуальних функцій.

Цільовою аудиторією для нового рішення є команди які складаються з 3-100 осіб, працюють паралельно над декількома проєктами, у яких присутня необхідність спланувати проєкт, декомпозувати завдання, розподілити їх між виконавцями, мати можливість знайти необхідне завдання за суттю, отримувати автоматизований звіт з деталями щодо бюджету, термінів виконання, стану проєкту та прогнозу розвитку на майбутнє. Інформацію по завдання які за змістом дублюються також буде економити час та ресурси.

Сегмент цільової аудиторії для якого рішення буде найбільше цінним це банки, страхові компанії, сервісні компаній у яких робота з інформаційними технологіями є додатковим напрямом діяльності, який використовується як інструмент для забезпечення функціонування бізнесу та його розвитку, а також безпека даних які циркулюють у локальному контурі є критичною для існування бізнесу.

Серед кінцевих користувачів системи можна виділити керівників підрозділів, зацікавлені особи зі сторони бізнесу без глибоких технічних знань, операційні керівники, технічні керівники та безпосередні виконавці завдань.

1.2 Аналіз існуючих програмних рішень

Серед ключових гравців на ринку ПО для керування проєктами з найбільшою часткою ринку можна виділити наступні:

- Jira [10] - рішення для проєктного менеджменту яке в процесі розвитку трансформувалося з трекеру багів. Фокус на плануванні роботи, моніторингу і контролю релізів.

- Asana [11] - позиціонується як платформа для керування роботою з фокусом на мінімізацію підготовки роботи, а саме координації, адміністрування процесів та пошуку інформації.
- Trello [12] - позиціонує себе як інструмент для візуалізації роботи, простоти та гнучкості в плануванні і моніторингу для будь-якого проєкту;
- monday.com [13] - позиціонується як операційна система для роботи, яка дозволяє налаштовувати і керувати процесами без необхідно щось програмувати кодом.
- Azure DevOps - платформа для керування життєвим циклом программного забезпечення, яка орієнтована комади розробки ПЗ і DevOps. Відносить до інструментів DevOps ринку. Включає в себе такі елементи як Boards (керування завданнями), Repos (репозиторії), Pipelines (пайплайни), Test Plans (тест плани, Artifacts (артефакти) [14].

На рисунку 1.2 показано розподіл ринку систем керування проєктами між найпопулярнішими продуктами.

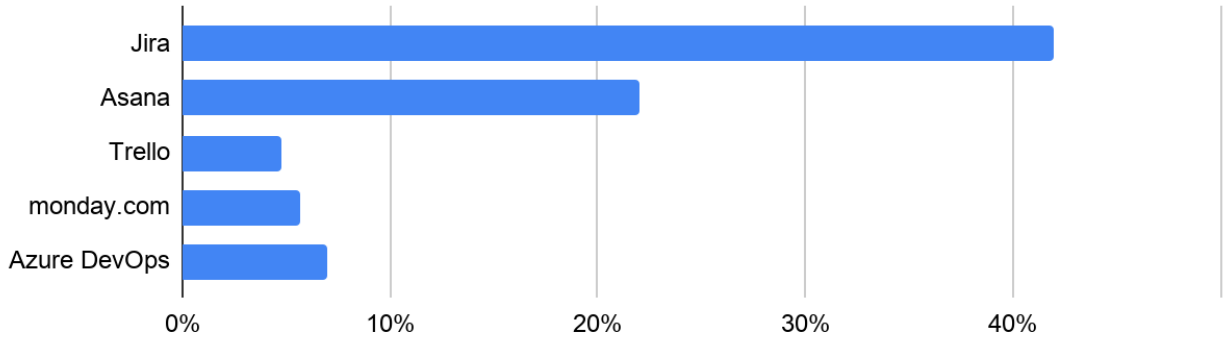

Рисунок 1.2 – Доля цифрового продукту на світовому ринку

Порівняльна таблиця 1.2 показує відмінності існуючих рішень на світовому ринку в категорії програмного забезпечення для керування проєктами.

Таблиця 1.2 – Аналіз лідерів ринку в категорії програмного забезпечення для керування проєктами

Параметр	**Jira**	**Asana**	**Trello**	**monday.com**	**Azure DevOps**
Присутність на ринку (дані за 2023-2025 роки)	~ 42-50 % - доля світового ринку [15, 16] 38,8% Австралія 19,8 США 6,8% Англія [17]	~ 22% - доля світового ринку [16, 18] 70,8% США 7,0% Австралія 5,3% Англія [19]	~ 4,8% - доля світового ринку [20, 21]	~ 5,7 % - доля світового ринку [22] 50% США 22% Європа 10% Англія [23]	~ 7-14% - доля світового ринку серед DevOps сервісів [24, 25] 49% США 10% Англія 6,7% Індія [24]
Активні користувачі (по світу)	Більше 300 тисяч компаній-користувачів [26]	Близько 150 тисяч компаній-користувачів [27]	50 млн зареєстрованих на 2019 рік [28]	Близько 250 тисяч компаній-користувачів [13]	Близько 27 тис. Компаній-користувачів [24]
Цільова аудиторія	B2B компанії, команди розробки ПЗ, техпідтримка, DevOps, сервісні команди (IT, маркетинг, дизайн) [29]	B2B компанії середнього і великого розміру (IT/цифрові сервіси, освіта, роздрібна торгівля, дозвілля та гостинність, виробництво) [19]	B2B/B2C користувачі, малі (<10) та середні команди (до 1000), фрілансери (IT, розробка ПЗ, маркетинг, фінансові послуги, освіта) [30]	B2B компанії малого і середнього розміру (маркетинг, операційна діяльність, продажі, розробка ПЗ, керування персоналом, виробництво) [31]	B2B компанії середнього та великого розміру (DevOps та команди розробки ПЗ), особливо ті які вже використовують сервіси Azure [32]
Стратегія розповсюдження	Понад 700 партнерів у 80+ країнах, які продають Jira, консультують і впроваджують[33].	Product-Led Growth (PLG) Прямі продажі великим компаніям, інтеграція з Google Workspace, MS Teams [34]	Freemium модель для швидкого запуску, продаж разом додатково до продуктів Atlassian [35]	Агресивний маркетинг у бренд та концепцію ОС для роботи, залучення реселерів. [36]	Freemium модель для 5 користувачів. Подальше розширення завдяки ліцензіям, канали Microsoft і партнерів компанії. [37]

Кінець таблиці 1.2

Параметр	**Jira**	**Asana**	**Trello**	**monday.com**	**Azure DevOps**
Вартість за 1 кор/міс, долари	Від 7 (Standard) [38]	Від 13,49 (Starter) [39]	Від 5 (Standard) [40]	Від 12 (Basic) [41]	Від 6 (Basic) [37]
Основні функ. можливості	Керування завданнями і проєктами Канбан, Скрам дошки, беклог, Дорожні карти, керування повноваженнями, звітність, автоматизація подій [42]	Дорожні карти, проєкти, цілі, беклог, ресурсне планування, автоматизація подій, звітність, шаблони для процесів [43]	Робоча область, канбан дошка, різні подання даних, кастомні поля, календар, шаблони для типових сценаріїв роботи, мобільний клієнт [44]	Керування завданнями на Канбан дошці, дорожні карти, інтеграції, автоматизація подій, звітність, форми для автоматичного створення завдань [41]	Керування завданнями, звітність, репозиторії, пайплайни, керування тестами, артефакти [37]

Аналіз ринку підкреслює, що найбільше розповсюдження і застосування має Jira, а Asana, Trello, monday.com та Azure DevOps орієнтовані на більш спеціалізовані сегменти ринку і потреби конкретної цільової аудиторії. Завдяки особливостям функціоналу кожного продукту, кожен з них дозволяє кінцевим користувачам задовольнити наявні бізнес потреби в межах їхньої сфери діяльності. Таблиця 1.3 показує придатність функціоналу цифрових продуктів до певної сфери діяльності кінцевих користувачів.

Таблиця 1.3 – Позиціонування продукту відповідно до сфери діяльності

Продукт / Сфера діяльності цільової аудиторії	**Jira**	**Asana**	**Trello**	**monday.com**	**Azure DevOps**
IT та розробка ПЗ	✓	✓	✓	✓	✓
DevOps	✓	-	○	○	✓
Техпідтримка (сервісні послуги)	✓	○	○	○	-
Маркетинг	✓	✓	✓	✓	-
Продажі	○	○	○	✓	-

Кінець таблиці 1.3

Продукт / Сфера діяльності цільової аудиторії	**Jira**	**Asana**	**Trello**	**monday.com**	**Azure DevOps**
HR	○	○	○	✓	-
Освіта	○	✓	✓	○	-
Виробництво	○	✓	○	✓	○
Роздрібна торгівля	○	✓	○	✓	-
Гостинність та дозвілля	○	✓	○	○	-
Дизайн / креативні команди	○	○	○	○	-
Фінансові послуги	○	○	○	○	○

- ✓ цільовий сегмент з яким позиціонується продукт.
- ○ присутня можливість для використання у сегменті.
- - використання можливе у певних випадках, але не вказується напряму виробником.

Для розуміння сильних і слабких сторін кожного продукту проведено порівняння їх переваг і недоліків у таблиці 1.4.

Таблиця 1.4 – Переваги і недоліки цифрових продуктів для керування проєктами

Система	**Переваги**	**Недоліки**
Jira [45]	● Підтримка Agile підходів ● Гнучкі налаштування ● Інтеграція з Atlassian-екосистемою ● Маректплейс з плагінами ● Звітність	● Складне навчання ● Перевантажений інтерфейс ● Складний пошук завдань ● Зниження продуктивності роботи у складних проєктах ● Висока вартість планів з розширеним функціоналом ● Обмеження по к-сті учасників у безкоштовному плані ● Платні плагіни
Asana [46]	● Гнучкий підхід Work Graph (завдання може належати до декількох проєктів) ● Портфоліо для керування декількома ініціативами ● Інструменти для OKR та цілей ● ШІ-учасники ● Інтеграції зі Slack, GDrive	● Складне навчання ● Висока вартість планів з розширеним функціоналом ● Надлишкові повідомлення ● Швидкодія ● Обмеження в дешевших планах (облік часу, к-сть учасників)

Кінець таблиці 1.4

Система	**Переваги**	**Недоліки**
Trello [47]	● Простий старт роботи ● Різноманітні подання (календра, карта і т.п.) ● Базових функцій достатньо для роботи в маленькій команді над відносно простим проєктом ● Вбудована автоматизація ● 200+ функцій розширення	● Відсутність залежностей між завданнями, ● Обмеженість для складних проєктів. ● Більшість функціоналу доступна в Premium/Enterprise планах ● Обмежена квота на автоматизацію
monday.com [48]	● Кастомізація відповідно до потреб робочого процесу ● Дашборди ● No-code автоматизації	● Цінова політика (мін 3 користувачі, далі збільшення з кроком в 5) ● Обмежена квота на автоматизацію ● Складні налаштування ● Відсутність просунутого функціоналу для керування завданнями
Azure DevOps [49]	● Інструменти для SDLC ● Підтримка методологій розробки ● Гнучкі налаштування ● Підтримка трасування від вимоги до реалізації в коді	● Складний для вивчення інтерфейс ● Складні налаштування ● Обмежена функціональність для класичного керування проєктами і планування ● Фокус на Azure-екосистемі

Подальше виокремлення критеріїв оцінювання дозволяє наглядно побачити особливості, наявність яких дозволяє продуктам бути цінними для своєї цільової аудиторії (у таблиці 1.5):

1. Масштабованість - здатність системи підтримувати роботу з багатьма користувачами і проєктами у разі необхідності.
2. Звітність і аналітика - наявність необхідних інструментів для контролю і моніторингу проєктів.
3. Зручність використання та поріг входження - на скільки процеси і інтерфейс user-centric і сприяють швидкому навчанню користувачів.

4. Інтеграції з іншими системами - наявність можливості передавати або обмінюватися даними з іншими системи (месенджери, інші сервіси).
5. Мобільна версія - наявність адаптації під мобільні пристрої.
6. Автоматизація процесів - підтримка налаштувань процесу автоматизації якихось дій.
7. Підтримка методологій розробки - можливість застосувати і використовувати певну методологію розробки на проєкті.

Таблиця 1.5 – Підсумкова порівняльна таблиця цифрових продуктів

Продукт / критерій оцінки	**Jira [50]**	**Asana [51]**	**Trello [52]**	**monday.com [52]**	**Azure DevOps [50]**
Масштабованість	5	4	2	4	4
Зручність використання та поріг входження	3	4	5	4	2
Підтримка методологій розробки	5	4	3	4	5
Мобільна версія	4	4	4	4	3
Звітність і аналітика	4	4	2	4	4
Інтеграції з іншими системами	5	4	4	4	4
Автоматизація процесів	4	4	3	5	4

Для порівняння використано наступну шкалу оцінювання (відповідно до загадок в джерелах інформації):

- 1 - функціоналу немає
- 2 - функціонал присутній, але дуже обмежений
- 3 - функціонал присутній в достатньому для комфортної роботи об’ємі
- 4 - реалізація функціоналу здатна покрити складні вимоги, але не є досконалою
- 5 - повноцінна реалізація яка надає перевагу на ринку.

Результати порівняння показано на рисунку 1.3.

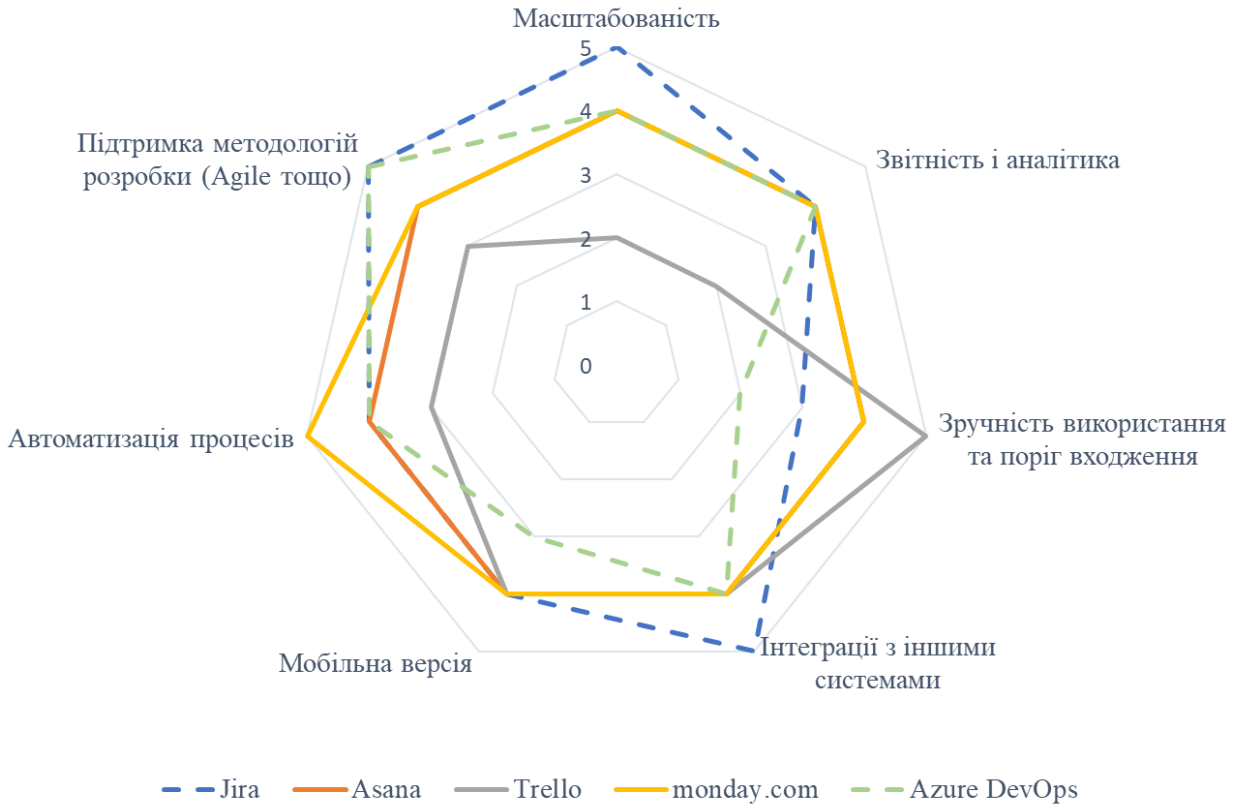

Рисунок 1.3 – Порівняння систем для керування проєктами за ключовими критеріями

Проведений аналіз показує, що кожна з проаналізованих систем має свої сильні і слабкі сторони. При виборі системи керування проєктами необхідно звертати увагу на величину проєкту, команди, цільову аудиторію, а також технічний рівень кінцевого користувача.

1.3 Формування мети та завдань роботи

Зважаючи на отримані результати проведеного аналізу предметної області та обгрунтування доцільності створення нових програмних рішень, визначено наступну мету і завдання роботи.

Метою кваліфікаційної роботи є розробка сучасної системи для керування завданнями, яка має забезпечувати виконання роботи в хмарному середовищі, забезпечувати безпеку чутливих даних, бути легкою для освоєння, дозволяти відслідковувати дублікати, дозволяти здійснювати семантичний пошук завань по суті завдання, а також надавати спрощені інтерпретації опису завдань які містять багато технічних деталей і є складними для розуміння спеціалістами не технічних спеціальностей.

Для виконання поставленої задачі, необхідно:

- виконати аналіз предметної області керування проєктами;
- дослідити існуючі програмні рішення;
- визначити проблеми користувачів та цільову аудиторію продукту;
- визначити сценарії використання, функціональні та нефункціональні вимоги до системи;
- спроєктувати архітектуру системи;
- обґрунтувати вибір технологічних рішень;
- виконати розробку програмного продукту.

1.4 Висновки до першого розділу

Фактори які зумовлюють подальше зростання ринку і потреб користувачі щодо ефективного рішення залишатимуться сталими у довготривалій перспективі. Сюди входять дистанційна робота, хмарні рішення, розвиток мобільних пристроїв, цифрових технологій та їх впровадження у різних сферах діяльності людини.

З часом цифрові продукти трансформуються і адаптуються під потреби кінцевих користувачів. З одного боку це може трактуватися як охоплення більшої аудиторії і розповсюдження у нових сферах. З іншого боку з появою

надлишкового функціоналу у кінцевих користувачів виникає потреба додатково навчатися для того щоб освоїти продукт і повноцінно з ним працювати. Велика частина компаній у США (77%) все ще надають перевагу більш простим інструментам як Microsoft Excel в противагу спеціалізованим інструментам для керування проєктами і завданнями. Серед керівників проєктів різних рівнів та власників компаній значну частку складають люди без технічної освіти для яких проста для розуміння звітність і інтелектуальні функції системи є критичними для ефективного керування проєктами.

Зважаючи на поточну ситуацію на ринку розробка системи для керування завданнями з інтеграцією штучного інтелекту є доцільною для визначеної цільової аудиторії та бізнес сегменту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

2.1 Сценарії варіантів використання

З метою забезпечення верифікації вимог, визначення границь системи та тестування виникає необхідність моделювати вимоги. Для моделювання вимог з точки зору кінцевого користувача доцільно використати UML діаграму типу Use Case. Діаграма сценаріїв використання чітко описує межі системи, перелік сценаріїв використання які доступні для кожного класу користувача для в взаємодії між користувачем і системою [53].

Перед початком виявлення і аналізу вимог доцільно визначити класи користувачів які будуть взаємодіяти з системою яка проєктується. Дана система фокусується на ключовому функціоналі який має цінність для кінцевого користувача.

Зважаючи на необхідність ізолювати керування і дані у середовищі з яким взаємодіє багато користувачів було виділено наступні 3 класи користувачів у таблиці 2.1.

Таблиця 2.1 – Класи користувачів системи

Класс	**Опис**
Гість	Неавторизований користувач без доступу до функціоналу системи. Метою цього користувача є реєстрація і автентифікація в системі
Власник	Авторизований користувач, якому доступний функціонал адміністративного рівня для повного контролю проєкту
Учасник	Авторизований користувач, якому доступні обмежені можливості при роботі з проєктом у якому він приймає участь

Кінець таблиці 2.1

Класс	**Опис**
Авторизований користувач	Користувач, який авторизований в системі

Для відображення груп сценаріїв варіантів використання та допоміжних кольорів використано функціонал інструменту plantuml [54]. Діаграму сценаріїв використання показано на рисунку 2.1.

Первинними акторами в залежності від сценарію використання виступають Гість, Власник, Учасник. Авторизований користувач, який в свою чергу є генералізованим актором, на що вказуються відповідні зв’язки на діаграмі. Деякі допоміжні сценарії використання винесені окремо, але пов’язані з основним сценарієм через зв’язок «extend», оскільки можуть виконуватися за певної умови. Така необхідність зумовлена наприклад, невизначеністю при роботі з новим проєктом, де потрібно сформувати завдання, але призначити виконавця може бути проблематично, оскільки учасників проєкту можуть бути тимчасово відсутні або інші причини.

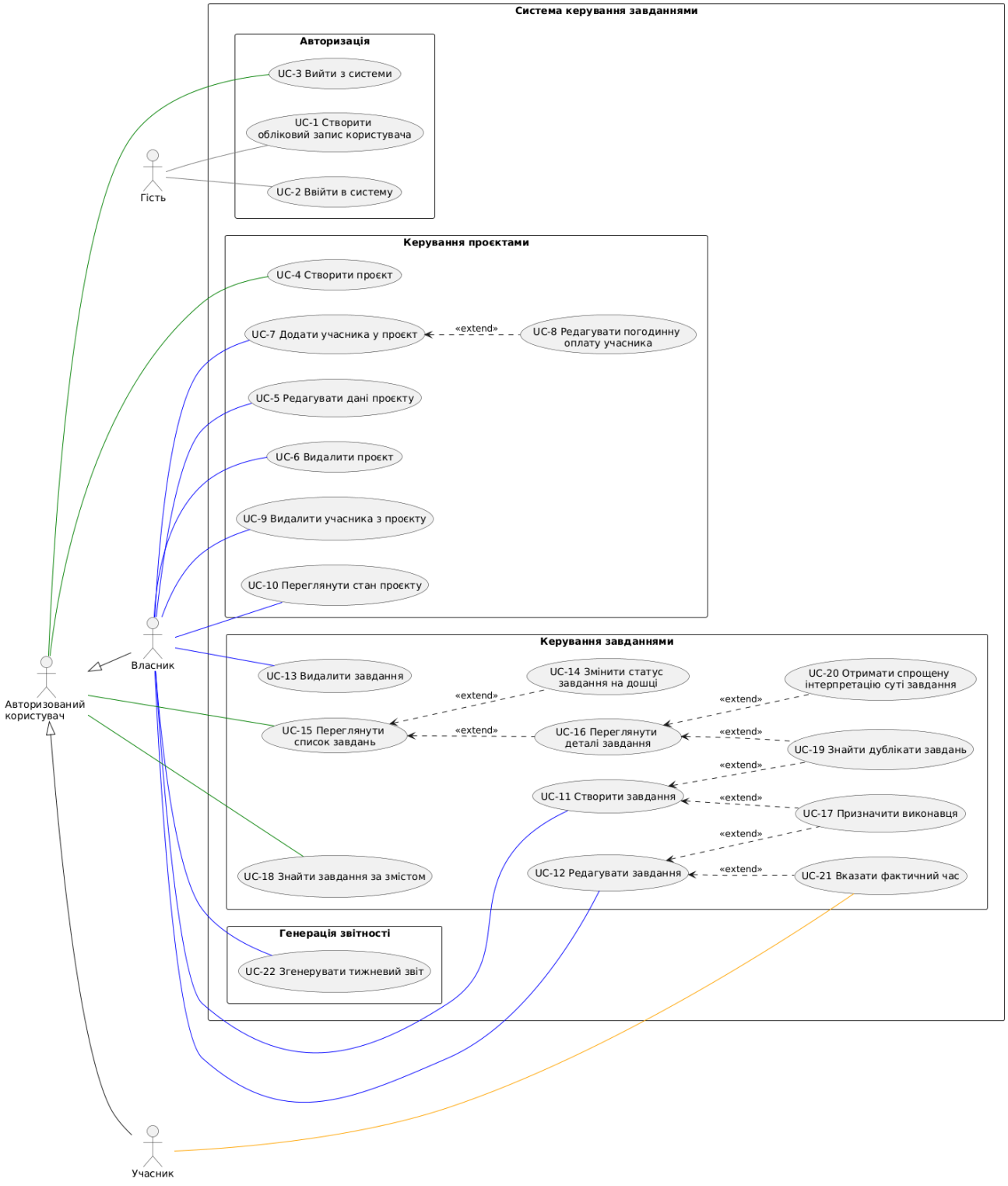

Рис 2.1 – Діаграма варіантів використання

Деталі кожного сценарію використання наведено в таблицях 2.2-2.23. Мінімальний набір даних для опису сценарія використання включає в себе опис:

- Первинний, вторинний актори

- Передумови
- Результат виконання
- Основний потік
- Альтернативний потік (за необхідності)

Також, певні джерела містять рекомендації щодо наповнення сценаріїв використання іншою додатковою функціональністю для повноти моделювання [55].

В зв’язку з тим, що вторинним актором з точки зору користувача в процесі поточного проєктування завжди буде система керування завдання, було прийнято рішення виключити згадку про вторинного актора із наведених в таблицях 2.2-2.23 сценаріїв використання.

Таблиця 2.2 – Сценарій створення облікового запису користувача

UC-1	**Створити обліковий запис користувача**
Актор	Гість
Передумови	Користувач не зареєстрований в системі
Результат виконання	1. Обліковий запис користувача створено
Основний потік	1. Користувач ініціює процедуру реєстрації та вводить електронну пошту і пароль 2. Система перевіряє коректність даних і унікальність електронної пошти 3. Система створює новий обліковий запис і відображає повідомлення про успішність реєстрації
Альтернативний потік	3а. Система виводить повідомлення про помилку, якщо ел. пошта вже використовується 3б. Система виводить повідомлення про помилку, Якщо пароль не відповідає критеріям до паролю

Таблиця 2.3 – Сценарій входу користувача в систему

UC-2	**Увійти в систему**
Актор	Зареєстрований користувач
Передумови	Користувач має обліковий запис

Кінець таблиці 2.3

UC-2	**Увійти в систему**
Результат виконання	Користувач авторизований
Основний потік	1. Користувач ініціює вхід в систему і вводить електронну пошту та пароль 2. Система перевіряє наявність облікового запису та правильність облікових даних 3. Система надає користувачу доступ до функціоналу системи
Альтернативний потік	3а. Система виводить повідомлення про помилку, якщо ел. пошта або пароль неправильні, або користувач не зареєстрований

Таблиця 2.4 – Сценарій виходу користувача з системи

UC-3	**Вийти з системи**
Актор	Зареєстрований користувач
Передумови	Користувача авторизовано в системі
Результат виконання	1. Користувацька сесія завершена
Основний потік	1. Користувач ініціює вихід з системи 2. Система завершує поточну сесію 3. Система обмежує доступ до функціоналу для користувача

Таблиця 2.5 – Сценарій створення проєкту

UC-4	**Створити проєкт**
Актор	Авторизований користувач
Передумови	Користувача авторизовано в системі
Результат виконання	1. Проєкт створено 2. Користувача призначено власником проєкту
Основний потік	1. Користувач ініціює створення проєкту та вводить дані проєкту та підтверджує створення 2. Система перевіряє правильність заповнення полів 3. Система створює проєкт 4. Система призначає користувача Власником проєкту
Альтернативний потік	3а. Система виводить повідомлення про помилку, якщо введені у поля значення некоректні.

Таблиця 2.6 – Сценарій редагування даних проєкту

UC-5	**Редагувати дані проєкту**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Користувач є Власником проєкту
Результат виконання	Дані проєкту змінено
Основний потік	1. Користувач ініціює редагування проєкту та змінює дані проєкту 2. Система перевіряє правильність заповнення полів 3. Система оновлює дані проєкту
Альтернативний потік	2а. Система виводить повідомлення про помилку, якщо введені у поля значення некоректні

Таблиця 2.7 – Сценарій видалення проєкту

UC-6	**Видалити проєкт**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Користувач є Власником проєкту
Результат виконання	Проєкт видалено
Основний потік	1. Користувач обирає опцію видалення проєкту 2. Система відображає вікно для підтвердження видалення 3. Користувач підтверджує видалення 4. Система видаляє проєкт з системи
Альтернативний потік	3а. Користувач скасовує видалення. Перехід на крок 1

Таблиця 2.8 – Сценарій додавання учасника у проєкт

UC-7	**Додати Учасника у проєкт**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Користувач якого потрібно додати в проєкт має обліковий запис в системі
Результат виконання	Користувача додано до проєкту в якості Учасника

Кінець таблиці 2.8

UC-7	**Додати Учасника у проєкт**
Основний потік	1. Користувач відкриває список учасників проєкту і вводить електронну адресу користувача та погодинний рейт 2. Система перевіряє існування облікового запису і що цей користувач не є учасником проєкту 3. Система додає учасника в проєкт
Альтернативний потік	3а. Система виводить повідомлення про помилку, якщо обліковий запис з введеною електронною поштою не існує

Таблиця 2.9 – Сценарій редагування погодинної оплати учасника проєкту

UC-8	**Редагувати погодинну оплату Учасника**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Учасник доданий до проєкту
Результат виконання	Погодинна вартість роботи учасника змінена
Основний потік	1. Користувач відкриває список учасників проєкту і встановлює нову погодинну вартість роботи для Учасника 2. Система зберігає нове значення погодинної оплати
Альтернативний потік	-

Таблиця 2.10 – Сценарій видалення Учасника з проєкту

UC-9	**Видалити Учасника з проєкту**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Учасник доданий до проєкту
Результат виконання	Учасника видалено з проєкту
Основний потік	1. Користувач відкриває список учасників проєкту і вибирає опцію для видалення Учасника 2. Система відображає вікно для підтвердження видалення. 3. Користувач підтверджує видалення 4. Система видаляє Учасника з проєкту і присвоює зна
Альтернативний потік	-

Таблиця 2.11 – Сценарій перегляду стану проєкту

UC-10	**Переглянути стан проєкту**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Проєкт створено
Результат виконання	Дані по стану проєкту відображено
Основний потік	1. Користувач ініціює перегляд аналітики свого проєкту 2. Система відображає дані по проєкту і обчислює стан проєкту
Альтернативний потік	2а. Система відображає повідомлення про відсутність проєктів, якщо проєкт не було створено

Таблиця 2.12 – Сценарій створення завдання

UC-11	**Створити завдання**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Проєкт створено
Результат виконання	Завдання створено
Основний потік	1. Користувач ініціює створення завдання і вводить обов’язкову інформацію 2. Система перевіряє правильність заповнення полів 3. Система створює завдання і відображає його у списку завдань проєкту
Альтернативний потік	2а. Система відображає повідомлення про помилку, якщо обов’язкові поля не заповнені або введені у поля значення некоректні.

