

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет _____ комп'ютерних наук
(повна назва)

Кафедра _____ програмної інженерії
(повна назва)

КОМПЛЕКСНИЙ КУРСОВИЙ ПРОЄКТ

Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Клієнт-серверний застосунок для керування проєктами з системою аналітики
виконання завдань.
(тема)

Виконав:

здобувач _____ 3 _____ курсу, групи _____ ПЗПІ-22-5
_____ Іван ГРОМОВ
(Власне ім'я, ПРІЗВИЩЕ)

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

Тип програми _____ освітньо-професійна
Освітня програма _____ Програмна інженерія
(повна назва освітньої програми)

Керівник _____ доц. кафедри ПІ
_____ Дмитро КОЛЕСНИКОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Члени комісії (Власне ім'я, ПРІЗВИЩЕ, підпис)

_____ Юрій НОВІКОВ

_____ Марія ШИРОКОПЕТЛЄВА

_____ Олексій НАЗАРОВ

2025 р.

Харківський національний університет радіоелектроніки

Факультет _____ комп'ютерних наук _____
 Кафедра _____ програмної інженерії _____
 Рівень вищої освіти _____ перший (бакалаврський) _____
 Спеціальність _____ 121 – Інженерія програмного забезпечення _____
 Тип програми _____ Освітньо-професійна _____
 Освітня програма _____ Програма Інженерія _____
 (шифр і назва)

Курс _____ 3 _____ Група _____ ПЗПІ-22-5 _____ Семестр _____ 6 _____

ЗАВДАННЯ
на курсовий проект(роботу) студента

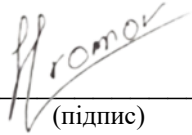
здобувачеві _____ Громову Івану Сергійовичу _____
 (прізвище, ім'я, по батькові)

1. Тема роботи Клієнт-серверний застосунок для керування проектами з системою аналітики виконання завдань.
2. Термін здачі студентом закінченої роботи „27” червня 2025 р.
3. Вихідні дані до проекту Розробити клієнт-серверний застосунок для управління проектами з можливістю створення завдань, призначення виконавців, відстеження статусу та перегляду аналітики. Система повинна підтримувати кількох користувачів з різними ролями в межах проекту. Клієнтська частина має взаємодіяти з REST API, підтримувати реєстрацію, редагування проектів і завдань, перегляд статистики. Серверна частина – обробляти запити, зберігати дані у Firebase, керувати доступом, формувати аналітику та логувати дії.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, постановка задачі, визначення функціональних вимог, аналіз предметної області, вибір технологічного стеку, розробка архітектури клієнтської та серверної частин, побудова REST API, авторизація та автентифікація користувачів, структура бази даних, реалізація аналітики виконання завдань, логування змін, реалізація клієнтського інтерфейсу, тестування, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	16.05.25 – 20.05.25	виконано
2	Розробка постановки задачі	21.05.25 – 24.05.25	виконано
3	Проектування ПЗ	25.05.25 – 01.06.25	виконано
4	Програмна реалізація	02.06.25 – 12.06.25	виконано
5	Аналіз результатів	13.06.25 – 15.06.25	виконано
6	Підготовка пояснювальної записки.	16.06.25 – 20.06.25	виконано
7	Перевірка на наявність ознак академічного плагіату	27.06.25	виконано
8	Захист роботи	27.06.25	виконано

Дата видачі завдання “27” лютого 2025р.

Здобувач 
(підпис)

Керівник роботи _____ доц. кафедри ПІ Дмитро КОЛЕСНИКОВ
(підпис) (посада, Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 68 стор., 17 рис., 9 джерел.

FIREBASE, JAVA, REACT, REST API, SPRING BOOT, VITE, WEB-ІНТЕРФЕЙС, CSS, КЛІЄНТ-СЕРВЕРНИЙ ЗАСТОСУНОК

Об'єкт розробки – клієнт-серверний вебзастосунок для керування проєктами, завданнями та учасниками з можливістю перегляду аналітики та інтеграцією з хмарним бек-ендом.

Мета розробки – створення функціонального застосунку для організації командної роботи над проєктами, що дозволяє користувачам ефективно планувати, виконувати й контролювати завдання в межах робочих просторів.

Метод рішення – розробка серверної частини на Java з використанням Spring Boot та збереженням даних у Firebase. Реалізація авторизації на базі Firebase-токенів. Створення клієнтської частини за допомогою React і Vite, з обробкою HTTP-запитів через REST API та стилізацією інтерфейсу за допомогою CSS.

FIREBASE, JAVA, REACT, REST API, SPRING BOOT, VITE, WEB INTERFACE, CSS, CLIENT-SERVER APPLICATION

The development object is a client-server web application for managing projects, tasks and participants with the ability to view analytics and integrate with a cloud backend.

The development goal is to create a functional application for organizing teamwork on projects, allowing users to effectively plan, execute and control tasks within workspaces.

The solution method is to develop the server part in Java using Spring Boot and save data in Firebase. Implementation of authorization based on Firebase tokens. Creation of the client part using React and Vite, with processing HTTP requests via REST API and styling the interface using CSS.

ЗМІСТ

Перелік скорочень	7
Вступ.....	8
1 Аналіз предметної галузі	10
1.1 Аналіз предметної галузі.....	10
1.2 Виявлення та вирішення проблем	11
1.3 Аналіз аналогів програмного забезпечення	12
2 Постановка задачі.....	16
3 Архітектура та проєктування програмного забезпечення	18
3.1 UML проєктування ПЗ.....	18
3.2 Проєктування архітектури ПЗ	22
3.3 Проєктування структури зберігання даних	25
3.4 Приклади найцікавіших алгоритмів та методів.....	27
3.4.1 Алгоритми фільтрації та пошуку завдань	27
3.4.2 Алгоритм розрахунку статистики виконання завдань	28
3.4.3 Метод авторизації через сторонній сервіс (Google Sign-In).....	28
3.5 Створення UI/UX дизайну системи.....	29
3.6 Опис підготовки даних	32
4 Опис прийнятих програмних рішень	34
4.1 Вибір стеку технологій.....	34
4.2 Організація серверної логіки	34
4.3 Реалізація клієнтської частини	35
4.4 Реалізація взаємодії з сервером	37
4.5 Забезпечення безпеки та контролю доступу	38

4.6 Реалізація аналітики.....	40
4.7 Логування подій	42
5 Аналіз отриманих результатів	44
5.1 Порівняння результатів з початковими вимогами	44
5.2 Оцінка якості роботи системи	45
5.3 Аналіз ефективності прийнятих рішень	46
5.4 Обмеження та недоліки	46
Висновки	48
Перелік джерел посилання	50
Додаток А Специфікація програмного забезпечення.....	51
Додаток Б Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ	62
Додаток В Слайди презентації.....	64

ПЕРЕЛІК СКОРОЧЕНЬ

SQL – Structured Query Language

API – Application Programming Interface

REST API – Representational State Transfer Application Programming Interface

CSS – Cascading Style Sheets

IT – Information Technology

DevOps – Development and Operations

CI/CD – Continuous Integration / Continuous Delivery

UML – Unified Modeling Language

HTTP – HyperText Transfer Protocol

DTO – Data Transfer Object

SDK – Software Development Kit

CRUD – Create, Read, Update, Delete

JSON – JavaScript Object Notation

UI – User Interface

UX – User Experience

ВСТУП

Стикаючись із багатьма паралельними завданнями, змінними пріоритетами й обмеженими ресурсами, учасники проєктів мають потребу в наочності, чіткості й упорядкованості. Замість нескінченних повідомлень у чатах, таблиць, що губляться в листуванні, або ручних записів, що швидко втрачають актуальність, все більше команд обирають веб-застосунки. Вони дозволяють структурувати потік інформації, бачити поточний стан задач, аналізувати активність і вчасно приймати рішення. Добре продуманий інтерфейс і прозора логіка взаємодії в таких системах дають змогу учасникам працювати ефективніше, а керівникам – оцінювати динаміку й швидко реагувати на зміни.

Особливої ваги набуває підтримка аналітики – автоматичне збирання статистики щодо прогресу, активності або навантаження кожного учасника. Такі функції дозволяють не лише оцінювати ситуацію в режимі реального часу, а й отримувати узагальнену картину – виявляти вузькі місця, ухвалювати зважені управлінські рішення, прогнозувати строки виконання та коригувати розподіл задач.

Темою курсового проєкту є розробка вебзастосунку CrewNote – клієнт-серверної системи для керування командними проєктами з підтримкою аналітики виконання задач. У межах застосунку користувачі можуть створювати проєкти, додавати учасників, формувати перелік завдань, змінювати їх статуси, переглядати статистику виконання та контролювати робочий процес. Усе це реалізовано в єдиному цифровому середовищі, доступному через вебінтерфейс із будь-якого пристрою.

Метою розробки є створення функціонального та зручного інструменту, який дає змогу організувати ефективну командну взаємодію. Застосунок має забезпечувати просту й зрозумілу взаємодію між користувачами та системою, швидкий доступ до актуальної інформації про проєкт, прозоре керування завданнями та мінімізацію ручної роботи, пов'язаної з відстеженням прогресу.

Архітектура системи реалізована за принципом розділення клієнтської та серверної частин. Бек-енд написаний на мові Java з використанням фреймворку Spring Boot та підключений до хмарної NoSQL-бази Firebase Firestore, яка також відповідає за автентифікацію користувачів. Клієнтська частина створена за допомогою бібліотеки React. Вся логіка обміну даними побудована на основі REST API. Стилзація інтерфейсу реалізована засобами CSS, а збирання клієнтської частини здійснюється через Vite.

Реалізований прототип включає базову функціональність: реєстрацію та вхід користувачів, створення й редагування проєктів, призначення виконавців, зміну статусів завдань і формування дашборду зі зведеною статистикою. Застосунок побудований на сучасному технологічному стеку з використанням Java, React і хмарного сервісу Firebase, що забезпечує гнучкість, масштабованість і готовність до подальшого розвитку. Архітектура проєкту дозволяє легко розширити його можливості – наприклад, додати комунікацію між користувачами, систему пріоритезації, розширену візуалізацію аналітики або інтеграцію з зовнішніми інструментами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сфера управління проєктами знаходить широке застосування в таких галузях, як інформаційні технології, будівництво, машинобудування, освіта, маркетинг, охорона здоров'я, культура та державне управління. Незалежно від галузі, проєктне управління базується на спільних принципах: визначення обсягу робіт, формування плану, управління ризиками, моніторинг виконання, контроль якості, управління комунікаціями та підготовка підсумкової звітності.

Центральним елементом у цій сфері є сам проєкт – сукупність взаємопов'язаних завдань, спрямованих на досягнення спільної мети. Кожне завдання в межах проєкту має опис, термін виконання, статус, виконавця та залежності від інших задач. Учасники команди можуть виконувати різні ролі – наприклад, менеджер, координатор, розробник, аналітик – і мати відповідні права доступу до інформації. Координацію командної роботи забезпечує менеджер проєкту, який відповідає за дотримання строків, ресурсну збалансованість, комунікацію та досягнення результатів.

Зважаючи на складність структури команд та високий обсяг інформації, актуальним є застосування рішень, що дозволяють централізовано зберігати дані, здійснювати розмежування прав доступу, синхронізувати зміни та надавати зведену інформацію в зручному вигляді. Відсутність подібного інструментарію або його недостатня функціональність призводить до інформаційного хаосу, дублювання задач, порушення термінів або перевантаження окремих учасників.

Особливу роль у проєктному середовищі відіграє моніторинг ходу виконання задач. Незалежно від якості планування, у процесі реалізації завжди виникають зміни: оновлення пріоритетів, зміщення строків, появу нових залежностей. Щоб оперативно реагувати на такі зміни, потрібні інструменти для відображення актуального статусу задач у зручній візуальній формі.

Ще одним важливим фактором є гнучкість – здатність оперативно перебудовувати план, додавати або змінювати задачі, оновлювати інформацію про команду, змінювати структуру проєкту або пріоритетність завдань. У нестабільних умовах, особливо в ІТ-проєктах, гнучкість часто є критичною для досягнення успіху.

Предметна область управління проєктами є складною, багаторівневою і критично залежною від інформаційної підтримки. Вона вимагає використання сучасних інструментів, які дозволяють автоматизувати ключові процеси, забезпечити контроль якості виконання задач і підвищити прозорість управлінських рішень.

1.2 Виявлення та вирішення проблем

Незважаючи на широкий розвиток цифрових інструментів, у сфері управління проєктами досі зберігаються типові проблеми, які ускладнюють ефективну реалізацію задач і досягнення цілей. Їх усунення є ключовою передумовою для підвищення прозорості та контрольованості проєктної діяльності.

Однією з основних проблем є відсутність єдиного простору для управління завданнями. Інформація про задачі часто зберігається в різних джерелах – таблицях, месенджерах, електронних листах – що призводить до дублювання, втрати актуальних даних або непорозумінь між виконавцями.

Ще одна поширена проблема – відсутність фіксації змін у статусах задач, дедлайнах або призначених виконавцях. Без журналу подій складно відслідковувати, хто і коли вніс зміни, що створює ризики втрати відповідальності та знижує прозорість процесу.

Нерівномірний розподіл навантаження – ще одна типова ситуація. За відсутності аналітики або системи контролю частина виконавців перевантажена, тоді як інші не мають достатньо задач. Це призводить до неоптимального використання ресурсів та порушення строків.

Також спостерігається недостатня наочність стану проєкту. Керівники часто не мають засобів для швидкої оцінки: скільки задач виконано, що в пріоритеті, які етапи потребують уваги. Відсутність візуального представлення статусів ускладнює ухвалення рішень.

Крім того, багато систем не передбачають гнучкого редагування задач і проєктів у реальному часі. Будь-яке оновлення вимагає декількох ручних дій або не оновлюється для всіх учасників синхронно, що знижує ефективність взаємодії.

Аналітика часто або взагалі відсутня, або представлена лише у вигляді таблиць без агрегованих показників. Це ускладнює оцінку ефективності та не дозволяє виявити проблемні зони – наприклад, частоту прострочення задач або перевантаження виконавців.

Для подолання вказаних проблем доцільно передбачити:

- централізоване зберігання та управління завданнями;
- автоматичне журналювання змін;
- балансування навантаження між учасниками;
- візуалізацію статусів, дедлайнів, прогресу виконання;
- аналітичні панелі з ключовими метриками (KPI);
- оперативне редагування проєкту без втрати цілісності.

Такі підходи дозволяють підвищити керованість, скоротити час на пошук і перевірку інформації, уникнути дублювань і втрат, а також підтримувати стабільний темп реалізації проєкту.

1.3 Аналіз аналогів програмного забезпечення

У сфері управління проєктами існує велика кількість програмних рішень, які пропонують різний рівень функціональності, інтеграції та зручності для користувача. Аналіз популярних платформ дозволяє виявити загальні тенденції розвитку цієї галузі, а також недоліки, які залишаються невирішеними в наявних системах.

Одним з найпотужніших інструментів є Jira — система управління проектами, орієнтована насамперед на IT-команди (див. рис. 1.1). Вона підтримує роботу за методологією Scrum і Kanban, дозволяє формувати спринти, вести backlog, контролювати навантаження на виконавців і автоматично генерувати аналітичні звіти (зокрема burndown chart, velocity). Jira має глибоку інтеграцію з інструментами DevOps, CI/CD, контролю версій тощо. До недоліків платформи відносяться складність у конфігурації, громіздкий інтерфейс і потреба у навчанні для нових користувачів.

