КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кафедра кібербезпеки та комп’ютерної інженерії
(назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему:

__Проект інформаційної системи для планування, координації та аналізу__
________________________командної діяльності________________________

Сахно Олександр Геннадійович
(прізвище, ім’я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ
____
„___" ______________20___ року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Проект інформаційної системи для планування, координації та аналізу командної діяльності

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Сахно Олександр Геннадійович
(прізвище, ім'я та по батькові повністю)

123 «Комп'ютерна інженерія»
(спеціальність)

«Комп'ютерні системи і мережі»
(освітня програма)

Група КСМм-24

Керівник Гуменний Д.О., Маринський К.В.
(прізвище та ініціали)

к.т.н., доцент; асистент
(вчене звання, науковий ступінь)

Рецензент Босенко І.В.
(прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерна інженерія
Освітній рівень: магістр
Спеціальність: 123 «Комп'ютерна інженерія»
ОПП: Комп'ютерні системи і мережі

ЗАТВЕРДЖУЮ

Завідувач кафедри
к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ
„___”_________20___ року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Сахно Олександр Геннадійович

(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Проект інформаційної системи для планування, координації та аналізу командної діяльності

затверджена наказом ректора КНУБА № 1636/23.2/25 від «30» 09 2025 року

2. Керівник роботи
Гуменний Дмитро Олександрович, к.т.н., доцент
Маринський Костянтин Володимирович, асистент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту____________________

4. Зміст пояснювальної записки за розділами:
Р. 1. Аналіз предметної області та постановка задачі
Р. 2. Проєктування інформаційного і програмного забезпечення
Р. 3. Розробка програмного забезпечення
Р. 4.
Р. 5.

5. Графічний матеріал за розділами:
Р. 1. 1 рисунок
Р. 2. 3 рисунки
Р. 3. 13 рисунків
Р. 4.
Р. 5.

6. Консультанти розділів кваліфікаційної випускної роботи

Розділи	Прізвища, ініціали та посади консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1.	20.10.2025
Розділ 2.	17.11.2025
Розділ 3.	08.12.2025
Остаточне оформлення роботи	13.12.2025
Направлення роботи на рецензування, перевірку на плагіат	13.12.2025
Попередній захист роботи на кафедрі	15.12.2025

8. Дата видачі завдання ___30.09.2025________________

Керівник ______________ _Гуменний Д.О__

(підпис) (прізвище та ініціали)

Керівник ______________ _Маринський К.В.

(підпис) (прізвище та ініціали)

Здобувач ______________ _Сахно О.Г._____

(підпис) (прізвище та ініціали)

<table>
<tr><td>РЕЗЮМЕ
(SUMMARY)
до кваліфікаційної випускової роботи здобувача</td><td colspan="3">ПІБ
Сахно Олександр Геннадійович
Sakhno Oleksandr Hennadiyovych</td></tr>
<tr><td>ЗВО</td><td colspan="3">Київський національний університет будівництва і архітектури</td></tr>
<tr><td>Тема (українською та англійською)</td><td colspan="3">Проект інформаційної системи для планування, координації та аналізу командної діяльності
Design of an Information System for Planning</td></tr>
<tr><td>Освітній ступінь</td><td colspan="3">Магістр</td></tr>
<tr><td>Факультет</td><td colspan="3">Автоматизації і інформаційних технологій</td></tr>
<tr><td>Випускова кафедра</td><td colspan="3">Кібебезпеки та комп’ютерної інженерії</td></tr>
<tr><td>Спеціальність</td><td colspan="3">123 «Комп’ютерна інженерія»</td></tr>
<tr><td>Освітня програма</td><td colspan="3">Комп’ютерні системи і мережі</td></tr>
<tr><td>Керівник</td><td colspan="3">Гуменний Д.О., Маринський К.В.</td></tr>
<tr><td rowspan="2">Обсяг роботи:</td><td>Поснювальна записка, стор.</td><td>Розділів</td><td>Презентація, кількість слайдів</td></tr>
<tr><td>90</td><td>3</td><td>15</td></tr>
<tr><td>Розділ 1</td><td colspan="3">Аналіз предметної області та постановка задачі</td></tr>
<tr><td>Розділ 2</td><td colspan="3">Проєктування інформаційного і програмного забазпечення</td></tr>
<tr><td>Розділ 3</td><td colspan="3">Розробка програмного забезпечення</td></tr>
<tr><td>Висновки по роботі</td><td colspan="3">У роботі створено повнофункціональну веб-систему, яка автоматизує управління проєктами та оптимізує продуктивність команди завдяки впровадженню унікальних алгоритмів розподілу навантаження і динамічної пріоритезації завдань.</td></tr>
<tr><td>Ключові слова:
Keywords:</td><td colspan="3">інформаційна система, управління проєктами, Kanban, автоматизація, автоматичний розподіл задач, веб-застосунок.
information system, project management, Kanban, automation, automatic task distribution, web application.</td></tr>
</table>

Здобувач ___Сахно О.Г.______/ ________________

Керівник ___Гуменний Д.О.____/ _______________

Керівник ___Маринський К.В. / _______________

АНОТАЦІЯ

Сахно О.Г. «Проєкт інформаційної системи для планування, координації та аналізу командної діяльності».

Робота присвячена вирішенню актуальної науково-практичної задачі створення веб-орієнтованої інформаційної системи, спрямованої на комплексну оптимізацію процесів управління проєктами та підвищення ефективності координації командної роботи. У рамках дослідження здійснено повний цикл проєктування та програмної реалізації системи, що включає розробку нормалізованої реляційної бази даних для забезпечення цілісності інформації, побудову масштабованої серверної частини на основі платформи Node.js із використанням архітектурного стилю REST API, а також створення інтерактивного клієнтського інтерфейсу Single Page Application. Візуалізація робочих процесів базується на методології Kanban, що забезпечує інтуїтивно зрозуміле управління життєвим циклом завдань. Особливу увагу в роботі приділено розробці та впровадженню інтелектуальних алгоритмів бізнес-логіки, зокрема механізму динамічної пріоритезації, який автоматично актуалізує статус терміновості задач залежно від часових обмежень, та унікального алгоритму автоматичного розподілу навантаження, що дозволяє оптимізувати використання людських ресурсів шляхом делегування завдань найменш завантаженим виконавцям. Практична цінність отриманих результатів полягає у створенні функціонального програмного продукту, здатного мінімізувати управлінські помилки та попередити професійне вигорання співробітників завдяки збалансованому розподілу робочого навантаження.

Ключові слова: Інформаційна система, управління проєктами, Kanban, автоматизація, автоматичний розподіл задач, веб-застосунок.

SUMMARY

Sakhno O.G. “Design of an Information System for Planning”.

The work is devoted to solving the urgent scientific and practical problem of creating a web-oriented information system aimed at comprehensive optimization of project management processes and improving the efficiency of teamwork coordination. As part of the research, a full cycle of system design and software implementation was carried out, including the development of a normalized relational database to ensure information integrity, the construction of a scalable server part based on the Node.js platform using the REST API architectural style, and the creation of an interactive Single Page Application client interface. The visualization of work processes is based on the Kanban methodology, which provides intuitive task lifecycle management. Particular attention in the work is paid to the development and implementation of intelligent business logic algorithms, in particular, a dynamic prioritization mechanism that automatically updates the urgency status of tasks depending on time constraints, and a unique algorithm for automatic load distribution, which allows optimizing the use of human resources by delegating tasks to the least loaded performers. The practical value of the results obtained lies in the creation of a functional software product capable of minimizing management errors and preventing employee burnout through a balanced distribution of workload.