Таблиця 2.13 – Сценарій редагування завдання

UC-12	**Редагувати завдання**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує
Результат виконання	Завдання відредаговано
Основний потік	1. Користувач ініціює редагування завдання і вносить зміни. 2. Система перевіряє правильність заповнення полів

Кінець таблиці 2.13

UC-12	**Редагувати завдання**
	Система оновлює дані завдання
Альтернативний потік	2а. Система відображає повідомлення про помилку, якщо обов’язкові поля не заповнені або введені у поля значення некоректні

Таблиця 2.14 – Сценарій видалення завдання

UC-13	**Видалити завдання**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Завдання існує
Результат виконання	Завдання видалено
Основний потік	1. Користувач обирає опцію видалення завдання 2. Система відображає вікно для підтвердження видалення 3. Користувач підтверджує видалення 4. Система видаляє завдання з проєкту
Альтернативний потік	3а. Користувач скасовує видалення. Перехід на крок 1

Таблиця 2.15 – Сценарій зміни статусу завдання на дошці

UC-14	**Змінити статус завдання на дошці**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує
Результат виконання	Статус завдання і положення на дошці змінено
Основний потік	1. Користувач переміщує завдання з одного стовпця у інший 2. Система оновлює статус завдання і положення завдання на дошці
Альтернативний потік	2а. Система скасовує зміну статусу і положення завдання, якщо зміна неможлива. Перехід на крок 1

Таблиця 2.16 – Сценарій перегляду списку завдань

UC-15	**Переглянути список завдань**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Проєкт існує, завдання існує.
Результат виконання	Список завдань проєкту відображено
Основний потік	1. Користувач відкриває проєкт 2. Система відображає список завдань на Канбан-дошці
Альтернативний потік	2а. Система відображає порожню Канбан-дошку, якщо завдання відсутні

Таблиця 2.17 – Сценарій перегляду деталей завдання

UC-16	**Переглянути деталі завдання**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує.
Результат виконання	Деталі завдання відображено
Основний потік	1. Користувач натискає на завдання 2. Система відображає форму з деталями завдання у режимі перегляду
Альтернативний потік	-

Таблиця 2.18 – Сценарій призначення виконавця завдання

UC-17	**Призначити виконавця**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Завдання існує
Результат виконання	Учасника завдання призначено
Основний потік	1. Користувач ініціює створення або редагування завдання і обирає опцію вибору виконавця завдання 2. Система відображає список доступних виконавців які є в проєкті 3. Користувач обирає виконавця зі списку і зберігає завдання 4. Система оновлює інформацію по завданню

Кінець таблиці 2.18

UC-17	**Призначити виконавця**
Альтернативний потік	-

Таблиця 2.19 – Сценарій семантичного пошуку завдання

UC-18	**Знайти завдання за змістом**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує.
Результат виконання	Результати пошуку завдання відображено
Основний потік	1. Користувач вводить запит у поле для пошуку і ініціює пошук 2. Система відображає список завдань які з вірогідністю більше 55% відповідають пошуковому запиту
Альтернативний потік	2а. Система відображає повідомлення про відсутність результату пошуку, якщо збігу за запитом не знайдено

Таблиця 2.20 – Сценарій для пошуку дублікатів завдань

UC-19	**Знайти дублікати завдань**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує.
Результат виконання	Результати пошуку завдання відображено
Основний потік	3. Користувач вводить запит у поле для пошуку і ініціює пошук 4. Система відображає список завдань які з вірогідністю більше 65% відповідають пошуковому запиту
Альтернативний потік	2а. Система відображає повідомлення про відсутність результату пошуку, якщо збігу за запитом не знайдено

Таблиця 2.21 – Сценарій для отримання інтерпретації завдання

UC-20	**Отримати спрощену інтерпретацію суті завдання**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі, Завдання існує.

Кінець таблиці 2.21

UC-20	**Отримати спрощену інтерпретацію суті завдання**
Результат виконання	Підсумок суті завдання відображено
Основний потік	1. Користувач відкриває завдання в режимі перегляду і вибирає опцію інтерпретації 2. Система відображає короткий підсумок суті завдання
Альтернативний потік	-

Таблиця 2.22 – Сценарій для вказування фактичного часу виконання завдання

UC-21	**Вказати фактичний час**
Актор	Власник проєкту/Учасник
Передумови	Користувача авторизовано в системі Завдання існує
Результат виконання	Фактичний час завдання змінено
Основний потік	1. Користувач відкриває завдання в режимі редагування і змінює значення фактичного часу 2. Система оновлює інформацію про завдання
Альтернативний потік	2a. Система виводить повідомлення про помилку, якщо введені у поля значення некоректні

Таблиця 2.23 – Сценарій для генерації тижневого звіту по проєкту

UC-22	**Згенрувати тижневий звіт**
Актор	Власник проєкту
Передумови	Користувача авторизовано в системі Проєкт існує Вебхук на Slack-канал додано до проєкту
Результат виконання	Звіт по поєкту відправлено в Slack-канал
Основний потік	1. Користувач відкриває список проєктів і обирає опцію для генерації звіту по проєкту 2. Система здійснює необхідні обчислення для генерації звіту і надсилає звіт у Slack-канал
Альтернативний потік	2a. Система виводить повідомлення про помилку, якщо вебхук на Slack-канал не вказано у записі проєкту

2.2 Вимоги до програмного продукту

Беручи до уваги визначені класи користувачів, сценарії використання і результати дослідження функціональних можливостей конкурентів та, визначено функціональні і нефункціональні вимоги до системи. Функціональні вимоги до системи наведено в таблиці 2.24.

Таблиця 2.24 – Функціональні вимоги до системи

№ п/п	**Опис вимоги**
ФВ-01	Система повинна дозволяти гостю зареєструватися за допомогою унікальної електронної пошти та пароля
ФВ-02	Система повинна автентифікувати користувача на основі введених електронної пошти та пароля
ФВ-03	Система повинна забезпечувати розмежування доступу до даних проєкту на основі ролей (Власник, Учасник)
ФВ-04	Система повинна дозволяти користувачеві створювати один або більше проєктів
ФВ-05	Система повинна дозволяти Власникові редагувати інформацію про проєкт
ФВ-06	Система повинна дозволяти Власникові видалити проєкт і супутню інформацію
ФВ-07	Система повинна забезпечувати ізоляцію даних різних проєктів
ФВ-08	Система повинна дозволяти Власнику проєкту запрошувати учасників до робочого простору за їх електронною адресою
ФВ-09	Система повинна дозволяти Власнику і Учаснику проєкту створення завдань з атрибутами: назва, опис, статус, пріоритет, виконавець, плановий час виконання, орієнтовна дата завершення завдання
ФВ-10	Система повинна дозволяти Власнику призначати завдання конкретному Учаснику проєкту
ФВ-11	Система повинна присвоювати завданню статус Backlog за замовчуванням при створенні завдання
ФВ-12	Система повинна дозволяти користувачеві присвоювати будь-який з доступних статусів при створенні або редагуванні (Backlog, ToDo, In Progress, In Review, Done)
ФВ-13	Система повинна дозволяти користувачеві зміну статусу завдання шляхом його переміщення між колонками на дошці Канбан
ФВ-14	Система повинна дозволяти Учаснику редагувати дані лише тих завдань, де він є Виконавцем
ФВ-15	Система повинна дозволяти видалення завдань тільки Власнику проєкту
ФВ-16	Система повинна забезпечити можливість організувати процес роботи на проєкті за допомогою Канбан дошки

Кінець таблиці 2.24

№ п/п	Опис вимоги
ФВ-17	Система повинна реалізувати семантичний пошук завдань на основі контексту назви та опису завдання (без необхідності точного співпадіння)
ФВ-18	Система повинна забезпечити за запитом від користувача виявлення дублікатів завдань на основі аналізу їх змісту (назви та опису)
ФВ-19	Система повинна відображати список потенційних дублікатів після пошуку, їхній ідентифікатор, назву і статус кожного дублікату завдання
ФВ-20	Система повинна забезпечувати генерацію звіту по проєкту з основними показниками за останні 7 календарних днів (к-сть виконанх завдань, використані години, бюджет, інформація про стан проєкту)
ФВ-21	Система повинна забезпечувати інтеграцію з месенджером Slack для надсилання звіту по проєкту за останні 7 днів у вказаний канал Slack.
ФВ-22	Система повинна відображати статус проєкту з ключовими показниками проєкту: прогрес, використаний бюджет, аналіз здобутої цінності, динаміка виконання завдань, навантаження команди, SPI, CPI, загальний стан проєкту.
ФВ-23	Система повинна забезпечувати інтерпретацію суті завдання без технічних подробиць за запитом користувача і формувати текстове пояснення для завдання

Для ефективного керування завданнями передбачено можливість гнучкої зміни статусів користувачами. Процес для роботи зі статусами завдань показано на рисунку 2.2.

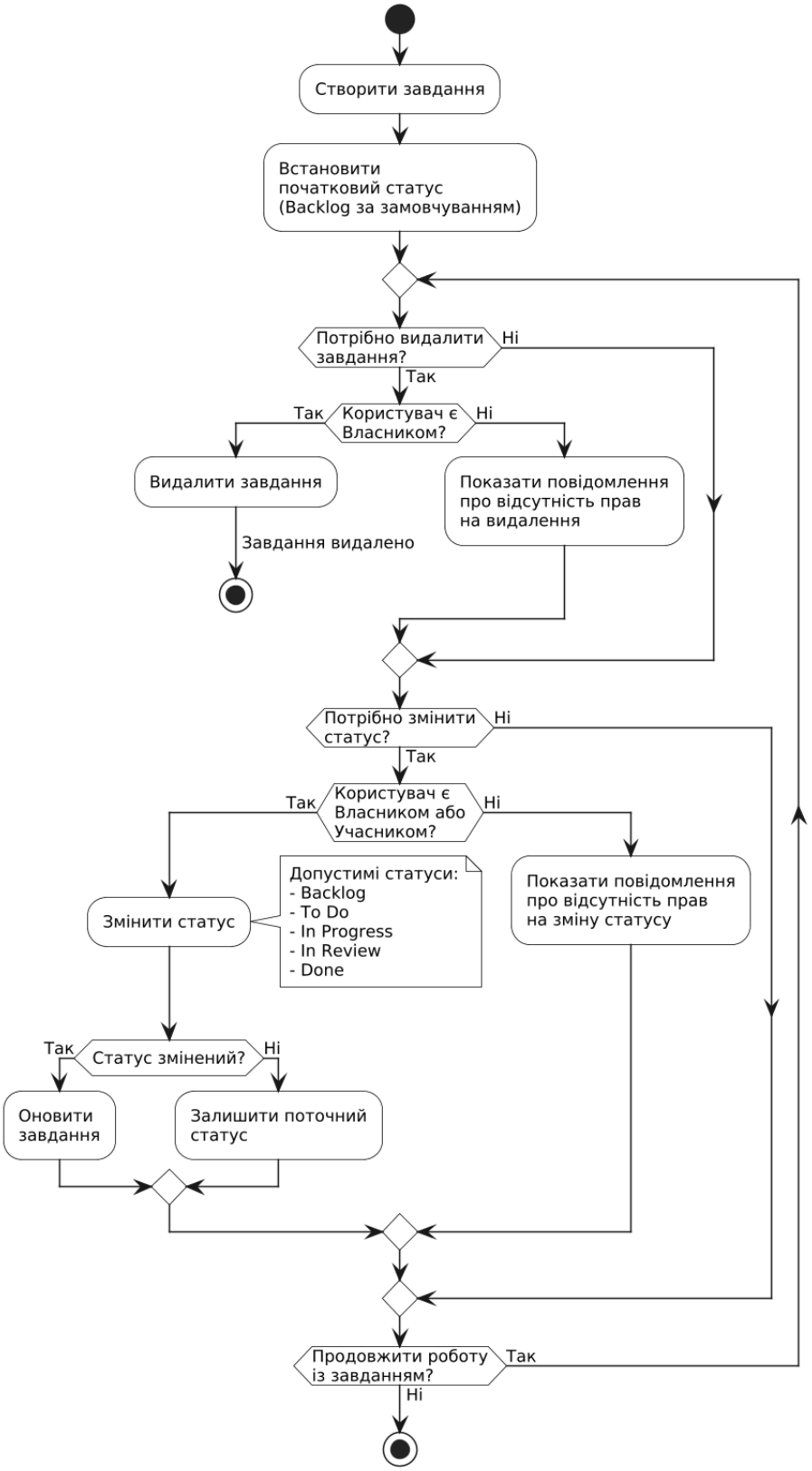

Рис 2.2 – Процес зміни статусів завдання

Визначення нефункціональних вимог до системи здійснюється відповідно до характеристик якості описаних у стандарті ISO/IEC 25010 забезпечує оптимальну якість програмного продукту. Таблиця 2.25 містить список нефункціональних вимог до системи.

Таблиця 2.25 – Нефункціональні вимоги до системи

№ п/п	**Опис вимоги**
НФВ-01	Система повинна інформувати користувача про системні події (успішне виконання дії, помилка, підтвердження дії) за допомогою попереджень у інтерфейсі користувача
НФВ-02	Час виконання семантичного пошуку повинен бути не більше 45 секунд при обсязі до 1 000 завдань у межах проєкту
НФВ-03	Веб-інтерфейс повинен коректно відображатися на пристроях із шириною екрана від 1280 px до 3840 px і висотою від 960 px
НФВ-04	Система повинна забезпечувати горизонтальне масштабування для розподілу навантаження, не менше 3-х екземплярів сервісу який масштабується
НФВ-05	Час життя токена автентифікації повинен бути обмежений (24 години) та має підтримуватися механізм оновлення доступу
НФВ-06	Система повинна забезпечувати час відгуку системи на CRUD операції повинен становити ≤ 1 секунди для 95% запитів при номінальному навантаженні при 100 одночасних користувачів
НФВ-07	Система повинна зберігати паролі користувачів у вигляді криптографічного хешу за допомогою алгоритму bcrypt
НФВ-08	Система повинна працювати без збоїв при збереженні до 10 000 завдань у базі даних
НФВ-09	Система повинна забезпечувати доступність у 99,9% часу цілодобово протягом календарного року
НФВ-10	У разі збою компонента системи (наприклад, семантичного пошуку) основний функціонал системи для керування завданнями повинен залишатися працездатним (наприклад, авторизація користувача в системі)
НФВ-11	Система повинна відображати результати семантичного пошуку при співпадінні більше 55%
НФВ-12	Система повинна забезпечувати розгортання як хмарний сервіс у контейнерах

В результаті декомпозиції вимог та їх документування визначено функціонал який система повинна мати для ефективного керування проєктами і завданнями [56].

2.3 Визначення архітектурного стилю системи

Для задоволення функціональних і нефункціональних вимог до системи необхідно визначити її архітектуру. Згідно міжнародного стандарту ISO-IEC-IEEE-42010-2022, поняття «архітектура» має наступне значення: «це фундаментальні концепції або властивості сутності в середовищі функціонування та керівні принципи для реалізації і еволюцій цієї сутності, а також пов’язаних з нею процесів життєвого циклу»[57].

Архітектура програмного забезпечення в сучасній інтерпретації являє собою сукупність наступних складових [58, 59]:

- Структура яка визначає архітектурний стиль і форму системи (шарова, клієнт-сервер, мікросервісна і т.п.).
- Архітектурні характеристики які визначаються можливостями які повинна підтримувати система (доступність, надійність, масштабованість та інші).
- Архітектурні рішення які реалізовуються у вигляді правил, що лежать в основі роботи компонентів системи).
- Принципи проєктування (директиви які визначають взаємодію компонентів).

Зі зміною або появою нових бізнес потреб які має задовільняти система виникає необхідність архітектурних змін в процесі експлуатації системи в процесі життєвого циклу розробки програмного забезпечення (SDLC). В свою чергу актуальність інформації яка описує архітектуру системи є критичним для подальшої її модифікації і нововведень.

Ієн Сомервіль (Ian Sommerville) приводить дані щодо тривалості життя інформаційних систем на основі своїх спостережень, де зазначає, що добре спроєктовані системи можуть експлуатуватися приблизно від 10 (для бізнес систем) до понад 30 років (для військових систем) [60].

Саме по собі поняття архітектури є абстракцією і для того щоб матеріалізувати це використовується опис архітектури (architecture description), що дає можливість зрозуміти яким чином влаштована система і оцінити чи така реалізація дозволяє [57]:

- задовольнити вимоги до системи з точки зору зацікавлених осіб;
- зрозуміти ключові особливості системи і її поведінки;
- оцінити середовище в якому функціонує система;
- аналізувати припущення та обмеження які впливатимуть на можливості модифікації системи;
- дослідити підгрунтя архітектурних рішень і передбачити можливі наслідки або альтернативні рішення.

Підґрунтям для застосування певного архітектурного стилю є вимоги до системи і для цього систему необхідно правильно декомпозувати. Модульність системи дозволяє розділити систему на логічні частини. В свою чергу компонент системи буде являти собою фізичне втілення модуля. В процесі розділення варто враховувати наступні залежності:

- зв’язність (cohesion) - характеристика рівня пов’язаності частин компонента при виконанні роботи над операціями в межах компонента системи;
- зв’язаність (coupling) - характеристика залежностей між окремими компонентами, де метою є знизити рівень залежностей;
- конасценція (connascence) - визначає рівень залежності між компонентами у випадках коли зміна всередині одного компоненту вимагає обов’язкової зміни в іншому компоненті (сила, локальність, ступінь).

Перше на що варто звернути увагу при аналізі архітектурних властивостей - це основні завдання предметної області і умови роботи системи. Кількість користувачів і можливості колаборації в реальному часі, особливо при роботі

десятків або сотень команд над різними проєктами, що буде створювати стрибки навантаження у певні проміжки часу, що в свою чергу не повинно призводити до зниження швидкодії системи, збоїв або недоступності функціоналу системи. Таким чином масштабованість і адаптація під зміни навантаження повинні регулюватися системою.

За допомогою предметно-орієнтованого проєктування розбиття архітектури по завданнях предметної області можна виділивши наступні предметні області [61]:

- Облікові записи користувачів
- Керування проєктами та завданнями
- Аналіз завдань
- Звітність і аналітика

Таким чином можна виділити логічні модулі або так звані обмежені контексти Domain-Driven Development (bounded context) [61]:

1. Модуль авторизації - відповідає за життєвий цикл облікового запису користувача, керування правами облікового запису та доступом до системи.
2. Модуль керування завданнями - відповідає за керування проєктами і завданнями. Цей модуль є ядром системи, оскільки містить основний функціонал який визначає призначення системи.
3. Модуль аналізу завдань - відповідає за перетворення завдань у формат зручний для семантичного пошуку, ідентифікації дублікатів завдань, формування резюме для нетехнічних користувачів.
4. Модуль звітності - збір інформації про зміни в проєкті, обчислення для ідентифікації прогресу, стану проєкту та прогнозу для керівника.

Для визначення типу архітектури (монолітна або розподілена) і подальшого вибору архітектурного стилю необхідно прийняти до уваги набори атрибутів якості, наведених в таблиці 2.5, для забезпечення ефективного функціонування модулів системи та їх еволюціонування.

Таблиця 2.26 – Ключові атрибути якості модулів

Модуль	**Характеристики які мають вплив на архітектуру**
Модуль авторизації	Конфіденційність, надійність, доступність
Модуль керування завданнями	Продуктивність CRUD операцій, цілісність даних, доступність, модифікованість, тестованість, відмовостійкість
Модуль аналізу завдань	Масштабованість (високі навантаження для роботи інтелектуальних функцій), адаптивність
Модуль звітності	Асинхронність, продуктивність

Беручи до уваги те, що кожен з модулів має різні потреби для свого функціонування об’єднання всіх модулів у моноліт матиме негативні наслідки для роботи системи і досвіду користувача (масштабування, доступність, очевидну технологічну гетерогенність у випадку аналітичного модуля та ін.). В свою чергу розподілена архітектура дозволяє ізолювати збої в модулях, підбирати технології під конкретні завдання, незалежне розгортання і масштабування.

Дослідження розподілених архітектурних стилів дозволяє згрупувати їх основні архітектурні властивості та надати оцінку [58]. Деталі порівняння наведено на рисунку 2.3. Оцінка від 1 до 5 позначає наскільки сильно архітектурна властивість підтримується відповідним архітектурним стилем.

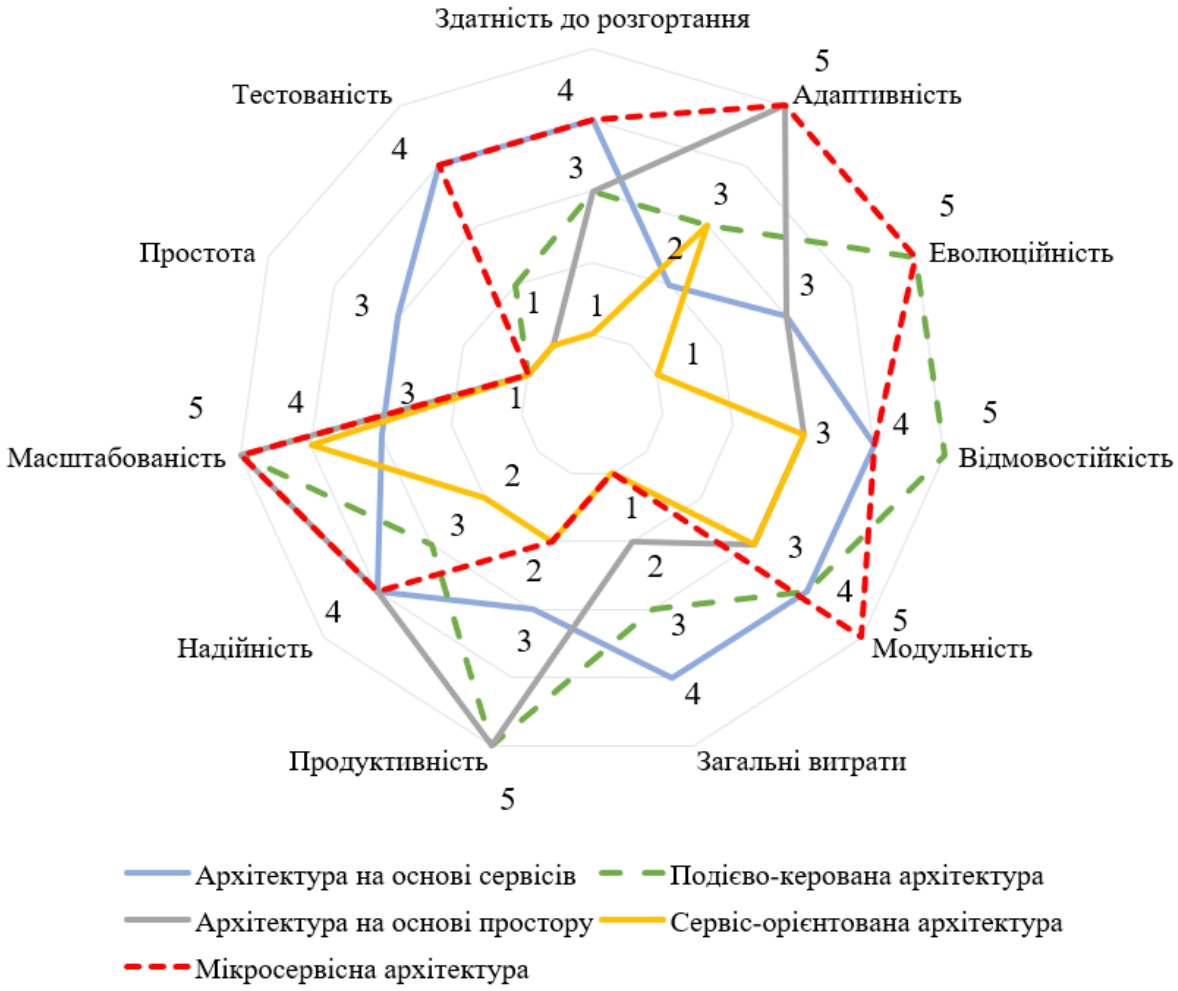

Рис 2.3 – Розподіл архітектурних властивостей відповідно до архітектурного стилю.

Порівняння підсумкових оцінок по архітектурних властивостях відповідно до архітектурного стилю показано на рисунку 2.4.

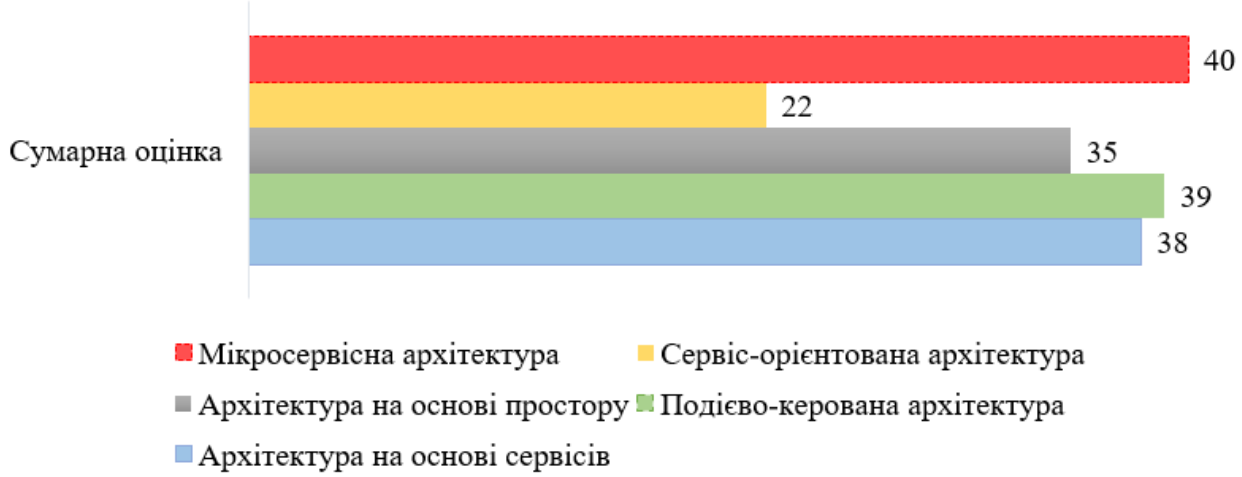

Рис 2.4 – Підсумкова оцінка по архітектурних стилях

Зважаючи на оцінку властивостей кожного стилю і вимог до системи, найбільш доцільним є використання стилю мікросервісної архітектури для побудови системи. Варто зазначити, що ключовими аспектами вибору є масштабованість, модульність, надійність, відмовостійкість, адаптивність та тестованість.

До принципів побудови системи з мікросервісною архітектурою відносять наступні [62]:

- Виокремлення і ізоляція функціоналу за бізнес контекстом в межах сервісу
- Ізоляція даних кожного сервісу
- Слабке зв’язування - коли зміни в одному сервісі не вимагають змін в іншому сервісі
- Сильна зв’язність - коли код в сервісі стосується тільки цього сервісу
- Технологічна гетерогенність в реалізації сервісів
- Незалежне розгортання елементів системи і їх масштабування

Вибір мікросервісної архітектури для розробки системи є оптимальним вибором серед розподілених архітектурних стилів, оскільки забезпечує створення гнучкої основи для подальшого розширення функціоналу системи, одночасно дозволяючи гарантувати високий рівень безпеки та відмовостійкості.

За принципом розподілу відповідальності виділено наступні мікросервіси, кожен з яких виконує певні функції відповідно до бізнес контексту:

1. Сервіс авторизації - виконує керування обліковими записами користувачів та контролю доступу до системи.
2. Сервіс керування завданнями - виконує керування завданнями та проєктами
3. Сервіс аналізу завдань - виконує інтелектуальну обробку даних завдань, виконує семантичний пошук, ідентифікацію потенційних дублікатів і інтерпретації завдань для користувачів без технічних знань.
4. Сервіс звітності - виконує формування аналітики для оцінки стану проєкту, а також підсумкових звітів для подальшої передачі в Slack
5. API-шлюз - виконує маршрутизацію запитів від клієнтського застосунку до відповідних мікросервісів і балансування навантаження.
6. Сервіс клієнтського застосунку - надає користувачеві функціонал для взаємодії з системою через графічний інтерфейс.
7. Інфраструктурні компоненти - виконують завдання з забезпечення асинхронної взаємодії з сервісами та передачі даних між сервісами.

Діаграму розгортання сервісів показано на рисунку 2.5.