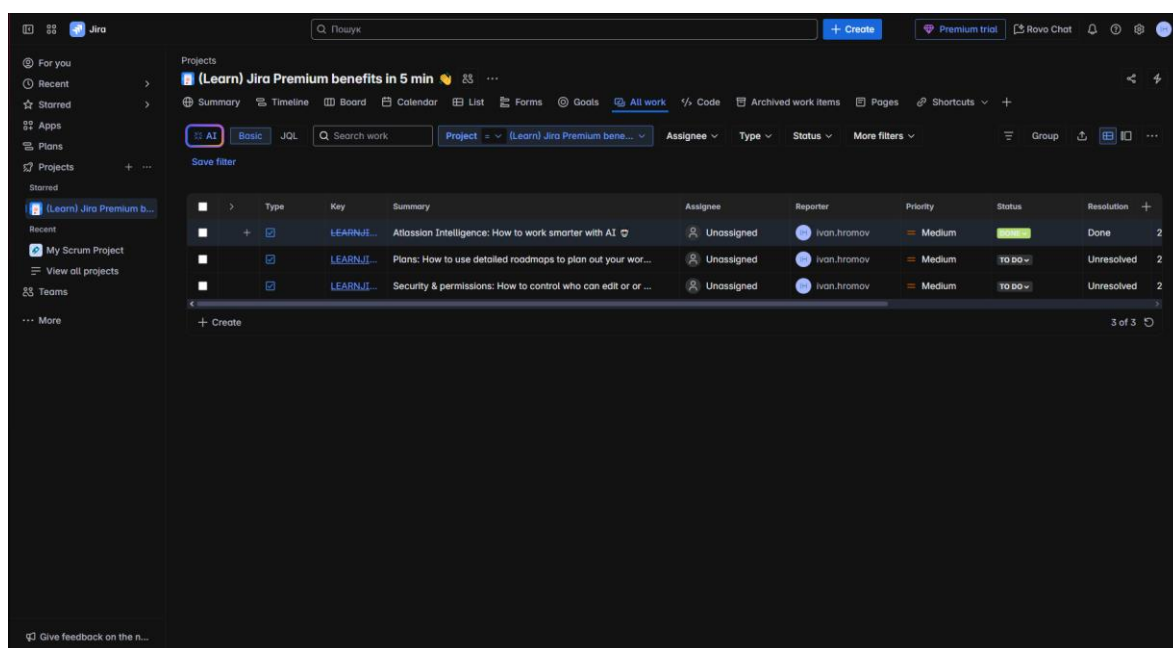


Рисунок 1.1 – Інтерфейс списку завдань Jira
(виконано самостійно)

Інша широко відома платформа – Asana, що орієнтована на широке коло користувачів: від невеликих стартапів до великих команд. Система підтримує різні формати представлення задач – у вигляді списку, Kanban-дошки або календаря. Кожне завдання можна детально описати, додати виконавця, дедлайн, підзадачі, коментарі та чеклісти. У базовій версії доступна аналітика за кількістю завершених задач і активністю учасників, але розширені функції звітності потребують платної підписки (див. рис. 1.2).

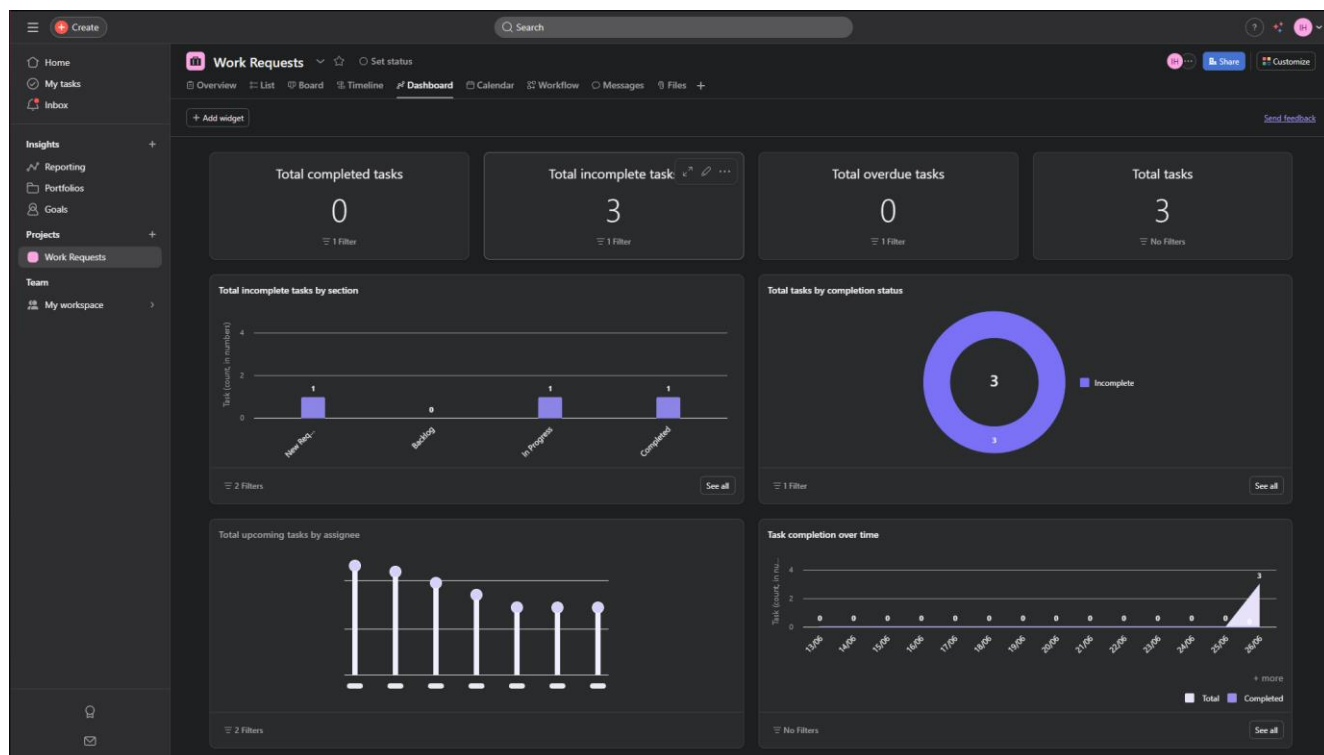


Рисунок 1.2 – Інтерфейс аналітики в Asana
(виконано самостійно)

Ще один популярний інструмент – Monday.com, який має яскравий інтерфейс із численними шаблонами для організації задач, ресурсів і термінів. Платформа дозволяє створювати візуальні Roadmap'и (див. рис. 1.3), використовувати таблиці, календарі та діаграми Ганта, а також підключати автоматизації для щоденних процесів. Monday.com підтримує аналітичні панелі, фільтрацію, групування даних і деталізоване керування доступом. Недоліком є перевантаженість функціоналу і часткова складність у навігації.

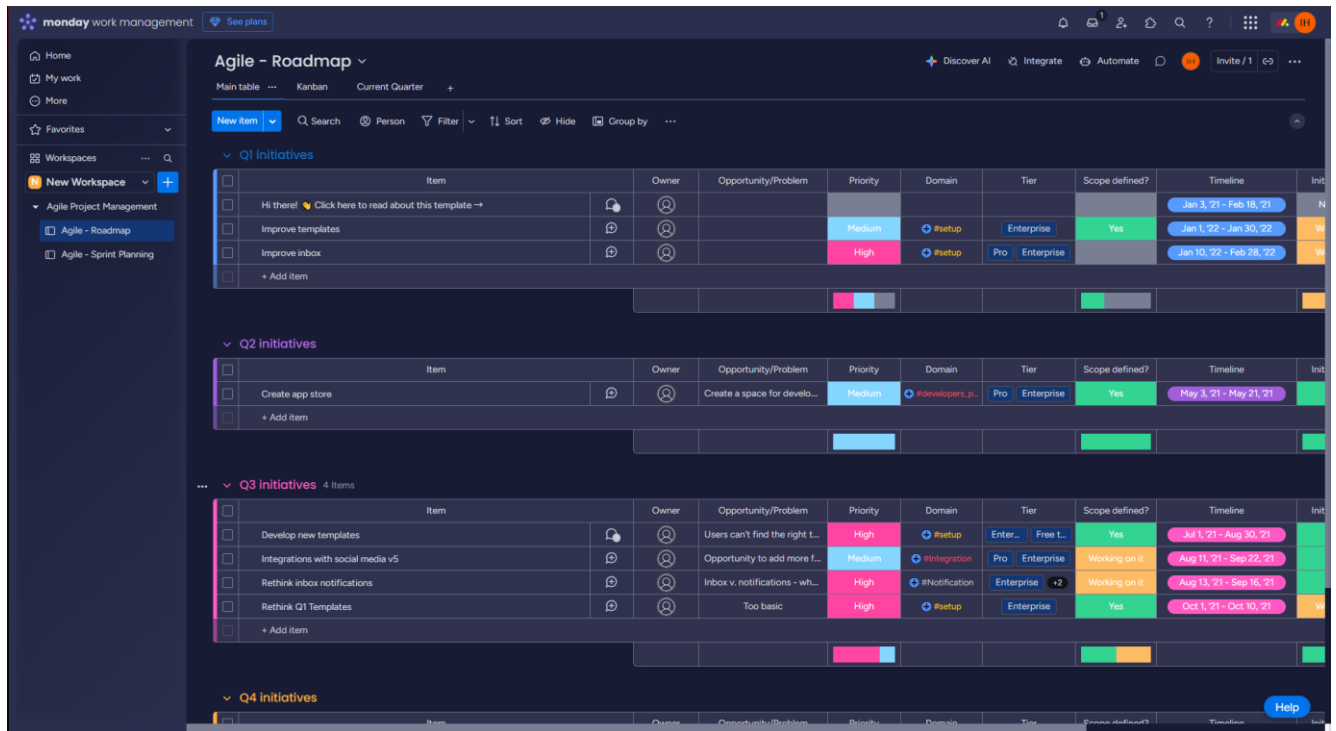


Рисунок 1.3 – Інтерфейс Roadmap у Monday.com
(виконано самостійно)

Аналіз наведених систем показує, що кожна з них має свою цільову аудиторію й специфічний набір функцій. Деякі сервіси орієнтовані на простоту та швидкий старт, інші – на гнучку конфігурацію, глибоку аналітику або масштабованість для великих команд. Разом із тим, потреба в доступному, інтегрованому й водночас інформативному інструменті управління проєктами залишається актуальною.

2 ПОСТАНОВКА ЗАДАЧІ

Метою даного проєкту є розробка клієнт-серверного вебзастосунок для управління командною проєктною діяльністю з підтримкою створення задач, контролю їх виконання, призначення відповідальних осіб і відображення аналітики щодо продуктивності користувачів. Застосунок має забезпечити інтуїтивну взаємодію користувача з системою, централізоване зберігання даних, гнучке управління завданнями та базову візуалізацію результатів проєктної діяльності.

Кінцевий продукт має включати дві ключові частини – клієнтську та серверну. Клієнтська частина відповідає за інтерфейс користувача, відображення структури даних і взаємодію з бек-ендом через REST API. Серверна частина реалізує логіку обробки запитів, авторизацію, управління базою даних і формування відповідей для клієнта. Уся інформація про користувачів, проєкти, задачі та аналітичні дані зберігається в хмарній NoSQL-базі Firebase Firestore.

Застосунок має працювати у багатокористувацькому режимі, підтримуючи рольову модель доступу, з можливістю створення декількох проєктів, в межах яких призначаються задачі певним користувачам. Очікується реалізація основних CRUD-операцій над проєктами та задачами, можливість зміни статусу задач, фіксація часу створення та оновлення, а також формування узагальненої статистики щодо активності команди.

Основними функціональними задачами системи є:

- реєстрація, вхід і вихід користувача із захищеним зберіганням облікових даних;
- створення та редагування проєктів;
- додавання учасників до проєкту з можливістю обмеження прав;
- створення задач з указанням назви, опису, відповідального, статусу й терміну виконання;
- оновлення статусу задач («Очікує», «У процесі», «Завершено»);
- перегляд задач з можливістю фільтрації за статусом або виконавцем;

- автоматичне збирання статистики за кількістю виконаних задач, поточним навантаженням і активністю;
- відображення аналітики в узагальненому вигляді (дашборд).

Клієнтська частина має бути реалізована на мові JavaScript із використанням бібліотеки React. Для стилізації інтерфейсу використовується CSS. Серверна частина створюється на мові Java з використанням Spring Boot. Для взаємодії з клієнтом передбачено використання REST API. Уся інформація зберігається у Firebase Firestore, що також використовується для автентифікації користувачів.

Додатковими вимогами до системи є:

- надійність і коректність обміну даними між клієнтом і сервером;
- зрозумілий та лаконічний інтерфейс;
- базова безпека при обробці персональних даних;
- масштабованість архітектури.

Основна задача – розробити гнучкий, функціональний та розширюваний вебзастосунок, що відповідає сучасним вимогам до організації командної взаємодії та контролю проєктної діяльності.

профілем. Менеджер, окрім функцій користувача, має розширені можливості щодо адміністрування: створення проєктів і завдань, редагування та видалення завдань, управління учасниками команди, перегляд усіх проєктів та аналітики.

Для деталізації внутрішньої логіки клієнтської частини було розроблено діаграму компонентів, яка ілюструє структуру основних елементів інтерфейсу (див. рис. 3.2). Кожен логічний блок представлений окремим компонентом: панель навігації (Sidebar), верхня панель (Topbar), сторінки проєктів (ProjectsPage), перегляд задач (TasksView), блок аналітики (AnalyticsCard), компоненти сповіщень (Toast), та ін. Особливу увагу приділено структурі взаємодії з REST API, що реалізується через спеціалізовані сервіси запитів.