Keywords: Information system, project management, Kanban, automation, automatic task distribution, web application.

ЗМІСТ

Перелік умовних позначень....……………………………………………………….10
ВСТУП.. 11
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 13
1.1 Постановка та аналіз проблеми 13
1.2 Дерево цілей... 16
1.3 Вимоги та особливості проєктування системи 19
1.4 Аналіз існуючих розробок... 21
1.5 Постановка задачі.. 24
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ.. 28
2.1 Основні питання проектування інформаційного забезпечення 28
2.2 Дерево функцій системи... 22
2.3 Аналіз та вибір програмних засобів реалізації... 35
2.3.1 Вибір технологічного стеку серверної частини 35
2.3.2 Архітектура клієнтського застосу 38
2.3.3 Огляд рішень для реалізації інтерактивної дошки та аналітики.................. 41
2.4 Проєктування бази даних 44
2.4.1 Аналіз та вибір моделі організації даних... 44
2.4.2 Концептуальна та логічна моделі даних.. 47
2.4.3 Фізична модель та типи даних 50
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ……………………………...54
3.1 Ініціалізація та конфігурація середовища розробки…………………..………54
3.1.1 Створення проєкту та встановлення залежностей………………...…..….54
3.1.2 Опис файлової архітектури проєкту …………………………………..…..56
3.2 Реалізація серверної частини……………………………………………...…….60
3.2.1 Розробка механізмів авторизації та Middleware...…………………….…….60
3.2.2 Реалізація REST API для управління проєктами та задачами……………..64

3.2.3 Програмна реалізація алгоритмів бізнес-логіки……………………………67
3.3 Реалізація клієнтської частини…………………………...…………………71
3.3.1 Організація маршрутизації та структури компонентів……………………..71
3.3.2 Реалізація інтерфейсу Kanban-дошки та механізму Drag-and-Drop……….74
3.3.3 Реалізація модулів аналітики та візуалізації даних ………………………..77
3.4 Інтерфейс користувача та сценарії взаємодії…………………………………80
ЗАГАЛЬНІ ВИСНОВКИ…..………………………………………………………86
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ...…………...………..87
ДОДАТОК А………………...………………………………………..………...91
ДОДАТОК Б……………………………………………………..………....92
ДОДАТОК В ..…………………………………………………………..……....94
ДОДАТОК Г ..…………………………………………………………..……....95

Перелік умовних позначень

ACID – Atomicity, Consistency, Isolation, Durability

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

DOM – Document Object Model

DRY – Don't Repeat Yourself

FK – Foreign Key (Зовнішній ключ)

HTTPS – HyperText Transfer Protocol Secure

I/O – Input/Output (Введення-виведення)

JSON – JavaScript Object Notation

JWT – JSON Web Token

NPM – Node Package Manager

PK – Primary Key (Первинний ключ)

RBAC – Role-Based Access Control

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

UI/UX – User Interface / User Experience

XSS – Cross-Site Scripting

3NF – Third Normal Form (Третя нормальна форма)

БД – База даних

ЗВО – Заклад вищої освіти

ІТ – Інформаційні технології

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

ВСТУП

Актуальність теми зумовлена необхідністю підвищення ефективності управління гібридними командами в умовах, коли існуючі рішення є або надмірно складними, або не забезпечують автоматизованого розподілу навантаження. Це створює потребу в розробці нової інформаційної системи, яка поєднує зручний візуальний інтерфейс із механізмами інтелектуального балансування ресурсів для усунення нерівномірної зайнятості виконавців. Впровадження такого інструменту дозволить автоматизувати рутинні управлінські рішення, мінімізувати ризики зриву дедлайнів та підвищити загальну продуктивність команди.

Метою роботи є розробка проєкту інформаційної системи, яка б не лише покривала базові потреби проєктних команд у плануванні та координації, а й пропонувала унікальні інструменти для оптимізації розподілу навантаження та підвищення прозорості робочих процесів через автоматизацію рутинних управлінських рішень.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести аналіз предметної області.
2. Оглянути існуючі ринкові аналоги.
3. Сформулювати функціональні вимоги до системи.
4. Розробити архітектуру системи.
5. Створити ключові компоненти системи.
6. Провести тестування розробленого продукту.

Об’єкт дослідження – процеси планування, координації та аналізу командної діяльності в рамках проєктного менеджменту.

Предмет дослідження – інформаційна система для автоматизації управління командною роботою з підтримкою ролей, задач, інтелектуального розподілу навантаження та аналітики виконання.

Методи дослідження. У роботі використано методи системного аналізу (для дослідження предметної області та аналогів), метод функціональної декомпозиції (для побудови дерева функцій), методи моделювання даних (для проєктування

схеми бази даних ERD), об’єктно-орієнтоване проєктування (для розробки архітектури програмних компонентів) та методи алгоритмізації (для реалізації логіки авторозподілу).

Наукова новизна одержаних результатів полягає у вдосконаленні підходів до управління задачами в малих групах шляхом поєднання візуальної методології Kanban з автоматизованим жадібним алгоритмом розподілу завдань, що дозволяє динамічно вирівнювати навантаження на виконавців у реальному часі без участі менеджера.

Практичне значення одержаних результатів. Розроблена інформаційна система є готовим програмним продуктом, який може бути впроваджений у діяльність IT-команд, маркетингових агентств або інших організацій з проєктною структурою. Використання системи дозволить скоротити час на планування робіт, зменшити ризик зриву дедлайнів завдяки системі динамічних пріоритетів та запобігти професійному вигоранню співробітників завдяки збалансованому розподілу задач.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

- У першому розділі проведено аналіз предметної області та існуючих рішень, обґрунтовано вибір технологій.
- У другому розділі спроєктовано архітектуру системи, базу даних та інтерфейси.
- У третьому розділі описано програмну реалізацію компонентів системи, алгоритмів та інтерфейсу користувача.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка та аналіз проблеми

Ефективне управління командною діяльністю є ключовим фактором успіху будь-якого проєкту в сучасному бізнес-середовищі. Зі зростанням складності завдань, поширенням віддаленої роботи та необхідністю прозорої координації зусиль багатьох учасників, роль інформаційних систем для управління проєктами стає дедалі важливішою. Такі системи дозволяють автоматизувати процеси планування, розподілу завдань, контролю виконання та аналізу продуктивності, що мінімізує ризики зриву термінів та підвищує загальну ефективність команди.

Для визначення напрямку даної роботи необхідно визначити об’єкт і предмет дослідження.

Об’єкт дослідження – процеси планування, координації та аналізу командної діяльності в рамках проєктного менеджменту.

Предмет дослідження – інформаційна система для автоматизації управління командною роботою з підтримкою ролей, задач та аналітики виконання.

Аналіз фахової літератури та сучасних досліджень у галузі програмної інженерії дозволяє виділити фундаментальну проблему управління розподіленими командами. Науковці зазначають, що зі збільшенням кількості комунікаційних каналів у гібридних командах (чати, пошта, відеозв'язок) виникає ефект «інформаційного шуму», який ускладнює об'єктивну оцінку завантаженості співробітників. Дослідження показують, що при ручному розподілі завдань менеджери схильні делегувати роботу тим виконавцям, які найчастіше перебувають «на зв'язку» або демонструють високу швидкість закриття простих задач, ігноруючи реальну складність роботи. Це призводить до викривлення балансу ресурсів: виникає ситуація, коли 20% команди виконують 80% критичної роботи, що неминуче веде до професійного вигорання ключових фахівців та демотивації інших учасників. Автори наголошують, що вирішення цієї проблеми лежить у площині переходу від інтуїтивного менеджменту до алгоритмічного, де

рішення про призначення виконавця базується на неупереджених метриках завантаженості [1, 3].