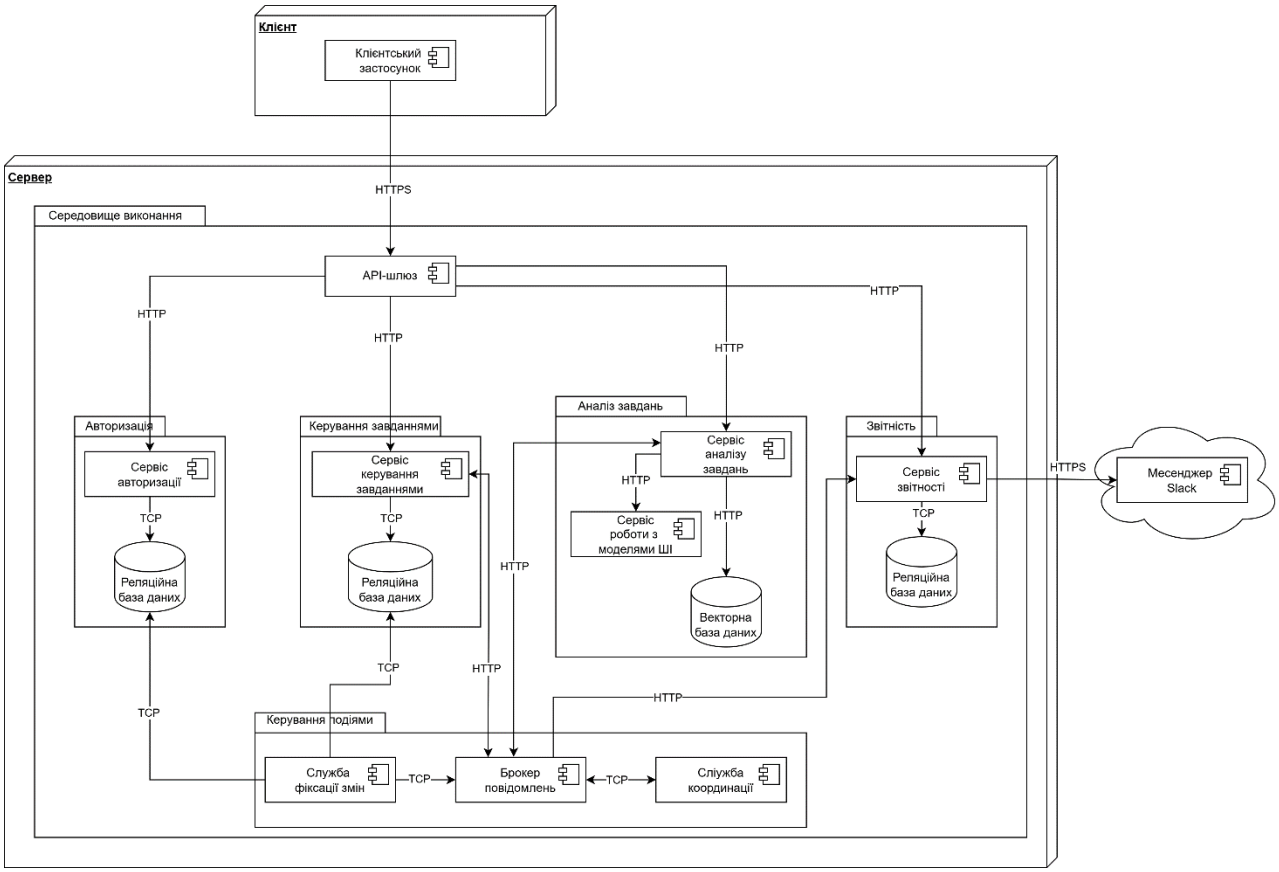

Рисунок 2.5 – Діаграма розгортання системи

2.4 Проєктування баз даних мікросервісів

Сервіс авторизації оперує набором даних необхідних для ідентифікації користувача і безпеки його облікових даних. Основною і єдиною сутністю буде користувач і його атрибути. Деталі показано на діаграмі сутності-зв’язку на рисунку 2.6.

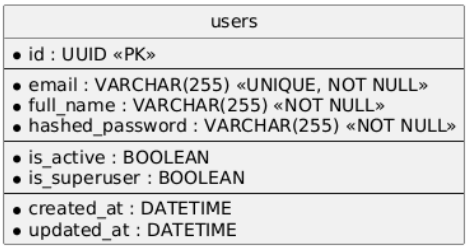

Рисунок 2.6 – Діаграма сутність-зв'язок бази даних сервісу авторизації

Завдяки такій структурі бази даних, сервіс авторизації забезпечує функціонал зі створення нового облікового запису користувача, перевірки даних під час автентифікації і авторизації, надання іншим компонентам системи можливість ідентифікувати користувача в разі необхідності. Таблиця users містить інформацію про системний ідентифікатор облікового запису, електронну пошту, повне ім’я, пароль у вигляді хешу та додаткові атрибути (дата створення, зміни, ознака активності облікового запису та наявності права суперюзера).

Сервіс керування завданнями оперує набором даних необхідних для створення і керування проєктами та завданнями. Таблиця user_cache призначена для зберігання і оновлення даних про користувачів системи, які використовуються в сервісі для створення проєктів, завдань, пошуку і призначення виконавців. Таблиця projects призначена для зберігання інформації про створені проєкти і їх деталі такі як назву, опис, власника проєкту, бюджет, дату початку і завершення проєкту, статус проєкту та вебхук для передачі звітності у месенджер Slack. Таблиця project_members призначення для встановлення приналежності команди користувачів до проєкту, їхньої ролі (повноваження) на проєкті та погодинної вартості роботи. Таблиця tasks призначена для збереження інформації про завдання. Один користувач мати 0 або більше проєктів, може бути бути учасником 0 або більше проєктів, може бути виконавцем 0 або більше завдань. Проєкт може мати 1 або більше учасників, 0 або більше завдань. Завдання має 0 або 1 одного учасника. Відображення таблиць та їх взаємозв'язків показано на рисунку 2.7.

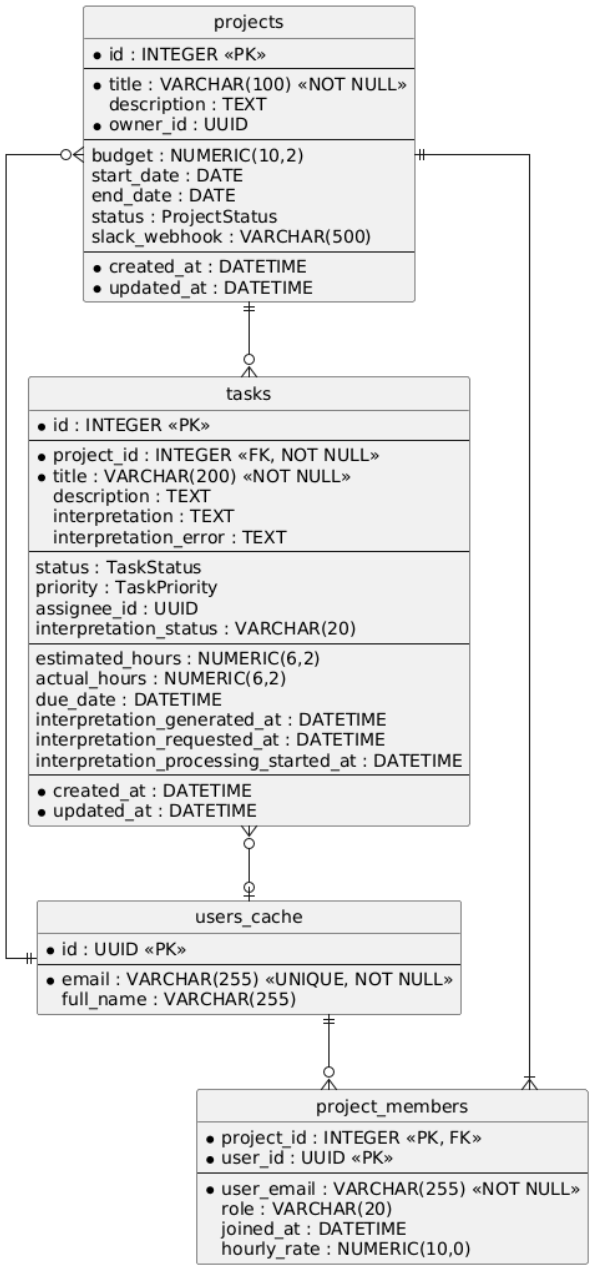

Рисунок 2.7 – Діаграма сутність-зв'язок бази даних сервісу керування завданнями

Суцільною лінією позначені зв’зяки між таблицями з зовнішніми ключем (foreign key), а пунктирною позначені логічні зв’язки з таблицею user_cache яка наповнюється даними користувачів асинхронно з сервісу авторизації, що є джерелом істини.

Сервіс аналізу оперує набором даних необхідних для створення векторів завдань для забезпечення функціоналу з семантичного пошуку завдань та

пошуку дублікатів. Колекція tasks_vectors дозволяє реалізувати цю логіку завдяки створення векторної точки в певному просторі на основі тексту завдання (назва та опис). Деталі колекції наведено в таблиці 2.27.

Таблиця 2.27 – Структура колекції векторної бази даних

Поле колекції	**Тип даних**	**Призначення**
point_id	integer	Унікальний ідентифікатор векторної точки яка визначає завдання в колекції
vector	float	Векторне представлення змісту завдання
distance	cosine	Метрика порівняння векторів
payload.id	integer	Ідентифікатор завдання для подальшої обробки результатів пошуку
payload.project_id	integer	Ідентифікатор проєкту
payload.title	text	Назва завдання
payload.description	text	Опис завдання
payload.status	text	Статус завдання
payload.status_norm	keyword	Нормалізоване значення статусу завдання (для фільтрації результатів пошуку)
payload.priority	text	Пріоритет завдання
payload.priority_norm	keyword	Нормалізоване значення пріоритету (для фільтрації результатів пошуку)
payload.assignee_id	UUID	Ідентифікатор виконавця завдання
payload.estimated_hours	numeric	Оцінка завдання
payload.actual_hours	numeric	Фактичні години
payload.due_date	datetime	Дедлайн завдання
payload.created_at	datetime	Дата і час створення завдання
payload.updated_at	datetime	Дата і час останнього оновлення завдання

Сервіс звітності оперує набором даних для збереження і подальшого обчислення резуль аналітичних обчислень щодо стану проєкту.

Таблиця project_snapshots призначена для зберігання інформації про поточний стан проєктів. Таблиця user_snapshots призначена для зберігання інформації про

користувачів (яку містить сервіс авторизації) і відображення відповідної інформації при формуванні звіту. Таблиця project_member_snapshots призначена для збереження інформації про учасників проєктів і погодинної вартості роботи учасника на проєкті. Таблиця task_snapshots містить інформацію про актуальний стан завдань і атрибути які використовуються для розрахунків при формуванні звітності. Таблиця project_daily_metrics зберігає агреговані метрики по проєкту за певну дату. Структуру бази даних сервісу звітності наведено на рисунку 2.8.

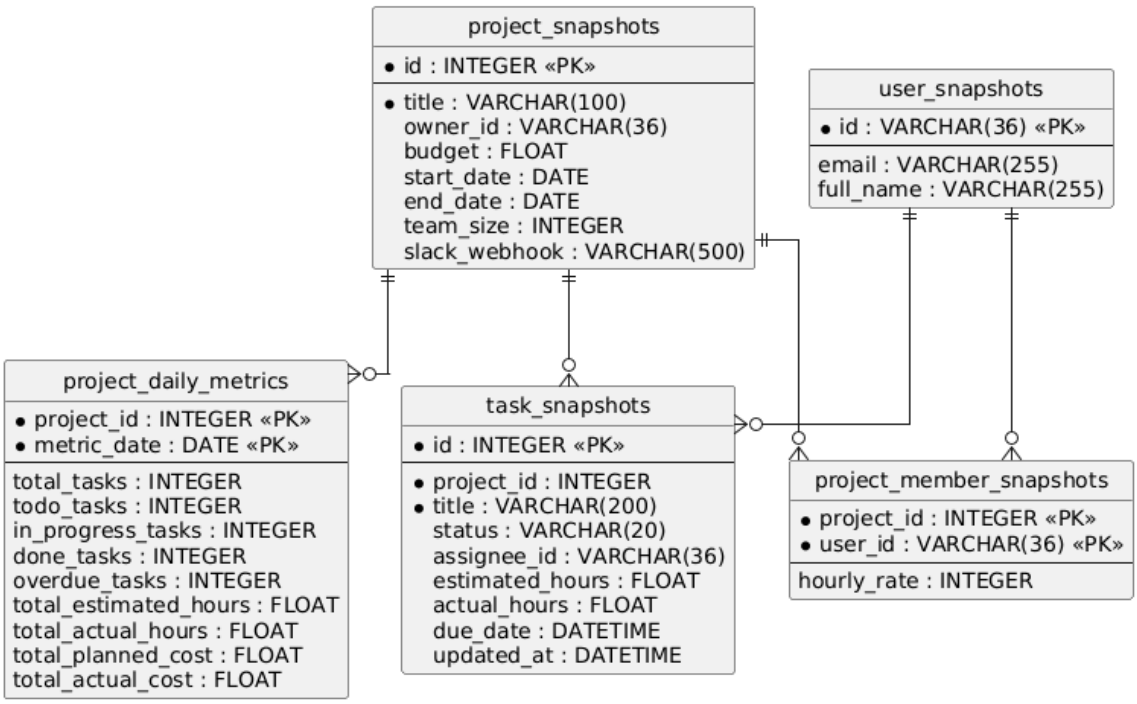

Рисунок 2.8 – Діаграма сутність-зв'язок бази даних сервісу звітності

Спроєктовані моделі даних для кожного сервісу системи дозволяють забезпечити необхідний функціонал для повноцінної роботи сервісів для керування завданнями, звітності та роботі інтелектуальних функцій.

2.5 Обґрунтування технологічних рішень для розробки системи

Для взаємодії користувача з системою, гарантування безпеки між бекендом та фронтендом, балансуванням навантаження обрано патерн API-шлюзу. API-шлюз виконує роль проксі-сервера який виконує маршрутизацію запитів. Порівняння сучасних технологій наведено в таблиці 2.28.

Таблиця 2.28 – Порівняння API-шлюзів

Параметр оцінювання	**Nginx [63]**	**Traefik [64]**	**Kong [65]**
Типове середовище	Веб сервер і reverse-proxy. Використовується як edge-proxy перед бекендами/SPA	Використовується для роботи у оркестраторах через інтеграцію з їхніми API	Великі мікросервісні платформи з вимогами до SSO, квотування
Модель конфігурації	Конфігурація в текстових файлах, наявність hot-reload, багато низькорівневих параметрів	Статична та динамічна конфігурація, оновлює маршрути без рестарту	Конфігурація через адмін-API або GUI. Потребує БД для роботи
Сильні сторони	Дуже висока продуктивність і масштабованість, тонкі налаштування	Авто-виявлення сервісів, авто-оновлення сертифікатів , мінімум ручних конфігурацій	Багато плагінів, безпеки, фокусується на керування саме АР
Слабкі сторони	Готовий до роботи у якості веб-серверу/reverse-proxy «з коробки». Конфігурацію потрібно будувати самостійно	Орієнтований на нативні хмарні сценарії роботи. Продуктивність та глибина налаштувань гірша в порівнянні з Nginx	Складне розгортання та експлуатація, потрбіна БД та керування міграціям

Оскільки система є досить невеликою, є потреба в простій реалізації, гнучкому налаштуванню, в якості API-шлюзу було обрано Nginx.

Для розробки бекенд частини мікросервісів обрано мову програмування Python. Такий вибір зумовлений простим синтаксисом, відсутності необхідності компіляції коду, продуктивністю і широким застосуванням в сфері машинного навчання та штучного інтелекту [66].

Порівняно з Java, C#, Go, швидкість розробки на Python є кращою, що дозволяє отримувати результати перевірок гіпотез у короткі проміжки часу. Серед доступних Python фреймворків проаналізовано Flask, Django та FastAPI та результати їхнього тестування [67-69]. У таблиці 2.29 показано оцінку по кожній категорії тестів і підсумкову оцінку.

Таблиця 2.29 – Порівняння Python фреймворків

Тип тесту	**FastApi**	**Flask**	**Django**
JSON	171,055	63,026	72,039
1 запит	66,185	34,217	20,021
20 запитів	13,022	6,647	1,599
Fortunes (комплексний тест) [70]	52,080	23,136	15,085
Оновлення	5,926	1,327	840
Простий текст	159,445	83,398	78,132

Серед доступних на Python фреймворків обрано FastAPI, оскільки фреймворк прискорює розробку, має готові рішення типових завдань і інтегровані компоненти які сприяють розробці [66]. До переваг FastAPI відносять наявність автоматичної документації за OpenAPI стандартом, висока продуктивність за рахунок асинхронного серверного шлюзового інтерфейсу, інтеграція з бібліотекою Pydantic для зручної валідації даних, ін’єкції залежностей для уникнення дубілювання коду та слабкою зв’язністю.

При виборі технології для побудови фронтенд частини варто звернути увагу на основні сценарії використання і призначення системи. Ключовим аспектом є ініціація і організація проєкту, тому при виборі доцільно керуватися найбільш зручним і оптимальним способом для цієї мети. В зв’язку з невеликими розмірами екранів мобільних пристроїв на практиці мобільна версія використовується у більшості випадків для якихось незначних змін в завданнях, типу зміни статусу завдання, або зміни виконавця, тощо. Повноцінне створення опису завдання з критеріями приймання, припущеннями та посиланнями на

додаткові ресурси є скоріше вийнятком ніж правилом, особливо у випадках активного наповнення беклога завданнями. Тому реалізація мобільної версії не є пріоритетною для даної системи.

Вибір з поміж нативного застосунку, гібридного або веб застосунку грунтується на декількох критеріях, а саме: простота розробки і підтримки, підтримка роботи як на десктопних так і на мобільних розширеннях.

Проаналізовані технологічні рішення включають Flutter, Vue, React. Vue позиціонується як прогресивний фреймворк для найбільш вживаних випадків який є гнучким у налаштуваннях та у впровадженні за рахунок виокремлення компонентів [71].

Офіційна документація React зазначає, що технологія є бібліотекою, якій бракує визначення того як має відбуватися маршрутизація та передача даних. Рекомендується використовувати додатковий фреймворк типу Next.js для того щоб закрити прогалини у самому React [72].

Flutter в свою чергу пропонує спільну кодову базу для веб, мобільного та десктоп застосунків. Базуючись на мові програмування Dart та сприймаючи UI елементи як віджети, Flutter має ряд переваг у випадку mobile-first підходу [73].

Вибір Vue в якості фреймворку для фронтенду є оптимальним з ряду причин:

1. Базується на стандартних технологіях HTML, SCC, Javascript, що спрощує розробку і підтримку продукту
2. Використовує компонентну модель і зручні для побудови інструменти
3. Є повноцінним фреймворком для розробки структурованого інтерфейсу
4. Присутня можливість зробити інтерфейс адаптивним під різні пристрої за необхідності
5. Офіційна підтримка маршрутизацій і керування станом є перевагою при масштабуванні.

Відповідно до інформації яка буде зберігатися і опрацьовуватися, для кожного логічного модуля системи доцільно вибрати правильну базу даних,

використовуючи підхід багатомовне зберігання (polyglot persistence). Зважаючи на необхідність здійснювати семантичний пошук, у таблиці 2.30 було проаналізовано векторні бази даних та їх бенчмарки [74-76]. Для реалізації семантичного пошуку необхідно зберігати векторні подання у векторній базі даних та використовувати алгоритм наближеного пошуку найближчих сусідів для отримання результатів пошуку.

Таблиця 2.30 – Порівняння переваг та недоліків векторних баз даних

Векторна база даних	**Переваги**	**Недоліки**
Milvus	- Затримка 6 мс - Може працювати з десятками мільярдів векторів - Підтримка різних типів алгоритмів	- Високе споживання ресурсів системи навіть у режимі простою - Потребує розгортання і підтримки etcd, MinIO, Kafka для роботи з векторами.
Qdrant	- Висока продуктивність яка забезпечується низькою затримкою (~ 4мс). - Швидкість і якість пошуку за допомогою алгоритму ACORN - Легкість розгортання в контейнері і масштабування	- Низький набір допоміжних інструментів порівняно з іншими БД. - Необхідність ручної конфігурації параметрів
PostgreSQL з розширенням pgvector	- Затримка 5-50 мс - Вектори оновлюються одночасно в метаданими через один SQL запит	- Векторні запити конкурують за ресурси RAM та CPU з SQL запитами - Складно реалізувати горизонтальне масштабування

Вибір Qdrant є оптимальним, оскільки демонструє найкращий баланс між функціональністю і підтримкою в довготривалій перспективі.

Для модулів авторизації, керування завданнями, звітності доцільно застосувати патерн наявності бази даних підкожен мікросервіс який потребує збереження даних та керування ними, що дозволяє оптимізувати моделі даних для потреб сервісу і забезпучує слабку зв’язність між мікросервісами [77].

Оптимальним вибором у цьому випадку є реляційна СУБД, оскільки кожен модуль має чіткий перелік сутностей які не змінюються, є потреба в підтримці складних запитів, підтримує масштабування, стабільна робота під високими навантаженнями операцій читання і запису (транзакційна надійність). Порівняння переваг і недоліків популярних реляційних баз даних наведено в таблиці 2.31 [78-80].

Таблиця 2.31 – Порівняння переваг та недоліків реляційних баз даних

Реляційна база даних	**Переваги**	**Недоліки**
PostgreSQL	- Підтримка складних запитів - Розширюваність (власні типи, функції, оператори) - Забезпечення ACID - Механізм конкурентного доступу - Секціонування та логічні реплікації	- Потребує налаштування та моніторингу продуктивності - Потребує контролю підключень (пул підключень) - Складне горизонтальне масштабування - Складне початкове налаштування
MySQL	- Просте розгортання - Зручна для простих застосунків з фокусом на читання даних - Можливість масштабування реплік для читання	- Триваліша обробка складних запитів в порівнянні з PostgreSQL - Для CDC потрібні додаткові налаштування
MariaDB	- Синхронна реплікація через Galera-кластер - Багатопотокова архітектура - Підтримка транзакцій через InnoDB	- Galera-кластер потребує складного налаштування - Необхідно вибрати рушії сховищ під транзакційні навантаження - Функціональність залежить від рушія сховища

Аналіз нереляційної бази даних MongoDB в порівнянні з реляційними базами даних показав, що у MongoDB зв’зяки між даними реалізуються або через посилання між документами, або через вкладені документи, що викликає надлишкове дублювання інформації і у разі необхідності змінити дані потрібно буде виконувати оновлення в багатьох місцях. При необхідності змінити декілька сутностей одночасно (проєкт, завдання, команда і т.п.) зростає

складність запиту і знижується продуктивність, що є критичним недоліком для системи керування проєктами [81].

Відповідно до результатів аналізу баз даних вибір PostgreSQL є оптимальним для побудови поточної системи, завдяки наявності розвинутих механізмів конкурентного доступу, можливостей логічної реплікації та розділення таблиць на секції, робота з складними запитами, робота з структурованими і напівструктурованих даними і є оптимальною для розробки системи і її розвитку [82].

Важливим аспектом взаємодії компонентів системи є їх взаємодія. Для процесу автентифікації користувача, створення завдання, перегляд та його оновлення, а також для інших процесів які критично впливають на користувацький досвід доцільно застосувати патерн синхронної взаємодії (Request-Response Communication [62]). Цей патерн спрямований на передачу запитів і відповідей від фронтенд чани до API-шлюзу та бекенду відповідно. Такий механізм дозволяє фронтенд частині очікувати поки не отримає синхронну відповідь, в той час коли з’єднання залишається відкритим.

Для фонових операцій та операцій які не вимагають негайної відповіді, забезпечення гарантованої доставки даних та можливості обміну даними асинхронно вирішено застосувати поєднання мікросервісіної архітектури з подієво-орієнтованим архітектурним стилем (event-driven architecture) [83]. Для забезпечення роботи цього патерну обрано сервіс Apache Kafka, який застосовується для потокової передачі даних завдяки своїй здатності обробляти великі об’єми даних та масштабуванню. Такий механізм буде доречним під час передачі даних між сервісами, як наприклад між сервісом завдань і сервісом аналітики, де сервіс аналітики буде готувати звітність і по мірі готовності зможе віддати відповідь користувачеві без блокування інших сценаріїв використання системи.

Для того щоб знизити залежність між сервісами які створюють якусь подію і передачею сервісу було обрано патерн захоплення змін (Change Data Capture

[83]), який дозволяє не створювати додаткову логіку для публікації подій і зменшує зв’язність між сервісами. Для реалізації цього підходу обрано Debezium який є спеціалізованим сервісом для відстеження змін в базах даних і передача інформації про зміни у вигляді потоку подій. Наприклад, при створенні завдання або його зміні в сервісі завдань Debezium публікує інформацію про зміну в топік Kafka, після чого сервіси які відслідковують такі зміни зможуть зчитати її асинхронно.

Для забезпечення інкапсуляції логіки виконання, ізоляції процесів та ефективного керування ресурсами використано патерн розгортання мікросервіса у вигляді контейнера (Service instance per container [84]). Для контейнеризації обрано Docker, який дозволяє швидко запустити або зупинити сервіс, зручний для подальшої інтеграції CI/CD і відносно простий при налаштуванні інфраструктури системи за рахунок файлу конфігурації.

З метою забезпечення векторних перетворень для семантичного пошуку вибрано Ollama у якості локального середовища виконання для моделі яка буде здійснювати ембедінги та моделі яка буде відповідальна за інтерпретацію тексту. Такий вибір зумовлений необхідністю мати високий рівень конфіденційності і не передавати дані стороннім сервісам типу OpenAI або Azure AI. Також важливим є вартість використання сервісу, що у випадку є Ollama є найнижчою порівняно з хмарними сервісами, доступність сервісу, яка у випадку з Ollama контролюється повністю, особливо у разі необхідності збільшити обчислювальні ресурси або перерозподілити виконання між CPU і GPU [85].

При порівнянні моделей для векторизації було враховано можливість роботи локально разом з Ollama, підтримка багатьох мов та якісь опрацювання вхідного тексту. Аналіз по показнику MRR, який показує наскільки високо модель ранжує перший релевантний до запиту результат, показано на рисунку 2.9 [86].

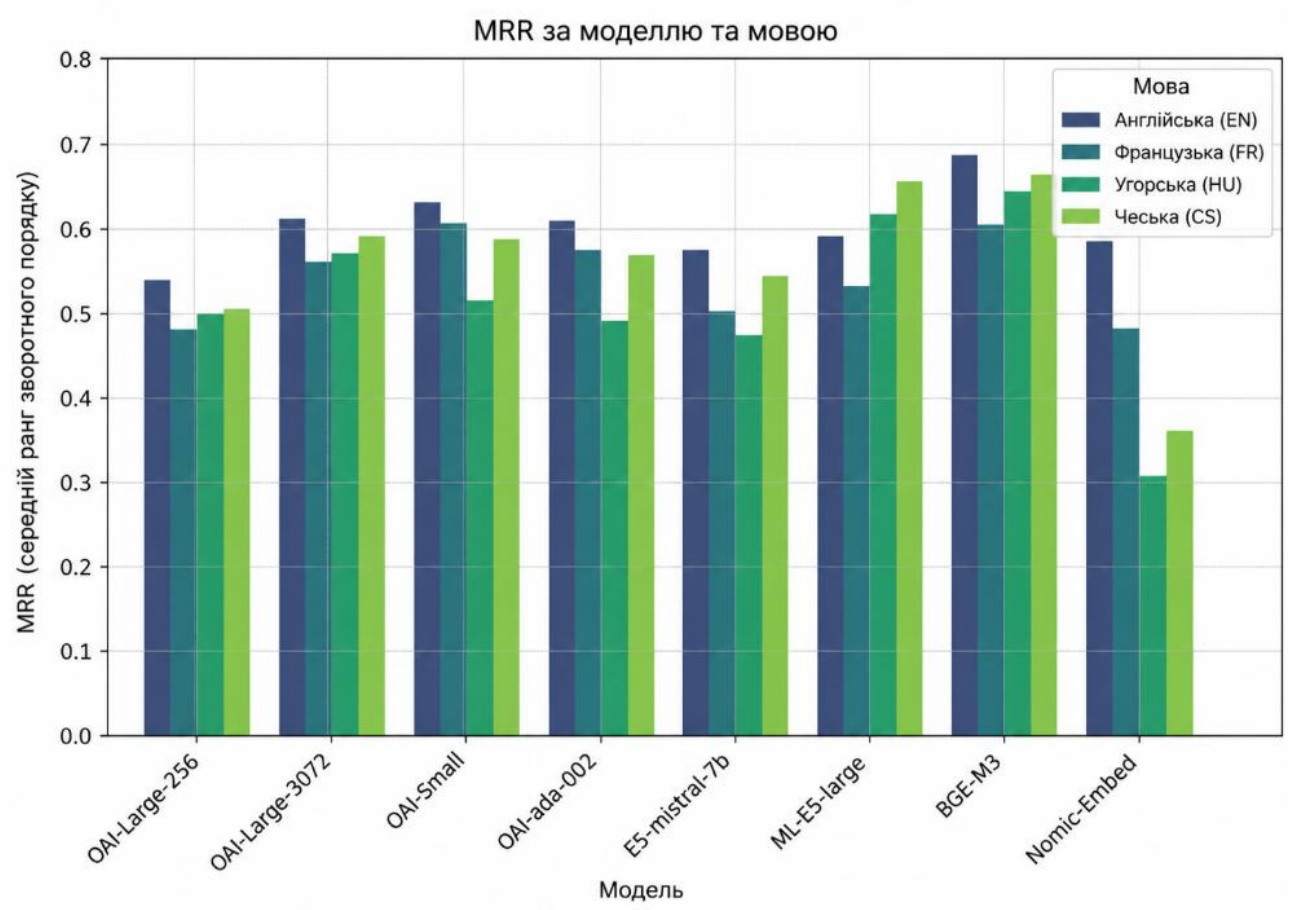

Рисунок 2.9 – Порівняння MRR моделей для векторизації інформації

Такі моделі як ML-E5-large, E5-mistral-7b, Nomic-Embed мають відносно високу якість при опрацюванні англомовних текстів, нижчу якість при опрацюванні неангломовних текстів [87]. ML-E5-large має суттєве обмеження в 512 токенів, що може бути критичним для довгих описів. У документації до моделі E5-mistral-7b та Nomic-Embed прямо зазначається, що модель рекомендована для використання тільки з англомовними текстами [88, 89].