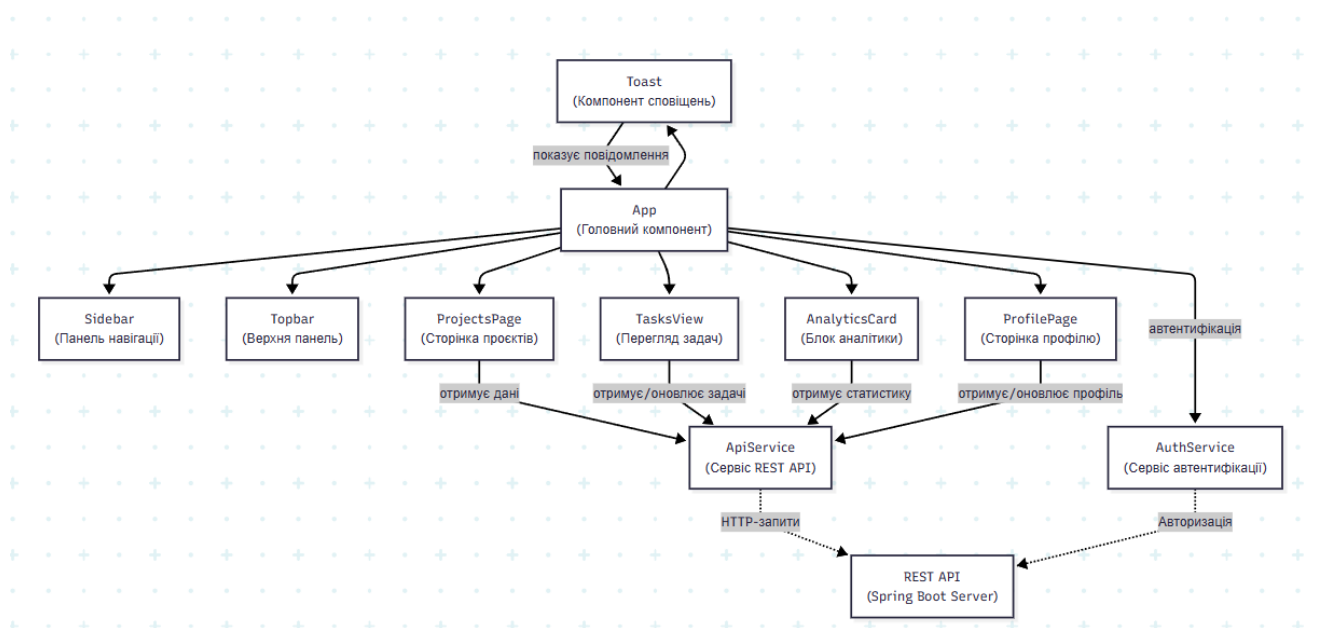


Рисунок 3.2 – Діаграма компонентів клієнтської частини застосунку
(виконано самостійно)

Серверна частина також проєктувалася з урахуванням чіткої структурованості, що відображено у діаграмі компонентів backend-архітектури. Вона включає контролери для обробки запитів (TaskController, ProjectController), сервіси (TaskService, AnalyticsService), репозиторії для доступу до Firebase, а також

модулі автентифікації та логування. Компоненти взаємодіють за шаблоном REST через Spring Boot (див. рис. 3.3).

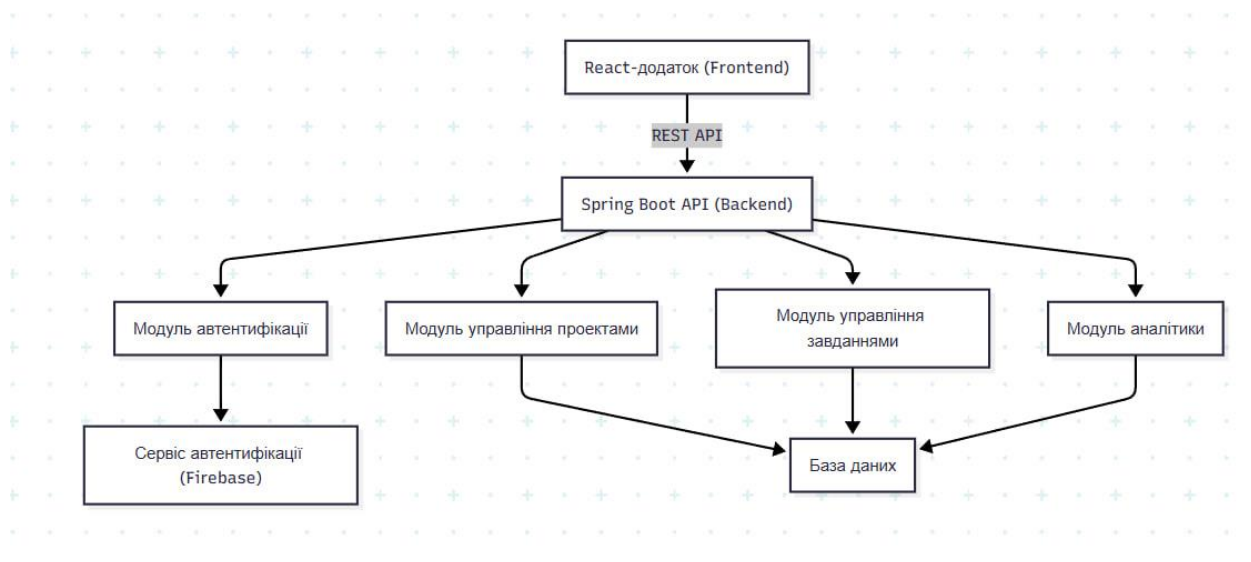


Рисунок 3.3 – Діаграма компонентів серверної частини застосунку

(виконано самостійно)

У рамках проєктування було також створено діаграму класів (див. рис. 3.4), яка формалізує основні сутності, що використовуються в системі. Зокрема:

- клас User містить ідентифікатор, email, ім'я та аватар користувача;
- клас Project описує проєкт: його ідентифікатор, назву, опис, дату створення, список учасників і логів;
- клас Task містить дані про завдання: статус, дедлайн, опис, список коментарів і тегів;
- Member відображає зв'язок між користувачем і проєктом, а саме роль у межах команди;
- Comment дає змогу вести обговорення задач, а також зберігає автора й дату створення;
- Log фіксує історію змін у проєкті — дії над завданнями, зміни статусів, додавання учасників.

Зв'язки між сутностями реалізують типові залежності: Task належить Project, має Comment; Member пов'язує User і Project; Log зберігається в контексті Project. Це дозволяє реалізувати складну бізнес-логіку в рамках структурованого зберігання.

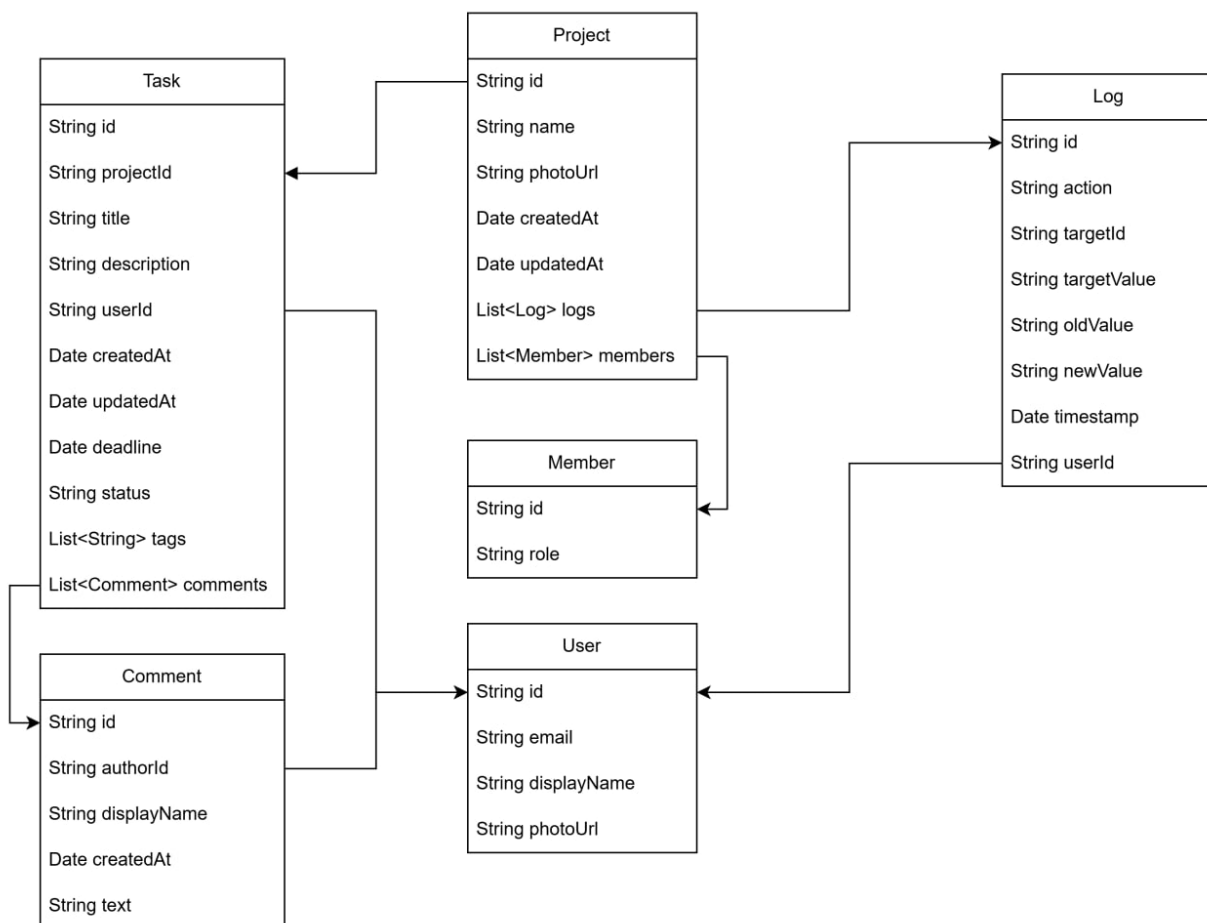


Рисунок 3.4 – Діаграма класів
(виконано самостійно)

Для демонстрації динаміки обміну даними між фронт-ендом і бек-ендом було змодельовано діаграму послідовності, яка відображає типову взаємодію під час створення завдання. Користувач (менеджер) ініціює дію через інтерфейс, клієнт формує запит до сервера, сервер проводить перевірки й зберігає дані у Firebase Firestore [2], після чого клієнт оновлює інтерфейс з урахуванням змін (див. рис. 3.5).

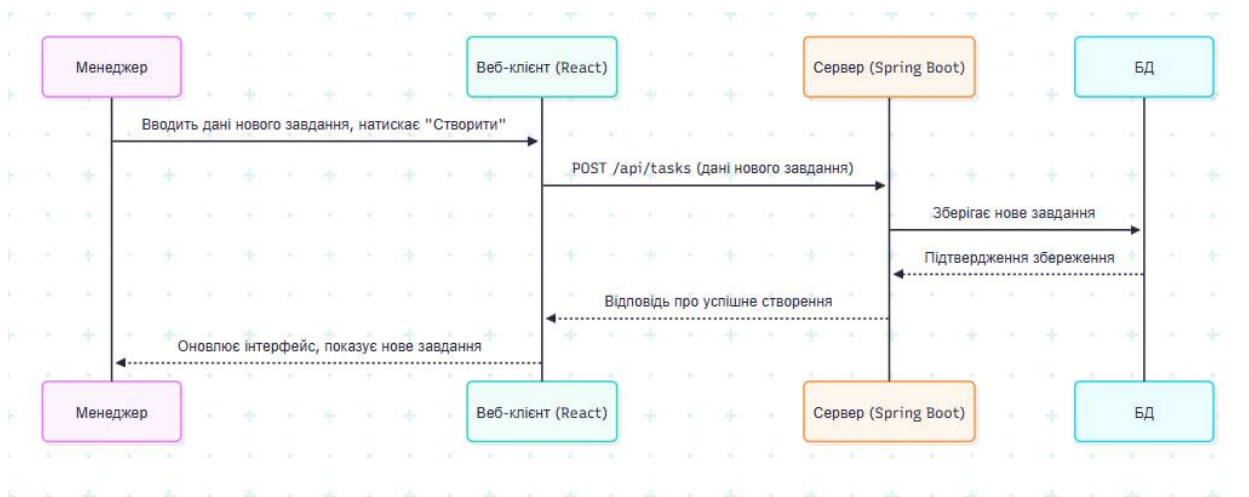


Рисунок 3.5 – Діаграму послідовності
(виконано самостійно)

Діаграми було побудовано на ранньому етапі проєктування, і вони відіграли важливу роль у розробці модульної, масштабованої та легко підтримуваної архітектури. Вони дозволили візуалізувати як логіку користувацької взаємодії, так і розподіл обов'язків між частинами системи.

3.2 Проектування архітектури ПЗ

Проектування архітектури програмного забезпечення CrewNote здійснювалося з урахуванням принципів модульності, масштабованості, зрозумілості та розділення відповідальностей. Було обрано клієнт-серверну архітектуру, що передбачає розподіл системи на фронт-енд-частину, яка відповідає за взаємодію з користувачем, і бек-енд-частину, що обробляє запити, зберігає дані та реалізує бізнес-логіку.

Клієнтська частина логічно поділяється на модулі, що відповідають за рендеринг інтерфейсу, обробку користувацьких дій, маршрутизацію, обмін даними з сервером і управління станом. Така структура забезпечує повторне використання компонентів та полегшує підтримку коду.

Бек-енд-частина реалізована у вигляді багатошарової архітектури. Вона включає:

- рівень контролерів (взаємодія з клієнтом через HTTP-запити);
- рівень сервісів (реалізація бізнес-логіки);
- рівень доступу до даних (взаємодія з базою даних).

Цей підхід дозволяє забезпечити слабке зв'язування між шарами та гнучке масштабування системи в майбутньому.

Для наочного представлення архітектури системи створено UML-діаграму компонентів (див. рис. 3.6), яка ілюструє структуру основних функціональних блоків обох частин системи, їхню взаємодію та точки інтеграції.

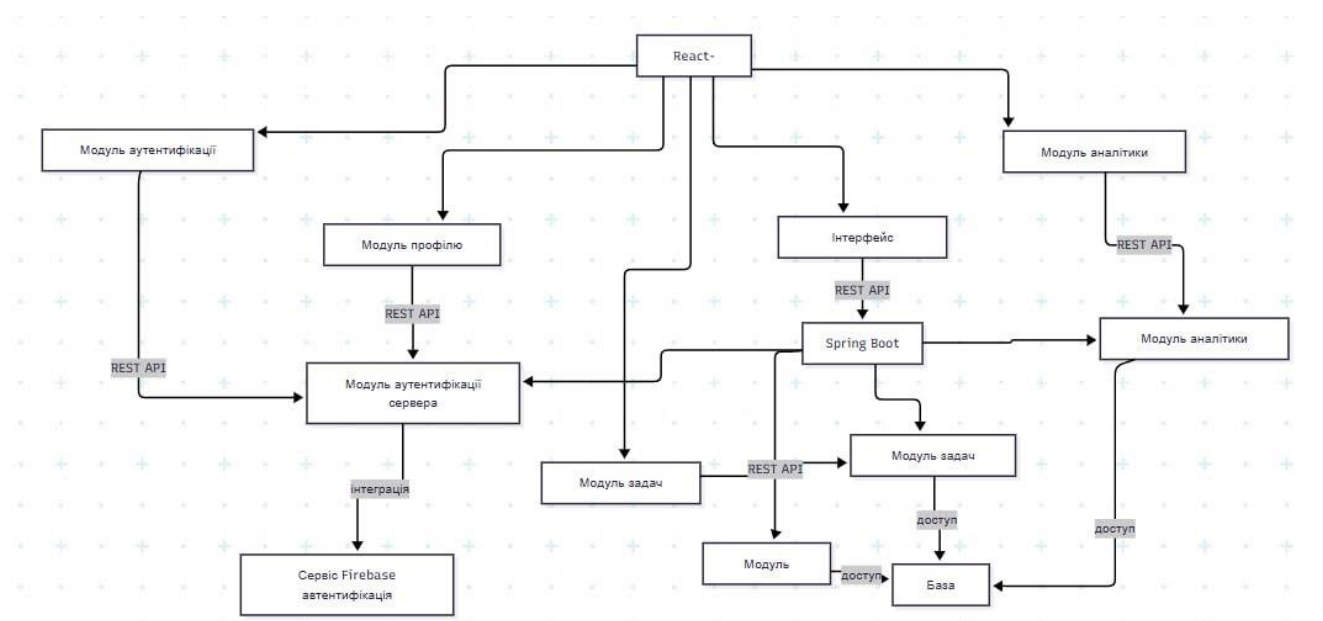


Рисунок 3.6 – Діаграма компонентів клієнтської та серверної частини системи (виконано самостійно)

Окремо було побудовано UML-діаграму розгортання (див. рис. 3.7), що відображає фізичне розміщення компонентів системи та канали комунікації між ними. На діаграмі показано фронт-енд-додаток, серверну частину, хмарне сховище Firestore та канали обміну даними (REST API).