Ситуація ускладнюється станом ринку існуючого програмного забезпечення. Популярні рішення хоч і є потужними інструментами, часто мають суттєві обмеження. Це вимагає від керівника постійного мікроменеджменту та ручного моніторингу черги задач, що забирає значний час і не гарантує оптимального результату.

Виходячи з аналізу літературних джерел та ринкової ситуації, виникає об’єктивна потреба у створенні інформаційної системи яка має поєднувати класичний функціонал таск-менеджера з інтелектуальними інструментами автоматизації. Ключовою відмінністю такого рішення має стати впровадження алгоритму автоматичного розподілу задач, який аналізує поточну завантаженість виконавців, дозволяючи оптимізувати робоче навантаження в команді без втручання людського фактора.

Таким чином, метою роботи є розробка проєкту інформаційної системи, яка б не лише покривала базові потреби проєктних команд у плануванні, а й пропонувала унікальні інструменти для оптимізації розподілу навантаження та підвищення прозорості робочих процесів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

7. Провести аналіз предметної області: дослідити основні методології управління проєктами та визначити ключові процеси, що потребують автоматизації.

8. Оглянути існуючі ринкові аналоги: проаналізувати функціональні можливості та недоліки популярних систем управління проєктами для виявлення незакритих потреб користувачів.

9. Сформулювати функціональні вимоги до системи: детально описати всі заплановані можливості, включаючи управління користувачами та ролями, менеджмент проєктів і завдань, інтерактивну дошку задач та модуль аналітики.

10. Розробити архітектуру системи: спроєктувати структуру бази даних, визначити основні програмні модулі та зв'язки між ними.

11. Створити ключові компоненти системи: реалізувати основний функціонал, приділяючи особливу увагу унікальному модулю автоматичного розподілу задач.

12. Провести тестування розробленого продукту: перевірити працездатність системи та відповідність її функціональності поставленим вимогам.

Поетапний процес розробки інформаційної системи для управління командною діяльністю можна представити у вигляді класичного життєвого циклу, що включає наступні основні етапи:

1. Етап планування. На цьому етапі визначаються основні цілі та завдання проєкту. Формулюється бачення майбутнього продукту: створення зручної, інтуїтивно зрозумілої системи, яка допомагає командам ефективно організовувати свою роботу та містить унікальну функцію для автоматизації розподілу навантаження.

2. Етап збору вимог. Це один з ключових етапів, на якому деталізуються всі функціональні та нефункціональні вимоги до системи. Визначаються ролі користувачів (адміністратор, проєктний менеджер, виконавець) та їхні права доступу. Описуються атрибути проєктів та завдань (статуси, пріоритети, дедлайни), логіка роботи дошки задач та вимоги до аналітичних звітів.

3. Етап проєктування. На основі зібраних вимог розробляється технічна архітектура системи. Проєктується структура бази даних для зберігання інформації про користувачів, проєкти, задачі та коментарі. Створюється дизайн користувацького інтерфейсу (UI/UX), включаючи макети дошки задач, сторінок проєктів та аналітичних дашбордів. Продумується взаємодія між клієнтською частиною та серверною.

4. Етап розробки програмного забезпечення. На цьому етапі відбувається написання програмного коду відповідно до проєктної документації. Розробники реалізують усі модулі системи: механізм автентифікації, створення та редагування проєктів і завдань, функціональність drag-and-drop для дошки задач, алгоритм автоматичного розподілу завдань та візуалізацію аналітичних даних.

5. Етап тестування. Команда тестування перевіряє розроблену систему на відповідність вимогам та наявність помилок. Проводиться функціональне тестування (чи коректно працює створення задач, зміна статусів), тестування інтерфейсу користувача (чи зручно користуватися дошкою), а також тестування безпеки (чи правильно працює розмежування прав доступу).

1.2 Дерево цілей

У сучасних умовах динамічного розвитку IT-сфери та бізнес-процесів ефективність роботи будь-якої організації безпосередньо залежить від якості управління командною діяльністю. Зростання обсягів інформації, розподіленість команд та необхідність оперативного реагування на зміни вимагають впровадження спеціалізованих інструментальних засобів. Відсутність єдиного інформаційного простору для комунікації та контролю часто призводить до зриву дедлайнів, нерівномірного розподілу навантаження та втрати важливих даних.

У зв'язку з цим, основною метою роботи є розробка інформаційної системи для планування, координації та аналізу командної діяльності. Ця система покликана автоматизувати рутинні процеси менеджменту, забезпечити прозорість виконання завдань та надати інструменти для оцінки продуктивності команди.

Для досягнення поставленої мети було проведено декомпозицію головної цілі на підцілі, що дозволило сформувати ієрархічну структуру завдань.

Функціональні вимоги: Ця група вимог визначає набір функцій, які система повинна виконувати для задоволення потреб користувачів.

1.1. Управління користувачами та доступом: Реалізація механізмів реєстрації, автентифікації та розмежування прав доступу на основі ролей (адміністратор, менеджер, виконавець).

1.2. Менеджмент проєктів: Можливість створення та редагування проєктів, а також налаштування команди для кожного окремого проєкту.

1.3. Управління завданнями: Забезпечення повного життєвого циклу задачі (створення, призначення виконавця, встановлення дедлайнів, зміна пріоритетів).

1.4. Інтерактивна дошка завдань: Реалізація візуалізації завдань у вигляді Kanban-дошки з підтримкою технології Drag-and-Drop для швидкої зміни статусів.

1.5. Аналітика та звітність: Автоматичний збір статистики щодо виконання завдань, побудова графіків продуктивності та розподілу навантаження.

1.6. Унікальна функціональність: Впровадження алгоритмів автоматичного розподілу задач та динамічного розрахунку пріоритетів на основі часових міток.

2. Вимоги до надійності та безпеки: Критично важливим аспектом є захист інформаційних активів та стабільність роботи.

2.1. Забезпечення конфіденційності та цілісності даних: Використання шифрування паролів та захищених протоколів передачі даних для захисту інформації про користувачів, проєкти та завдання.

2.2. Захист системи: Реалізація заходів протидії несанкціонованому доступу та базовим веб-вразливостям (таким як SQL-ін'єкції, XSS).

2.3. Забезпечення стабільної роботи: Розробка механізмів обробки помилок та резервного копіювання даних для запобігання їх втрати у разі технічних збоїв.

3. Вимоги до продуктивності та масштабованості: Система повинна бути готова до зростання навантаження.

3.1. Швидкодія: Забезпечення швидкого відгуку інтерфейсу користувача (SPA), особливо при роботі з інтерактивною дошкою, де зміни мають відбуватися миттєво.

3.2. Архітектурна гнучкість: Проєктування архітектури (клієнт-сервер), що дозволяє в майбутньому збільшувати кількість користувачів та обсяг даних без суттєвого погіршення продуктивності.

4. Вимоги до користувацького інтерфейсу (UI/UX): Ергономіка системи безпосередньо впливає на ефективність роботи персоналу [11].

4.1. Сучасний дизайн: Розробка інтуїтивно зрозумілого, чистого та естетично привабливого інтерфейсу, що знижує когнітивне навантаження.

4.2. Зручна навігація: Забезпечення логічної структури меню та швидкого переходу між проєктами, завданнями та аналітичними звітами.