Для поточної системи обрано BGE-M3 яка є більш оптимальною порівняно з аналогами, є open-source, підтримує понад 100 мов, завдяки чому можна робити запити на мові яка відрізняється від тої на якій написане завдання і все одно отримати його у результатах пошуку. Також підтримка текстів довжиною 8192 токенів дозволяє ефективно опрацьовувати великі обсяги інформації. Серед особливостей варто відмітити можливості пошуку на основі векторних відстаней, лексичний пошук та багатовекторний пошук [90].

При виборі моделі для інтерпретації інформації з аналізу виключено моделі які доступні за підпискою через API, моделі розмір яких перевищує 10 Гб, не мають процесу розмірковування та мають обмежену підтримку мов які використовуються для інтерпретації. Це зумовлено необхідністю контролювати інформацію яка передається і контролювати ресурси системи. Також, моделі з малим розміром роблять інтерпретацію тексту з нижчою якістю порівняно з більш великими моделями. Серед розглянутих сімейств мовних моделей були проаналізовані характеристики Llama, Qwen, Gemma. Результати порівняння наведено в таблиці 2.32 [91-94].

Таблиця 2.32 – Порівняння моделей для інтерпретації інформації

Параметр	**Gemma 4 E4B**	**Qwen3 14B**	**Llama 3.1 8B**
Розмір, Гб	9,6	9,3	4,9
Квантизація	Q4_K_M	Q4_K_M	Q4_K_M
Підтримка мов	140 мов	119 мов	8 мов
Контекстне вікно, тисячі токенів	128	40	128
Локальний запуск	Висока ефективність	Висока ефективність	Висока ефективність

Відповідно до результатів аналізу у якості моделі для інтерпретації інформації обрано модель Gemma-4-E4B, яка завдяки підтримці мов, обдумуванню відповіді та швидкості генерації токенів є оптимальним варіантом для задоволення вимоги швидко і якісно інтерпретувати інформацію.

2.6 Перспективи розвитку системи

Архітектурний стиль та технічні рішення системи передбачають можливість її подальшого вдосконалення і розвитку у випадку появи нових

вимог, зміни поточних вимог та необхідності впровадити додаткові конкурентні переваги. Серед потенційних напрямків розвитку у короткостроковій перспективі можна виділити наступні:

1. Розробка адаптивного інтерфейсу для мобільних пристроїв, що дозволить зручно керувати проєктами та завданнями з мобільних пристроїв.
2. Впровадження функціоналу для додавання коментарів до завдання і інтелектуальний режим автоматичного формування підсумку обговорення і потенційну рекомендацію для керівника.
3. Сповіщення щодо нових коментарів, можливість згадувати учасників в коментарях.
4. Формування професійного профілю учасника проєкту або проєктів, який буде підсумовувати технічні виклики та рішення які впровадив користувач і технології які були використані. Після чого система може пропонувати певного виконавця для схожих завдань, додаючи обгрунтування по фактично виконаних завданнях.
5. Інтеграція з системами автентифікації сторонніх виробників, таких як Google, Microsoft, що в свою чергу дозволить додати додатковий рівень безпеки через організацію облікових записів у цих сервісах.
6. Розробка можливості керувати обліковим записом користувача на рівні системи
7. Розробка функціоналу для фільтрації завдань по виконавцях

Реалізація цих потенційних рішень дозволить оптимізувати процеси керування проєктами, що підвищить цінність системи для цільової аудиторії завдяки спрощенню процесу керування проєктами і завданнями.

2.7 Висновки до другого розділу

У цьому розділі було виконано аналіз вимог до системи керування завданнями, архітектурних стилів, архітектурних рішень та вибір технологій для реалізації системи.

Використані архітектурні рішення забезпечують масштабованість, відмовостійкість, модульність та подальшу еволюцію системи. Підхід до виокремлення функціоналу по домену дозволив чітко структурувати бізнес логіку компонентів завдяки обмеженому контексту (bounded context).
Мікросервісна архітектура в поєднанні з подієво-орієнтованою архітектурою дозволяє досягти сильної зв’язності і слабкого зв’язування компонентів системи, а також ізоляцію зони відповідальності компонентів і забезпечити незалежне розгортання.

Вибір технологій для реалізації системи грунтувався на попередньому аналізі сценаріїв використання системи, особливостях роботи системи, функціональних та нефункціональних вимогах і архітектурних особливостях.
Додатково запропоновано функціонал для майбутньої ітерації розробки системи, який дозволить системі підвищити цінність рішень для цільової аудиторії.

Можливість відносно швидкої адаптації під технічні виклики і нові ідеї є ключовою для успіху будь-якої системи. Зважаючи на сучасні реалії можна стверджувати, що здатність системи до масштабування і інтеграції з новими компонентами які розширюють існуючий або надають новий функціонал є запорукою стабільності роботи системи і її розвитку в довготривалій перспективі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ ПРОЄКТАМИ

3.1 Методологія розробки

Під час аналізу підходу [1] до розробки системи враховувалися такі критерії як технічна новизна, ризики реалізації, необхідність додаткового дослідження і пошуку рішень в процесі розробки.

Зважаючи на це було обрано адаптивний (гнучкий) підхід до розробки [95], оскільки в розробці даної системи присутній елемент інноваційності і невизначеності, який вимагає експериментальних досліджень для пошуку оптимального рішення, а також фокусуватися лише на найціннішому фукнціоналу для кінцевого користувача.

В якості методології розробки обрано Scrum (хоча Scrum Guide наголошує, що це є «легкий фреймворк» [96]). Scrum дозволяє керувати керувати високим ступенем невизначеності завдяки використанню ітеративного підходу та інкрементальної розробці. Scrum дозволяє розділити період розробки на ітерації які забезпечують отримання швидкого зворотнього зв’язку на основі отриманих результатів розробки, а також дозволяє зміну пріоритетів і послідовності реалізації функціоналу, в залежності від зміни контексту.

Серед основних переваг можна виділити наступні:

- Фокусування на цінності для цільової аудиторії яка буде використовувати розроблену систему.
- Прозорість процесу розробки за рахунок щоденного аналізу виконаної роботи і аналізу інкремента.
- Можливість почати з версії мінімально життєздатного продукту (MVP) і працювати з гіпотезами через адаптацію успішних рішень і швидкий вихід на ринок.

Таким чином Scrum дозволяє зосередитись на розробці найважливішого функціоналу і адаптувати план розробки відповідно до результатів досліджень та виникаючих ризиків.

3.2 Реалізація мікросервісу авторизації

Мікросервіс авторизації реалізує такі основні сценарії як реєстрація користувача та автентифікацію користувача. При перевірці облікових даних сервіс видає пару JWT токенів, який дозволяє користувачеві звертатися до інших захищених компонентів системи. Мікросервіс реалізовано у контейнері Docker у вигляді окремого контейнера для бекенд частини auth-service, написаної на FastAPI та контейнера бази даних postgres_auth створеного на основі образу postgres:15-alpine. При запуску мікросервісу в базі даних автмоаично створюються необхідні таблиці у разі їх відсутності на момент запуску завдяки асинхронному контекстному менеджеру lifespan.

Зберігання постійних даних реалізовано за допомогою Docker volume. Запуск контейнера бази даних postgres_auth містить параметр wal_level=logical, який дозволяє використовувати механізм Change Data Capture для читання змін в базі даних. Для передачі змін в базі даних сервісу використовується окремий контейнер з платформою Debezium яка створює події у топіку auth_db_server.public.users брокера повідомлень. Відслідковується таблиця users і інформація передається в топік асинхронно. Далі сервіси яким необхідні дані про користувачів (ідентифікатор користувача, електронна адреса, повне ім’я) використовують консюмери для читання і оновлення відповідних даних. Для роботи з базою даних використовується SQLAlchemy. Підключення до бази даних здійснюється за допомогою облікових даних мікросервісу і реалізовано за допомогою сесій підключення до бази даних. Це дозволяє контролювати ресурси системи і вивільняти ресурси при завершення сесії роботи з даними. Таблиця users містить облікові дані користувачів системи. Під час реєстрації користувач

передає своє повне ім’я, електронну адресу та пароль, далі система перевіряє чи користувач такою адресою вже існує, після чого хешує пароль та створює обліковий запис користувача в базі даних. Для хешування паролів використано алгоритм bcrypt з вбудованим додавання солі (salting) для підвищення рівня криптологічного захисту.

В процесі автентифікації мікросервіс користувач передає електронну адресу та пароль через стандартну форму OAuth2PasswordRequestForm, де електронна адреса передається у полі username , а система, після підтвердження що обліковий запис існує і пароль правильний, створює токен доступу та токен оновлення (access token, refresh token). Окрема змінна середовища містить криптографічний секретний ключ який використовується для генерації JWT токена. Тривалість дії токена доступу (access token) встановлена на 30 хвилин. Тривалість дії токена оновлення встановлена на 14 днів. Ендпойнти сервісу показано у таблиці 3.1.

Таблиця 3.1 – API ендпойнти мікросервісу авторизації

API ендпойнт	**Метод**	**Призначення**
/auth/register	POST	Реєстрація нового користувача. Взаємодію між об’єктами системи під час реєстрації показано на діаграмі послідовності на рисунку 3.1
/auth/login	POST	Автентифікація користувача
/auth/refresh	POST	Оновлення токена доступу і токена оновлення.

Для валідації даних використовується бібліотека Pydantic. Схеми Pydantic містять вимоги до даних які надходять від користувача.

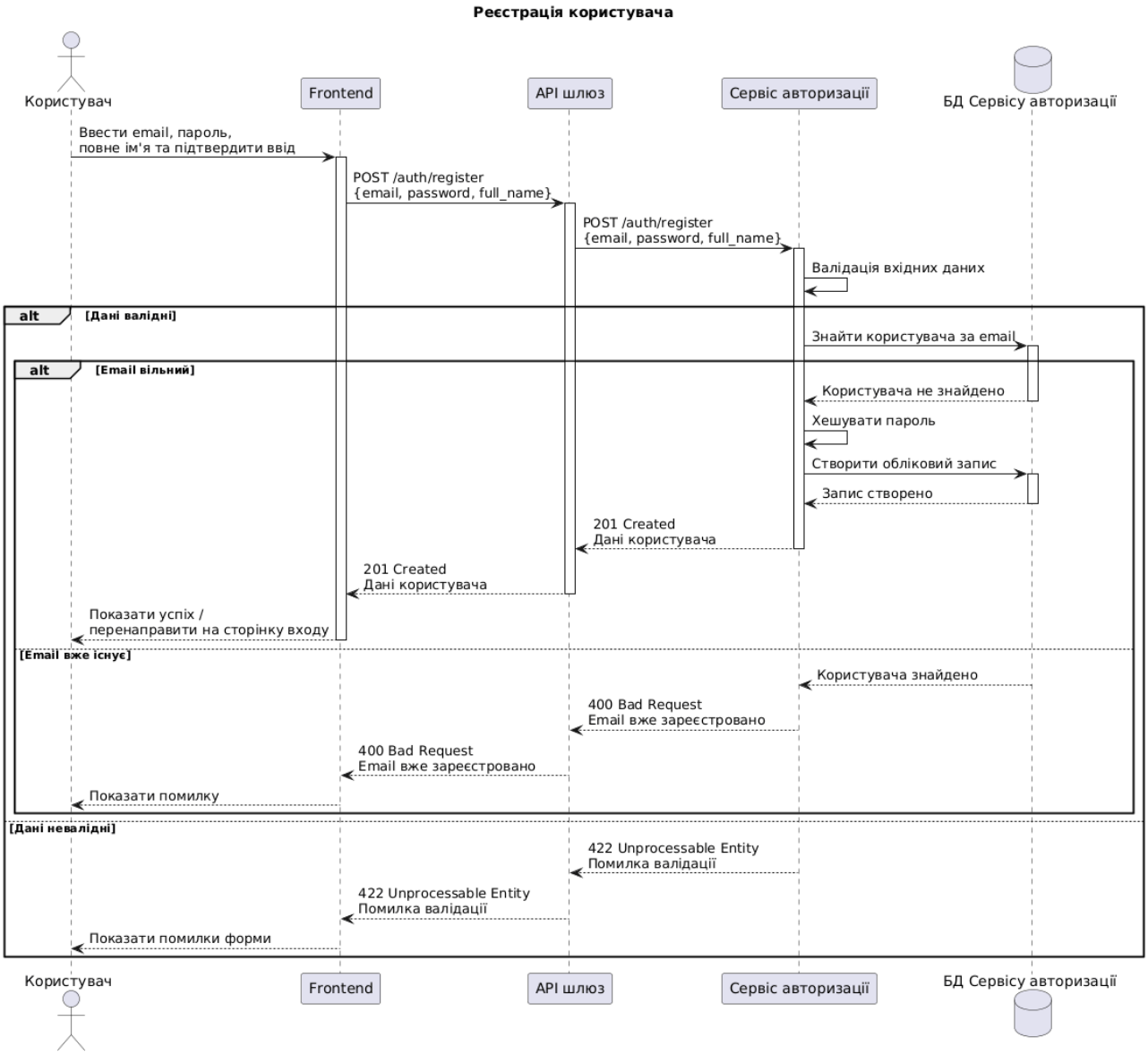

Рисунок 3.1 – Діаграма послідовності процесу реєстрації

Така реалізація роботи мікросервісу дозволяє знизити зв’язність з іншими сервісами які залежать від даних сервісу авторизації, забезпечує необхідну ізоляцію даних, безпеку і відмовостійкість. Структура проєкту наведена у додатку А. Лістинг програмного коду наведено в додатку Б.

3.3 Реалізація мікросервісу керування завданнями

Основним мікросервісом на який припадає максимальне навантаження є мікросервіс керування завданнями. Він забезпечує CRUD операції для роботи з проєктами, завданнями і керування учасниками проєкту. Ендпойнти сервісу показано у таблиці 3.2. Лістинг програмного коду наведено в додатку Б.

Таблиця 3.2 – API ендпойнти мікросервісу

API ендпойнт	**Метод**	**Призначення**
/projects/	GET	Отримати список проєктів, де користувач є власником або учасником
/projects/	POST	Створити новий проєкт
/projects/{project_id}	GET	Отримати дані проєкту
/projects/{project_id}	PATCH	Оновити частину змінених даних проєкту.
/projects/{project_id}	DELETE	Видалити проєкт
/projects/{project_id}/members	GET	Отримати список учасників проєкту
/projects/{project_id}/add_members	POST	Додати нового учасника до проєкту
/projects/{project_id}/members/{user_id}/rate	PATCH	Зміна значення погодинної вартості роботи учасника
/projects/{project_id}/members/{user_id}	DELETE	Видалити учасника з проєкту
/projects/{project_id}/tasks/	GET	Отримати список завдань
/projects/{project_id}/tasks/	POST	Створити завдання
/projects/{project_id}/tasks/{task_id}	PATCH	Оновити частину змінених даних завдання
/projects/{project_id}/tasks/{task_id}	DELETE	Видалити завдання
/projects/{project_id}/tasks/{task_id}/interpretation/request	POST	Відправити запит на AI-інтерпретацію завдання. Відбувається перевірка проєкту і учасника та формується запит з даними до сервісу аналізу завдань

Кінець таблиці 3.2

API ендпойнт	**Метод**	**Призначення**
/tasks/hydrate	POST	Приймає список ідентифікаторів завдань і повертає повну інформацію про завдання. Використовується при семантичному пошуку через мікросервіс аналізу

Для авторизації користувача мікросервіс використовує JWT токен сформований сервісом авторизації раніше і при декодуванні отримує ідентифікатор користувача та його електронну адресу. Ці дані використовуються у ключових операціях сервісу для дотримання необхідного рівня безпеки при роботі з даними. Контроль доступу до функціоналу реалізовано за допомогою ролей owner і member, що дозволяє обмежувати доступ до певного функціоналу для користувачів з роллю member.

Завдяки наявності консюмера, сервіс здійснює зчитування актуальної інформації про користувачів сервісу авторизації через топік auth_db_server.public.users з Kafka. Якщо дані користувача вже присутні в сервісі керування завданнями, то здійснюється порівняння ідентичності, після чого дані оновлюються якщо є відмінності.

Для валідації даних використовуються Pydantic схеми, які дозволяють здійснити перевірку правильності даних введених користувачем при передачі даних від клієнтського застосунку і формування стандартизованих API відповідей. Керування даними при взаємодії з базою даних виконується за допомогою ORM моделі (SQLAlchemy), що спрощує реалізацію бізнес логіки.

Взаємодія між користувачем і системою для створення проєкту показана на рисунку 3.2.

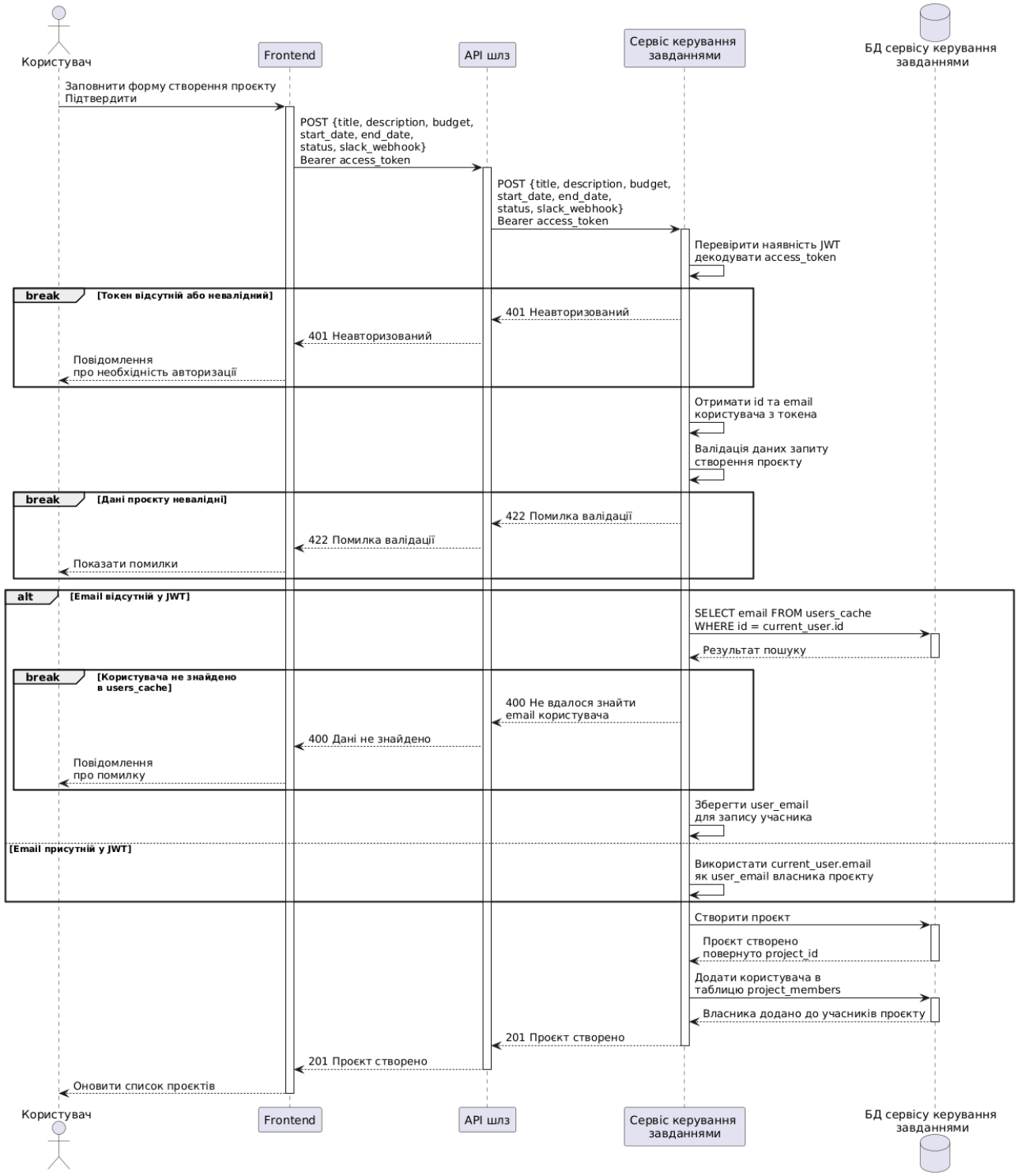

Рисунок 3.2 – Діаграма послідовності процесу створення проєкту

Мікросервіс реалізовано яка FastAPI застосунок з окремою базою даних, які розміщені в окреми Docker контейнерах. Бекенд частина в task-service та контейнер бази даних postgres_task створеного на основі образу postgres:15-alpine. Зберігання постійних даних реалізовано за допомогою Docker volume.

Контейнер бази даних також запущено з параметром wal_level=logical для використання механізму Change Data Capture. Відповідні події створюються у наступних топіках брокера повідомлень для асинхронної передачі даних:

- task_db_server.public.projects для публікації змін у проєктах;
- task_db_server.public.project_members для публікації змін які стосуються учасників проєктів;
- task_db_server.public.tasks для публікації змін які стосуються завдань.

Дані з цих топіків потрібні для подальшого формування звітності по проєкту і обчислень для визначення стану проєкту. Набір даних по суті той який ідентичний до таблиць projects, project_members, tasks в базі даних сервісу керування завданнями.

Використання механізмів для роботи з подіями допомогає забезпечити слабку зв’язність між компонентами системи, що в свою чергу дозволяє мікросервісу ефективно працювати.

3.4 Реалізація мікросервісу аналізу завдань

Мікросервіс аналізу завдань реалізує інтелектуальну обробку даних які пов’язані з завданнями проєкту. Мікросервіс виконує функції з векторизації та індексації завдань у векторній базі даних, пошук потенційних дублікатів завдань, семантичний пошук, генерацію бізнес-орієнтованих пояснень технічних (і не тільки) завдань.

Мікросервіс складається з трьох компонентів які знаходяться в Docker контейнерах. Бекенд частина (FastAPI) , контейнер для роботи Ollama та контейнер векторної бази даних Qdrant зі зберіганням даних через Docker volume. Під час запуску мікросервісу виокнується ініціалізація колекції у векторній базі даних та запуск фонових споживачів Kafka.

Консюмер мікросервісу аналізу завдань слухає події які стосуються змін в завданнях у топіку task_db_server.public.tasks та створює або оновлює завдання у векторній базі даних, попередньо формуючи ембедінги на основі комбінації тексту назви завдання та його опису. Процес створення цифрового представлення інформації (емебедінгу) відбувається за допомогою допоміжного сервісу Ollama який за допомогою ембедінг моделі BGE-M3 формує вектор для подальшого збереження. Для реалізації семантичного пошуку завдань та перевірки на наявність дублікатів використовується векторна база даних яка містить векторні представлення завдань.

Завдяки такій реалізації користувач у своєму пошуковому запиті має можливість передати саму суть завдання без необхідності вказувати якісь точні визначення або цитувати назву чи опис завдання. У якості розмірності встановлено оптимальне значення 1024 (к-сть координат у векторному просторі), яке забезпечує високу точність при порівняннях. Це значення є параметром за замовчуванням для моделі BGE-M [90].

Для порівняння векторів використовується параметр Cosine який порівнює вектори за їхнім напрямком, що дозволяє яраз порівнювати смислову схожість, не звертаючи увагу на довжину опису чи запиту.

В процесі підбору межі схожості була проведена оцінка точності результатів пошуку для вибору оптимального значення межі схожості. Результати дослідження показано на рисунку 3.3. Тестування здійснювалося на тестовому проєкті з повноцінним описом тестових завдань. Точність отриманих результатів визначена як відношення кількості релевантних результатів до всіх знайдених завдань.

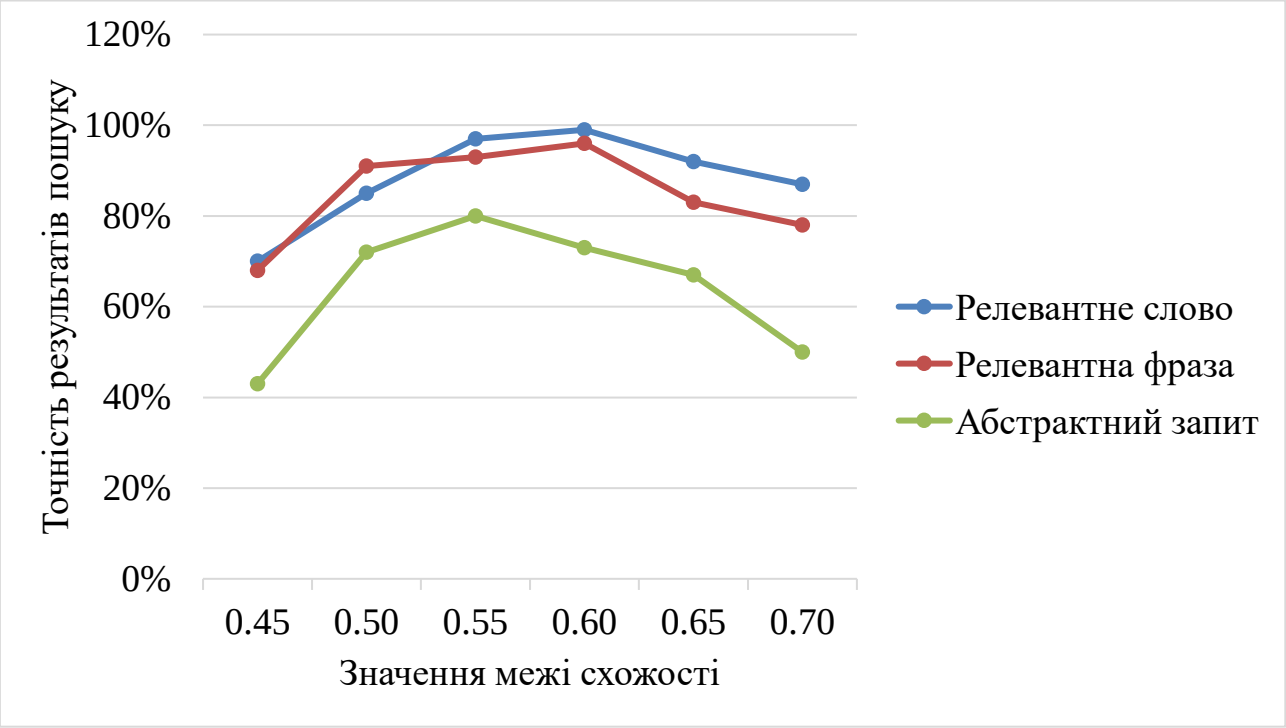

Рисунок 3.3 – Результати тестування для підбору оптимальної межі схожості при семантичному пошуку завдань

Оптимальним значенням межі схожості при семантичному пошуку для роботи сервісу обрано 0.55. Саме таке значення дозволяє отримувати оптимальні результати пошуку при нечітких формулюваннях і запитах на мові яка відрізняється від мови якою описане завдання що шукають. Послідовність взаємодії між об’єктами системи під час семантичного пошуку відображено на рисунку 3.4.