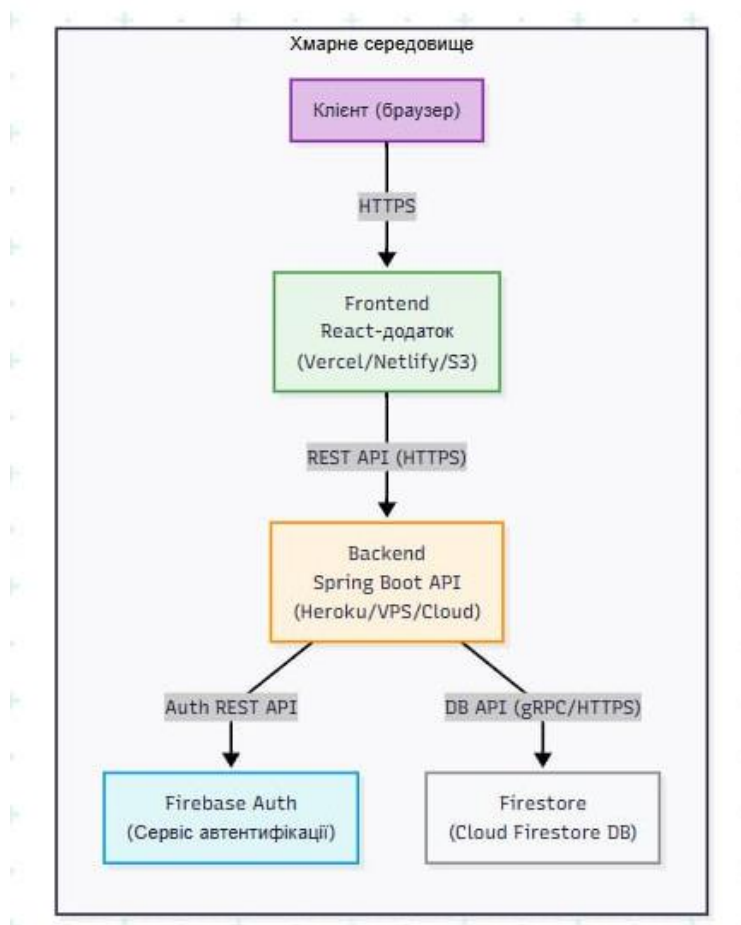


Рисунок 3.7 – Діаграма розгортання системи
(виконано самостійно)

Архітектура також передбачає ізоляцію обробки запитів від доступу до даних, що дозволяє легко замінити або розширити окремі модулі. Це особливо важливо у разі впровадження нових функцій, таких як аналітика, штучний інтелект або інтеграція з іншими системами.

При проєктуванні було дотримано принципів:

- Separation of Concerns: кожен модуль має окрему відповідальність;
- Modularity: компоненти ізольовані й взаємодіють через чітко визначені інтерфейси;
- Scalability: архітектура дозволяє масштабування як у глибину (розширення логіки), так і в ширину (додавання нових модулів).

Архітектурне проєктування дало змогу сформувати узгоджену структуру системи, що задовольняє функціональні та нефункціональні вимоги й забезпечує основу для подальшої розробки, тестування та розгортання програмного забезпечення.

3.3 Проєктування структури зберігання даних

Зберігання даних є критичним елементом функціонування застосунку CrewNote, оскільки саме в базі даних зберігається інформація про користувачів, проєкти, завдання, коментарі, аналітику та історію змін. У межах цього проєкту було обрано Firebase Firestore як основну технологію збереження даних. Це документоорієнтована база, що підтримує масштабування, синхронізацію в реальному часі, гнучку структуру колекцій і документів [2].

Firestore має ієрархічну модель даних, яка базується на колекціях (collections) і документах (documents). Документи зберігаються у вигляді пар "ключ–значення", можуть містити вкладені об'єкти, списки та підколекції (див. рис. 3.8). Це дозволяє гнучко структурувати дані без необхідності створення складних схем, характерних для реляційних баз.

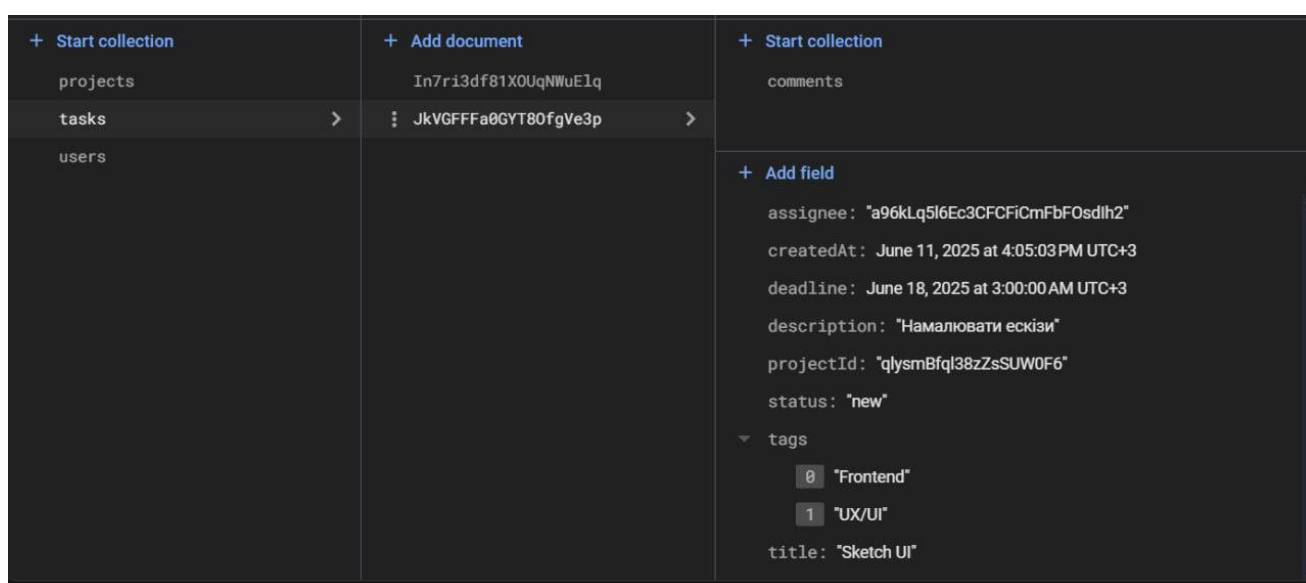


Рисунок 3.8 – Структура колекцій та документів у Firestore
(виконано самостійно)

Основні колекції, які було спроектовано для зберігання в системі:

- users: містить інформацію про кожного користувача системи, зокрема id, email, displayName, avatarUrl, дата створення, а також права доступу.
- projects: описує кожен проєкт: id, name, description, createdAt, updatedAt, createdBy, а також список учасників (members).
- tasks: зберігає завдання, пов'язані з проєктами: title, description, status, dueDate, projectId, assigneeId, createdAt, updatedAt, tags, comments.

У системі передбачено зв'язки між сутностями. Наприклад:

- один користувач може бути учасником кількох проєктів;
- кожен проєкт містить множину завдань;
- кожне завдання може мати коментарі, що належать різним користувачам;
- кожна дія записується у лог, пов'язаний із відповідним проєктом або завданням.

Особливу увагу приділено ідентифікації документів: для кожного запису використовується унікальний id, згенерований Firestore, що дозволяє уникнути колізій. Також використовуються індекси для полегшення фільтрації задач за статусом, дедлайном та виконавцем.

Firestore підтримує автоматичне оновлення в реальному часі, що дає змогу миттєво відображати зміни у фронт-енді без додаткових запитів. Це особливо важливо для мультикористувацьких систем [2].

Проектована структура зберігання:

- відповідає принципам нормалізації;
- дозволяє масштабування без змін схеми;
- підтримує історію змін та відновлення попередніх станів.

Оскільки система не використовує реляційні залежності в класичному сенсі, акцент зроблено на правильному групуванні даних та оптимізації запитів до Firestore через REST API. Це дозволяє гнучко працювати з даними як на клієнтському, так і на серверному рівні.

3.4 Приклади найцікавіших алгоритмів та методів

У процесі розробки клієнт-серверного застосунку для керування проєктами було використано низку алгоритмів та методів, які забезпечують ефективну роботу системи, виконання бізнес-логіки, обробку даних та підвищення зручності для користувачів. Деякі з них реалізовані у вигляді окремих функцій, інші – є частиною загальних процесів і модулів застосунку.

3.4.1 Алгоритми фільтрації та пошуку завдань

Однією з найбільш затребуваних функцій є фільтрація та пошук завдань у межах проєкту. Це дозволяє користувачам швидко знаходити потрібні завдання за певними критеріями: статус, відповідальний, дедлайн, ключові слова тощо.

```
let filtered = tasks.filter(t =>
  (filterStatus === "all" || t.status === filterStatus)
  && ( t.title?.toLowerCase().includes(search.toLowerCase()) ||
    t.description?.toLowerCase().includes(search.toLowerCase()) ||
    (t.tags &&
t.tags.join(", ").toLowerCase().includes(search.toLowerCase()) ) )
);

if (sort === "deadline_asc") filtered.sort((a, b) => (a.deadline ||
"").localeCompare(b.deadline || ""));
if (sort === "deadline_desc") filtered.sort((a, b) => (b.deadline ||
"").localeCompare(a.deadline || ""));
if (sort === "title_asc") filtered.sort((a, b) =>
(a.title || "").localeCompare(b.title || ""));
if (sort === "title_desc") filtered.sort((a, b) =>
(b.title || "").localeCompare(a.title || ""));
```

Після обробки запиту отримані результати повертаються назад до клієнтської частини для подальшого відображення користувачеві в інтерфейсі.

3.4.2 Алгоритм розрахунку статистики виконання завдань

Модуль аналітики системи включає алгоритми агрегації даних, які призначені для аналізу виконання завдань користувачами або цілими командами. Ця функціональність надає менеджерам можливість відстежувати рівень прогресу проєктів та оцінювати загальну ефективність роботи команди.

```
const resTasks = await
  apiFetch(`/api/tasks?projectId=${currentProject.id}`);
let tasksArr = [];
if (resTasks.ok) tasksArr = await resTasks.json();
setTasks(tasksArr);

const percentDone = tasks.length
  ? Math.round((tasks.filter(t => t.status === "done").length /
    tasks.length) * 100)
  : 0;
```

На основі отриманої інформації система підраховує загальну кількість завдань у проєкті, а також визначає, скільки з них вже виконано, скільки перебуває в процесі виконання, і скільки ще планується до реалізації. При необхідності алгоритм може сформулювати більш деталізовану статистику, яка показує індивідуальні показники ефективності для кожного учасника команди окремо.

3.4.3 Метод авторизації через сторонній сервіс (Google Sign-In)

Також система реалізує метод авторизації через сторонній сервіс Google Sign-In, що значно спрощує процес входу користувачів до системи. Цей алгоритм авторизації через Google працює за чітко визначеною схемою взаємодії між клієнтом і сервером [3].

```
async function handleGoogleLogin() {
  setError("");
  try {
    const provider = new GoogleAuthProvider();
    const result = await signInWithPopup(auth, provider);
    const idToken = await result.user.getIdToken();

    setAuthToken(idToken);
  }
}
```

```

const res = await fetch("/api/auth/verify", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ idToken }),
});

if (res.ok) {
  const data = await res.json();
  setUser(data);

  await apiFetch("/api/users", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      id: data.uid, // або data.id
      email: data.email,
      displayName: data.displayName,
      photoUrl: data.photoUrl
    }),
  });

  navigate("/home");
} else {
  setError("Google авторизація не вдалася.");
}
} catch (err) {
  setError("Google помилка: " + err.message);
}
}

```

Використання такого підходу має декілька важливих переваг. По-перше, це підвищує загальний рівень безпеки системи, оскільки автентифікація відбувається через надійні сервіси Google [3]. По-друге, значно знижується кількість помилок при вході, адже користувачам не потрібно запам'ятовувати додаткові паролі. Нарешті, цей метод істотно спрощує життя користувачам, надаючи їм можливість швидко та зручно отримувати доступ до системи.

3.5 Створення UI/UX дизайну системи

У рамках розробки системи CrewNote проектування інтерфейсу проводилось на основі принципів UI/UX-дизайну, орієнтованого на зручність, простоту використання та ефективну організацію робочого простору.

Фронт-енд частина реалізована з використанням бібліотеки React, що дозволило застосувати компонентну модель побудови інтерфейсу. Кожна функціональна частина системи представлена окремим компонентом: панель навігації, список завдань, перегляд статистики, вікно створення або редагування задачі, тощо. Такий підхід забезпечує високу модульність, повторне використання елементів та зручність у підтримці коду. Стилiзація реалізована за допомогою звичайних CSS-файлів, що дозволило точно контролювати зовнішній вигляд кожного компонента [4].

Основним екраном взаємодії є дашборд, на якому представлено поточні проєкти та узагальнені аналітичні показники щодо виконання завдань (див. рис. 3.9). Тут користувач може оцінити динаміку роботи команди, побачити найбільш активних учасників і переглянути розподіл завдань за статусами. Зазначений функціонал реалізований у вигляді окремих візуальних блоків: графіків, таблиць та карток.