4.3. Адаптивність: Оптимізація інтерфейсу для коректного відображення на стандартних екранах настільних комп'ютерів та ноутбуків.

Дерево цілей, що описує структуру даної мети, наведено на рис. 1.1.

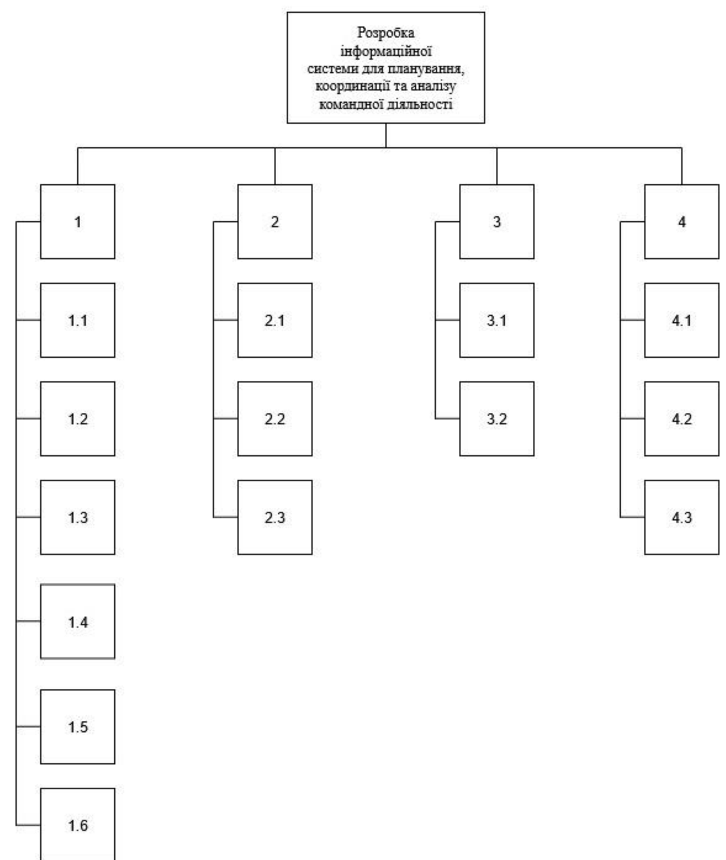

Рис. 1.1. Дерево цілей

У даному підрозділі було визначено стратегічну мету дипломного проєкту та деталізовано її через побудову дерева цілей. Сформований перелік вимог охоплює всі ключові аспекти розробки програмного забезпечення: від функціонального наповнення до питань безпеки, продуктивності та ергономіки.

Визначені вимоги слугуватимуть базою для подальшого проєктування архітектури системи, вибору технологічного стеку та розробки схеми бази даних. Реалізація повного комплексу зазначених вимог дозволить створити конкурентоспроможний програмний продукт, здатний суттєво підвищити ефективність планування та координації діяльності команд.

1.3 Вимоги та особливості проєктування системи

Проєктування інформаційної системи для управління командною діяльністю є комплексним завданням, що охоплює розробку трьох ключових компонентів: бази даних, серверної частини та клієнтської частини. Цей процес вимагає чіткого визначення вимог, які поділяються на функціональні та нефункціональні.

Функціональні вимоги визначають архітектуру бізнес-логіки та структуру даних. У контексті розроблюваної системи вони групуються за двома напрямками: зберігання даних та операції над ними.

Зберігання даних передбачає створення надійної інформаційної моделі для наступних сутностей:

- Інформація про користувачів. Система повинна зберігати не лише облікові дані для входу (логін, хеш пароля), але й розширений профіль: повне ім'я, контактні дані та, що найважливіше, роль у системі. Рольова модель є основою для подальшого розмежування прав доступу.
- Інформація про проєкти. Проєкт виступає контейнером для завдань. Необхідно зберігати його назву, детальний опис, глобальні дедлайни та список учасників, що дозволить формувати ізольовані робочі групи.
- Інформація про завдання. Це центральна сутність системи. Для кожного завдання фіксується назва, технічний опис, поточний статус, пріоритет виконання, чіткий дедлайн, ID відповідального виконавця та зв'язок з конкретним проєктом.
- Історія та файли. Для забезпечення аудиту та командної роботи необхідно зберігати хронологію змін (хто і коли змінив статус), коментарі користувачів та посилання на вкладені файли (документацію, макети).

Операції з даними описують динаміку системи та взаємодію користувача з інформацією:

- Базові операції (CRUD). Надання інтерфейсу для створення, читання, оновлення та видалення всіх ключових сутностей. Це фундамент, на якому будується вся подальша логіка.

- Бізнес-логіка зміни статусів (Kanban). Реалізація механізму переходу завдань між станами. Це не просто зміна поля в базі даних, а процес, що може запускати супутні дії. Візуалізація цього процесу через інтерфейс Kanban-дошки робить управління інтуїтивним.
- Аналітика та дашборди. Агрегація "сирих" даних для формування звітів. Це включає статистику по завершених задачах та розподілу задач по статусам.
- Автоматичний розподіл. Реалізація алгоритму, який може автоматично призначати завдання виконавцям, базуючись на їх поточній завантаженості.

Окрім функціонала, критично важливими є нефункціональні вимоги, які визначають якість та стабільність рішення:

- Продуктивність. Система повинна забезпечувати миттєвий відгук інтерфейсу. Особливо це стосується інтерактивної роботи з Kanban-дошкою: затримка при перетягуванні картки руйнує користувацький досвід (UX).
- Масштабованість. Архітектура має бути спроєктована з запасом міцності. Зростання кількості користувачів або проєктів не повинно вимагати переписання коду. Це передбачає можливість горизонтального масштабування сервера або оптимізацію запитів до БД.
- Надійність. Гарантія стабільного доступу та збереження даних досягається через регулярне резервне копіювання (бекапи), використання механізмів автоматичного відновлення після збоїв та ретельну обробку виключних ситуацій (errors handling) як на сервері, так і в браузері.
- Безпека. Оскільки система містить корпоративні дані, вимоги до безпеки є суворими: захист від несанкціонованого доступу (надійна автентифікація), чітка авторизація дій (перевірка прав на кожен запит), використання шифрування (HTTPS) та превентивний захист від атак типу SQL-injection та XSS [19].

Технічна реалізація базується на базі даних, сервері та клієнті.

Проєктування бази даних вимагає вибору реляційної моделі (наприклад, MySQL або PostgreSQL), оскільки дані є високоструктурованими та взаємопов'язаними.

- Нормалізація. Схему БД слід привести мінімум до третьої нормальної форми (3NF) для усунення дублювання інформації.
- Зв'язки. Використання зовнішніх ключів (Foreign Keys) гарантує цілісність даних (неможливо створити задачу для неіснуючого проєкту).

Проєктування серверної частини (Backend) відповідає за бізнес-логіку та обробку запитів.

- Технологічний стек. Вибір надійних технологій.
- Сервісний шар. Виділення бізнес-логіки в окремі сервіси, щоб контролери займалися лише обробкою HTTP-запитів.
- Безпека API. Впровадження JWT або сесій для захисту ендпоінтів.

Проєктування клієнтської частини є "обличчям" системи.

- Компонентний підхід. Використання фреймворків типу React або Vue дозволяє будувати інтерфейс з незалежних блоків (компонентів).
- Управління станом. Використання глобального сховища стану забезпечує синхронізацію: коли один користувач змінює статус задачі, інтерфейс повинен оновитися без перезавантаження.
- UI/UX Kanban-дошки. Особливу увагу слід приділити ергономіці: Drag-and-Drop має бути плавним, а візуальна ієрархія завдань — чіткою.