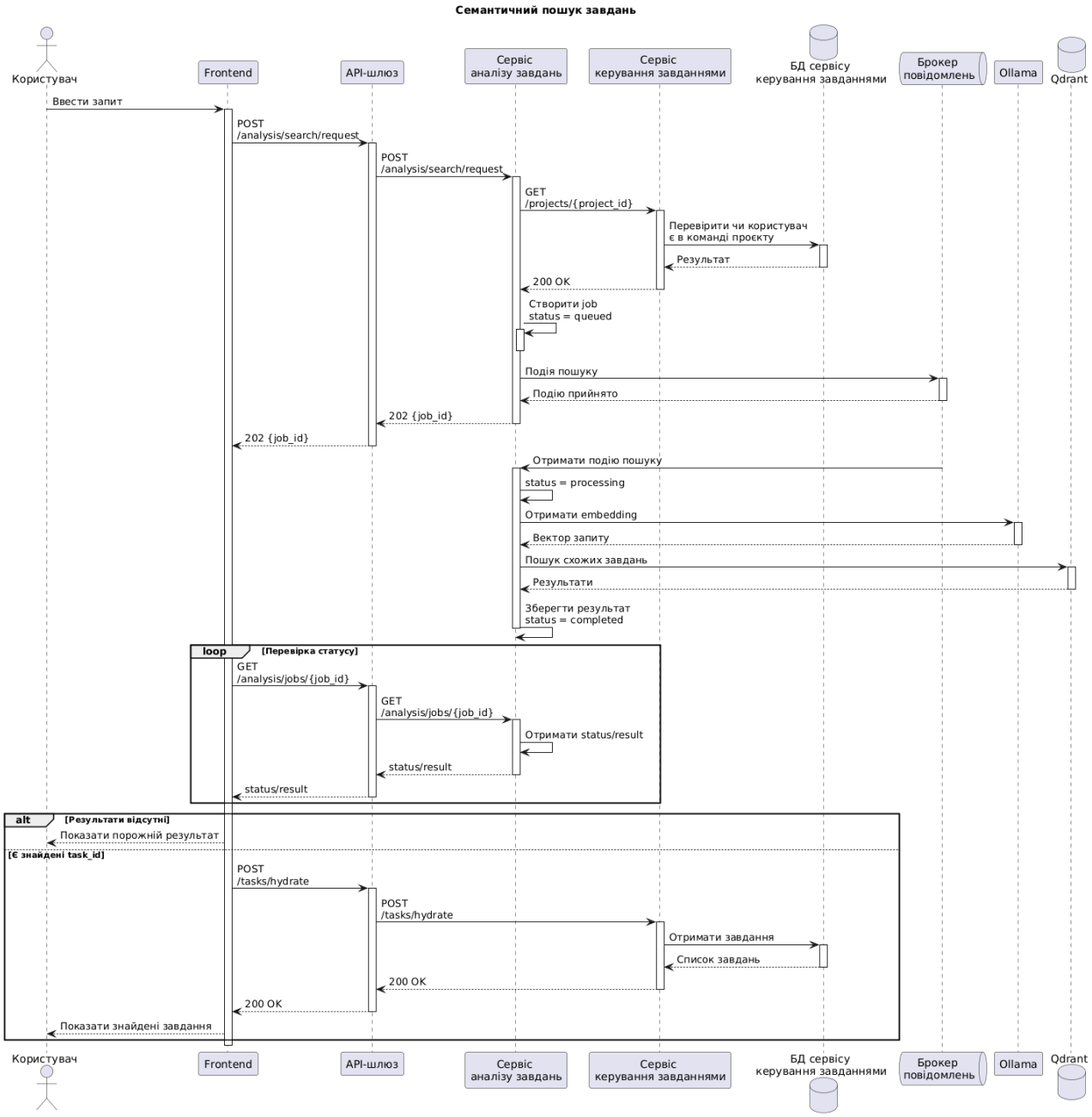

Рисунок 3.4 – Діаграма послідовності семантичного пошуку завдання

Така реалізація семантичного пошуку використовує так звану гідратацію через сервіс керування завданнями, тобто доповнення ідентифікаторів завдань актуальною інформацією з сервісу через окремий запит (/tasks/hydrate), що дозволяє отримати достовірну інформацію з джерела. Це дозволяє розділити відповідальність між сервісами відповідно до їх бізнес контексту.

При пошуку дублікатів за допомогою того самого методу необхідно знаходити більш точні результати. Результати тестування для визначення оптимальної межі схожості показані на на рисунку 3.5.

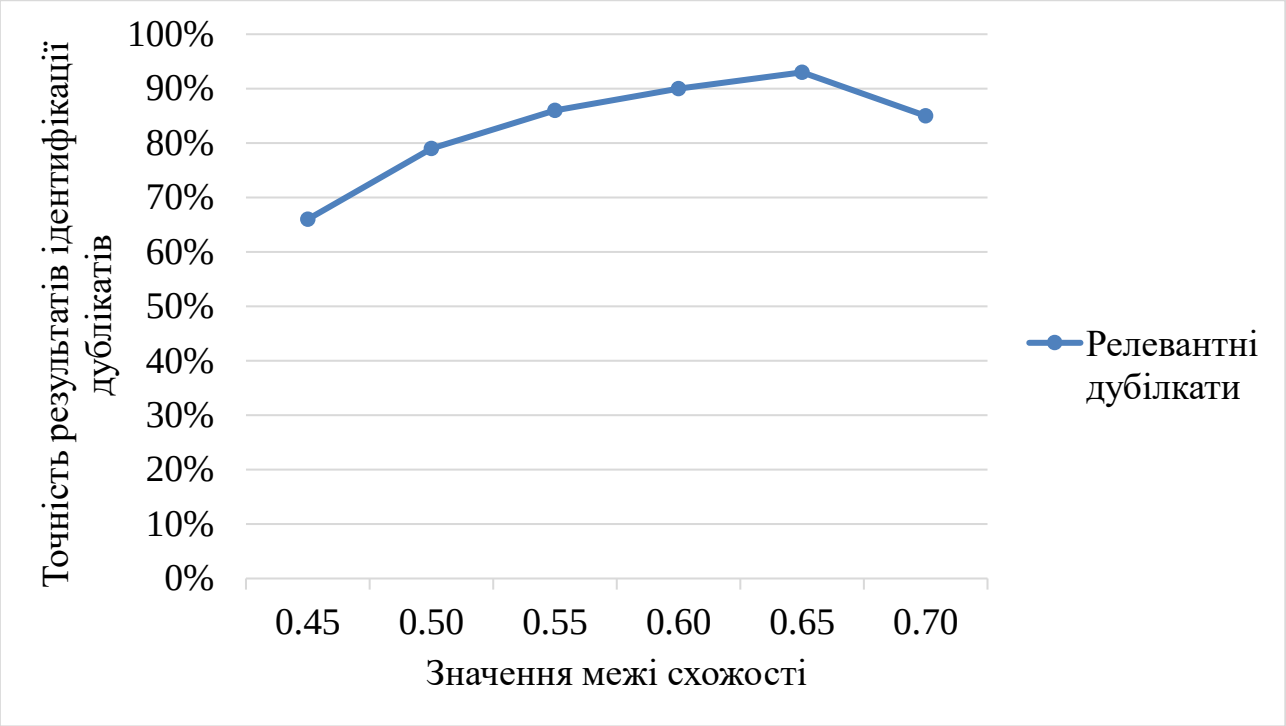

Рисунок 3.5 – Результати тестування для підбору оптимальної межі схожості при пошуку дублікатів завдань

Межа схожості завдання рівна 0.65 знижує ризик помилково визначених потенційних дублікатів . Процес пошуку аналогічний до семантичного пошуку з невеликими відмінностями які стосуються інформації яка відображається користувачеві, а саме ідентифікатор завдання, назва завдання, статус завдання.

Qdrant виконує пошук за допомогою алгоритму Hierarchical Navigable Small World і в поточній реалізації значення параметру для цього алгоритму встановлено рівним 128. Цей параметр визначає ретельність пошуку у векторній базі даних.

Для інтерпретації завдань використовується інша мовна модель, а саме Gemma-4-E4B. Результати тестування параметру температури показані на

рисунку 3.6. На тестовій вибірці проводилася оцінка якості інтерпретації відповідно до початкового опису завдання. Наявність зайвих формулювань, відхилення від інструкції та галюцинації позначалися як такі які негативно впливають на сумарну оцінку якості згенерованого тексту.

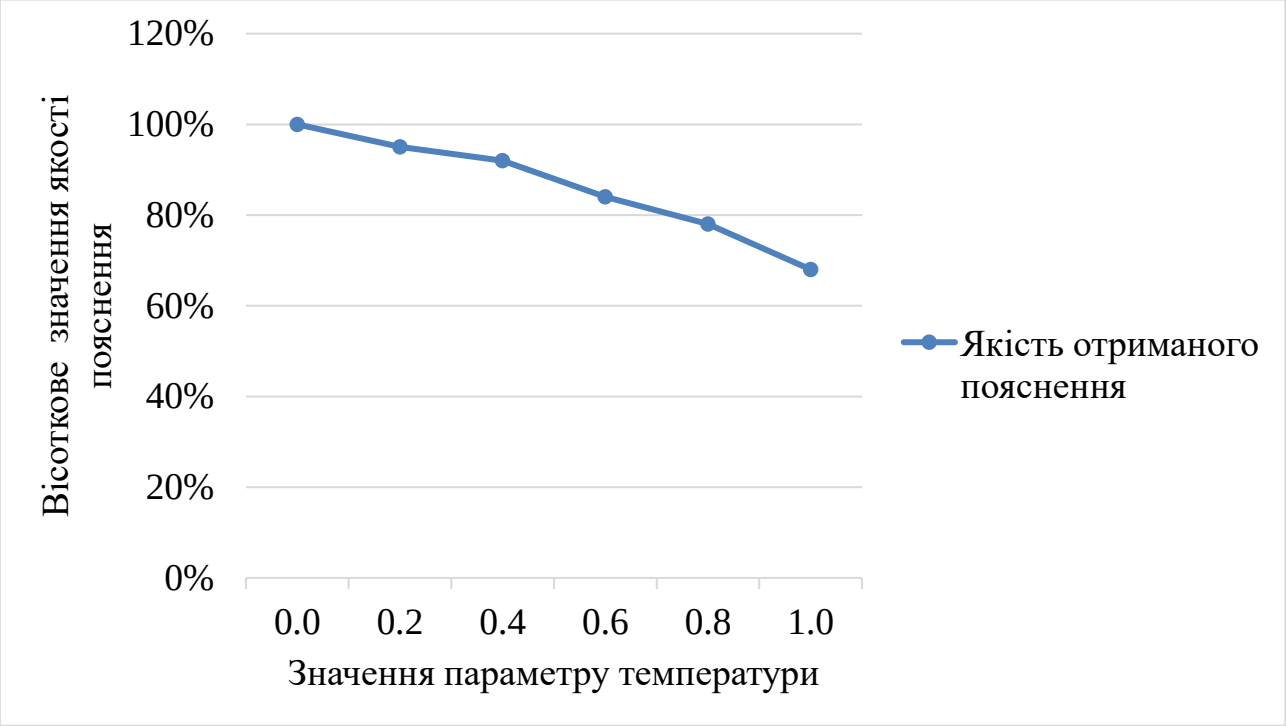

Рисунок 3.6 – Вплив параметру температури на якісь інтерпретації завдання

В результаті аналізу для роботи обрано параметр температури рівний 0.2 для формування менш креативного трактування тексту з фокусом на точність. Незважаючи на меншу точність, порівняно з параметром температури 0.0, параметри 0.2 дозволяє робити пояснення більш зрозумілими для сприйняття, а також забезпечує певну варіативність тектсу пояснення у випадку, коли опис завдання змінився навіть незначним чином і потрібно повторно згенерувати пояснення.

Інструкція стосовно того що має зробити мовна модель відправляється одночасно з описом завдання яке потрібно інтерпретувати. Користувач отримує

у відповідь структуроване пояснення без технічної термінології. Ендпойнти сервісу показано у таблиці 3.3. Для безпеки операцій сервіси використовує токен доступу який формується сервісом авторизації. Pydantc схеми використано для опису стурктури запитів і відповідей. Для запитів які використовують ШІ можливості використано механізм створення завдань (job) і публікацію подій в Kafka через наступні топіки:

- analysis.ai_requests використовується для запитів які стосуються пошуку дублікатів завдань та семантичного пошуку завдань;
- analysis.task_interpretation_requests використовується для запитів на інтерпретацію завдань;
- analysis.task_interpretations використовується для передачі інформації щодо інтерпретації (пояснення) завдання.

Таблиця 3.3 – API ендпойнти мікросервісу аналізу завдань

API ендпойнт	**Метод**	**Призначення**
/analysis/search/request	POST	Додавання запиту на семантичний пошук в чергу
/analysis/duplicates/request	POST	Додавання запиту на пошук дублікатів завдань в чергу
/analysis/interpretation	POST	Додавання запиту на пояснення завдання в чергу
/analysis/jobs/{job_id}	GET	Отримання статусів і результатів операцій

При інтерпретації завдання ШІ, відповідь публікується у Kafka топік analysis.task_interpretations. Після чого консюмер сервісу керування завданнями зберігає цю інформацію для завдання по id. При наступному запиті від користувача щодо інформації про завдання сервіс керування завданнями передасть вже готове пояснення до завдання. Лістинг програмного коду наведено в додатку Б.

З метою підвищення швидкості операції інтерпретації завдання, семантичного пошуку і пошуку дублікатів система налаштована на

використання Nvidia GPU у Docker контейнері через блок deploy.resources.reservations.devices, що дозволяє використовувати наявні потужності відеокарти.

Для перевірки доступності апаратного забезпечення для використання у Docker використано команду docker run --rm --gpus all nvidia/cuda:12.9.0-base-ubuntu22.04 nvidia-smi. Результат виконання команди показано на рисунку 3.7.

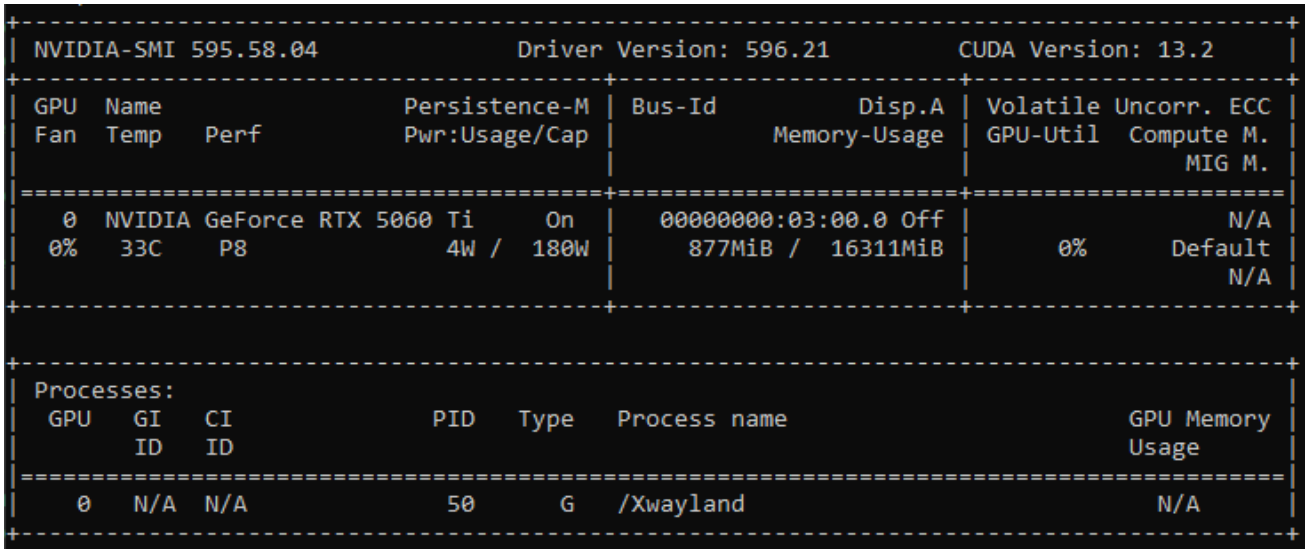

Рисунок 3.7 – Результат перевірки доступності GPU для подальшого використання

Реалізований мікросервіс аналізу завдань додає цінний функціонал для роботи з завданнями і є однією з ключових переваг використання системи. До обмежень які має поточна реалізація відносяться доступні обчислювальні ресурси які використовує Ollama. У випадку використання тільки CPU швидкість обчислень буде нижчою ніж при використанні GPU, відповідно, інтелектуальні функції системи будуть працювати повільно. Поточна реалізація використовує окремий GPU для роботи. Окремий контейнер ollama-model-loader виконує роль сервісу який завантажує моделі в пам’ять GPU для можливості швидкої роботи у відповідь на запити користувача.

3.5 Реалізація мікросервісу звітності

Мікросервіс звітності складається з двох частин які розміщені в Docker контейнерах: бекенд частина (FastAPI-застосунок) та база даних бази даних postgres-reporting створеної на основі образу postgres:15-alpine. Дані які зберігає сервіс він отримує завдяки консюмеру який слухає топіки auth_db_server.public.users, task_db_server.public.projects, task_db_server.public.project_members, task_db_server.public.tasks з даними про користувачів, проєкти та завдання і оновлює дані в базі даних відповідно. Для можливості формувати аналітичні дашборди та звіти сервіс надає ендпойнти які наведено у таблиці 3.4. Всі ендпойнти захищені JWT токеном.

Таблиця 3.4 – API ендпойнти мікросервісу звітності

API ендпойнт	**Метод**	**Призначення**
/reports/analytics/{project_id}	GET	Аналітика по проєкту для швидкого аналізу показників у клієнтському застосунку
/reports/generate/{project_id}	POST	Формування звіту за тиждень і відправка в Slack за допомогою Webhook посилання яке користувач додає під час створення або редагування проєкту

Для формування аналітики використовуються дані які зафіксував сервіс в процесі прослуховування топіків брокера повідомлень. Умовні позначення які використовуються при обчисленнях метрик наведено в таблиці 3.5.
Метрики та відповідні формули для обчислень наведено в таблиці 3.6. Частина метрик є рекомендованою PMBOK як такі по яких доцільно оцінювати стан проєкту [1].

Таблиця 3.5 – Умовні позначення які використовуються при обчисленнях метрик

Умовне позначення	**Опис**	**Формула для обчислення**
N_t	Загальна кількість завдань проєкту	N_t = Сума завдань у всіх статусах
N_{done}	Кількість завершених завдань	Сума завдань у статусі Done
N_{open}	Сумарна к-сть незавершених завдань (всі, крім виконаних у статусі Done)	N_t- N_{done}
$N_{overdue}$	К-сть завдань з простроченою датою виконання	Сума завдань у яких дата менша сьогоднішньої і статус не рівний Done
B_p	Плановий бюджет проєкту	-
H_a	Фактично використані години у завданні	-
H_e	Планова оцінка виконання завдання у годинах	-
$Rate$	Погодинна ставка виконавця завдання	-
$Days_{elapsed}$	К-сть календарних днів які минули від дати початку проєкту	Сьогоднішня дата - Дата початку проєкту
$Days_{total}$	К-сть календарних днів всього доступних для виконання проєкту	Дата завершення проєкту - Дата початку проєкту
AC	Фактична вартість робіт	$\sum_{i=1}^{N_t}(H_{a,i} \times Rate_i)$
PV	Планова вартість робіт	$\sum_{i=1}^{N_t}(H_{e,i} \times Rate_i)$
EV	Вартість виконаної частини проєкту (освоєний обсяг)	$B_p \times \frac{N_{done}}{N_t}$
CV	«Сума дефіциту або надлишку бюджету на певний момент часу, виражена як різниця між освоєним обсягом і фактичною вартістю» [1]	$EV - AC$

Кінець таблиці 3.5

Умовне позначення	**Опис**	**Формула для обчислення**
SV	«Величина, на яку проєкт випереджає або відстає від запланованої дати постачання на певний момент часу, виражена як різниця між освоєним обсягом і плановою вартістю» [1]	$EV - PV$

Таблиця 3.6 – Додаткові метрики для аналітики проєкту

Обчислене значення	**Призначення**	**Обчислення**
Прогрес	Показує відсоток завершеності проєкту	$\frac{N_{done}}{N_t}$ x 100
Витрати часу, %	Показує частину від календарного проєкту яка вже використана	$\frac{Days_{elapsed}}{Days_{total}}$ x 100
Використання бюджету, %	Показує відсоток використання бюджету проєкту та значення у грошах	$\frac{AC}{B_p}$ x 100
SPI (Schedule Performance Index)	«вказує, наскільки ефективно виконують заплановану роботу» [1] > 1.0 - Випередження графіку = 1.0 - Згідно графіку < 1.0 - Відставання від графіку	$\frac{\text{Прогрес, \%}}{\text{Витрати часу, \%}}$
CPI (Cost Performance Index	«вказує, наскільки ефективно виконують роботу з урахуванням закладеної в бюджет вартості роботи» [1] > 1.0 - Економія витрат = 1.0 - Витрати згідно плану < 1.0 - Перевищення витрат	$\frac{\text{Прогрес, \%}}{\text{Використання бюджету, \%}}$
Час до дедлайну	Показує к-сть робочих днів які залишилися до дедлайну.	Сума робочих днів між датою початку проєкту і датою закінчення (включно)
Частка прострочених завдань	Індикатор для виявлення проблем з простроченими завданнями. 0 - 10% - Допустиме відхилення 10 - 30 % - Потребує уваги > 30% - Критичний стан	$\frac{N_{overdue}}{N_{open}}$

Продовження таблиці 3.6

Обчислене значення	**Призначення**	**Обчислення**
Здоров'я проєкту	Показник здоров'я проєкту. 0 < Потрібна увага < 45 45 < Помірний ризик < 75 75 < Стабільний темп < 100	SPI x 40% + CPI x 40% + Risk x 20%
Аналіз здобутої цінності (EVM)	Лінії тренду які показують відхилення показників CV та SV в динаміці за останні 60 днів	-
Навантаження команди	Список учасників і сума фактично використаних годин	Сума годин по учаснику проєкту
Команда	К-ть чоловік які працюють над проєктом	Сума к-сті осіб які присутні в проєкті
Базовий темп (Baseline)	Кількість завдань які завершує команда за 7 днів. *I* - к-сть завершених завдань в ітерації.	1 тиждень: значення відсутнє 2 тижні: I_1 3 тижні: $\frac{I_1 + I_2}{2}$ Більше 6 тижнів: $\frac{I_1 + I_2 + I_3}{2}$
Темп команди (Velocity)	Показує середній темп завершення завдань за 7 днів (к-сть виконаних завдань)	1 тиждень: I_1 2, 3 тижні: $I_{\text{останнього тижня}}$ 4,5 тижні: $\frac{I_{\text{останній т}} + I_{\text{останній т}-1}}{2}$ Більше 6 тижнів: $\frac{I_{\text{останній т}} + I_{\text{останній т}-1} + I_{\text{останній т}-2}}{2}$
Динаміка виконання завдань	Відображає кількість завдань на дату: - Завершені завдання (Done) - Активні (To Do, In Progress) - Прострочені завдання	-
Дедлайн	Планова дата завершення проєкту	-
Структура статусів	Підсумок к-сті завдань відповідно до статусу	Сума завдань по кожному статусу
К-сть оцінених годин	Планова к-сть годин на виконання всіх завдань	$\sum_{i=1}^{N_t}(H_{e,i})$
К-сть факичних годни	Фактична к-сть використаних годин	$\sum_{i=1}^{N_t}(H_{a,i})$

Кінець таблиці 3.6

Використання часового ресурсу, %	Співвідношення між фактичними і плановими годинами	$\frac{H_a}{H_e}$ x 100
Фінансове навантаження, %	Частка використаного бюджету	$\frac{AC}{B_p}$ x 100
Залишок бюджету	Бюджет який залишився для освоєння	$B_p - AC$

На рисунку 3.8 наведено приклад опублікованого в Slack канал звіту. На рисунку 3.9 показано послідовність взаємодії між користувачем та компонентами системи при генеруванні текстового звіту та відправки його в Slack-канал. Лістинг програмного коду наведено в додатку Б.

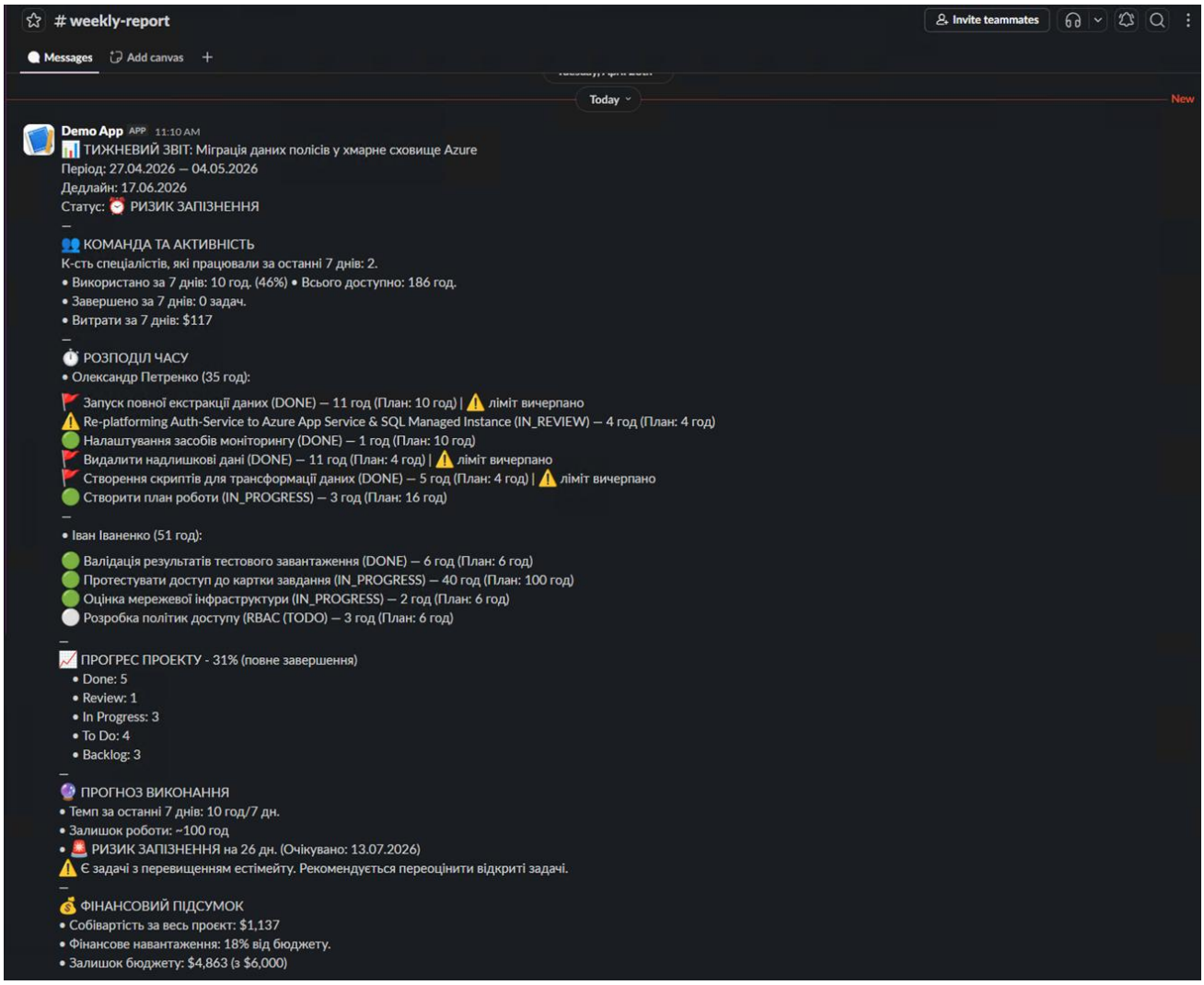

Рисунок 3.8 – Приклад отриманого в Slack звіту по проєкту

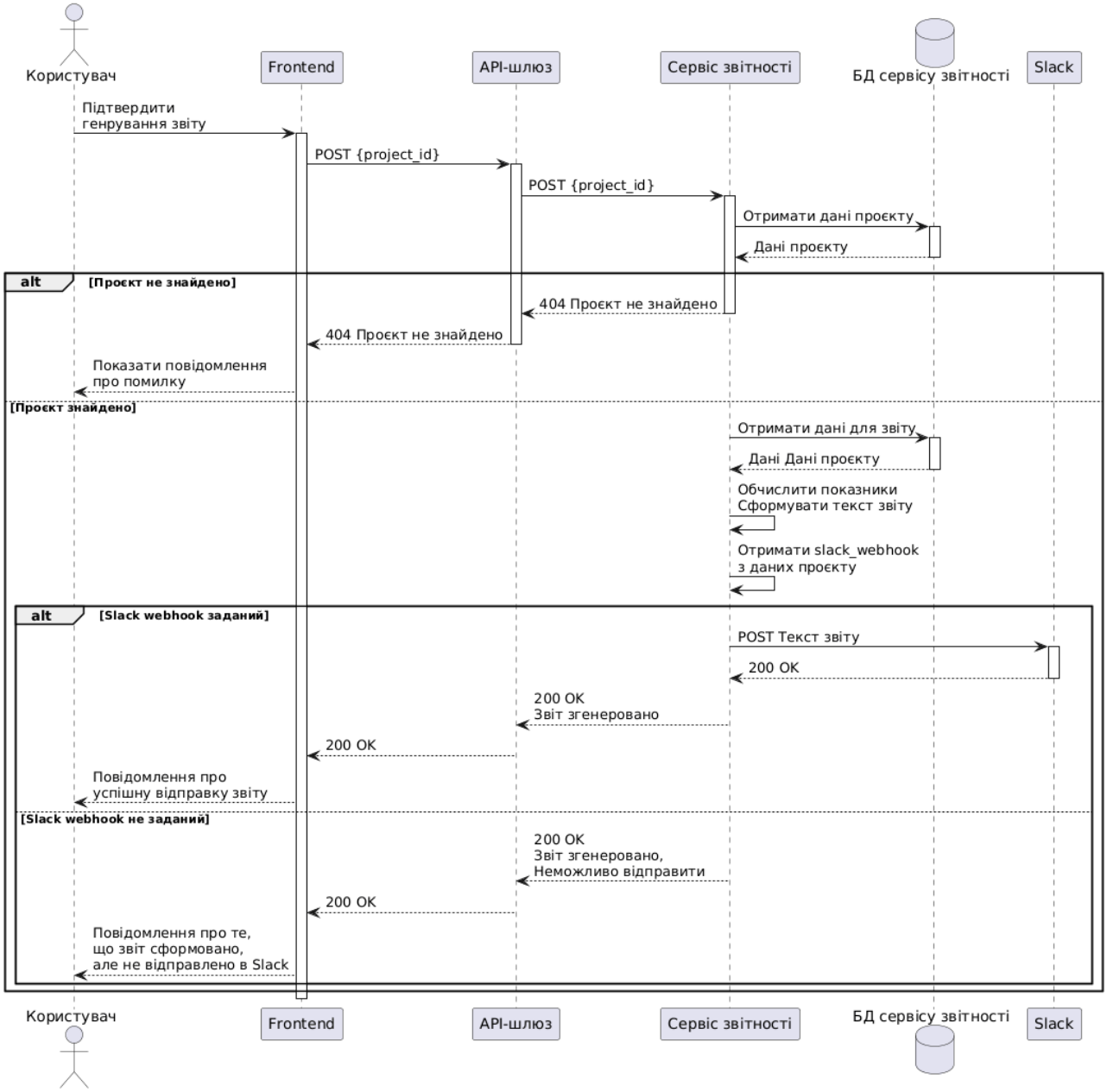

Рисунок 3.9 – Діаграма послідовності при генеруванні та відправці звіту в Slack

Така реалізація мікросервісу звітноті забезпечує асинхронне отримання даних і подальші обчислення метрик які потрібні для звітності та подальшої візуалізації даних у клієнтському застосунку та месенджері Slack.