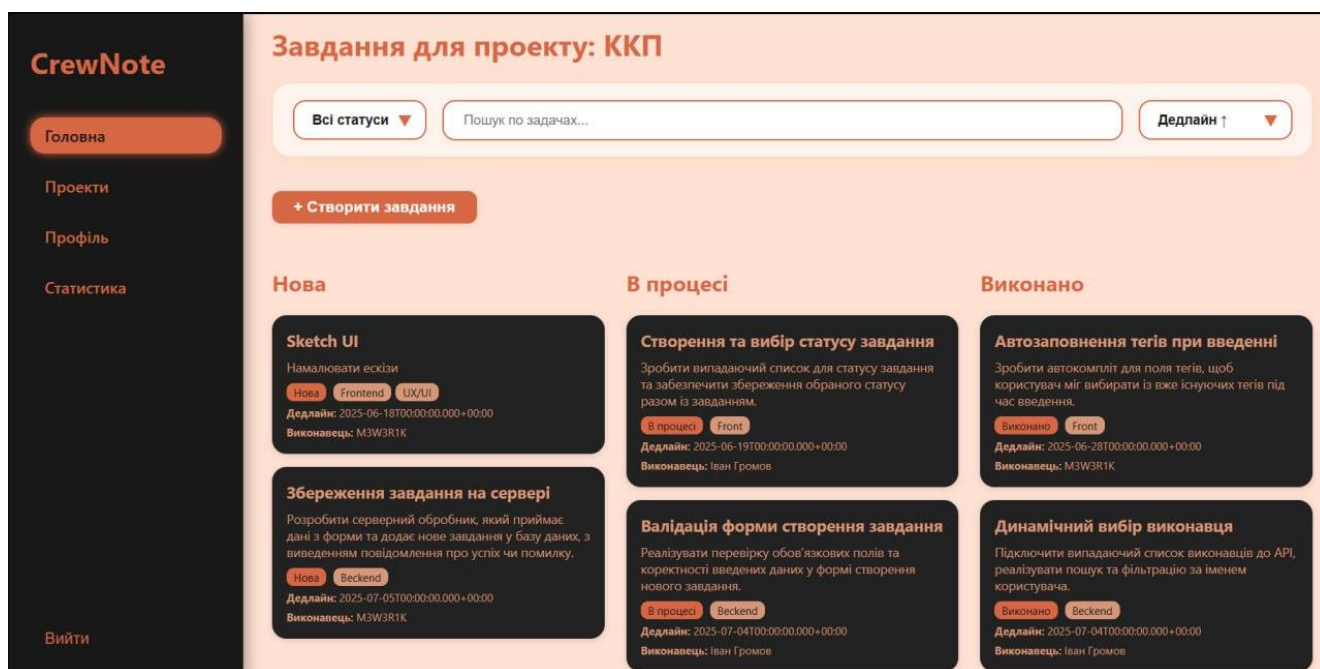
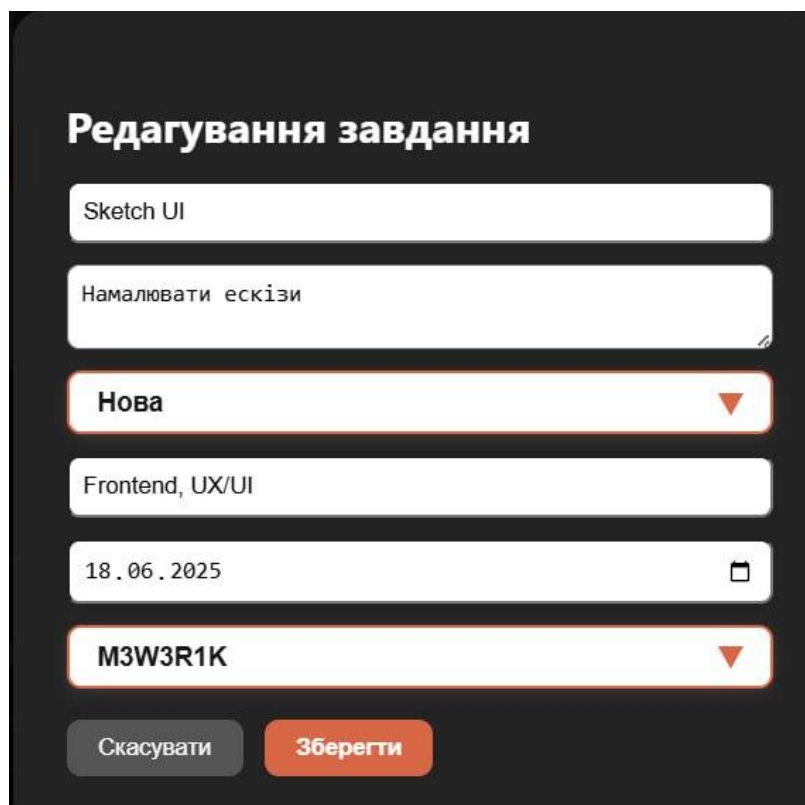


Рисунок 3.9 – Інтерфейс головної сторінки застосунку
(виконано самостійно)

Для роботи із завданнями реалізовано окремий інтерфейс, що включає фільтрацію, сортування, статусну індикацію та кнопки редагування (див. рис. 3.10). Завдання згруповані за проєктами, а інтерфейс дозволяє швидко орієнтуватися в загальному обсязі роботи, визначати пріоритети та перемикатися між видами представлення даних.



Редагування завдання

Sketch UI

Намалювати ескізи

Нова ▼

Frontend, UX/UI

18.06.2025

M3W3R1K ▼

Скасувати Зберегти

Рисунок 3.10 – Інтерфейс редагування завдань у системі
(виконано самостійно)

Навігація в системі реалізована за допомогою бічної панелі (Sidebar), яка надає швидкий доступ до ключових розділів, зокрема до проєктів, завдань, аналітики та профілю користувача (див. рис. 3.11). Всі кнопки та інтерфейсні елементи мають логічне групування та зрозуміле візуальне оформлення, що дозволяє знизити час на навчання і покращити продуктивність роботи.

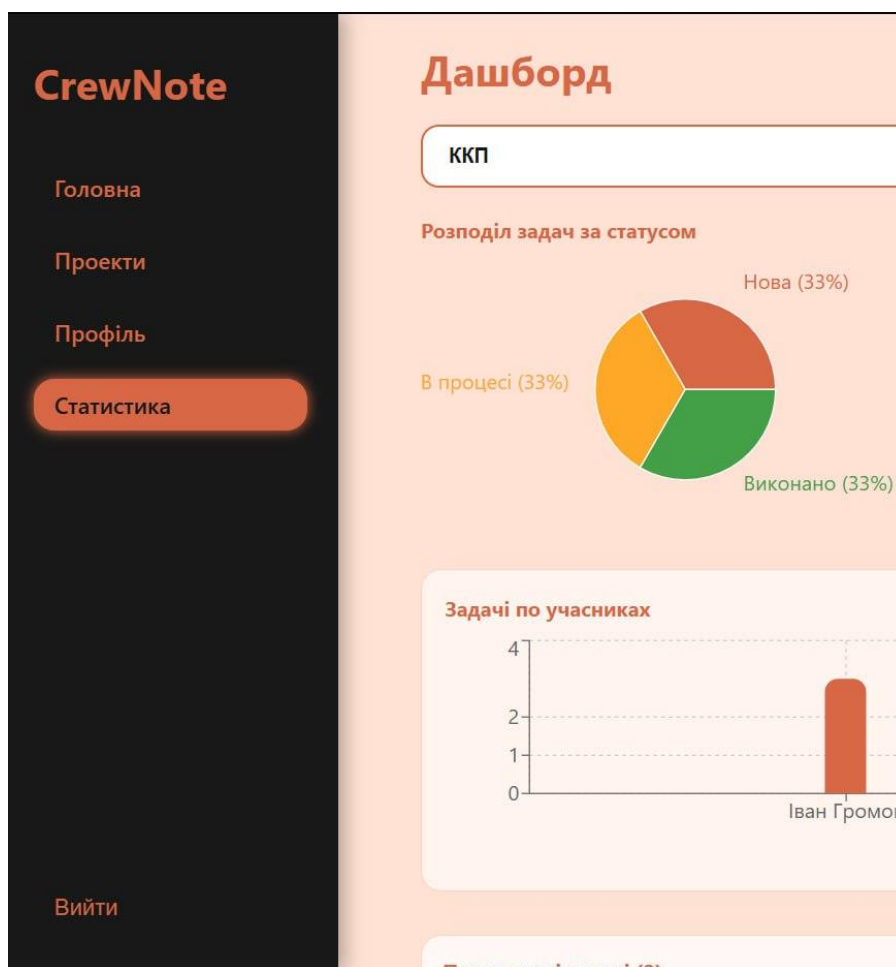


Рисунок 3.11 – Навігаційна структура інтерфейсу
(виконано самостійно)

Таким чином, створення інтерфейсу користувача CrewNote базується на сучасних підходах до UI/UX-дизайну, що забезпечує зручну та ефективну взаємодію користувача з системою. Компонентна архітектура, логічна структура екранів, візуальна ясність та інформативність – усе це дозволяє зробити систему доступною як для звичайних користувачів, так і для менеджерів команд.

3.6 Опис підготовки даних

На етапі проєктування та розробки клієнт-серверного застосунку CrewNote особливу увагу було приділено підготовці вхідних і структурованих даних, які використовуються в межах основної функціональності системи. Підготовка даних передбачає визначення структури зберігання, форматів обміну, типів даних для

окремих сутностей та наповнення бази актуальними тестовими записами з метою перевірки роботи застосунку.

Основним сховищем інформації в системі є база даних Firebase Firestore, яка працює як NoSQL-рішення й організовує дані у вигляді колекцій і документів. Такий підхід дозволяє гнучко моделювати зв'язки між сутностями та масштабувати структуру даних без жорсткої прив'язки до схем. Для збереження та подальшої обробки були виділені наступні основні колекції: users, projects, tasks [2].

Проекти (projects) містять назву, опис, ідентифікатори учасників, дату створення та активність. Для кожного проекту були створені завдання (tasks) з урахуванням різних статусів (нове, у процесі, завершене), дедлайнів, авторів і тегів, що дозволило протестувати роботу аналітики.

Також був підготовлений набір коментарів (comments) для імітації колективного обговорення завдань. Це дозволило перевірити коректність відображення вкладених структур і механізмів оновлення вмісту.

Важливим етапом стала також підготовка логів змін (logs), які фіксують дії користувачів у проекті: створення, редагування або видалення завдань, додавання учасників, зміну статусів тощо. Вони необхідні для реалізації історії змін і подальшого аналізу активності.

Під час тестування REST API всі запити (GET, POST, PUT, DELETE) перевірялись на реальних прикладах даних через інструменти Postman і фронт-енд-застосунок [5]. Дані вводились вручну через інтерфейс або напряму в Firestore з дотриманням форматів, визначених у DTO-моделях.

Підготовка даних у системі включала створення структури колекцій, ініціалізацію ключових записів, перевірку форматів обміну між клієнтом і сервером та тестування логіки на реалістичних прикладах. Це дозволило забезпечити стабільність роботи застосунку та валідність обробки користувацьких даних у різних сценаріях.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір стеку технологій

У проєкті використано сучасний клієнт-серверний стек: Java [6] + Spring Boot для серверної частини, Firebase як хмарний сервіс, React для клієнта, CSS для стилізації та Vite як інструмент збірки.

Серверна частина реалізована на Java з використанням Spring Boot через його чітку підтримку REST API, просту конфігурацію та модульність [7]. Цей фреймворк дозволяє швидко створювати структуровані сервіси, що масштабуються. Для обробки запитів використовуються контролери (наприклад, ProjectController, TaskController), а бізнес-логіка винесена в окремі сервіси (ProjectService, TaskService тощо).

Клієнтська частина створена на React, що забезпечує компонентну архітектуру та зручну побудову інтерфейсу. Стан користувача обробляється через AuthContext, запити до API виконуються через fetch. React обрано через його гнучкість, популярність і просту інтеграцію з бек-ендом [4].

Для стилізації застосовано чистий CSS – рішення, що дозволяє повністю контролювати вигляд без залежності від сторонніх фреймворків. Це забезпечує кращу адаптацію дизайну під специфіку інтерфейсу.

Збірка клієнта відбувається через Vite, що забезпечує швидкий запуск, гаряче оновлення модулів і високу продуктивність у продакшн-збірці [8].

Обраний стек забезпечує швидкість розробки, безпечну автентифікацію, просту інтеграцію клієнта з сервером і гнучкість у подальшому розширенні системи.

4.2 Організація серверної логіки

Серверна частина програмного забезпечення реалізована на основі фреймворку Spring Boot, що дозволило побудувати масштабовану клієнт-серверну архітектуру з чітким розділенням відповідальностей. У структурі проєкту

визначено основні логічні блоки, які відповідають за обробку запитів, реалізацію бізнес-логіки, передачу даних і взаємодію з базою даних.

Усі дані, які передаються між клієнтською частиною та сервером, обробляються за допомогою спеціальних DTO-класів. Це дає змогу уникнути передачі зайвої інформації та забезпечити чіткий контракт між компонентами системи. DTO включають як скорочені версії об'єктів для списків (TaskShortDto), так і агреговані структури для аналітики (DashboardStatsDto, StatusCount, MemberStats).

Зберігання даних здійснюється у хмарній базі даних Firebase Firestore. Робота з базою реалізована через Firebase Admin SDK, який дозволяє виконувати основні CRUD-операції з документами у відповідних колекціях. Наприклад, створення завдання передбачає генерацію документа у колекції tasks, прив'язку його до відповідного проєкту, додавання міток часу та пов'язаних ідентифікаторів. Оновлення задачі або проєкту передбачає часткову зміну полів документа, а видалення – видалення з основної колекції разом із відповідними підколекціями (наприклад, коментарями).

Використання Firestore також дає змогу будувати вкладені структури, що відображають зв'язки між об'єктами. Наприклад, всередині документа проєкту зберігається список учасників, а всередині завдання – колекція коментарів. Такий підхід дозволяє зменшити обсяг запитів і покращити читаність даних [2].

Загалом, серверна логіка організована відповідно до принципів чистої архітектури: контролери не містять логіки обробки даних, усі перетворення виконуються в сервісах, а моделі й DTO чітко розмежовані. Це забезпечує підтримуваність, можливість розширення функціоналу та інтеграції нових сервісів у майбутньому.

4.3 Реалізація клієнтської частини

Клієнтська частина застосунку CrewNote реалізована з використанням бібліотеки React, яка забезпечує компонентний підхід до побудови інтерфейсу. Архітектура проєкту передбачає чіткий поділ логіки на візуальні компоненти,

контексти стану, модулі обробки запитів та сторінки з відповідною маршрутизацією [4].

Основу інтерфейсу складають багаторазові компоненти, що відповідають за відображення навігації, інформаційних блоків та контенту. Наприклад, Sidebar відповідає за перемикання між сторінками, Dashboard – за загальний огляд проєктів та завдань, а TaskView – за детальну взаємодію з окремими задачами. Компоненти згруповано за функціональністю у відповідні директорії, що полегшує підтримку та розширення системи.

Для управління глобальним станом автентифікації використовується контекст React (AuthContext). Контекст дозволяє зберігати інформацію про поточного користувача, стан авторизації, а також реалізує функції входу та виходу з облікового запису. Це забезпечує єдиний підхід до керування доступом у межах усіх сторінок застосунку.

Маршрутизація в реалізації здійснюється за допомогою react-router-dom. Визначено окремі маршрути для авторизації, домашньої сторінки, дашборду, списку завдань та інших сторінок. Для обмеження доступу до внутрішніх розділів реалізовано компонент PrivateRoute, який перевіряє, чи користувач автентифікований, і перенаправляє його у разі відсутності дозволу [4].

Інтерактивна частина забезпечується за рахунок асинхронних запитів до REST API, що реалізовані через модуль apiFetch.js. Цей модуль містить функції для отримання проєктів, завдань, створення та оновлення даних на сервері.

```
export async function apiFetch(url, options = {}) {
  const user = auth.currentUser;
  if (!user) throw new Error("Не авторизовано");
  const token = await user.getIdToken(true); // оновлює токен!
  const headers = {
    ...options.headers,
    Authorization: `Bearer ${token}`,
    "Content-Type": "application/json"
  };
  return fetch(url, { ...options, headers });
}
```

Уся клієнтська логіка структурується відповідно до принципів повторного використання компонентів, інкапсуляції та чистоти коду, що сприяє підтримуваності застосунку та подальшому масштабуванню.

4.4 Реалізація взаємодії з сервером

У застосунку CrewNote взаємодія між клієнтською та серверною частиною реалізована за допомогою REST API. Кожна функціональна операція – отримання списку завдань, створення нового проєкту, оновлення профілю користувача — реалізована через окремий HTTP-запит до відповідного API-ендпоінту на сервері.