Успішне поєднання цих компонентів дозволить створити цілісний продукт, що задовольняє потреби сучасних команд у плануванні та координації.

1.4 Аналіз існуючих розробок

У сучасному світі ефективне управління проєктами є неможливим без використання спеціалізованих інформаційних систем. Вони дозволяють централізувати комунікацію, структурувати робочі процеси, відстежувати прогрес та аналізувати продуктивність команди. Ринок пропонує широкий вибір

інструментів, від простих таск-менеджерів до комплексних корпоративних платформ. Проте, незважаючи на різноманіття, більшість рішень фокусуються на фіксації стану завдань, залишаючи функцію оптимізації розподілу ресурсів на плечах менеджера.

У цьому аналізі розглянуто найпопулярніші системи управління проєктами (Jira, Trello, Asana), щоб визначити їхні ключові можливості та виявити критичні недоліки, які покликана вирішити проєктована система.

Jira є беззаперечним галузевим стандартом для IT-команд, глибоко інтегрованим з методологіями Agile (Scrum/Kanban) [27, 28].

- Переваги: Висока гнучкість, налаштування workflow будь-якої складності, потужна звітність та величезний маркетплейс плагінів.
- Недоліки: Відсутність "розумного" розподілу: Jira дозволяє бачити завантаженість команди, але не пропонує інструменту для автоматичного вирівнювання навантаження. Розподіл завдань відбувається вручну. Хоча в Jira є поле "Priority", воно не змінюється автоматично з плином часу. Для невеликих команд інтерфейс Jira перевантажений, а налаштування автоматизації вимагає значних часових витрат.

2. Trello – це інструмент, що популяризував візуальний підхід Kanban. Його філософія — цифрова імітація стікерів на дошці [4].

- Переваги: Максимальна простота, низький поріг входження, візуальна наочність.
- Недоліки: У Trello відсутнє поняття "алгоритмічного управління". Всі переміщення та призначення робляться виключно руками. Базова версія не надає звітів про продуктивність чи розподіл навантаження. Це робить неможливим об'єктивний аналіз ефективності співробітників без купівлі платних "Power-Ups". При великій кількості задач дошка перетворюється на хаос, де важко візуально виділити термінові завдання без ручного проставляння міток ("Labels").

3. Asana

Універсальна платформа для управління роботою, яка намагається поєднати простоту списків з потужністю діаграм Ганта.

- Переваги: Зручний інтерфейс, різні режими перегляду (List, Board, Timeline), розвинені правила автоматизації (Rules).
- Недоліки: Автоматизація на основі тригерів, а не стану. Вона не вміє робити: "Призначити на того, у кого зараз найменше задач". Це критична відмінність для балансування навантаження. Повноцінні дашборди для відстеження ефективності доступні лише в дорогих корпоративних тарифах, що є бар'єром для малих команд.

Для наочного порівняння функціональних розривів наведено таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей

Критерій	**Jira**	**Trello**	**Asana**	**Проєктована система**
Цільова аудиторія	Великі IT-команди	Малі команди, особисте використання	Середній та великий бізнес	Малі та середні проєктні команди
Основна методологія	Agile (Scrum, Kanban)	Kanban	Гнучка	Kanban + Smart Assignment
Візуалізація терміновості	Статична (потребує налаштувань)	Ручна (кольорові мітки)	Статична (дати)	Динамічна (зміна кольору від часу)
Вбудована аналітика	Потужна, але складна	Відсутня в базі	Доступна в Premium	Базова вбудована (безкоштовна)
Автоматичний розподіл задач	Ні (тільки через плагіни)	Ні	Ні (лише тригерні правила)	Так (алгоритм балансування навантаження)
Складність впровадження	Висока	Низька	Середня	Низька

Проведений аналіз демонструє, що на ринку існує розрив між простими інструментами візуалізації (Trello) та складними системами управління процесами (Jira). Жодна з розглянутих систем у своїй базовій комплектації не вирішує проблему нерівномірного розподілу навантаження на учасників команди.

Ключові "незакриті" потреби, на вирішення яких спрямована розробка власної системи:

1. Проблема "Bottleneck" (Вузького місця): У існуючих системах часто виникає ситуація, коли один ефективний виконавець перевантажений завданнями, а інші простоюють. Менеджер повинен відстежувати це вручну. Проєктована система вирішує це шляхом впровадження алгоритму автоматичного розподілу, який призначає нові задачі найменш завантаженому співробітнику.
2. Проблема візуального шуму: У стандартних системах прострочена задача часто виглядає так само, як і звичайна, поки користувач не подивиться на дату. Власна розробка пропонує динамічний пріоритет, де картка змінює свій візуальний стиль (колір, індикатор) автоматично при наближенні дедлайну, без втручання користувача.
3. Доступність аналітики: Малим командам потрібні прості метрики ефективності (хто скільки зробив, яка динаміка), але вони не готові платити за Enterprise-тарифи Asana чи налаштовувати BI-системи для Jira. Власна система надає ці дані "з коробки".

Таким чином, розробка власної інформаційної системи є доцільною, оскільки вона поєднає зручність Kanban-дошки з унікальними механізмами інтелектуального менеджменту ресурсів, які наразі відсутні або важкодоступні в існуючих аналогах.

1.5 Постановка задачі

Ефективність командної роботи в сучасному IT-середовищі критично залежить від якості інструментів управління. Як було визначено в попередніх підрозділах, існуючі на ринку рішення часто поляризовані: вони або занадто складні та дорогі для невеликих команд, або функціонально обмежені, пропонуючи лише базову візуалізацію без можливостей автоматизації. Це створює потребу в розробці спеціалізованого програмного забезпечення, яке б поєднувало візуальну простоту Kanban-методології з інтелектуальними алгоритмами розподілу ресурсів.

Основною задачею є створення веб-застосунку на основі клієнт-серверної архітектури, що включає три взаємопов'язані компоненти:

1. Реляційна база даних: Для надійного зберігання структурованої інформації та забезпечення цілісності зв'язків між проєктами, користувачами та завданнями.

2. Серверна частина: Для реалізації REST API, механізмів автентифікації та унікальної бізнес-логіки (зокрема, алгоритмів автоматичного розподілу завдань).

3. Клієнтська частина: Для надання користувачам інтерактивного інтерфейсу Single Page Application (SPA) з підтримкою технології Drag-and-Drop та візуалізації аналітичних даних у реальному часі.

Для досягнення поставленої мети необхідно виконати наступний ряд завдань:

1. Системний аналіз предметної області: Провести аналіз існуючих аналогів, виявити їхні недоліки та сформулювати детальні функціональні й нефункціональні вимоги до системи, фокусуючись на потребах менеджерів проєктів та виконавців.

2. Обґрунтування вибору засобів розробки: Провести порівняльний аналіз сучасних технологій та обрати оптимальний стек для реалізації поставлених вимог.

3. Проєктування архітектури системи: Розробити інфологічну та датологічну моделі бази даних, спроєктувати структуру RESTful API та визначити компонентну ієрархію клієнтського застосунку.

4. Програмна реалізація базового функціоналу: Створити механізми реєстрації та авторизації користувачів, реалізувати CRUD-операції для управління проєктами та задачами.

5. Розробка алгоритмів бізнес-логіки: Імплементувати алгоритм динамічного розрахунку пріоритетів на основі часових міток (дедлайнів) та алгоритм автоматичного балансування навантаження для розподілу задач між виконавцями.