3.6 Реалізація фронтенду

Клієнтський частина реалізована у вигляді веб-застосунку. Веб-застосунок завдяки наявному інтерфейсу дозволяє користувачеві взаємодіяти з усіма мікросервісами системи. Веб-застосунок реалізовано за допомогою фреймворку Vue та конфігурації Vite. Застосунок працює у окремому Docker контейнері і передає запити на бекенд через API-шлюз, який реалізовано на Nginx у окремому контейнері.

Маршрутизація між сторінками відбувається за допомогою роутера (vue-router). Маршрути захищені перевіркою токена доступу. Токен доступу та токен оновлення зберігаються у локальному сховищі.

Інтерфейс користувача розділений на компоненти та основні сторінки. Сторінка входу в систему відображено на рисунку 3.10.

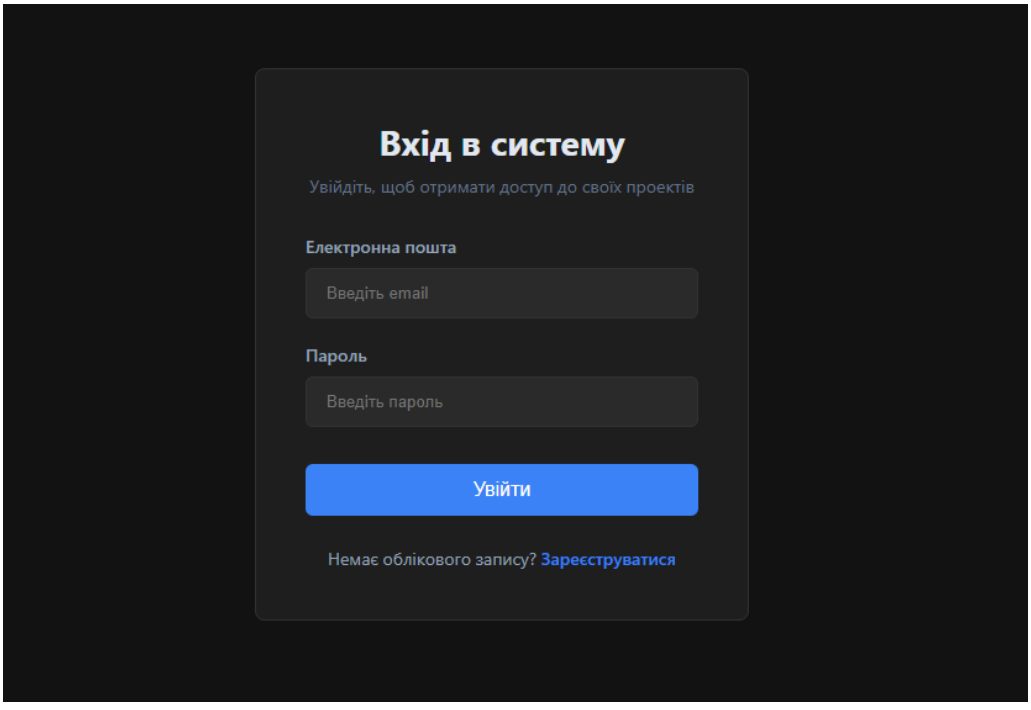

Рисунок 3.10 – Сторінка входу в систему

Користувач може перейти на сторінку реєстрації у разі відсутності існуючого облікового запису. Сторінка реєстрації показана на рисунку 3.11.

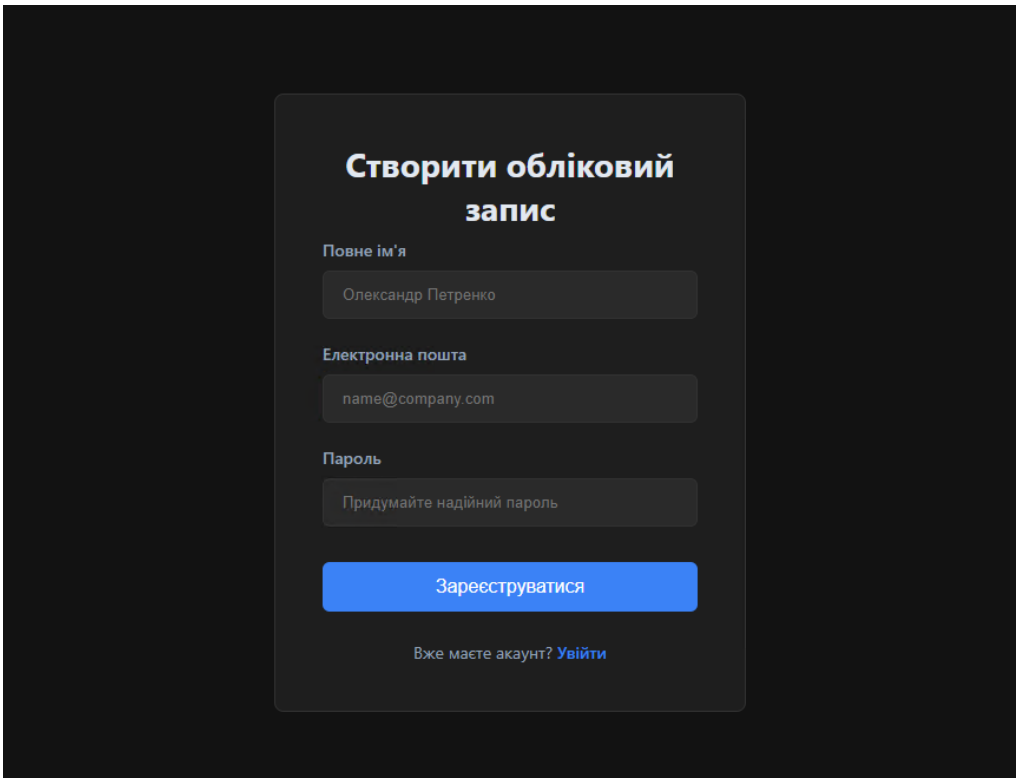

Рисунок 3.11 – Сторінка реєстрації

Після успішного входу в систему відображається головна сторінка з поточними проєктами у яких користувач є власником. Сторінку з проєктами показано на рисунку 3.12.

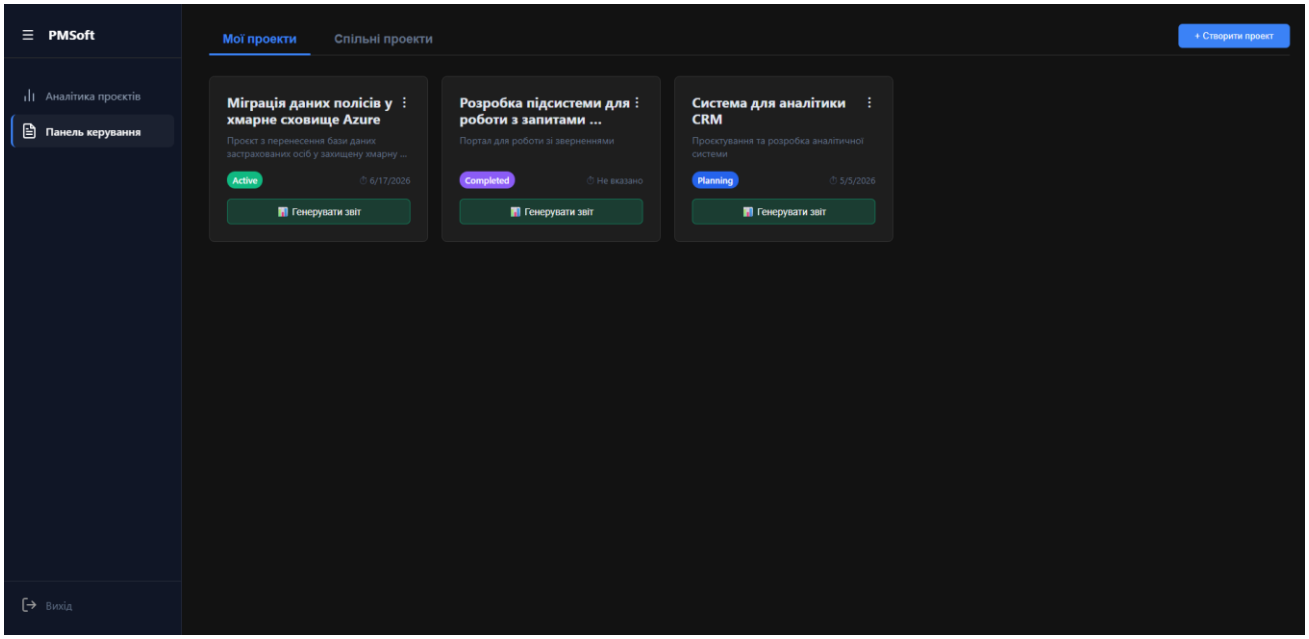

Рисунок 3.12 – Панель керування проєктами

Панель меню зліва містить елемент для переходу на сторінку аналітики проєктів та можливість вийти з системи. Робоча область сторінки містить картки проєктів. Присутня можливість редагувати або видалити існуючі проєкти. На сторінці присутня можливість для створення нового проєкту. Модальне вікно створення проєкту показано на рисунку 3.13.

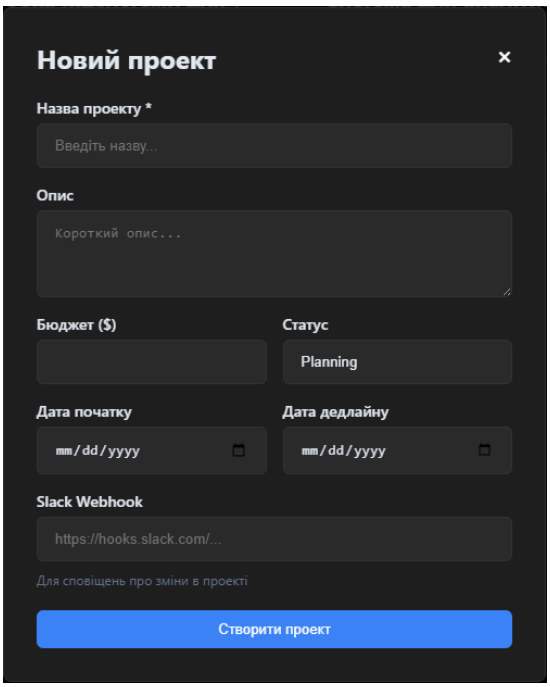

Рисунок 3.13 – Модальне вікно створення нового проєкту

На сторінці Спільні проєкти відображено список проєктів у яких користувача запросили у якості учасника. Вміст сторінки зі спільними проєктами відображено на рисунку 3.14.

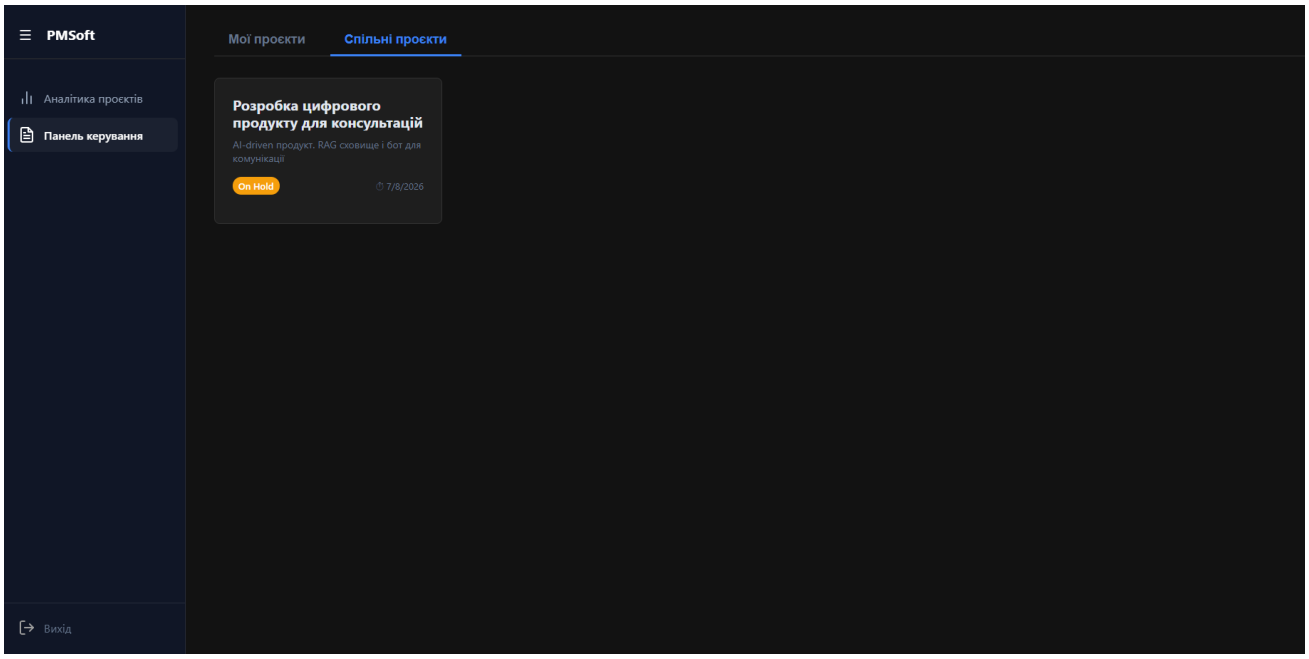

Рисунок 3.14 – Сторінка зі спільними проєктами.

Після натискання на картку проєкту система відображає Канбан-дошку з завданнями. Дошку з завданнями показано на рисунку 3.15.

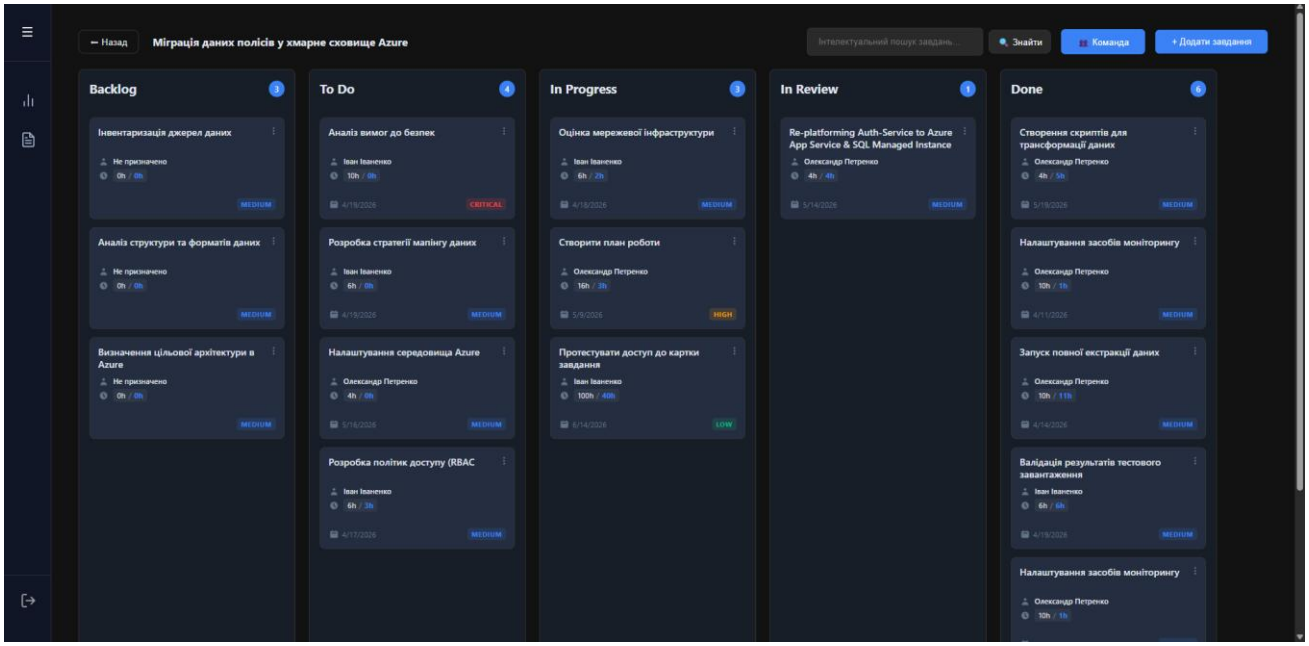

Рисунок 3.15 – Канбан дошка проєкту.

Модальне вікно для створення нового завдання з отриманими результатами пошуку дублікатів відображено на рисунку 3.16.

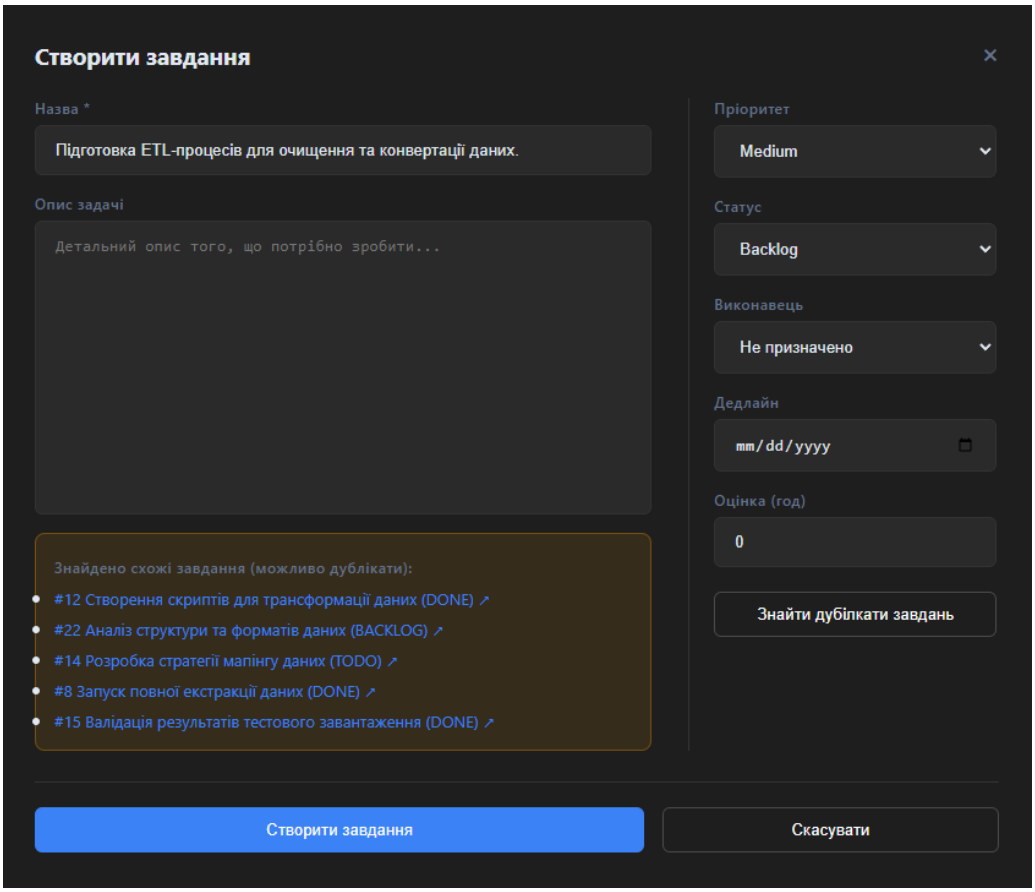

Рисунок 3.16 – Модальне вікно створення проєкту

При створенні проєкту користувач має можливість вказати основні атрибути завдання, а також здійснити пошук дублікатів за потреби ще до завершення створення завдання.

Редагування або видалення завдання відбувається за допомогою контекстного меню на картці завдання, показаного на рисунку 3.17.

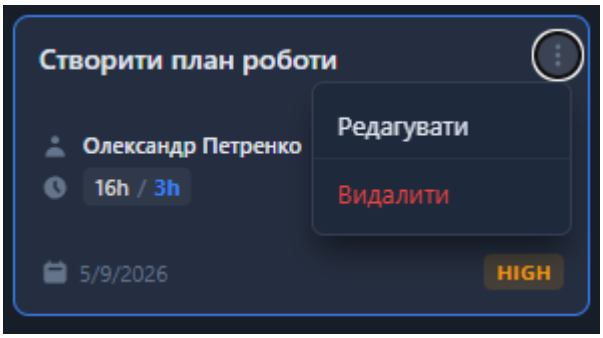

Рисунок 3.17– Контекстне меню картки завдання

Перегляд вмісту існуючого завдання відбувається при натисканні на картку завдання. Модальне вікно з деталями завдання показано на рисунку 3.18.

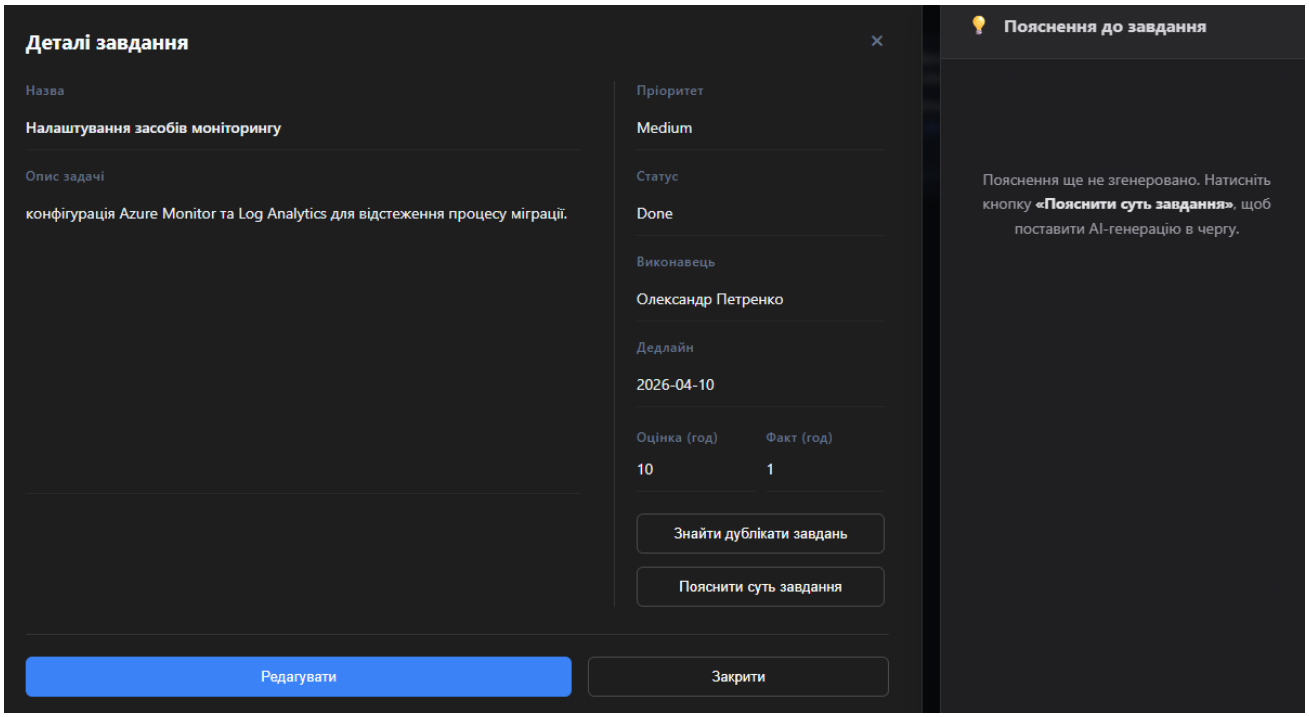

Рисунок 3.18 – Модальне вікно в режимі перегляду завдання

В такому режимі користувач має можливість скористатися функцією для інтерпретації завдання. Результат інтерпретації показано на рисунку 3.19.

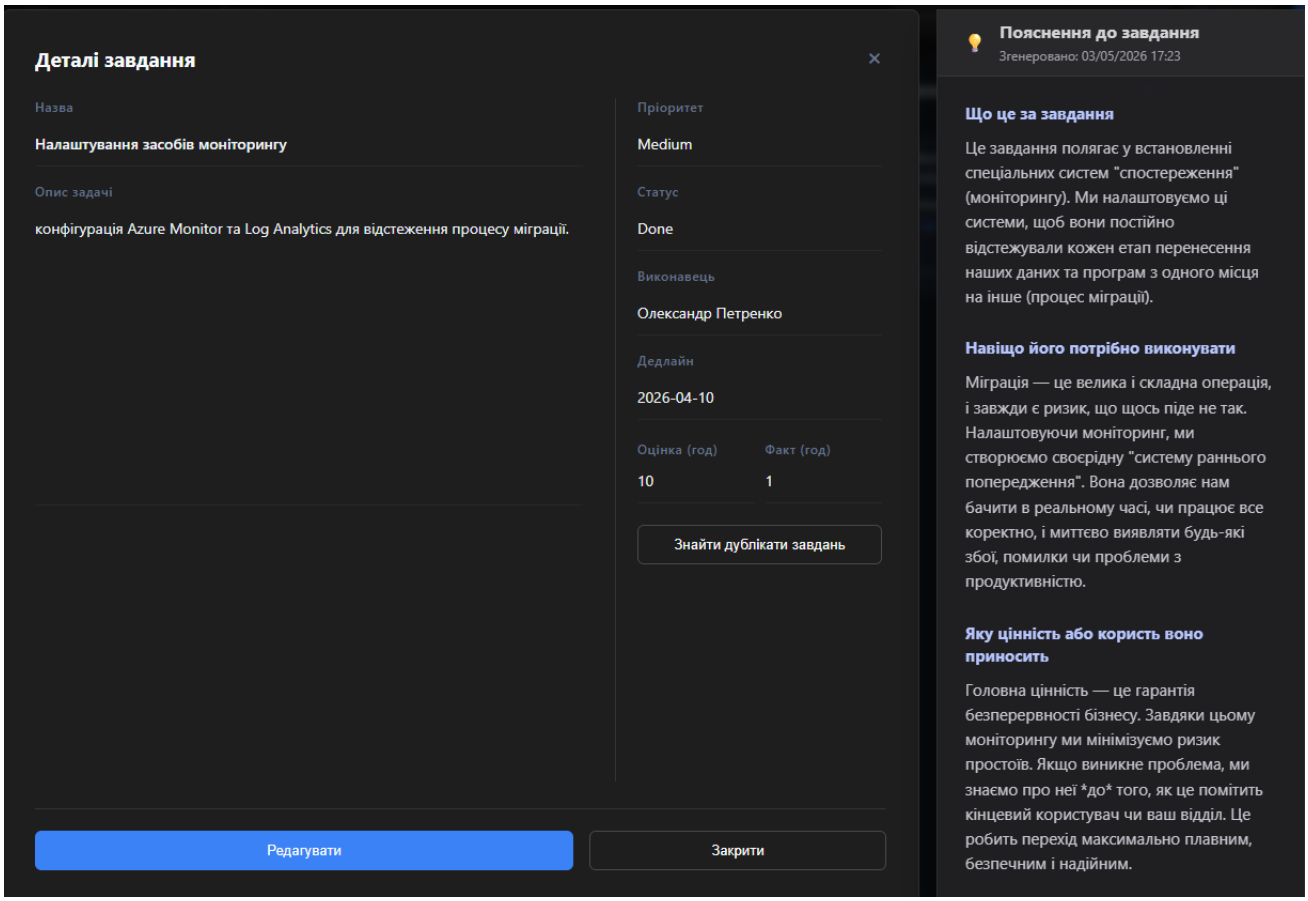

Рисунок 3.19 – Модальне вікно в режимі перегляду завдання з поясненням

Пояснення розділено на 3 секції які дозволяють зрозуміти і оцінити доцільність завдання.

Результати семантичного пошуку за комбінацією назви завдання та його опису показано на рисунку 3.20

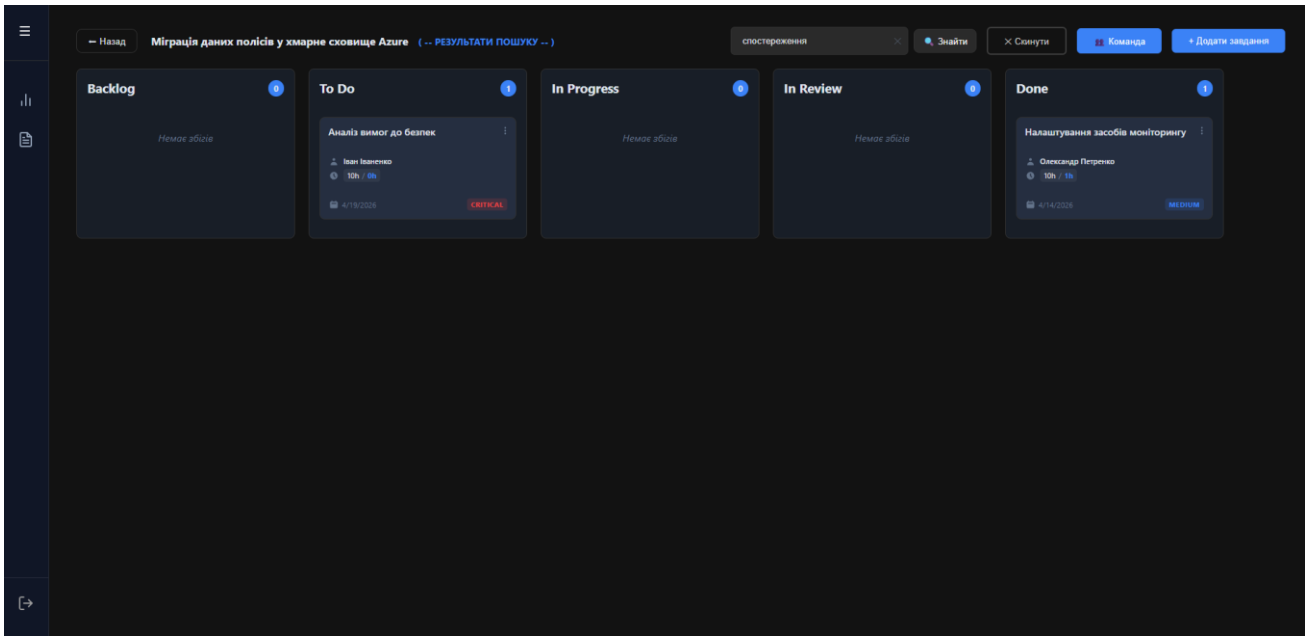

Рисунок 3.20 – Результати семантичного пошуку

В результаті семантичного пошуку було виявлено найбільш релевантні завдання і відфільтровано завдання які за суттю не відповідають суті сформульованій запиті.