У клієнтській частині для виконання таких запитів використовується модуль `apiFetch.js`, який інкапсулює логіку виконання HTTP-запитів методом `fetch`. Це дозволяє централізовано додавати заголовки, обробляти помилки, виконувати перевірку автентифікації та обробку відповідей сервера.

Запити формуються у форматі JSON. У більшості випадків використовується метод GET для отримання даних, POST для створення нових об'єктів (наприклад, задач або проєктів), PUT або PATCH для оновлення існуючих записів і DELETE для їх видалення. Кожен запит містить токен Firebase у заголовку `Authorization`, що дозволяє серверу ідентифікувати користувача та перевірити його права доступу [5].

Наприклад, виконується GET-запит до ендпоінта `/api/projects/{id}`, де `{id}` – унікальний ідентифікатор проєкту. Цей запит ініціюється після натискання на картку проєкту або при відкритті сторінки перегляду. Після успішного виконання запиту сервер повертає у форматі JSON інформацію про відповідний проєкт, включаючи його назву, опис, список учасників, статус та дати створення і оновлення (див. рис. 4.1).

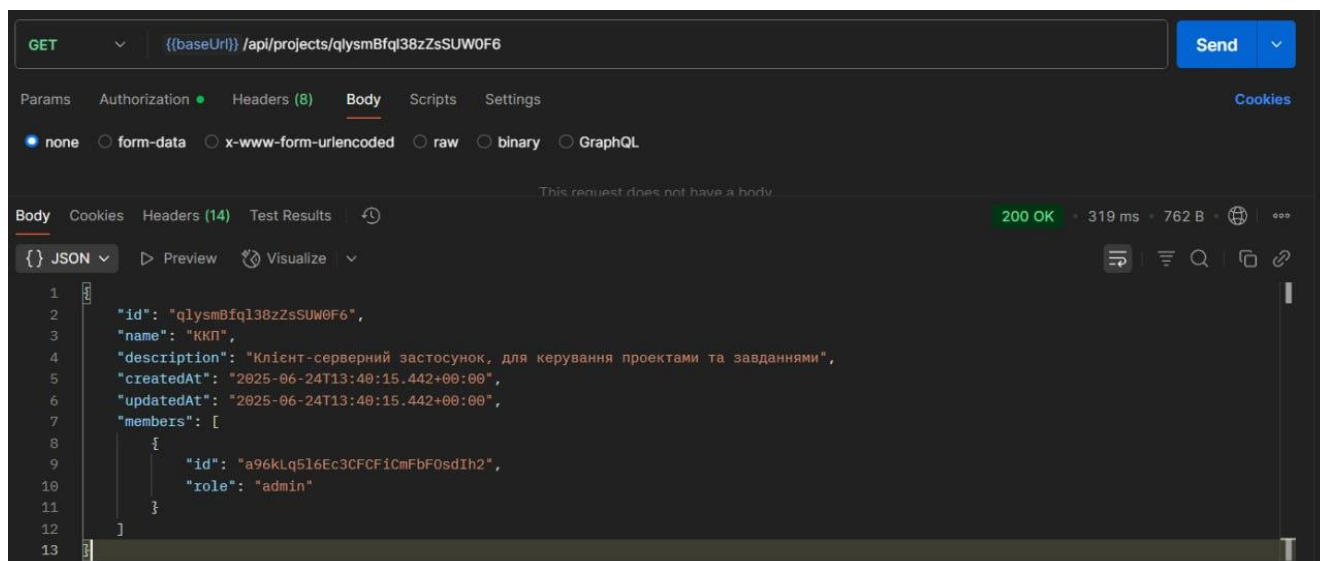


Рисунок 4.1 – запиту до API для отримання інформації про проєкт за ідентифікатором (виконано самостійно)

Отримані дані обробляються клієнтською частиною й оновлюють локальний стан інтерфейсу. В окремих випадках, якщо запит не проходить через автентифікацію, користувач перенаправляється на сторінку входу або отримує повідомлення про помилку доступу.

```

@GetMapping("/{id}")
public Mono<ProjectWithMembers> getProject(@PathVariable String id) {
    return projectService.getById(id);
}

```

Клієнт-серверна взаємодія в CrewNote реалізована за принципами REST, із чітко визначеними ендпоінтами та структурованими запитамі, що забезпечує гнучкість, розширюваність та зрозумілість при підтримці та розвитку системи.

4.5 Забезпечення безпеки та контролю доступу

Для автентифікації користувачів у системі CrewNote використовується сервіс Firebase Authentication, який забезпечує надійне управління доступом на основі токенів. Після успішного входу користувача на клієнтській частині, Firebase формує ID-токен, який додається до кожного запиту у заголовку Authorization.

Серверна частина, реалізована на основі Spring Boot, перевіряє валідність цього токена перед обробкою запиту.

```
@PostMapping("/verify")
public AuthResponse verify(@RequestBody TokenRequest req) throws
Exception {
    FirebaseToken decoded =
FirebaseAuth.getInstance().verifyIdToken(req.getIdToken());
    AuthResponse resp = new AuthResponse();
    resp.setUid(decoded.getUid());
    resp.setEmail(decoded.getEmail());
    resp.setDisplayName(decoded.getName());
    resp.setPhotoUrl(decoded.getPicture());
    log.info("Login success for {}", resp.getEmail());
    return resp;
}
```

У структурі Spring Security інтеграція Firebase реалізована через спеціально створений фільтр безпеки – `FirebaseAuthenticationFilter` [2]. Цей фільтр перехоплює кожен вхідний HTTP-запит, перевіряє наявність токена, здійснює його валідацію через Firebase SDK та, у випадку успішної перевірки, додає відповідного користувача до контексту безпеки Spring (`SecurityContextHolder`) [7]. Такий підхід дозволяє забезпечити прозору інтеграцію зовнішньої системи автентифікації в рамках Java-базованого серверного застосунку.

```
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain chain)
    throws IOException, ServletException {
    String header = request.getHeader("Authorization");
    System.out.println("Authorization header: " + header);
    if (header == null || !header.startsWith("Bearer ")) {
        System.out.println("No Bearer header, skip");
        chain.doFilter(request, response);
        return;
    }
    String idToken = header.substring(7);
    try {
        FirebaseToken decoded =
FirebaseAuth.getInstance().verifyIdToken(idToken);
        System.out.println("Firebase token decoded UID: " +
decoded.getUid());
        var auth = new UsernamePasswordAuthenticationToken(
            decoded.getUid(),
            null,
```

```

        Collections.emptyList()
    );
    SecurityContextHolder.getContext().setAuthentication(auth);
} catch (Exception e) {
    System.out.println("Firebase token error: " + e.getMessage());
    throw new AuthenticationCredentialsNotFoundException("Invalid
Firebase ID token", e);
}
chain.doFilter(request, response);
}

```

Окрім перевірки токенів, система підтримує механізм авторизації залежно від ролі користувача. Ролі поділяються на звичайного користувача (user) та менеджера проєкту (manager). Перевірка ролей відбувається як на рівні контролерів (через умовну логіку в методах), так і на рівні доступу до окремих ресурсів. Наприклад, лише менеджери мають змогу створювати нові проєкти або призначати учасників, тоді як звичайні користувачі можуть переглядати та оновлювати лише власні завдання.

Розмежування прав базується на перевірці вмісту токена (ідентифікатора користувача) та співставленні його з інформацією в базі даних Firestore щодо конкретного проєкту або завдання. Також враховується інформація про зв'язки типу «учасник проєкту», яка зберігається у вкладених структурах даних.

Реалізований механізм безпеки гарантує як надійність автентифікації, так і гнучкість у реалізації авторизаційних сценаріїв. Це дозволяє ефективно захищати дані користувачів і забезпечувати коректне функціонування системи відповідно до ролей і прав доступу.

4.6 Реалізація аналітики

У системі CrewNote реалізовано базову аналітику для оцінювання прогресу виконання задач у рамках проєктів. Основною метою аналітичного модуля є надання користувачеві (зокрема менеджеру) узагальненої інформації про статуси завдань, активність учасників і загальну динаміку проєкту.

Розрахунок статистики відбувається на серверній стороні за допомогою окремого сервісу DashboardService, який обробляє запити до відповідного REST-

ендпоінту (наприклад, GET /api/dashboard/{projectId}) і формує відповідь у вигляді узагальнених показників [5].

Для передачі аналітичних даних між бек-ендом і фронт-ендом використовуються спеціалізовані DTO-моделі:

- DashboardStatsDto: головний об'єкт відповіді, що містить усю узагальнену статистику для конкретного проєкту;
- StatusCount: підмодель, що відображає кількість задач у кожному зі статусів (наприклад, "to-do", "in progress", "done");
- MemberStats: статистика за учасниками проєкту: кількість задач, що призначені на кожного користувача, кількість завершених завдань, рівень активності.

```
public class DashboardStatsDto {
    private List<StatusCount> statusCounts;
    private int percentDone;
    private List<MemberStats> memberStats;
    private List<TaskShortDto> overdueTasks;
    private List<TaskShortDto> upcomingTasks;
    private List<MemberShortDto> members;
}
```

На клієнтській частині ці дані відображаються у вигляді окремих компонентів аналітики – карток (<AnalyticsCard />), діаграм або таблиць. Після отримання відповіді з сервера інтерфейс автоматично оновлюється, відображаючи актуальні метрики проєкту (див. рис. 4.2).

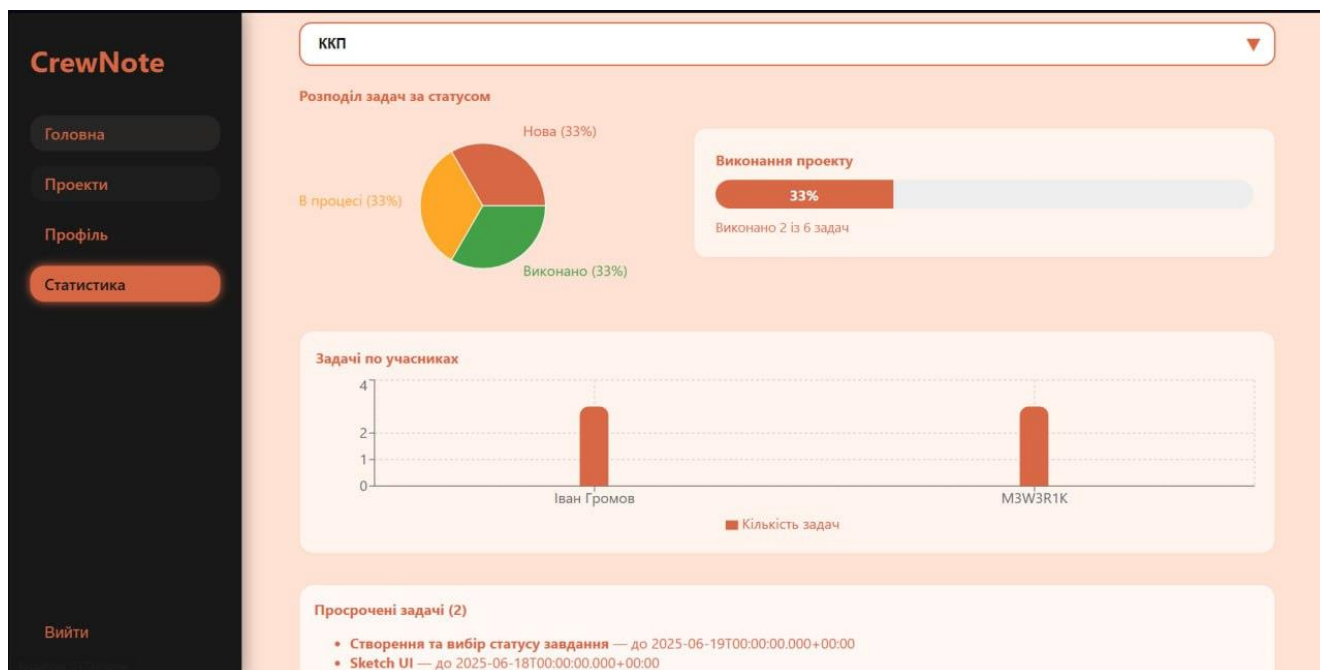


Рисунок 4.2 – Візуалізація аналітики у клієнтському інтерфейсі
(виконано самостійно)

Завдяки модульному підходу реалізації та чіткому розділенню відповідальностей між фронт-ендом і бек-ендом, аналітичний функціонал може бути легко розширений – наприклад, шляхом додавання нових метрик або візуалізацій.

4.7 Логування подій

У системі реалізовано механізм логування дій користувачів для забезпечення прозорості процесів, відстеження змін і подальшого аналізу активності в рамках проектів. Такий підхід дозволяє фіксувати всі ключові події, пов’язані з взаємодією користувача із задачами, проектами та загальною системою.

На стороні серверу функціонал логування реалізовано за допомогою спеціалізованого сервісу LogService, що оперує об’єктами класу LogEntry. Кожен запис журналу містить наступну інформацію:

- тип події (створення, редагування, видалення, зміна статусу тощо);
- об’єкт, до якого застосовано дію (проект, завдання, користувач);

- автор дії (ідентифікатор і e-mail користувача);
- дата та час виконання дії (timestamp);
- додаткові дані (наприклад, старе і нове значення).

Записи журналу зберігаються в колекції Firestore, структуровано за проектами (див. рис. 4.3). Це дозволяє швидко отримати лог усіх змін, що відбувались у рамках одного проекту.

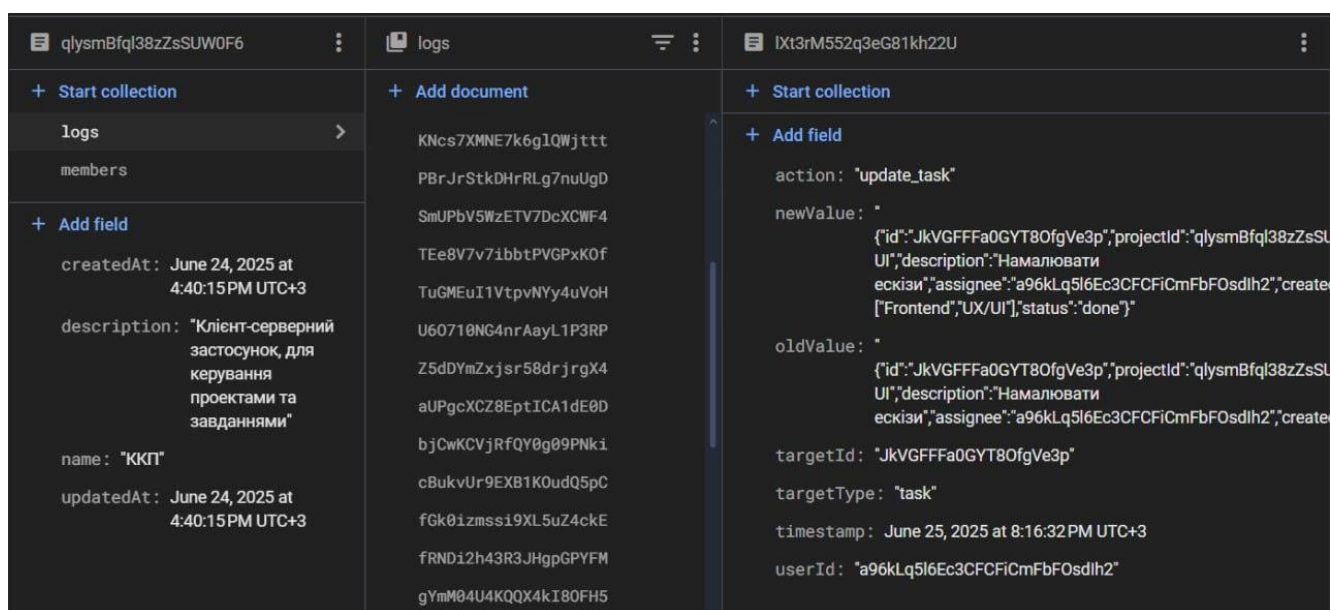


Рисунок 4.3 – Приклад збереженого об'єкта LogEntry у Firestore
(виконано самостійно)

Завдяки логуванню забезпечується додаткова безпека та зручність у контролі за змінами, особливо у випадках командної роботи над проектами. Це також дозволяє у перспективі реалізувати функціонал відкату змін або формування звітів.