6. Створення інтерактивного інтерфейсу: Розробити Kanban-дошку з можливістю переміщення карток, реалізувати модальні вікна для роботи з даними та інтегрувати графічні модулі для візуалізації статистики.

7. Забезпечення безпеки та надійності: Реалізувати рольову модель доступу, захист маршрутів API, хешування паролів та валідацію вхідних даних.

8. Тестування та налагодження: Провести перевірку коректності роботи всіх модулів системи, наповнити базу даних тестовим набором даних для демонстрації сценаріїв використання.

Для коректного функціонування системи визначено потоки вхідної та вихідної інформації.

Вхідні дані:

- Реєстраційні дані: Персональна інформація користувачів (ім'я, email, пароль) та їхні системні ролі.
- Метадані проєктів: Назви, описи, граничні терміни виконання проєктів.
- Параметри завдань: Заголовки, детальні описи, дати дедлайнів, ручне призначення виконавців (у разі відмови від авторозподілу).
- Системні події: Дії користувача в інтерфейсі (переміщення карток Drag-and-Drop, натискання кнопок зміни статусів, вибір фільтрів).

Вихідні дані:

- Інтерактивна візуалізація: Динамічна Kanban-дошка, де задачі згруповані за статусами («Planned», «In Progress», «Review», «Completed») та відсортовані за пріоритетом.
- Індикатори терміновості: Візуальне маркування карток (кольорові індикатори), що автоматично змінюється залежно від близькості дедлайну (наприклад, червоний колір для прострочених задач).
- Аналітична звітність: Графічні діаграми, що відображають розподіл завдань за статусами та динаміку продуктивності команди у реальному часі.
- Результати автоматизації: Списки завдань, автоматично призначені конкретним виконавцям на основі аналізу їхнього поточного навантаження.

Успішне виконання окресленого комплексу завдань дозволить досягти головної мети дипломного проєкту — створення повнофункціональної інформаційної системи, що відповідає сучасним вимогам до управління IT-проєктами. Реалізація проєкту не лише автоматизує рутинні процеси планування та контролю, але й запропонує нові підходи до менеджменту ресурсів.

Впровадження унікальних функціональних модулів, зокрема алгоритму автоматичного розподілу навантаження та системи динамічної пріоритезації, забезпечить розробці суттєві конкурентні переваги. На відміну від існуючих на ринку спрощених рішень, проєктована система дозволить мінімізувати вплив людського фактора при розподілі завдань, попередити професійне вигорання учасників команди та нівелювати ризики зриву дедлайнів через неефективний менеджмент.

Крім того, використання сучасного технологічного стеку гарантує створеному продукту високу продуктивність, масштабованість та безпеку даних. Таким чином, розроблена система зможе зайняти вільну ринкову нішу проміжного ланцюга між простими таск-трекерами та складними корпоративними платформами, пропонуючи збалансоване рішення для малих та середніх команд.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Основні питання проектування інформаційного забезпечення

Проєктування інформаційного забезпечення є фундаментальним та системоутворюючим етапом у життєвому циклі розробки автоматизованої системи планування, координації та аналізу командної діяльності. Цей етап виступає сполучною ланкою між аналізом вимог, проведеним у першому розділі, та безпосередньою програмною реалізацією продукту. Від якості виконання цього етапу, глибини опрацювання структур даних та коректності побудови інформаційних потоків залежить не лише коректність функціонування майбутнього програмного продукту, а і його здатність до масштабування, надійність зберігання даних, швидкість реакції на дії користувачів та можливість подальшої модернізації.

Метою даного підрозділу є визначення ключових векторів проєктування та формування детального "креслення" системи, яке дозволить трансформувати абстрактні бізнес-вимоги та користувацькі сценарії у чітку технічну структуру, придатну для реалізації у програмному коді.

Для створення ефективного інформаційного забезпечення, що відповідає сучасним стандартам розробки ПЗ, необхідно комплексно розглянути та вирішити ряд взаємопов'язаних концептуальних питань:

1. Побудова дерева функцій системи Цей крок є необхідним для структурування, логічного впорядкування та ієрархічної організації всіх можливостей платформи. Метод функціональної декомпозиції дозволяє розділити складну мету — «підвищення ефективності роботи команди» — на ряд простих, зрозумілих та технічно реалізованих підзадач.

- Деталізація процесів: Розбиття системи на модулі (управління проєктами, робота з Kanban-дошкою, модуль аналітики, система адміністрування) дозволяє чітко окреслити межі кожного функціонального блоку.

- Рольова модель: Побудова дерева функцій є основою для проєктування системи розмежування прав доступу (RBAC). Визначення того, які функції доступні для ролей «Адміністратор», «Менеджер» та «Виконавець», дозволяє гарантувати безпеку даних та уникнути ситуацій несанкціонованого втручання в критичні бізнес-процеси (наприклад, видалення проєкту виконавцем).
- Основа інтерфейсу: Структура дерева функцій у подальшому трансформується в навігаційну систему меню та логіку переходів між екранами застосунку.

2. Проєктування архітектури клієнтської та серверної частин На цьому етапі відбувається визначення технічної парадигми реалізації взаємодії між користувачем та системою. Для забезпечення високої інтерактивності та швидкодії обґрунтовується вибір триланкової архітектури з використанням принципів REST API [2, 6].

- Розділення відповідальності (Separation of Concerns): Такий підхід забезпечує чітке розмежування логіки обробки даних та бізнес-правил (Backend) від логіки їх візуалізації та взаємодії з користувачем (Frontend).
- Реактивність: Використання Single Page Application (SPA) підходу на клієнті є критично важливим для реалізації Kanban-дошки, де переміщення задач (Drag-and-Drop) та оновлення статусів має відбуватися миттєво, без повного перезавантаження сторінки.
- Універсальність: REST-архітектура дозволяє серверній частині виступати універсальним джерелом даних, що в майбутньому спростить розробку мобільної версії системи або інтеграцію зі сторонніми сервісами.

3. Вибір типу та засобів зберігання даних Це питання є ключовим для забезпечення цілісності, консистентності та довговічності інформації. Враховуючи специфіку предметної області, яка вимагає оперування чітко структурованими даними зі складними зв'язками (один проєкт — багато задач, один користувач — багато ролей), необхідно провести аналіз та вибір між реляційними (SQL) та нереляційними (NoSQL) базами даних [10].

- Цілісність даних: Вибір реляційної СУБД (наприклад, MySQL) дозволить використовувати механізми зовнішніх ключів та транзакцій, що є критично важливим для запобігання появи «сирітських» записів (наприклад, задача без виконавця або проєкту).
- Аналітичні можливості: Структурована модель даних значно спрощує виконання складних SQL-запитів для агрегації статистики, необхідної для побудови графіків продуктивності та звітів.

4. Проєктування моделі бази даних (концептуальне, логічне та фізичне представлення) Для успішної технічної реалізації системи важливо пройти повний шлях моделювання даних — від абстрактного опису сутностей до конкретних таблиць у СУБД.

- Концептуальний рівень: Визначення основних об'єктів системи (Користувач, Проєкт, Задача, Роль) та характеру зв'язків між ними (1:1, 1:N, M:N).
- Логічний рівень: Нормалізація бази даних для усунення надлишковості інформації, що дозволяє зменшити обсяг бази та пришвидшити операції оновлення даних.
- Фізичний рівень: Визначення типів даних для кожного атрибута, створення індексів для оптимізації пошукових запитів. Ефективна схема БД є запорукою швидкої роботи алгоритмів, зокрема унікального алгоритму автоматичного розподілу навантаження, який потребує миттєвого доступу до історії активності користувачів.