Для керування учасниками проєкту передбачено окреме модальне вікно яке відображене на рисунку 3.21.

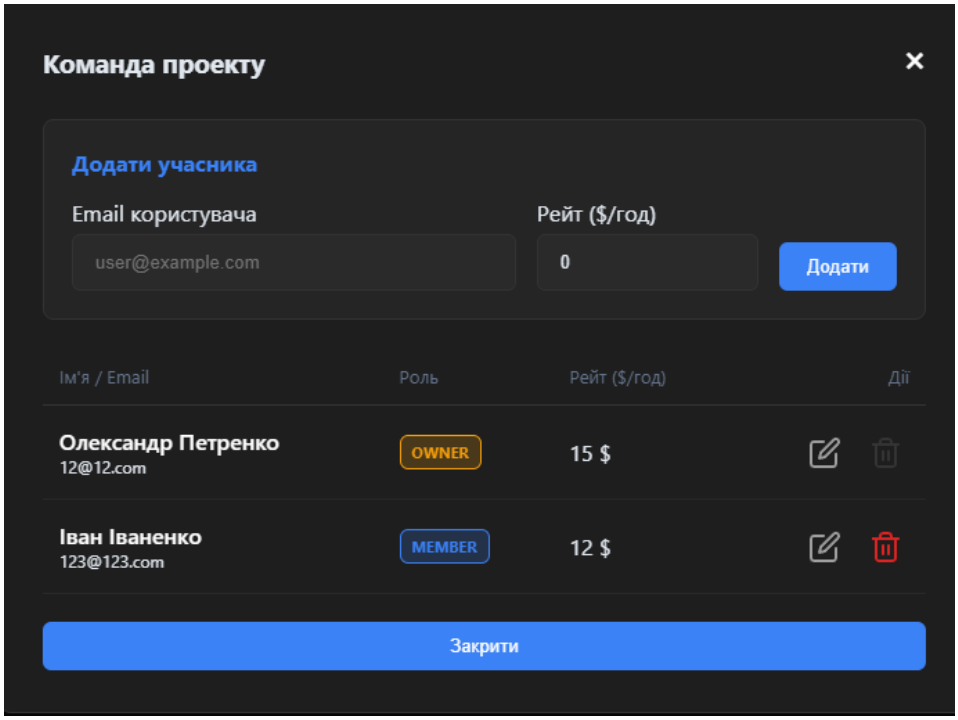

Рисунок 3.21 – Модальне вікно керування учасниками проєкту

Користувач у ролі власника проєкту має можливість додавати, видаляти учасників, редагувати їхню вартість погодинної оплати.

Після переходу на сторінки аналітики проєкту користувач має можливість побачити окремий дашборд для кожного зі своїх проєктів. Приклад дашборду відображено на рисунку 3.22.

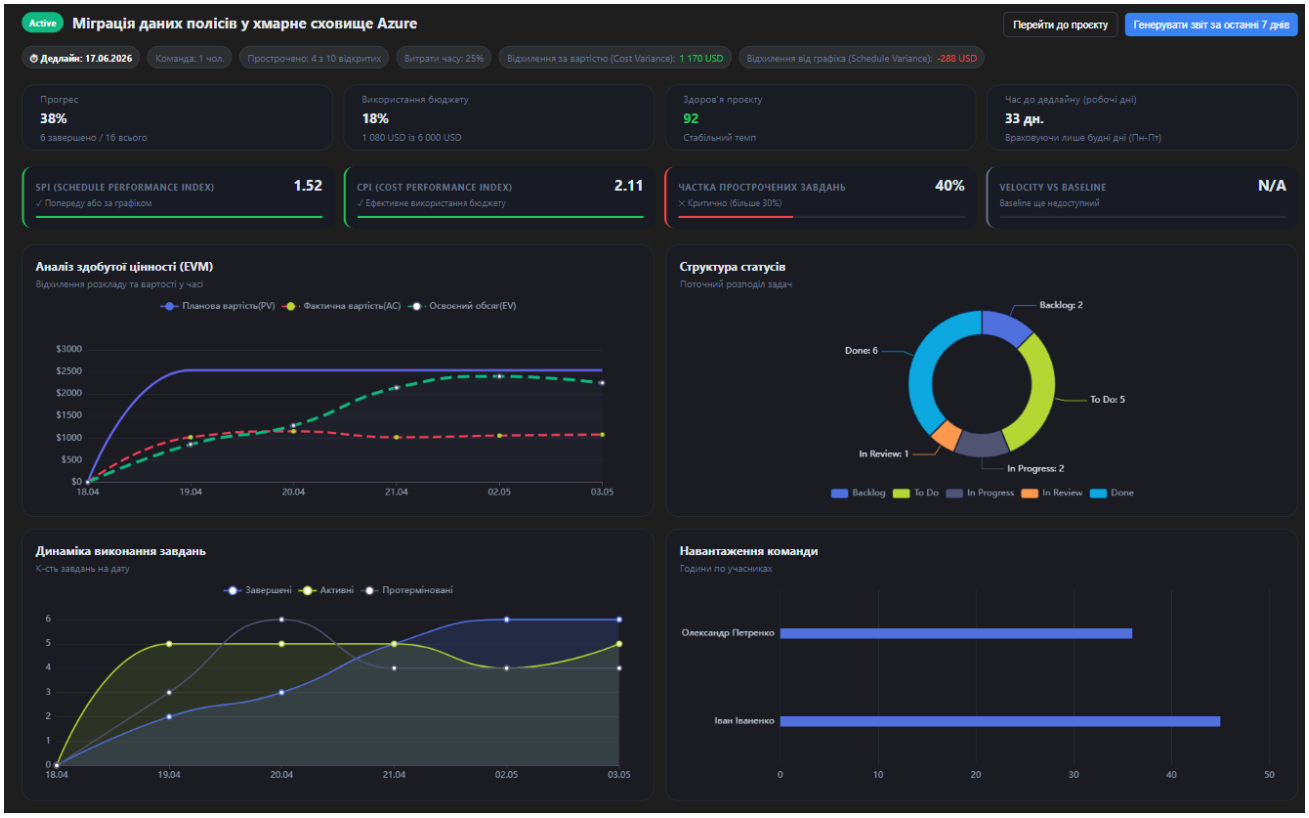

Рисунок 3.22 – Дашборд аналітики проєкту

У клієнтському застосунку присутній набір інтерфейсів для повноцінної роботи з проєктами та завданнями. Взаємодія з бекендом забезпечується API-шлюзом і спрощує процес розгортання системи. Компонентний підхід до побудови фронтенду дозволяє будувати структуровані інтерфейси.

3.7 Висновки до третього розділу

Результатом розробки є система керування проєктами побудована за принципами мікросервісної архітектури з інтеграцією штучного інтелекту. В розділі наведено інформацію про особливості реалізації мікросервісів системи, підкреслено їх особливості та можливості кожного мікросервісу. Завдяки подієво-орієнтованого підходу при роботі запитами до ШІ і асинхронній взаємодії, досвід користувача від взаємодії з системою є прийнятним і оптимальним в умовах роботи релізованої системи. Клієнтський застосунок за рахунок захищеної маршрутизації є зручною точкою доступу до функціоналу системи.Завдяки використанню FastAPI у кожного мікросервісу є готова API документація створена на основі OpenAPI специфікації.

ВИСНОВКИ

Результатом роботи є проєктування та розробка системи керування проєктами на основі мікросервісної архітектури з інтеграцією штучного інтелекту. Розроблена система орієнтована на команди в яких є потреба керувати проєктами та завданнями з можливостями ідентифікувати дублікати завдань, знаходити завдання за суттю, мати можливість отримати спрощену інтерпретацію опису завдання та можливість ефективно аналізувати стан проєктів.

У межах роботи було досліджено предметну область керування проєктами, існуючі рішення та типові проблеми для цільової аудиторії такого роду продуктів. Аналіз існуючих рішень дозволив визначити ключовий функціонал який дозволяє системі повноцінно виконувати своє призначення, а також дозволив виявити функціонал який надає конкурентну перевагу над аналогами.

В процесі проєктування системи було визначено ключові сценарії використання системи, функціональні та нефункціональні вимоги. Для задоволення виявлених вимог проведено аналіз архітектурних стилів та архітектурних рішень. За допомогою предметно-орієнтованого підходу до проєктування визначено сервіси системи. Для кожного сервісу, відповідно до потреб, розроблено стуктуру для зберігання даних. Вибір мікросервісної архітектури дозволив розділити відповідальність між окреми сервісами, ізоляцію даних в межах бізнес контексту, незалежне розгортання системи і масштабування її частин в залежності від потреби.

Реалізацію системи було виконано за допомогою мови програмування Python і фреймворку FastAPI для бекенд частини системи та фреймворка Vue для фронтенду системи у виглядів веб-застосунку. Збереження основних даних реалізовано за допомогою використання PostgreSQL. Для роботи з векторними предстравленнями даних використано векторну базу даних Qdrant. Для роботи з моделлю для ембедінгу та моделлю для генерації тексту використано сервіс

Ollama. Асинхронна взаємодія між сервісами реалізована за допомогою сервісу Kafka. Збір даних про зміни реалізовано за допомогою Debezium. Контейнеризація сервісів та розгортання реалізовано за допомогою Docker. Для маршрутизації запитів між веб-застосунком та мікросервісервісами використано Nginx в якості API-шлюза.

Під час написання кваліфікаційної роботи були виконані наступні завдання:

- проаналізовано літературні джерела;
- виконано аналіз предметної області продукту;
- виконано аналіз конкурентів серед існуючих на ринку рішень та визначено цільову аудиторію продукту;
- визначено ключові сценарії використання, вимоги до системи для отримання конкурентної переваги;
- спроєктовано архітектуру системи та системні компоненти;
- обґрунтовано вибір технологічних рішень для реалізації системи;
- визначено перелік потенційного функціоналу для подальшого розвитку системи;
- розроблено систему для керування проєктами.

Розроблена система дозволяє використовувати основний функціонал для роботи з проєктами та завданнями, поєднуючи його з інтелектуальними можливостями обробки завдань та аналітикою розширеною аналітикою проєкту. Подальший розвиток системи завдяки впровадженню можливості роботи з коментарями, створення професійного профілю учасників проєктів на основі виконаних завдань та адаптація інтерфейсу під мобільні пристрої дозволить підвищити цінність для кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 6th ed. Newtown Square, PA : Project Management Institute, 2017. 756 p.
2. Task Management Software Market [Електронний ресурс] // Market Growth Reports. URL: https://www.marketgrowthreports.com/market-reports/task-management-software-market-106919 (дата звернення: 04.05.2026).
3. Software Project Manager Demographics and Statistics [Електронний ресурс] // Zippia. URL: https://www.zippia.com/software-project-manager-jobs/demographics/ (дата звернення: 04.05.2026).
4. Owner/Project Manager Demographics and Statistics [Електронний ресурс] // Zippia. URL: https://www.zippia.com/owner-project-manager-jobs/demographics/ (дата звернення: 04.05.2026).
5. Global Project Manager Demographics and Statistics [Електронний ресурс] // Zippia. URL: https://www.zippia.com/global-project-manager-jobs/demographics/ (дата звернення: 04.05.2026).
6. Task Management Software Market [Електронний ресурс] // Astute Analytica. URL: https://www.astuteanalytica.com/industry-report/task-management-software-market (дата звернення: 04.05.2026).
7. Task Management Software Market Size, Share & Industry Analysis [Електронний ресурс] // Fortune Business Insights. URL: https://www.fortunebusinessinsights.com/task-management-software-market-102249 (дата звернення: 04.05.2026).
8. Project Management Software Market Size, Share & Trends Analysis Report [Електронний ресурс] // Grand View Research. URL: https://www.grandviewresearch.com/industry-analysis/project-management-software-market-report (дата звернення: 04.05.2026).

9. Hodun L. Project Management Statistics: Trends, Insights & Real-World Lessons for Modern Teams [Електронний ресурс] / L. Hodun // Chanty Blog. 2025. 11 Dec. URL: https://www.chanty.com/blog/project-management-statistics/ (дата звернення: 16.04.2026).
10. Jira: Issue & Project Tracking Software [Електронний ресурс] // Atlassian. URL: https://www.atlassian.com/software/jira (дата звернення: 04.05.2026).
11. Asana Product Guide [Електронний ресурс] // Asana. URL: https://asana.com/product (дата звернення: 04.05.2026).
12. About Trello [Електронний ресурс] // Trello. URL: https://trello.com/about (дата звернення: 04.05.2026).
13. About Us [Електронний ресурс] // monday.com. URL: https://monday.com/p/about/ (дата звернення: 04.05.2026).
14. What is Azure DevOps? [Електронний ресурс] // Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops (дата звернення: 04.05.2026).
15. Project Management Statistics [Електронний ресурс] // Scoop Market. URL: https://scoop.market.us/project-management-statistics/ (дата звернення: 04.05.2026).
16. Project Management Vendors [Електронний ресурс] // Ramp. URL: https://ramp.com/vendors/categories/project-management (дата звернення: 04.05.2026).
17. Atlassian Jira Customers [Електронний ресурс] // Apps Run The World. URL: https://www.appsruntheworld.com/customers-database/products/view/atlassian-jira (дата звернення: 04.05.2026).
18. Companies using Asana [Електронний ресурс] // Enlyft. URL: https://enlyft.com/tech/products/asana (дата звернення: 04.05.2026).
19. Asana Customers [Електронний ресурс] // Apps Run The World. URL: https://www.appsruntheworld.com/customers-database/products/view/asana (дата звернення: 04.05.2026).

20. Trello Pricing & Reviews [Електронний ресурс] // Ramp. URL: https://ramp.com/vendors/trello (дата звернення: 04.05.2026).
21. Trello Statistics [Електронний ресурс] // ElectroIQ. URL: https://electroiq.com/stats/trello-statistics/ (дата звернення: 04.05.2026).
22. Companies using monday.com [Електронний ресурс] // Enlyft. URL: https://enlyft.com/tech/products/monday-com (дата звернення: 04.05.2026).
23. monday.com Statistics [Електронний ресурс] // ElectroIQ. URL: https://electroiq.com/stats/monday-com-statistics/ (дата звернення: 04.05.2026).
24. DevOps Statistics [Електронний ресурс] // Radixweb. URL: https://radixweb.com/blog/devops-statistics (дата звернення: 04.05.2026).
25. Azure Statistics [Електронний ресурс] // Turbo360. URL: https://turbo360.com/blog/azure-statistics (дата звернення: 04.05.2026).
26. Who uses Jira? [Електронний ресурс] // Atlassian. URL: https://www.atlassian.com/software/jira/guides/getting-started/who-uses-jira (дата звернення: 04.05.2026).
27. Asana (ASAN) Statistics [Електронний ресурс] // WallStreetZen. URL: https://www.wallstreetzen.com/stocks/us/nyse/asan/statistics (дата звернення: 04.05.2026).
28. Trello Statistics [Електронний ресурс] // SQ Magazine. URL: https://sqmagazine.co.uk/trello-statistics/ (дата звернення: 04.05.2026).
29. Jira Service Desk Usage [Електронний ресурс] // Landbase. URL: https://data.landbase.com/technology/jira-service-desk/ (дата звернення: 04.05.2026).
30. Trello Target Market [Електронний ресурс] // Canvas Business Model. URL: https://canvasbusinessmodel.com/blogs/target-market/trello-target-market (дата звернення: 04.05.2026).

31. monday.com Target Market [Електронний ресурс] // Canvas Business Model. URL: https://canvasbusinessmodel.com/blogs/target-market/monday-com-target-market (дата звернення: 04.05.2026).
32. How to Use Azure DevOps for Effective Project Management [Електронний ресурс] // OpsNexa. URL: https://www.opsnexa.com/how-to-use-azure-devops-for-effective-project-management/ (дата звернення: 04.05.2026).
33. Atlassian Marketing Strategy [Електронний ресурс] // Porter's Five Forces. URL: https://portersfiveforce.com/blogs/marketing-strategy/atlassian (дата звернення: 04.05.2026).
34. Asana Partners [Електронний ресурс] // Asana. URL: https://asana.com/partners (дата звернення: 04.05.2026).
35. Trello Comparison with Asana, Monday, and Jira [Електронний ресурс] // Hipporello. URL: https://www.hipporello.com/blog/trello-comparison-with-asana-monday-and-jira (дата звернення: 04.05.2026).
36. monday.com Partnerships [Електронний ресурс] // monday.com. URL: https://monday.com/w/partnership (дата звернення: 04.05.2026).
37. Azure DevOps Services Pricing [Електронний ресурс] // Microsoft Azure. URL: https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/ (дата звернення: 04.05.2026).
38. Jira Pricing Guide [Електронний ресурс] // The Digital Project Manager. URL: https://thedigitalprojectmanager.com/tools/jira-pricing/ (дата звернення: 04.05.2026).
39. Asana Pricing Guide [Електронний ресурс] // The Digital Project Manager. URL: https://thedigitalprojectmanager.com/tools/asana-pricing/ (дата звернення: 04.05.2026).
40. Trello Pricing Guide [Електронний ресурс] // The Digital Project Manager. URL: https://thedigitalprojectmanager.com/tools/trello-pricing/ (дата звернення: 04.05.2026).

41. monday.com Pricing 2026 [Електронний ресурс] // Waymaker OS. URL: https://www.waymakeros.com/learn/monday-com-pricing-2026 (дата звернення: 04.05.2026).
42. Jira Pricing Plans [Електронний ресурс] // SaaSworthy. URL: https://www.saasworthy.com/blog/jira-pricing-plans (дата звернення: 04.05.2026).
43. Asana subscriptions and pricing [Електронний ресурс] // Asana Help Center. URL: https://help.asana.com/s/article/asana-subscriptions-and-pricing?language=en_US (дата звернення: 04.05.2026).
44. Trello Pricing 2025 Complete Guide [Електронний ресурс] // UserJot. URL: https://userjot.com/blog/trello-pricing-2025-complete-guide (дата звернення: 04.05.2026).
45. Jira Reviews: Pros & Cons [Електронний ресурс] // Capterra. URL: https://www.capterra.com/p/19319/JIRA/#pros-cons (дата звернення: 04.05.2026).
46. Asana Review [Електронний ресурс] // PCMag UK. URL: https://uk.pcmag.com/productivity-2/1670/asana (дата звернення: 04.05.2026).
47. Trello Reviews [Електронний ресурс] // G2. URL: https://www.g2.com/products/trello/reviews (дата звернення: 04.05.2026).
48. monday.com Review [Електронний ресурс] // TechRadar. URL: https://www.techradar.com/reviews/mondaycom (дата звернення: 04.05.2026).
49. Azure DevOps Project Management [Електронний ресурс] // ProjectManager. URL: https://www.projectmanager.com/blog/azure-devops-project-management (дата звернення: 04.05.2026).
50. Azure DevOps vs Jira: An Executive’s Dilemma [Електронний ресурс] // AlphaBOLD. URL: https://www.alphabold.com/pioneering-the-digital-frontier-azure-devops-vs-jira-an-executives-dilemma/ (дата звернення: 04.05.2026).

51. Asana vs Monday vs Trello: A B2B Project Management Smackdown [Електронний ресурс] // DEV Community. URL: https://dev.to/michaelaiglobal/asana-vs-monday-vs-trello-a-b2b-project-management-smackdown-for-engineers-3j8l (дата звернення: 04.05.2026).
52. Comparison of Top Project Management Tools for 2025 [Електронний ресурс] // Workflo. URL: https://workflo.com/blogs/project-management/comparison-of-top-project-management-tools-for-2025/ (дата звернення: 04.05.2026).
53. OMG Unified Modeling Language (OMG UML). Version 2.5.1 [Електронний ресурс] // Object Management Group. URL: https://www.omg.org/spec/UML/2.5.1/PDF (дата звернення: 04.05.2026).
54. Use Case Diagram [Електронний ресурс] // PlantUML. URL: https://plantuml.com/use-case-diagram (дата звернення: 04.05.2026).
55. Cockburn A. Writing Effective Use Cases. Boston : Addison-Wesley Professional, 2000. 270 p.
56. Wiegers K., Beatty J. Software Requirements. 3rd ed. Redmond : Microsoft Press, 2013. 672 p.
57. ISO/IEC/IEEE 42010:2022. Software, systems and enterprise — Architecture description. Geneva : International Organization for Standardization, 2022. 35 p.
58. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 416 p.
59. Raju G., Richards M., Ford N. Head First Software Architecture. Sebastopol : O'Reilly Media, 2024. 500 p.
60. Sommerville I. Software Engineering. 9th ed. Boston : Pearson, 2010. 792 p.
61. Эванс Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем. М. : Вильямс, 2011. 448 с.
62. Ньюмен С. Создание микросервисов. 2-е изд. СПб. : Питер, 2022. 624 с.

63. NGINX's Event-Driven Architecture [Електронний ресурс] // Engineering at Scale. URL: https://engineeringatscale.substack.com/p/nginx-millions-connections-event-driven-architecture (дата звернення: 04.05.2026).
64. Building and Testing Traefik [Електронний ресурс] // Traefik Documentation. URL: https://doc.traefik.io/traefik/v2.0/contributing/building-testing/ (дата звернення: 04.05.2026).
65. Kong Gateway [Електронний ресурс] // Kong. URL: https://konghq.com/products/kong-gateway (дата звернення: 04.05.2026).
66. Любанович Б. FastAPI: веб-разработка на Python. СПб. : Питер, 2025. 416 с.
67. TechEmpower Framework Benchmarks [Електронний ресурс] // TechEmpower. URL: https://www.techempower.com/benchmarks/ (дата звернення: 04.05.2026).
68. Podoba V. Django vs Flask vs FastAPI for Software Founders [Електронний ресурс] / Vitaliy Podoba // SoftFormance Blog. 2022. URL: https://www.softformance.com/blog/django-vs-flask/ (дата звернення: 04.05.2026).
69. Benchmarks [Електронний ресурс] // FastAPI Documentation. URL: https://fastapi.tiangolo.com/benchmarks/ (дата звернення: 04.05.2026).
70. TechEmpower Framework Benchmarks [Електронний ресурс] // TechEmpower. URL: https://www.techempower.com/benchmarks/ (дата звернення: 04.05.2026).
71. Vue.js: The Progressive JavaScript Framework [Електронний ресурс] // Vue.js. URL: https://vuejs.org/index.html (дата звернення: 04.05.2026).
72. React: The library for web and native user interfaces [Електронний ресурс] // React. URL: https://react.dev/ (дата звернення: 04.05.2026).
73. Flutter User Interface [Електронний ресурс] // Flutter Documentation. URL: https://docs.flutter.dev/ui (дата звернення: 04.05.2026).

74. Best Vector Databases [Електронний ресурс] // Encore. URL: https://encore.dev/articles/best-vector-databases (дата звернення: 04.05.2026).
75. Cernja I. Best Vector Databases in 2026 [Електронний ресурс] / Ivan Cernja // Encore. 2026. URL: https://encore.dev/articles/best-vector-databases (дата звернення: 04.05.2026).
76. Tahir H. We Tried and Tested 10 Best Vector Databases for RAG Pipelines [Електронний ресурс] / Hamza Tahir // ZenML Blog. 2025. URL: https://www.zenml.io/blog/vector-databases-for-rag (дата звернення: 04.05.2026).
77. Peralta J. H. Microservice APIs. Shelter Island : Manning Publications, 2023. 360 p.
78. Codex A. C. PostgreSQL in the Microservices Architecture [Електронний ресурс] / Arthur C. Codex // Reintech. 2024. URL: https://reintech.io/blog/postgresql-microservices-architecture (дата звернення: 04.05.2026).
79. Rogers J. PostgreSQL vs MySQL (2026): Key Differences, Detailed Comparison, & Performance Insights [Електронний ресурс] / Jeremy Rogers // NetCom Learning. 2026. URL: https://www.netcomlearning.com/blog/postgresql-vs-mysql (дата звернення: 04.05.2026).
80. Silén R. MariaDB vs PostgreSQL: Understanding the Architectural Differences That Matter [Електронний ресурс] / Robert Silén // MariaDB.org. 2025. URL: https://mariadb.org/mariadb-vs-postgresql-understanding-the-architectural-differences-that-matter/ (дата звернення: 04.05.2026).
81. Malik A., Burney A., Ahmed F. A Comparative Study of Unstructured Data with SQL and NO-SQL Database Management Systems [Електронний ресурс] // Journal of Computer and Communications. 2020. Vol. 8, No. 4. P. 59–71. DOI: 10.4236/jcc.2020.84005. URL:

https://www.scirp.org/journal/paperinformation?paperid=99539 (дата звернення: 04.05.2026).

82. PostgreSQL in Microservices Architecture [Електронний ресурс] // Reintech. URL: https://reintech.io/blog/postgresql-microservices-architecture (дата звернення: 04.05.2026).
83. Rocha H. F. O. Practical Event-Driven Microservices Architecture. New York : Apress, 2022. 343 p.
84. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. СПб. : Питер, 2023. 544 с.
85. Ollama Documentation [Електронний ресурс] // Ollama. URL: https://docs.ollama.com/ (дата звернення: 04.05.2026).
86. bge-m3 [Електронний ресурс] // Ollama. URL: https://ollama.com/library/bge-m3 (дата звернення: 04.05.2026).
87. intfloat/multilingual-e5-large Model [Електронний ресурс] // Hugging Face. URL: https://huggingface.co/intfloat/multilingual-e5-large (дата звернення: 04.05.2026).
88. intfloat/e5-mistral-7b-instruct Model [Електронний ресурс] // Hugging Face. URL: https://huggingface.co/intfloat/e5-mistral-7b-instruct (дата звернення: 04.05.2026).
89. nomic-ai/nomic-embed-text-v1.5 Model [Електронний ресурс] // Hugging Face. URL: https://huggingface.co/nomic-ai/nomic-embed-text-v1.5 (дата звернення: 04.05.2026).
90. BAAI/bge-m3 Model [Електронний ресурс] // Hugging Face. URL: https://huggingface.co/BAAI/bge-m3 (дата звернення: 04.05.2026).
91. Gemma 4 model card [Електронний ресурс] // Google AI for Developers. URL: https://ai.google.dev/gemma/docs/core/model_card_4 (дата звернення: 04.05.2026).
92. Manik M. M. H., Wang G. Gemma 4, Phi-4, and Qwen3: Accuracy–Efficiency Tradeoffs in Dense and MoE Reasoning Language Models [Електронний

ресурс] // arXiv. 2026. arXiv:2604.07035v1. URL: https://arxiv.org/html/2604.07035v1 (дата звернення: 04.05.2026).

93. lama 4 Model Card [Електронний ресурс] // Meta Llama / GitHub. 2025. URL: https://github.com/meta-llama/llama-models/blob/main/models/llama4/MODEL_CARD.md (дата звернення: 04.05.2026).
94. google/gemma-4-E4B Model [Електронний ресурс] // Hugging Face. URL: https://huggingface.co/google/gemma-4-E4B (дата звернення: 04.05.2026).
95. Principles behind the Agile Manifesto [Електронний ресурс] // Agile Manifesto. URL: https://agilemanifesto.org/principles.html (дата звернення: 04.05.2026).
96. The Scrum Guide [Електронний ресурс] // Scrum.org. URL: https://www.scrum.org/resources/scrum-guide (дата звернення: 04.05.2026).