5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

5.1 Порівняння результатів з початковими вимогами

В процесі реалізації клієнт-серверного вебзастосунку для управління проєктами було проаналізовано і виконано всі ключові вимоги, що були визначені на етапі постановки задачі. Зокрема, створено дві основні частини системи – клієнтську (React, JavaScript) і серверну (Spring Boot, Java), які взаємодіють через REST API. Для зберігання даних та автентифікації використано хмарну NoSQL-базу Firebase Firestore.

Усі основні функції, зазначені у постановці задачі, були реалізовані:

- забезпечена багатокористувацька робота із підтримкою роліової моделі доступу (менеджери та учасники проєкту);
- реалізовано повний набір CRUD-операцій для проєктів і задач, а також механізм призначення виконавців;
- реалізовано оновлення статусу задач, а також фіксацію часу створення і зміни;
- здійснено фільтрацію задач за статусом та пошук по назві, тексту та тегам задач;
- автоматизовано збір статистики щодо виконання задач, поточного навантаження, а також реалізовано аналітичний дашборд для відображення показників активності команди;
- забезпечено надійний обмін даними між клієнтом і сервером із валідацією введеної інформації та захистом персональних даних користувачів;
- інтерфейс користувача виконано зрозумілим та лаконічно.

Додатково дотримано вимог щодо масштабованості, базової безпеки (автентифікація через Firebase), та розширюваності архітектури. Таким чином, результати розробки у повній мірі відповідають початково поставленим завданням і вимогам технічного завдання.

5.2 Оцінка якості роботи системи

Якість роботи системи оцінювалася за кількома критеріями: коректність функціонування основних модулів, надійність збереження та обробки даних, інтуїтивність інтерфейсу користувача, продуктивність і відгук системи при роботі з декількома користувачами.

В результаті тестування встановлено:

- система коректно виконує всі основні операції: створення, редагування, видалення проєктів і задач, зміну статусів, призначення виконавців, перегляд аналітики;
- дані проєктів, задач і користувачів надійно зберігаються у Firebase Firestore, без втрат чи дублювання;
- передбачено основні валідації на рівні клієнта та сервера (наприклад, запобігання створенню задач без назви чи некоректних дат);
- система демонструє стабільну роботу як у випадку індивідуального використання, так і при одночасній роботі декількох користувачів;
- інтерфейс побудовано з урахуванням сучасних вимог до UX: логічна структура сторінок, швидкий доступ до основних функцій, інформативність панелі аналітики;
- швидкість обробки операцій відповідає вимогам до сучасних вебзастосунків: більшість дій виконується миттєво, затримки практично відсутні.

Окремо відзначається висока гнучкість системи для подальшого масштабування – за рахунок розділення логіки на фронт-енд і бек-енд, а також використання REST API для взаємодії.

5.3 Аналіз ефективності прийнятих рішень

Під час розробки були прийняті ефективні архітектурні та технологічні рішення:

- використання зв'язки React + Spring Boot дало змогу чітко відокремити клієнтську та серверну частину, що спрощує обслуговування, тестування і майбутній розвиток системи;
- застосування Firebase Firestore дозволило забезпечити надійне хмарне зберігання даних, підтримку багатокористувацького режиму, масштабованість та просту інтеграцію з модулем автентифікації;
- інтеграція Google Sign-In/реєстрації спростила вхід до системи для користувачів і підвищила безпеку;
- вибір REST API як механізму обміну даними дав змогу створити універсальний та розширюваний інтерфейс для зв'язку клієнта і сервера, що підвищує гнучкість при зміні чи розширенні функціоналу;
- впровадження гнучкої ролі доступу (менеджер/учасник) та аналітичного дашборду підвищило зручність управління проєктами, прозорість роботи команди та мотивацію користувачів.

Завдяки цим рішенням система виявилася простою у використанні, ефективною, стійкою до збоїв і зручною для подальшої модифікації.

5.4 Обмеження та недоліки

В процесі реалізації та тестування системи були виявлені певні обмеження:

- поточна версія не містить розширених можливостей інтеграції з іншими сторонніми сервісами (наприклад, календарями чи месенджерами);
- безпека даних користувача базується переважно на інструментах Firebase; у разі потреби підвищеної безпеки чи специфічних політик може виникнути необхідність доопрацювання;

- візуалізація аналітики наразі реалізована у базовому вигляді; для великої кількості проєктів чи задач може знадобитись оптимізація дашборду і розширення функціоналу графіків;
- не реалізовано систему сповіщень для користувачів про зміни у задачах чи проєктах (наприклад, email чи push-повідомлення);
- масштабування системи для дуже великої кількості одночасних користувачів може вимагати додаткових оптимізацій на рівні серверної частини та бази даних.

Загалом ці обмеження не критичні для базового функціонування системи, але визначають напрямки для її подальшого розвитку.

ВИСНОВКИ

У межах виконання комплексного курсового проєкту було реалізовано повнофункціональний клієнт-серверний вебзастосунок CrewNote, призначений для організації командної роботи над проєктами та управління пов'язаними із ними завданнями [9]. Система орієнтована на малий та середній бізнес, команди розробників, стартапи та інших користувачів, які потребують ефективного інструменту для координації дій, моніторингу прогресу та обліку активностей у межах конкретного проєкту.

У результаті аналізу предметної галузі було виявлено ключові потреби цільової аудиторії, пов'язані з прозорим розподілом обов'язків, зберіганням історії змін, аналітикою ефективності та гнучкістю у роботі з проєктними даними. Порівняння існуючих аналогів дозволило виокремити функціональні можливості, які варто включити до системи, а також окреслити недоліки, які можна усунути завдяки кастомній розробці.

Для реалізації серверної частини було обрано Java з використанням фреймворку Spring Boot. Це дало змогу забезпечити гнучкість, масштабованість, високий рівень безпеки та чітке розмежування логіки відповідно до архітектурних принципів. Усі дані зберігаються в хмарній базі даних Firestore, що надає зручний доступ до структурованої інформації, забезпечує масштабованість та легкість розгортання.

Клієнтська частина реалізована за допомогою React, з використанням Vite як інструменту збирання, а також класичного CSS для стилізації. Інтерфейс підтримує маршрутизацію, контекстну авторизацію, перегляд і фільтрацію задач, ведення журналу активностей та побудову базової аналітики. Було створено адаптивну та зрозумілу для користувача структуру сторінок із відповідним розмежуванням прав доступу для різних ролей (звичайний користувач та менеджер проєкту).

Особливу увагу було приділено реалізації ключових функцій: створення, редагування та видалення задач і проєктів, контроль дедлайнів, фіксація логів змін,

виведення статистичних даних та перевірка прав доступу. Значна частина логіки роботи з даними винесена в окремі сервіси, що забезпечує модульність та легкість подальшої підтримки. Також реалізовано зручну інтеграцію між клієнтом і сервером через REST API з урахуванням обробки помилок і статусів відповіді.

Завдяки використанню сучасних технологій та структурованому підходу до архітектурного проєктування, отриманий застосунок демонструє стабільну роботу, високу швидкість завантаження та зручність користування. Його можливо легко масштабувати, доповнювати новим функціоналом і інтегрувати з іншими сервісами при потребі.

У ході реалізації було опрацьовано повний цикл розробки програмного забезпечення: від аналізу проблемної області та проєктування архітектури до реалізації й тестування ключових функціональних компонентів. Було створено UML-діаграми, структуровано логіку взаємодії між компонентами системи, сформовано зручний UI, що відповідає потребам користувачів.

Розроблений застосунок повністю відповідає поставленим вимогам, демонструє ефективну реалізацію функціоналу та має потенціал до подальшого розвитку як сучасний інструмент для управління проєктами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. UML Diagrams [Електронний ресурс]. – Режим доступу: <https://www.uml-diagrams.org/> (дата звернення 29.05.2025)
2. Firebase Documentation [Електронний ресурс]. – Режим доступу: <https://firebase.google.com/docs> (дата звернення 01.06.2025)
3. OAuth 2.0 (Google Sign-In) [Електронний ресурс]. – Режим доступу: <https://developers.google.com/identity/protocols/oauth2> (дата звернення 06.06.2025)
4. React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: <https://react.dev/> (дата звернення 25.05.2025)
5. RESTful API Design [Електронний ресурс]. – Режим доступу: <https://restfulapi.net/> (дата звернення 03.06.2025)
6. Java SE Documentation [Електронний ресурс]. – Режим доступу: <https://docs.oracle.com/en/java/> (дата звернення 04.06.2025)
7. Spring Boot Documentation [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> (дата звернення 27.05.2025)
8. Vite – Next Generation Frontend Tooling [Електронний ресурс]. – Режим доступу: <https://vitejs.dev/guide/> (дата звернення 26.05.2025)
9. Архів програмного коду. GitHub-репозиторій автора. Режим доступу: https://github.com/MeW3R1K/2025_B_KKP_PZPI-22-5_Hromov_I_S

ДОДАТОК А
Специфікація програмного забезпечення

Vision and Scope Document

for

**«Клієнт-серверний застосунок для керування
проектами з системою аналітики виконання завдань»**

Version 1.0

Prepared by

Громов Іван Сергійович

ХНУРЕ

20.06.2025

ЗМІСТ

1 ВИМОГИ ДО БІЗНЕСУ.....	53
1.1 Передумови.....	53
1.2 Бізнес-можливість	53
1.3 Бізнес-цілі та критерії успіху	53
1.4 Потреби ринку / користувачів	54
1.5 Бізнес-ризика.....	55
2 БАЧЕННЯ РІШЕННЯ.....	56
2.1 Заява про бачення.....	56
2.2 Основні функції.....	56
2.3 Припущення та залежності	57
3 МЕЖІ ТА ОБМЕЖЕННЯ	58
3.1 Межі першого релізу	58
3.2 Межі наступних релізів	58
3.3 Обмеження та виключення	59
4 БІЗНЕС-КОНТЕКСТ	60
4.1 Профілі зацікавлених сторін.....	60
4.2 Пріоритети проєкту.....	60
4.3 Операційне середовище.....	61

Історія ревізій

Name	Date	Reason For Chages	Version
Громов І. С.	20.06.2025	Розробка документу	Version 1.0

1 ВИМОГИ ДО БІЗНЕСУ

1.1 Передумови

У сучасному середовищі командної роботи особливої актуальності набувають цифрові інструменти для управління проєктами, які дозволяють координувати дії учасників, контролювати виконання завдань, здійснювати розподіл відповідальності та оцінювати результати. У більшості команд виникає потреба в системі, яка дозволяє централізовано зберігати інформацію про задачі, відображати статуси в реальному часі та генерувати аналітику для управлінських рішень.

Традиційні методи ведення проєктів (через електронні таблиці, месенджери або документи) втрачають актуальність у зв'язку з відсутністю системності, контролю версій, журналів змін і візуалізації. У відповідь на ці виклики виникла необхідність розробити сучасний вебзастосунок з клієнт-серверною архітектурою, який би дозволяв ефективно управляти проєктами, задачами, учасниками команд і формувати динамічну статистику виконання.

1.2 Бізнес-можливість

Програмне забезпечення має задовольнити потребу у зручному, інтуїтивному середовищі для планування, розподілу та моніторингу завдань в рамках командної роботи. Система дозволяє створювати проєкти, додавати завдання, призначати виконавців, змінювати статуси, вести облік виконання та переглядати аналітику. Це дозволяє підвищити прозорість командної роботи, ефективно планувати ресурси та уникати простоїв.

1.3 Бізнес-цілі та критерії успіху

Основною метою проєкту є створення інтуїтивного та ефективного рішення для управління проєктами, яке дозволить командам організувати свою роботу більш структуровано та продуктивно. Система повинна стати центральним хабом

для координації робочих процесів, де кожен учасник команди зможе чітко розуміти свої обов'язки та відстежувати прогрес виконання завдань.

Ключові бізнес-цілі включають:

- надати користувачам простий інструмент для управління проєктами;
- забезпечити централізоване зберігання інформації про завдання;
- реалізувати систему ролей (менеджер, виконавець);
- дозволити перегляд статистики виконання задач;
- досягнути стабільної та захищеної роботи застосунку.

Для оцінки успіху проєкту важливо визначити ключові показники, які дозволять оцінити рівень готовності та стабільності розробленого рішення.

Критерії успіху:

- завершення MVP з основним функціоналом;
- позитивні результати внутрішнього тестування;
- можливість масштабування без переписування архітектури.

1.4 Потреби ринку / користувачів

Користувачами системи можуть бути як внутрішні команди компаній, так і окремі фрілансери чи малі організації, які шукають безкоштовний або простий спосіб управління задачами.

Потреби:

- доступ до завдань з будь-якого пристрою;
- зрозумілий інтерфейс без потреби в навчанні;
- можливість швидко переглянути статуси по команді;
- безпечний вхід у систему (зокрема, через Google-акаунт).

1.5 Бізнес-ризика

Успішна реалізація будь-якого програмного проєкту вимагає ретельного аналізу та планування можливих ризиків, які можуть вплинути на досягнення поставлених цілей. Раннє виявлення потенційних проблем дозволяє розробити стратегії їх мітигації та забезпечити більш передбачуваний результат проєкту.

Бізнес-ризика:

- нестача часу або ресурсів для впровадження додаткового функціоналу;
- можливі труднощі з масштабуванням при великій кількості користувачів;
- складність із забезпеченням безпеки без повноцінної системи авторизації;
- ризики втрати даних без регулярного резервного копіювання.

Проактивне управління цими ризиками дозволить не тільки уникнути потенційних проблем, але й забезпечить більш стабільний та передбачуваний розвиток проєкту, що критично важливо для досягнення поставлених бізнес-цілей та забезпечення довгострокового успіху системи.

2 БАЧЕННЯ РІШЕННЯ

2.1 Заява про бачення

Система надає засіб для централізованого управління проєктами з можливістю розподілу задач, відслідковування їх виконання, контролю командної продуктивності та ведення аналітики. Застосунок працює у веббраузері, має інтуїтивний інтерфейс і дозволяє користувачам взаємодіяти в режимі реального часу.

2.2 Основні функції

Функціональність системи управління проєктами спроектована для забезпечення повного циклу роботи з проєктами та завданнями. Кожна функція була ретельно відібрана на основі аналізу потреб користувачів та найкращих практик у сфері управління проєктами, щоб забезпечити оптимальний баланс між простотою використання та функціональною повнотою.