Таким чином, детальний розгляд та опрацювання вищезазначених питань дозволяє сформувати цілісне, системне бачення майбутнього програмного продукту та мінімізувати ризики виникнення архітектурних помилок на етапі кодування. Визначення функціонального складу, обґрунтування архітектурного підходу та ретельне моделювання даних закладають надійний фундамент інформаційного забезпечення. Це гарантує, що розроблена система не лише відповідатиме поставленим вимогам щодо автоматизації процесів планування та аналізу, але й буде надійною, зручною у використанні та готовою до подальшого

розвитку. Детальна реалізація кожного з окреслених етапів проєктування буде наведена у наступних підрозділах.

2.2 Дерево функцій системи

Побудова дерева функцій є фундаментальним етапом проєктування будь-якої складної інформаційної системи, що передує етапам моделювання бази даних та написання програмного коду. Цей процес базується на методі функціональної декомпозиції — послідовному розчленуванні головної мети системи на ієрархічну структуру підцілей та конкретних функцій. Такий підхід дозволяє трансформувати абстрактні вимоги замовника у чіткий перелік технічних задач, які підлягають реалізації.

У контексті розробки «Інформаційної системи для планування, координації та аналізу командної діяльності» дерево функцій виступає як "дорожня карта" проєкту. Воно дозволяє чітко розмежувати зони відповідальності між клієнтською та серверною частинами, визначити необхідні API-ендпоінти та спроєктувати інтерфейси користувача. Головна мета системи — підвищення ефективності командної роботи — є надто загальною для безпосередньої реалізації, тому вона декомпозується на п'ять ключових функціональних блоків: безпека, управління проєктами, координація задач, автоматизація бізнес-логіки та аналітика.

Функціональна модель визначає межі системи та описує сценарії взаємодії різних ролей користувачів (Адміністратора, Менеджера, Виконавця) з програмними модулями. На рисунку 2.1 наведено повне дерево функцій, яке охоплює всі аспекти роботи веб-застосунку.

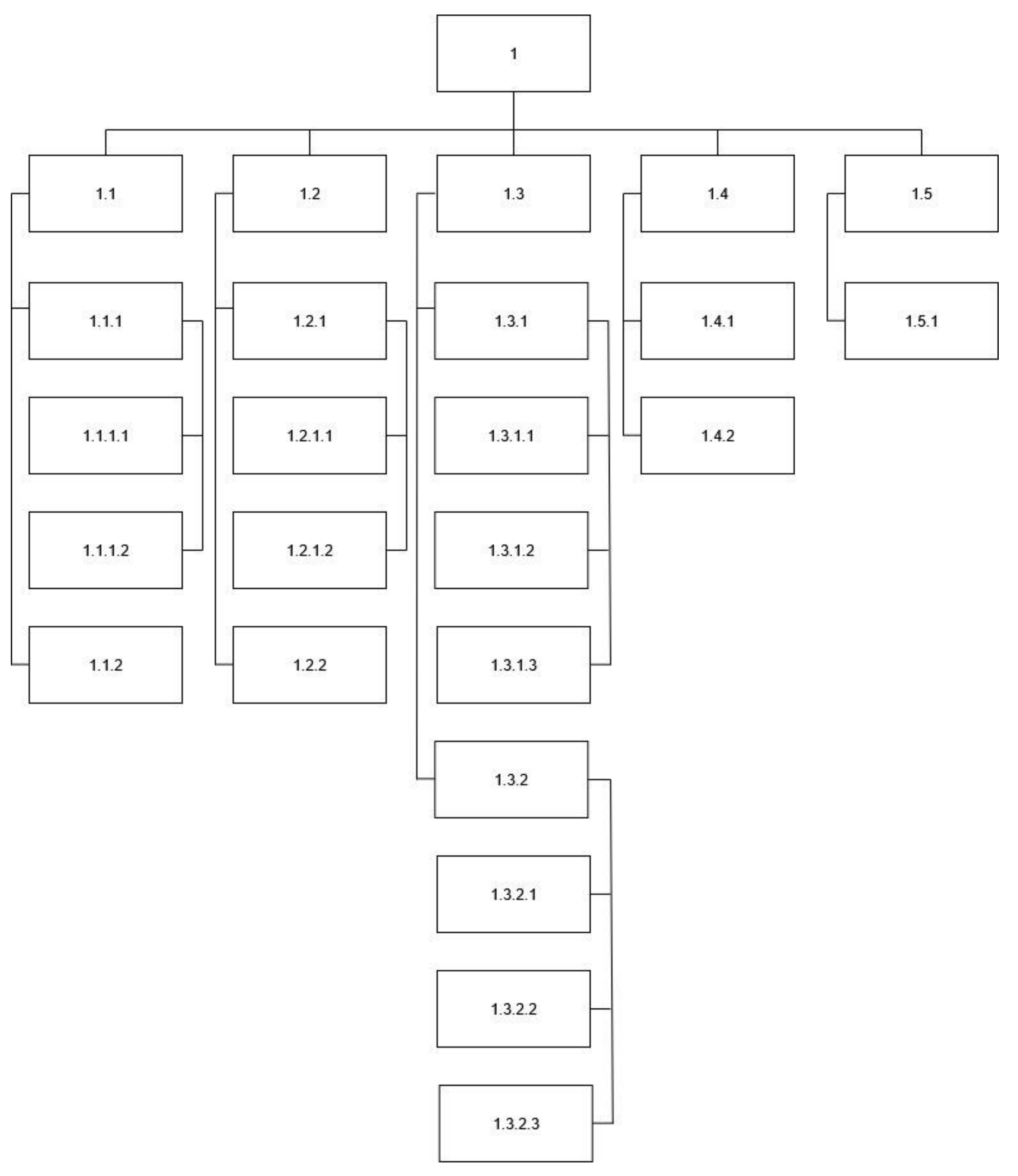

Рисунок 2.1 – Дерево функцій

Нижче наведено детальний опис кожного вузла дерева функцій з поясненням його призначення та логіки роботи.

Організація доступу та безпеки Цей модуль є "фундаментом" системи, що забезпечує захист даних та персоналізацію робочого простору.

1.1.1 Ідентифікація користувачів:

1.1.1.1 Реєстрація та автентифікація: Забезпечує створення нових облікових записів з валідацією унікальності email та складності пароля. Процес входу

включає перевірку хешу пароля (за допомогою бібліотеки bcrypt) та підтвердження особи.

1.1.1.2 Генерація токенів доступу: Реалізація механізму Stateless-авторизації. Після успішного входу сервер генерує JWT-токен, який підписується секретним ключем. Цей токен використовується клієнтським застосунком для підтвердження прав при кожному наступному HTTP-запиті.

1.1.2 Розмежування прав доступу за ролями: Система перевіряє роль користувача перед виконанням критичних дій. Це гарантує, що звичайний виконавець не зможе видалити проєкт або змінити глобальні налаштування.

1.2 Управління проєктним середовищем Модуль відповідає за організацію верхнього рівня ієрархії даних — контейнерів для задач (проєктів).

1.2.1 Адміністрування проєктів:

1.2.1.1 Створення та налаштування проєктів: Функціонал для менеджерів, що дозволяє ініціалізувати новий робочий простір, задати його назву, опис та часові рамки.

1.2.1.2 Розрахунок прогресу виконання: Сервер автоматично агрегує статистику (кількість виконаних задач до загальної кількості) і передає ці дані на клієнт для відображення прогрес-барів у списку проєктів.