ДОДАТОК А

Структура проєкту

```
PMSoft/
├── docker-compose.yaml
├── run-create-connector.bat
├── tasks-connector.ps1
├── users-connector.ps1
├── analysis-service/
│   ├── .dockerignore
│   ├── .env.docker
│   ├── Dockerfile
│   ├── auth.py
│   ├── job_store.py
│   ├── kafka_consumer.py
│   ├── kafka_producer.py
│   ├── main.py
│   └── requirements.txt
├── api-gateway/
│   └── nginx.conf
├── auth-service/
│   ├── .env.docker
│   ├── Dockerfile
│   ├── database.py
│   ├── main.py
│   ├── models.py
│   ├── requirements.txt
│   ├── router.py
│   ├── schemas.py
│   └── security.py
├── frontend-service/
│   ├── .env.docker
```

```
│   ├── Dockerfile
│   ├── index.html
│   ├── package.json
│   ├── package-lock.json
│   ├── vite.config.js
│   └── src/
│       ├── App.vue
│       ├── main.js
│       ├── style.css
│       ├── api/
│       │   └── http.js
│       ├── assets/
│       │   ├── edit.svg
│       │   ├── hero.png
│       │   ├── log-out.svg
│       │   ├── trash-2.svg
│       │   ├── vite.svg
│       │   └── vue.svg
│       ├── components/
│       │   ├── AppSidebar.vue
│       │   ├── CreateTaskModal.vue
│       │   ├── EditTaskModal.vue
│       │   ├── HelloWorld.vue
│       │   ├── KanbanBoard.vue
│       │   ├── ProjectAnalyticsCard.vue
│       │   ├── ProjectAnalyticsHeader.vue
│       │   ├── ProjectChartsGrid.vue
│       │   ├── ProjectEvmGrid.vue
│       │   ├── ProjectModal.vue
│       │   ├── ProjectSummaryGrid.vue
│       │   ├── ProjectTeam.vue
│       │   ├── TaskCard.vue
│       │   ├── TaskDetailsModal.vue
│       │   └── TaskSearch.vue
```

```
│       ├── router/
│       │   └── index.js
│       └── views/
│           ├── AnalyticsBoard.vue
│           ├── Dashboard.vue
│           ├── Login.vue
│           ├── ProjectBoard.vue
│           └── Register.vue
├── reporting-service/
│   ├── .env.docker
│   ├── Dockerfile
│   ├── analytics_calculations.py
│   ├── auth.py
│   ├── database.py
│   ├── kafka_consumer.py
│   ├── main.py
│   ├── models.py
│   ├── report_builder.py
│   ├── requirements.txt
│   └── schemas.py
└── task-service/
    ├── .env.docker
    ├── Dockerfile
    ├── auth.py
    ├── database.py
    ├── kafka_consumer.py
    ├── main.py
    ├── models.py
    ├── requirements.txt
    └── schemas.py
```

ДОДАТОК Б

Лістинг програмного коду

Лістинг коду мікросервісу аналізу даних

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    await init_qdrant_collection()
    await ensure_analysis_topics_exist()
    background_tasks = [
        asyncio.create_task(run_supervised_background_task("Task CDC consumer",
consume_task_events)),
        asyncio.create_task(run_supervised_background_task("Interpretation worker",
consume_interpretation_requests)),
        asyncio.create_task(run_supervised_background_task("Analysis AI worker",
consume_analysis_ai_requests)),
        asyncio.create_task(run_supervised_background_task("Job cleanup    loop",
cleanup_jobs_loop)), ]
    try:
        yield
    finally:
        for task in background_tasks:
            task.cancel()
            with suppress(asyncio.CancelledError):
                await task
        await close_kafka_producer()

app = FastAPI(
    title="Analysis Service API",
    description=(
        "Мікросервіс аналізу завдань"),
    version="1.0.0",
```

```
    docs_url="/docs/analysis",
    openapi_url="/analysis/openapi.json",
    lifespan=lifespan,
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
def _format_results(raw_results: List[Dict[str, Any]]) -> List[SearchResult]: return [
        SearchResult(
            id=result["id"],
            score=result["score"],
            payload=result.get("payload", {}),
        )
        for result in raw_results
    ]
@app.post(
    "/analysis/search/request",
    response_model=JobQueuedResponse,
    status_code=status.HTTP_202_ACCEPTED,
    tags=["Search"],
)
async def request_semantic_search(
    request: SearchRequest,
    current_user: CurrentUser = Depends(get_current_user), token: str = Depends(oauth2_scheme),
):
    await ensure_project_access(request.project_id, token)
    query = request.query.strip()
    if not query:
        raise HTTPException(status_code=400, detail="Потрібно передати query")
    job_id = str(uuid.uuid4())
```

```
    await create_job(job_id, "semantic_search", {**request.model_dump(), "requested_by":
str(current_user.id)})
    try:
        await publish_analysis_search_request(
            job_id=job_id,
            query=query,
            limit=request.limit,
            project_id=request.project_id,
            status_filter=request.status,
            priority=request.priority,
            assignee_id=request.assignee_id,
        )
    except Exception as exc:
        raise HTTPException(status_code=500, detail=str(exc))
    return JobQueuedResponse(
        job_id=job_id,
        status="queued",
        message="Запит на семантичний пошук поставлено в чергу",
    )
@app.post(
    "/analysis/duplicates/request",
    response_model=JobQueuedResponse,
    status_code=status.HTTP_202_ACCEPTED,
    tags=["Duplicates"],
)
async def request_duplicate_search(
    request: DuplicateRequest,
    current_user: CurrentUser = Depends(get_current_user),
    token: str = Depends(oauth2_scheme),):
    await ensure_project_access(request.project_id, token)
    title = (request.title or "").strip()
    description = (request.description or "").strip()
    if not title:
        raise HTTPException(status_code=400, detail="Потрібно передати title")
```

```
    job_id = str(uuid.uuid4())
    await create_job(job_id, "duplicate_search", {**request.model_dump(), "requested_by":
str(current_user.id)})
    try:
        await publish_duplicate_search_request(
            job_id=job_id,
            title=title,
            description=description,
            project_id=request.project_id,
            limit=request.limit or 5,
        )
    except Exception as exc:
        raise HTTPException(status_code=500, detail=str(exc))
    return JobQueuedResponse(
        job_id=job_id,
        status="queued",
        message="Запит на пошук дублікатів поставлено в чергу",
    )
@app.post(
    "/analysis/interpretation",
    response_model=InterpretationQueuedResponse,
    status_code=status.HTTP_202_ACCEPTED,
    tags=["Interpretation"],
)
async def request_task_interpretation(
    request: InterpretationRequest,
    current_user: CurrentUser = Depends(get_current_user),
    token: str = Depends(oauth2_scheme),):
    await ensure_project_access(request.project_id, token)
    try:
        title = (request.title or "").strip()
        description = (request.description or "").strip()
        if not title and not description:
            raise HTTPException(status_code=400, detail="Потрібно передати title або description")
```

```
        request_id = await publish_task_interpretation_request(
            task_id=request.task_id,
            project_id=request.project_id,
            title=title,
            description=description,
            source_updated_at=request.source_updated_at.isoformat() if request.source_updated_at else
None,
        )
        return InterpretationQueuedResponse(
            request_id=request_id,
            task_id=request.task_id,
            status="queued",
            message="Запит на AI-пояснення поставлено в чергу",
        )
    except HTTPException:
        raise
    except Exception as exc:
        raise HTTPException(status_code=500, detail=str(exc))
@app.get("/analysis/jobs/{job_id}", response_model=JobStatusResponse, tags=["Jobs"]) async def
get_analysis_job(
    job_id: str,
    current_user: CurrentUser = Depends(get_current_user),
    token: str = Depends(oauth2_scheme),
):
    job = await get_job(job_id)
    if not job:
        raise HTTPException(status_code=404, detail="AI job не знайдено або результат вже
очищено")
    payload = job.get("payload") or {}
    requested_by = payload.get("requested_by")
    if requested_by and str(requested_by) != str(current_user.id) and not current_user.is_superuser:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Доступ до цього AI job
            заборонено",
```

```
        )
    project_id = payload.get("project_id")
    if project_id is not None:
        try:
            project_id = int(project_id)
        except (TypeError, ValueError):
            raise HTTPException(status_code=400, detail="Некоректний project_id у job payload")
        await ensure_project_access(project_id, token)
    return JobStatusResponse(**job)
```

Лістинг коду мікросервісу авторизації

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    Base.metadata.create_all(bind=engine)
    yield
app = FastAPI(
    title="Auth Service",
    description="Мікросервіс для реєстрації, авторизації та управління JWT токенами",
    version="1.0.0",
    lifespan=lifespan,
    docs_url="/docs/auth",
    openapi_url="/auth/openapi.json",
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(auth_router)
router = APIRouter(prefix="/auth", tags=["Authentication"])
def build_token_response(user: User) -> dict[str, str | int]:
```

```
    access_token = create_access_token(subject=str(user.id), email=user.email)
    refresh_token = create_refresh_token(subject=str(user.id), email=user.email)
    return {
        "access_token": access_token,
        "refresh_token": refresh_token,
        "token_type": "bearer",
        "expires_in": ACCESS_TOKEN_EXPIRE_MINUTES * 60, }
@router.post(
    "/register",
    response_model=UserResponse,
    status_code=status.HTTP_201_CREATED,
    summary="Реєстрація нового користувача",
)
def register(user_in: UserCreate, db: Session = Depends(get_db)):
    existing_user = db.query(User).filter(User.email == user_in.email).first() if existing_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Користувач з таким email вже існує.",
        )
    hashed_password = get_password_hash(user_in.password)
    new_user = User(
        email=user_in.email,
        full_name=user_in.full_name,
        hashed_password=hashed_password,
    )
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user
@router.post(
    "/login",
    response_model=Token,
    summary="Авторизація та отримання токенів",
)
```

```
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db),
):
    user = db.query(User).filter(User.email == form_data.username).first()
    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Невірний email або пароль", headers={"WWW-Authenticate": "Bearer"},
        )
    if not user.is_active:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Користувач неактивний",
        )
    return build_token_response(user)
@router.post(
    "/refresh",
    response_model=Token,
    summary="Оновлення access та refresh токенів",
)
def refresh_tokens(
    token_in: RefreshTokenRequest,
    db: Session = Depends(get_db),
):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Недійсний або прострочений refresh_token", headers={"WWW-Authenticate":
        "Bearer"},
)
    try:
        payload = decode_token(token_in.refresh_token, expected_type="refresh")
        user_id = uuid.UUID(str(payload["sub"]))
    except (ValueError, KeyError):
```

```
        raise credentials_exception
    user = db.query(User).filter(User.id == user_id).first()
    if not user or not user.is_active:
        raise credentials_exception
    return build_token_response(user)
SECRET_KEY = os.getenv("SECRET_KEY")
if not SECRET_KEY:
    raise RuntimeError("SECRET_KEY environment variable is required for Auth Service")
ALGORITHM = os.getenv("JWT_ALGORITHM", "HS256")
ACCESS_TOKEN_EXPIRE_MINUTES =
int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "30"))
REFRESH_TOKEN_EXPIRE_DAYS = int(os.getenv("REFRESH_TOKEN_EXPIRE_DAYS",
"14"))
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
def verify_password(plain_password: str, hashed_password: str) -> bool: return
    pwd_context.verify(plain_password, hashed_password)
def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)
def create_access_token(
    subject: Union[str, Any],
    expires_delta: Optional[timedelta] = None,
    email: Optional[str] = None,
) -> str:
    expire = datetime.utcnow() + (
        expires_delta if expires_delta
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    to_encode = {
        "exp": expire,
        "sub": str(subject),
        "type": "access",
    }
    if email:
        to_encode["email"] = email
```

```
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
def create_refresh_token(subject: Union[str, Any], email: Optional[str] = None) -> str:
    expire = datetime.utcnow() + timedelta(days=REFRESH_TOKEN_EXPIRE_DAYS)
    to_encode = {
        "exp": expire,
        "sub": str(subject),
        "type": "refresh",
    }
    if email:
        to_encode["email"] = email
def decode_token(token: str, expected_type: Optional[str] = None) -> dict[str, Any]: try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
    except ExpiredSignatureError as exc:
        raise ValueError("Token has expired") from exc
    except InvalidTokenError as exc:
        raise ValueError("Invalid token") from exc
    token_type = payload.get("type")
    if expected_type and token_type != expected_type:
        raise ValueError(f"Invalid token type. Expected {expected_type}")
    if not payload.get("sub"):
        raise ValueError("Token subject is missing")
    return payload
```

Лістинг коду мікросервісу звітності

```
async def send_to_slack(text: str, webhook_url: str | None) -> None:
    if not webhook_url:
        return
    async with httpx.AsyncClient(timeout=15.0) as client:
        response = await client.post(webhook_url, json={"text": text})
        response.raise_for_status()
@asynccontextmanager
async def lifespan(app: FastAPI):
    Base.metadata.create_all(bind=engine)
    kafka_task = asyncio.create_task(consume_debezium_events())
    try:
```

```
        yield
    finally:
        kafka_task.cancel()
        with suppress(asyncio.CancelledError): await kafka_task
app = FastAPI(
    title="Reporting Service", version="2.0.0", docs_url="/docs/reports",
    openapi_url="/reports/openapi.json", lifespan=lifespan,
)
app.add_middleware( CORSMiddleware, allow_origins=["*"], allow_credentials=True,
    allow_methods=["*"], allow_headers=["*"],
)
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally: db.close()
def normalize_user_id(user_id: Any) -> str: return str(user_id)
    def get_project_or_404(db: Session, project_id: int) -> ProjectSnapshot:
    project = db.query(ProjectSnapshot).filter(ProjectSnapshot.id == project_id).first() if not project:
        raise HTTPException(status_code=404, detail="Project not found")
    return project
def get_project_member(
    db: Session,
    project_id: int,
    user_id: Any,
) -> ProjectMemberSnapshot | None:
    return (
        db.query(ProjectMemberSnapshot)
        .filter(
            ProjectMemberSnapshot.project_id == project_id,
            ProjectMemberSnapshot.user_id == normalize_user_id(user_id),
        )
        .first()
    )
```

```
def require_project_member(
    db: Session,
    project_id: int,
    current_user: CurrentUser,
) -> ProjectSnapshot:
    project = get_project_or_404(db, project_id)
    if current_user.is_superuser:
        return project
    current_user_id = normalize_user_id(current_user.id)
    if project.owner_id and str(project.owner_id) == current_user_id:
        return project
    member = get_project_member(db, project_id, current_user.id)
    if not member:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Access denied",
        )
    return project
def require_project_owner(
    db: Session,
    project_id: int,
    current_user: CurrentUser,
) -> ProjectSnapshot:
    project = get_project_or_404(db, project_id)
    if current_user.is_superuser:
        return project
    current_user_id = normalize_user_id(current_user.id)
    if project.owner_id and str(project.owner_id) == current_user_id: return project
    member = get_project_member(db, project_id, current_user.id)
    if member and member.role == "owner":
        return project
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN, detail="Only project owner can perform this
        action",
    )
```

```
@app.get("/reports/analytics/{project_id}")
def get_project_analytics(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
) -> Dict[str, Any]:
    require_project_member(db, project_id, current_user)
    return calculate_project_analytics(project_id=project_id, db=db)
@app.post("/reports/generate/{project_id}")
async def generate_weekly_report(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user)
    try:
        result = build_weekly_report(db, project_id)
        if not result:
            raise HTTPException(status_code=404, detail="Project not found")
        if isinstance(result, tuple):
            final_text, webhook_url = result
        else:
            final_text = result
            webhook_url = None
        if not final_text:
            raise HTTPException(status_code=404, detail="Project not found")
        await send_to_slack(final_text, webhook_url)
        return {
            "status": "Report generated",
            "project_id": project_id,
            "preview": final_text,
            "sent_to_slack": bool(webhook_url),
        }
    except HTTPException:
```

```
        raise
    except Exception as exc:
        raise HTTPException(status_code=500, detail=f"Failed to generate report: {exc}")
@app.post("/reports/seed-metrics/{project_id}")
def seed_test_metrics(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    project = require_project_owner(db, project_id, current_user)
    db.query(ProjectDailyMetric).filter(ProjectDailyMetric.project_id == project_id).delete()
    budget = float(project.budget or 10000.0)
    days_to_generate = 30
    base_date = date.today() - timedelta(days=days_to_generate)
    total_tasks = 50
    done_tasks = 0
    actual_cost = 0.0
    for i in range(days_to_generate + 1):
        current_date = base_date + timedelta(days=i)
        progress_ratio = i / days_to_generate
        pv = budget * (progress_ratio ** 1.5)

        if i > 3 and random.random() > 0.2:
            done_tasks += random.randint(0, 3) done_tasks = min(done_tasks, total_tasks)
        if i > 3:
            cost_increment = (budget / days_to_generate) * random.uniform(0.7, 1.6) actual_cost +=
            cost_increment
        metric = ProjectDailyMetric(
            project_id=project_id, metric_date=current_date, total_tasks=total_tasks,
            done_tasks=done_tasks, total_planned_cost=round(pv, 2),
            total_actual_cost=round(actual_cost, 2), backlog_tasks=total_tasks - done_tasks,
            todo_tasks=0,
            in_progress_tasks=random.randint(2, 6) if done_tasks < total_tasks else 0,
            in_review_tasks=0,
```

```
            overdue_tasks=random.randint(0, 3),
            unassigned_tasks=0,
            total_estimated_hours=total_tasks * 4.0, total_actual_hours=round(actual_cost / 25.0, 2),
            team_size=project.team_size or 5,
        )
        db.add(metric)
    db.commit()
    return {"status": "Test historical metrics generated", "days": days_to_generate}
```

Лістинг коду мікросервісу кеування завданнями

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    init_db(models.Base)
    kafka_tasks = [
        asyncio.create_task(consume_user_events()),
        asyncio.create_task(consume_task_interpretation_events()), ]
    try:
        yield
    finally:
        for kafka_task in kafka_tasks:
            kafka_task.cancel()
            with suppress(asyncio.CancelledError): await kafka_task
app = FastAPI(
    title="Task Service API",
    version="1.0.0",
    docs_url="/docs/tasks",
    openapi_url="/tasks/openapi.json", lifespan=lifespan,
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"], allow_headers=["*"],
)
```

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
def get_project_or_404(db: Session, project_id: int) -> models.Project:
    project = db.query(models.Project).filter(models.Project.id == project_id).first() if not project:
        raise HTTPException(status_code=404, detail="Проект не знайдено") return project
def require_project_member(
    db: Session,
    project_id: int,
    user_id: uuid.UUID,
) -> models.ProjectMember:
    get_project_or_404(db, project_id)
    member = (
        db.query(models.ProjectMember)
        .filter(
            models.ProjectMember.project_id == project_id,
            models.ProjectMember.user_id == user_id,
        )
        .first()
    )
    if not member:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Доступ заборонено",
        )
    return member
def require_project_owner(
    db: Session,
    project_id: int,
    user_id: uuid.UUID,
) -> models.Project:
```

```
    project = get_project_or_404(db, project_id)
    if project.owner_id != user_id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Лише власник проекту може виконувати цю дію",
        )
    return project
def get_task_or_404(db: Session, project_id: int, task_id: int) -> models.Task: task = (
        db.query(models.Task)
        .filter(models.Task.id == task_id, models.Task.project_id == project_id) .first()
    )
    if not task:
        raise HTTPException(status_code=404, detail="Завдання не знайдено") return task
def ensure_assignee_is_project_member(
    db: Session,
    project_id: int,
    assignee_id: Optional[uuid.UUID],
) -> None:
    if not assignee_id:
        return
    member = (
        db.query(models.ProjectMember)
        .filter(
            models.ProjectMember.project_id == project_id,
            models.ProjectMember.user_id == assignee_id,
        )
        .first()
    )
    if not member:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Виконавець має бути учасником проекту",
        )
def resolve_user_email(db: Session, user_id: uuid.UUID, token_email: Optional[str]) -> str:
```

```
    if token_email:
        return token_email
    cached_user = db.query(models.UserCache).filter(models.UserCache.id == user_id).first()
    if cached_user and cached_user.email:
        return cached_user.email
    raise HTTPException(
        status_code=status.HTTP_400_BAD_REQUEST,
        detail=(
            "Не вдалося визначити email поточного користувача. "
            "Увійдіть знову, щоб отримати токен з email, або дочекайтеся синхронізації
UserCache."
        ),
    )
def task_to_dict_with_assignee(db: Session, task: models.Task) -> dict:
    t_dict = schemas.TaskResponse.model_validate(task).model_dump()
    if task.assignee_id:
        user = db.query(models.UserCache).filter_by(id=task.assignee_id).first()
        if user:
            t_dict["assignee_name"] = user.full_name or user.email return t_dict
def project_member_to_dict(db: Session, member: models.ProjectMember) -> dict:
    m_dict = schemas.ProjectMemberResponse.model_validate(member).model_dump()
    user = db.query(models.UserCache).filter(models.UserCache.id == member.user_id).first()
    m_dict["user_name"] = (user.full_name or user.email) if user else member.user_email return
    m_dict
@app.get("/projects/", response_model=List[dict])
def get_user_projects(
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    memberships = (
        db.query(models.ProjectMember)
        .filter(models.ProjectMember.user_id == current_user.id)
        .all()
    )
```

```
    result = []
    for member in memberships:
        proj = db.query(models.Project).filter(models.Project.id == member.project_id).first() if proj:
            p_dict = schemas.ProjectResponse.model_validate(proj).model_dump() p_dict["role"] =
            member.role
            result.append(p_dict)
    return result
@app.get("/projects/{project_id}", response_model=schemas.ProjectResponse)
def get_project(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
    return get_project_or_404(db, project_id)
@app.post("/projects/",                                    response_model=schemas.ProjectResponse,
status_code=status.HTTP_201_CREATED) def create_project(
    project: schemas.ProjectCreate,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    db_project = models.Project(**project.model_dump(), owner_id=current_user.id)
    db.add(db_project)
    db.commit()
    db.refresh(db_project)
    owner_email = resolve_user_email(db, current_user.id, current_user.email)
    owner_member = models.ProjectMember(
        project_id=db_project.id,
        user_id=current_user.id,
        user_email=owner_email,
        role="owner",
    )
    db.add(owner_member)
    db.commit()
```

```
    return db_project
@app.patch("/projects/{project_id}", response_model=schemas.ProjectResponse)
def update_project(
    project_id: int,
    project_update: schemas.ProjectUpdate,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    project = require_project_owner(db, project_id, current_user.id)
    update_data = project_update.model_dump(exclude_unset=True)
    for field, value in update_data.items():
        setattr(project, field, value)
    db.commit()
    db.refresh(project)
    return project
@app.delete("/projects/{project_id}", status_code=status.HTTP_204_NO_CONTENT) def
delete_project(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    project = require_project_owner(db, project_id, current_user.id)
    db.delete(project)
    db.commit()
    return None
@app.post("/projects/{project_id}/add_members",
response_model=schemas.ProjectMemberResponse)
def add_project_member(
    project_id: int,
    request: schemas.ProjectMemberAddRequest,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_owner(db, project_id, current_user.id)
```

```
    user_in_cache = db.query(models.UserCache).filter(models.UserCache.email
request.email).first()
    if not user_in_cache:
        raise HTTPException( status_code=404, detail=f"Користувача {request.email} ще немає в
            системі (UserCache).",
        )
    existing_member = (
        db.query(models.ProjectMember)
        .filter_by(project_id=project_id, user_id=user_in_cache.id)
        .first()
    )
    if existing_member:
        raise HTTPException(status_code=400, detail="Користувач вже є учасником цього
проекту")
    new_member = models.ProjectMember(
        project_id=project_id,
        user_id=user_in_cache.id,
        user_email=user_in_cache.email,
        role=request.role or "member",
        hourly_rate=request.hourly_rate or 0,
    )
    db.add(new_member)
    db.commit()
    db.refresh(new_member)
    return project_member_to_dict(db, new_member)
@app.delete("/projects/{project_id}/members/{user_id}",
status_code=status.HTTP_204_NO_CONTENT)
def delete_project_member(
    project_id: int,
    user_id: uuid.UUID,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    project = require_project_owner(db, project_id, current_user.id)
```

```
    member = (
        db.query(models.ProjectMember)
        .filter(
            models.ProjectMember.project_id == project_id,
            models.ProjectMember.user_id == user_id,
        )
        .first()
    )
    if not member:
        raise HTTPException(status_code=404, detail="Учасника не знайдено")
    if member.user_id == project.owner_id:
        raise HTTPException(status_code=400, detail="Неможливо видалити власника проекту")
    db.delete(member)
    db.commit()
    return None
@app.get("/projects/{project_id}/members",
response_model=List[schemas.ProjectMemberResponse])
def get_project_members(
    project_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
    members = (
        db.query(models.ProjectMember)
        .filter(models.ProjectMember.project_id == project_id)
        .all()
    )
    return [project_member_to_dict(db, member) for member in members]
@app.post("/projects/{project_id}/tasks/", response_model=schemas.TaskResponse)
def create_task(
    project_id: int,
    task: schemas.TaskCreate,
    db: Session = Depends(get_db),
```

```
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
    ensure_assignee_is_project_member(db, project_id, task.assignee_id)
    db_task = models.Task(**task.model_dump(), project_id=project_id)
    db.add(db_task)
    db.commit()
    db.refresh(db_task)
    return task_to_dict_with_assignee(db, db_task)
@app.get("/projects/{project_id}/tasks/", response_model=List[schemas.TaskResponse]) def
get_tasks(
    project_id: int,
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
    tasks = (
        db.query(models.Task)
        .filter(models.Task.project_id == project_id)
        .offset(skip)
        .limit(limit)
        .all()
    )
    return [task_to_dict_with_assignee(db, t) for t in tasks]
@app.get("/projects/{project_id}/tasks/{task_id}", response_model=schemas.TaskResponse) def
get_task(
    project_id: int,
    task_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
```

```
    task = get_task_or_404(db, project_id, task_id)
    return task_to_dict_with_assignee(db, task)
@app.post(
    "/projects/{project_id}/tasks/{task_id}/interpretation/request",
    response_model=schemas.TaskResponse,
)
async def request_task_interpretation(
    project_id: int,
    task_id: int,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
    token: str = Depends(oauth2_scheme),
):
    require_project_member(db, project_id, current_user.id)
    task = get_task_or_404(db, project_id, task_id)
    if not task.title and not task.description:
        raise HTTPException(
            status_code=400,
            detail="Для генерації пояснення потрібна назва або опис завдання",
        )
    task.interpretation = None
    task.interpretation_generated_at = None
    task.interpretation_status = "queued"
    task.interpretation_requested_at = datetime.utcnow()
    task.interpretation_processing_started_at = None
    task.interpretation_error = None
    db.commit()
    db.refresh(task)
    source_updated_at = task.updated_at.isoformat() if task.updated_at else None
    payload = {
        "task_id": task.id,
        "project_id": task.project_id,
        "title": task.title or "",
        "description": task.description or "",
```

```
    "source_updated_at": source_updated_at,
}
try:
    async with httpx.AsyncClient(timeout=60.0) as client:
        response = await client.post(
            f"{ANALYTICAL_SERVICE_URL}/analysis/interpretation",
            json=payload,
            headers={"Authorization": f"Bearer {token}"},
        )
        response.raise_for_status()
    data = response.json() if response.content else {}
    immediate_interpretation = (
        data.get("interpretation")
        or data.get("explanation")
        or data.get("result")
    )
    if immediate_interpretation:
        task.interpretation = str(immediate_interpretation) task.interpretation_generated_at =
        datetime.utcnow() task.interpretation_status = "completed"
        task.interpretation_error = None
        db.commit()
        db.refresh(task)
    return task_to_dict_with_assignee(db, task)
except httpx.HTTPStatusError as exc:
    error_text = exc.response.text or str(exc)
except httpx.RequestError as exc:
    error_text = f"Analysis Service недоступний: {exc}"
except Exception as exc:
    error_text = str(exc)
task.interpretation_status = "failed"
task.interpretation_error = error_text
db.commit()
db.refresh(task)
raise HTTPException(
```

```
        status_code=status.HTTP_502_BAD_GATEWAY, detail=error_text,
    )
@app.patch("/projects/{project_id}/tasks/{task_id}", response_model=schemas.TaskResponse) def
update_task(
    project_id: int,
    task_id: int,
    task_update: schemas.TaskUpdate,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_member(db, project_id, current_user.id)
    task = get_task_or_404(db, project_id, task_id)
    update_data = task_update.model_dump(exclude_unset=True)
    if "assignee_id" in update_data:
        ensure_assignee_is_project_member(db, project_id, update_data.get("assignee_id"))
    description_changed = False
    if "description" in update_data:
        current_description = task.description or ""
        new_description = update_data.get("description") or ""
        description_changed = current_description != new_description
    for field, value in update_data.items(): setattr(task, field, value)
     if description_changed:
        task.interpretation = None
        task.interpretation_generated_at = None task.interpretation_status = "not_started"
        task.interpretation_requested_at = None task.interpretation_processing_started_at = None
        task.interpretation_error = None
    db.commit()
    db.refresh(task)
    return task_to_dict_with_assignee(db, task)
@app.delete("/projects/{project_id}/tasks/{task_id}",
status_code=status.HTTP_204_NO_CONTENT) def delete_task(
    project_id: int,
    task_id: int,
    db: Session = Depends(get_db),
```

```
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_owner(db, project_id, current_user.id) task = get_task_or_404(db, project_id,
    task_id)
    db.delete(task)
    db.commit()
    return None
@app.patch("/projects/{project_id}/members/{user_id}/rate",
response_model=schemas.ProjectMemberResponse)
def set_project_member_rate(
    project_id: int,
    user_id: uuid.UUID,
    request: schemas.ProjectMemberRateUpdate,
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    require_project_owner(db, project_id, current_user.id)
    member = (
        db.query(models.ProjectMember)
        .filter(
            models.ProjectMember.project_id == project_id,
            models.ProjectMember.user_id == user_id,
        )
        .first()
    )
    if not member:
        raise HTTPException(status_code=404, detail="Учасника проекту не знайдено")
    member.hourly_rate = request.hourly_rate
    db.commit()
    db.refresh(member)
    return project_member_to_dict(db, member)
@app.post("/tasks/hydrate", response_model=List[schemas.TaskResponse])
def hydrate_tasks(
    task_ids: List[int],
```

```
    db: Session = Depends(get_db),
    current_user: CurrentUser = Depends(get_current_user),
):
    if not task_ids:
        return []
    accessible_project_ids = (
        db.query(models.ProjectMember.project_id)
        .filter(models.ProjectMember.user_id == current_user.id)
        .subquery()
    )
    tasks = (
        db.query(models.Task)
        .filter(
            models.Task.id.in_(task_ids),
            models.Task.project_id.in_(accessible_project_ids),
        )
        .all()
    )
    task_map = {task.id: task for task in tasks}
    ordered_tasks = [task_map[task_id] for task_id in task_ids if task_id in task_map]
    return [task_to_dict_with_assignee(db, task) for task in ordered_tasks]
```