Ключові функціональні можливості:

- реєстрація, вхід у систему (включаючи Google Sign-In);
- створення/редагування/видалення проєктів;
- додавання завдань, призначення виконавців;
- зміна статусу задач;
- перегляд аналітики (виконано/прострочено/в роботі);
- керування профілем користувача (аватар, ім'я);
- базова рольова система (менеджер, учасник).

Представлений набір функцій формує основу MVP, який дозволить користувачам ефективно організовувати свою роботу з проєктами від початкового планування до аналізу результатів. Інтеграція з Google Sign-In спрощує процес реєстрації, а рольова система забезпечує необхідний рівень контролю доступу без надмірного ускладнення системи.

2.3 Припущення та залежності

Успішна реалізація та функціонування системи базується на ряді технічних та організаційних припущень, які визначають архітектурні рішення та обмеження проєкту. Ці припущення були сформульовані на основі аналізу цільової аудиторії, технічних можливостей та сучасних стандартів веб-розробки.

Основні припущення проєкту:

- користувачі мають доступ до інтернету;
- застосунок працює в сучасному браузері;
- firebase забезпечує зберігання даних та авторизацію;
- frontend і Backend обмінюються даними через REST API;
- система не потребує локального встановлення.

Ці припущення дозволяють сфокусуватися на розробці веб-орієнтованого рішення, яке не потребує складного розгортання та підтримки локальної інфраструктури. Використання Firebase як основної платформи для бек-енд-сервісів значно спрощує архітектуру та прискорює розробку, хоча і створює залежність від зовнішнього провайдера сервісів. Орієнтація на сучасні браузери дозволяє використовувати новітні веб-технології та забезпечити кращий користувацький досвід.

3 МЕЖІ ТА ОБМЕЖЕННЯ

3.1 Межі першого релізу

Перший реліз системи зосереджений на створенні стабільної основи з усіма ключовими функціями, необхідними для базового управління проєктами. Цей етап закладає фундамент для подальшого розвитку системи та забезпечує користувачів мінімально необхідним набором інструментів.

Функціональність першого релізу:

- повноцінна підтримка створення проєктів і задач;
- автентифікація користувачів;
- перегляд списку задач з фільтрацією;
- рольова модель доступу;
- візуалізація аналітики;
- створення та оновлення профілю користувача.

Цей функціональний набір забезпечує повний цикл роботи з проєктами, від створення до аналізу результатів. Особлива увага приділена надійності автентифікації та гнучкості системи ролей, що створює безпечне середовище для колаборативної роботи команд.

3.2 Межі наступних релізів

Подальші релізи системи будуть зосереджені на розширенні функціональності та покращенні користувацького досвіду. Ці удосконалення базуватимуться на зворотному зв'язку користувачів та аналізі метрик використання системи.

Планований функціонал майбутніх версій:

- реалізація push-сповіщень або email-нагадувань;
- розширена аналітика (графіки, діаграми);
- коментарі до задач у реальному часі;
- інтеграція з календарем або зовнішніми сервісами;
- мобільна адаптація інтерфейсу.

Ці функції значно розширяють можливості системи, зробивши її більш інтерактивною та інтегрованою з робочими процесами користувачів. Система сповіщень та розширена аналітика підвищують ефективність управління проєктами, а мобільна адаптація забезпечить доступ до системи в будь-який час та в будь-якому місці.

3.3 Обмеження та виключення

Реалістичне розуміння обмежень проєкту дозволяє уникнути нереалістичних очікувань та забезпечити прозору комунікацію з усіма зацікавленими сторонами. Ці обмеження обумовлені архітектурними рішеннями, бюджетними рамками та стратегічними пріоритетами проєкту.

Основні обмеження системи:

- немає офлайн-режиму;
- не передбачено інтеграції з зовнішніми CRM-системами;
- безпека базується на засобах Firebase;
- можливі обмеження масштабованості у безкоштовному тарифі Firebase.

Ці обмеження відображають стратегічний вибір на користь швидкої розробки та розгортання за рахунок деяких розширених можливостей. Залежність від Firebase спрощує розробку та підтримку системи, але створює технічні та економічні обмеження, які необхідно враховувати при плануванні масштабування. Відсутність офлайн-режиму є свідомим рішенням, яке дозволяє зосередитися на основній функціональності, хоча може обмежити використання системи в умовах нестабільного інтернет-з'єднання.

4 БІЗНЕС-КОНТЕКСТ

4.1 Профілі зацікавлених сторін

Зацікавлені на стороні	Основна цінність	Ставлення	Основні інтереси	Обмеження
Менеджер проєкту	Керування процесом, контроль задач	Активний	Статистика, призначення виконавців	Інтерфейс, зручність
Учасник команди	Отримання і виконання задач	Залучений	Проста робота із завданнями	Обмежений доступ
Розробник	Можливість розширення системи	Технічний	API, підтримка, безпека	Чітка архітектура
Замовник/керівник	Контроль ефективності роботи команд	Аналітичний	Аналітика, продуктивність	Актуальність інформації

4.2 Пріоритети проєкту

Параметр	Ключовий драйвер	Обмеження	Ступінь свободи
Час розробки	Завершення MVP	Обмеження календарем	Середній
Функціональність	Повна базова версія	Пріоритет основного ядра	Розширюється у майбутньому
Якість	Мінімум багів	Самотестування	Середній
Зручність	Простий UX	Інтерфейс без перевантаження	Гнучкий

4.3 Операційне середовище

Система працює як вебзастосунок, що запускається у будь-якому сучасному браузері (Chrome, Firefox, Edge). Дані зберігаються у хмарній NoSQL-базі Firebase Firestore. Клієнтська частина розроблена на React + Vite, серверна — Spring Boot. Взаємодія реалізована через REST API. Жодне програмне забезпечення не потребує встановлення на пристрій користувача. Серверна частина може бути розміщена на будь-якому хостингу із підтримкою Java.

ДОДАТОК Б

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 6/27/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_ПІ_ККП_ПЗПІ_22_5_Громов_І_С_скорочений
АвторНауковий керівник / Експерт
підрозділ
каф. ПІ

Обсяг знайдених подібностей

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

1.02%
1.02% КП 1

1.14%
1.14% КЦ

25
Довжина фрази для коефіцієнта подібності 2

6559
Кількість слів

55243
Кількість символів

Тривога

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

Заміна букв		0
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		3

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Копір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Копір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://stackoverflow.com/questions/53605255/spring-boot-jwt-cors-with-angular-6	16 0.24 %
2	Ігнатенко І-робота 6/17/2024 Bohdan Khmelnytsky National University of Cherkasy (Bohdan Khmelnytsky National University of Cherkasy)	14 0.21 %

3	Ігнатенко_І-робота 6/17/2024 Bohdan Khmelnytsky National University of Cherkasy (Bohdan Khmelnytsky National University of Cherkasy)	10 0.15 %
4	https://openarchive.nure.ua/bitstreams/6980f2d3-90a8-496b-92d6-2608b79845db/download	10 0.15 %
5	https://stackoverflow.com/questions/53605255/spring-boot-jwt-cors-with-angular-6	6 0.09 %
6	https://stackoverflow.com/questions/53605255/spring-boot-jwt-cors-with-angular-6	6 0.09 %
7	Ігнатенко_І-робота 6/17/2024 Bohdan Khmelnytsky National University of Cherkasy (Bohdan Khmelnytsky National University of Cherkasy)	5 0.08 %
з бази даних RefBooks (0.00 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з домашньої бази даних (0.00 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з програми обміну базами даних (0.44 %)		
ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Ігнатенко_І-робота 6/17/2024 Bohdan Khmelnytsky National University of Cherkasy (Bohdan Khmelnytsky National University of Cherkasy)	29 (3) 0.44 %
з Інтернету (0.58 %)		
ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://stackoverflow.com/questions/53605255/spring-boot-jwt-cors-with-angular-6	28 (3) 0.43 %
2	https://openarchive.nure.ua/bitstreams/6980f2d3-90a8-496b-92d6-2608b79845db/download	10 (1) 0.15 %
Список прийнятих фрагментів (немає прийнятих фрагментів)		
ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

ДОДАТОК В

Слайди презентації

Клієнт-серверний застосунок для
керування проєктами з системою
аналітики виконання завдань

Громов Іван Сергійович, ПЗПП-22-5
Керівник: доц. кафедри ПІ Дмитро КОЛЕСНИКОВ

Мета роботи

Чітке визначення мети роботи: Створити комплексне веб-рішення для управління проєктами, що забезпечує командну співпрацю, моніторинг прогресу та аналіз ефективності виконання завдань.

Актуальність роботи: Необхідність створення централізованого інструменту управління завданнями з розширеними аналітичними можливостями та прозорістю робочих процесів.

Аналіз проблеми (аналіз існуючих рішень)

Перелік досліджених конкурентів: Jira, Asana, Monday.com

Прогалин аналогів:

Складність налаштування (Jira)

- Платні розширені функції (Asana)
- Перевантажений інтерфейс (Monday)



Постановка задачі та опис системи

Чітке формулювання проблеми: Відсутність єдиного простору для управління проектами та завданнями з аналітикою.

Опис очікуваних результатів:

- Створення проектів і завдань
- Призначення відповідальних
- Зміна статусів
- Перегляд аналітики та історії змін



Вибір технологій розробки

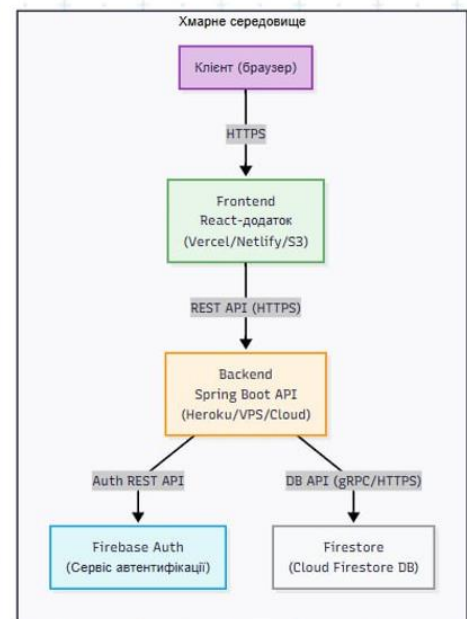
- **Клієнтська частина:** React + Vite + CSS
- **Серверна частина:** Java + Spring Boot
- **База даних та автентифікація:** Firebase Firestore + Firebase Authentication
- **API:** REST API для обміну даними між клієнтом і сервером

Архітектура створеного програмного забезпечення

Клієнт-серверна архітектура

Модулі:

- Клієнт: UI, запити, маршрутизація
- Сервер: контролери, сервіси, доступ до даних
- База даних: Firestore – зберігання даних
- Комунікація: HTTP-запити через REST API



Опис програмного забезпечення, що було використано у дослідженні

Мови та фреймворки:

- Java + Spring Boot (бекенд)
- React + CSS (фронтенд)
- Firebase (база даних, автентифікація)

Процес розробки:

- Проектування архітектури
- Створення бази даних
- Реалізація API
- Розробка UI
- Тестування функціоналу



Дизайн системи

Методи:

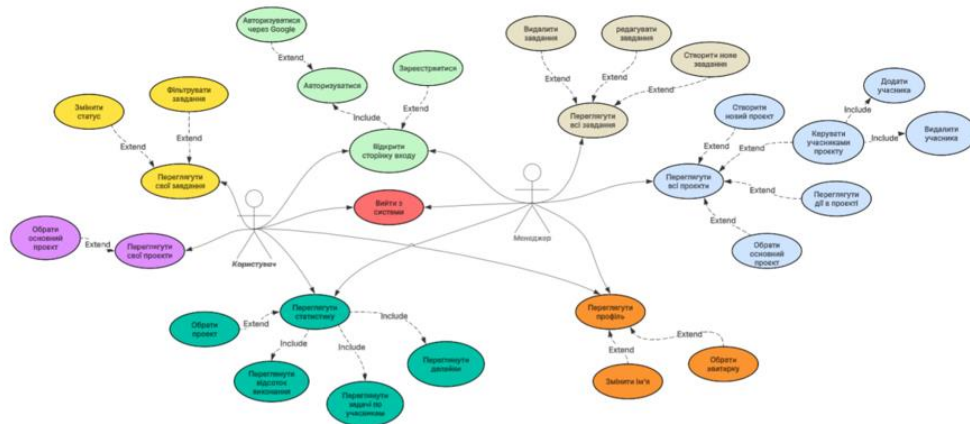
- Компонентна структура React
- Контекст для керування станом (AuthContext)
- Адаптивний дизайн з CSS

Інструменти:

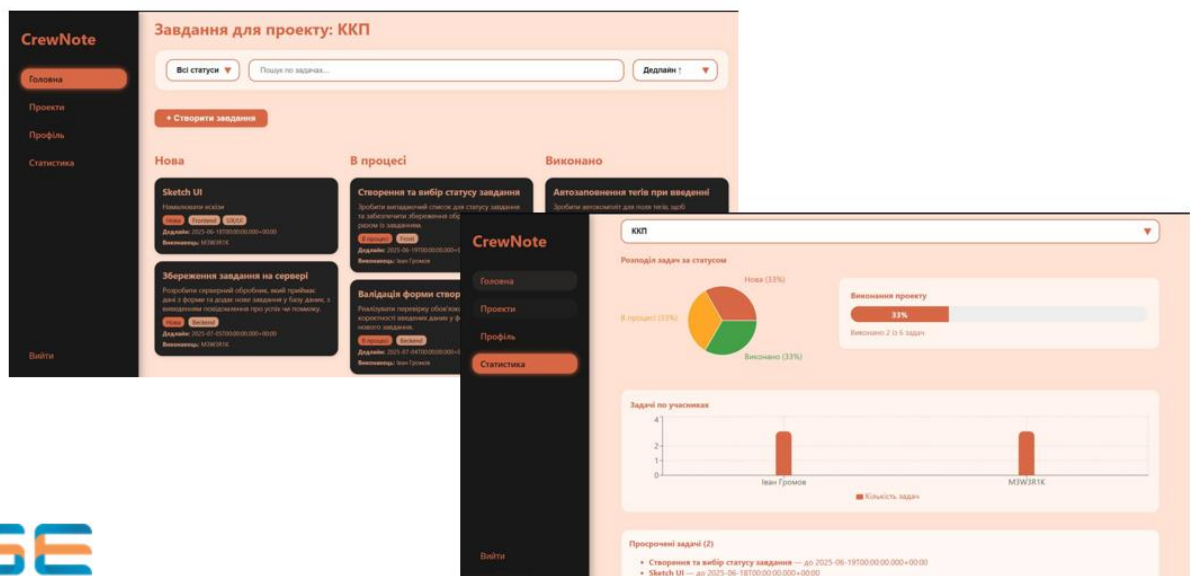
- Figma (прототипування)
- React DevTools (налагодження)



Приклад реалізації



Інтерфейс користувача



Підсумки

- Реалізовано повноцінний вебзастосунок для керування проектами.
- Система підтримує аналітику, історію змін, контроль статусів.

Можливий розвиток:

- Додати чат між користувачами
- Розширити візуалізацію аналітики
- Інтеграція зі сторонніми сервісами