1.2.2 Управління командою проєкту: Функції для додавання користувачів до конкретного проєкту та призначення їм локальних ролей. Це дозволяє формувати ізольовані робочі групи.

1.3 Координація виконання задач Центральний функціональний блок, що забезпечує основний робочий процес команди.

1.3.1 Оперативне управління:

1.3.1.1 Створення задач з дедлайнами: Можливість детального опису завдання з обов'язковим зазначенням граничного терміну виконання, що є базою для подальшої пріоритезації.

1.3.1.2 Візуалізація на Kanban-дошці: Графічне представлення задач у вигляді карток, розподілених по колонках («Planned», «In Progress», «Review», «Completed»).

1.3.1.3 Інтерактивна зміна статусів: Реалізація Drag-and-Drop інтерфейсу, що дозволяє змінювати статус задачі простим перетягуванням, з миттєвою синхронізацією змін у базі даних.

1.3.2 Автоматизація процесів: Унікальна бізнес-логіка системи.

1.3.2.1 Динамічний розрахунок пріоритетів: Алгоритм, який на клієнтській стороні порівнює поточну дату з дедлайном задачі. Якщо до завершення залишилося менше 48 годин, картка автоматично отримує візуальний індикатор високого пріоритету («Терміново»), якщо час вичерпано — статус «Прострочено».

1.3.2.2 Автоматичний розподіл навантаження (Auto-assign): Серверний алгоритм, який аналізує кількість активних задач у кожного учасника команди та розподіляє нові завдання («Planned») на виконавців з найменшим навантаженням.

1.3.2.3 Контроль правил бізнес-процесу (Review flow): Програмна заборона для виконавців переміщувати задачі зі статусу «Completed» назад або встановлювати цей статус без проходження етапу перевірки, що забезпечує контроль якості.

Аналіз результативності Модуль моніторингу, що надає інструменти для оцінки ефективності роботи.

1.4.1 Аналіз розподілу задач за статусами: Побудова кругової діаграми, що показує поточний стан проєкту (яка частка роботи виконана, а яка ще в планах).

1.4.2 Аналіз динаміки виконання робіт: Побудова стовпчастої діаграми, яка відображає кількість закритих задач у розрізі календарних днів (Velocity chart).

Адміністрування системи Спеціалізований блок функцій для супер-адміністратора.

1.5.1 Управління глобальними ролями користувачів: Інтерфейс для перегляду всіх зареєстрованих користувачів та зміни їхніх глобальних прав (наприклад, підвищення користувача до статусу Менеджера, що дає йому право створювати власні проєкти).

Побудоване дерево функцій демонструє, що розроблювана система є комплексним програмним продуктом, який поєднує стандартні механізми управління задачами (CRUD, Kanban) з інтелектуальними алгоритмами

автоматизації (Auto-assign, динамічні пріоритети). Така декомпозиція дозволила чітко визначити обсяг робіт для кожного етапу розробки.

Представлена функціональна модель є основою для подальшого проєктування структури бази даних (оскільки кожна функція вимагає відповідних сутностей для зберігання даних) та архітектури API (кожен листовий вузол дерева фактично відповідає одному або декільком програмним методам). Реалізація повного набору описаних функцій гарантує досягнення головної мети проєкту — створення ефективного інструменту для координації командної діяльності.

2.3 Аналіз та вибір програмних засобів реалізації

2.3.1 Вибір технологічного стеку серверної частини

Серверна частина (бекенд) інформаційної системи є її обчислювальним ядром, яке відповідає за обробку бізнес-логіки, управління потоками даних, автентифікацію користувачів та взаємодію з базою даних. Від архітектурної досконалості та правильного вибору технологічних засобів реалізації серверної частини безпосередньо залежать такі критичні характеристики системи, як продуктивність, здатність до масштабування (scalability), безпека та швидкість відгуку на запити клієнта.

Враховуючи специфіку проєктованої системи — «Інформаційна система для планування, координації та аналізу командної діяльності», — до серверної платформи висувається ряд специфічних вимог:

1. Висока швидкість обробки I/O запитів (Input/Output). Система передбачає інтенсивну взаємодію з базою даних: оновлення статусів задач при переміщенні на Kanban-дошці, створення нових проєктів, реєстрація користувачів. Сервер повинен ефективно обробляти велику кількість коротких асинхронних запитів одночасно, не створюючи черг.

2. Підтримка REST-архітектури. Необхідність побудови гнучкого, стандартизованого API для взаємодії з клієнтським SPA-застосунком (Single Page

Application). Сервер повинен виступати виключно постачальником даних у форматі JSON.

3. Уніфікація розробки (Isomorphic JavaScript). Використання єдиної мови програмування як на клієнті, так і на сервері дозволяє знизити когнітивне навантаження на розробника, використовувати спільні бібліотеки та спростити підтримку коду.

На основі детального аналізу цих вимог [37], для реалізації серверної частини було обрано платформу Node.js у зв'язці з веб-фреймворком Express.js.

Платформа Node.js Node.js — це програмна платформа, заснована на високопродуктивному рушії V8 (розробка Google) [38], що транслює JavaScript у машинний код. Вибір Node.js для даного дипломного проєкту обумовлений його ключовою архітектурною особливістю — подійно-орієнтованою моделлю (Event-driven architecture) [7] та неблокуючим вводом-виводом.

У традиційних серверних технологіях (наприклад, класичні підходи в PHP або Java) для кожного нового підключення клієнта часто створюється окремий потік (thread) або процес. Це призводить до значного споживання оперативної пам'яті та процесорного часу на перемикання контексту при великій кількості одночасних користувачів. Натомість Node.js працює в одному потоці, використовуючи нескінченний цикл подій (Event Loop).

Для системи управління проєктами це надає суттєві переваги:

- Ефективність при роботі з БД: Коли користувач переміщує задачу на дошці, сервер отримує запит і відправляє команду UPDATE в базу даних MySQL. Завдяки асинхронності, Node.js не "завмирає" в очікуванні відповіді від диску, а продовжує обробляти запити інших користувачів (наприклад, авторизацію). Коли база даних завершить запис, Node.js повернеться до цього запиту через механізм Promise або async/await.

- Нативна робота з JSON: Оскільки Node.js виконує JavaScript [20], серіалізація (перетворення об'єктів у рядок) та десеріалізацію (розбір рядка в об'єкт) даних у форматі JSON відбуваються максимально швидко та без додаткових бібліотек. Це критично важливо для REST API.

- Екосистема NPM: Доступ до найбільшого у світі репозиторію пакетів NPM (Node Package Manager) дозволяє інтегрувати готові, протестовані рішення для стандартних задач (безпека, робота з файлами, валідація), зосереджуючись на унікальній бізнес-логіці проєкту.

Веб-фреймворк Express.js Чистий Node.js надає лише низькорівневі інструменти для роботи з мережею. Для побудови повноцінної архітектури застосунку доцільно використати мікрофреймворк Express.js [13], який є де-факто стандартом у індустрії.

Ключові аргументи на користь Express.js у даному проєкті:

1. Потужна система маршрутизації: Express надає зручний інтерфейс для опису ендпоінтів API (Routes), таких як /kanban/:projectId або /create-task. Це дозволяє логічно структурувати застосунок, розділяючи обробку різних сутностей.
2. Архітектура Middleware (Проміжного ПЗ): Це одна з найсильніших сторін фреймворку. Обробка запиту відбувається через ланцюжок функцій. У проєкті це дозволило елегантно реалізувати систему безпеки: кожен запит спочатку проходить через cors (дозвіл доступу), потім через express.json (парсинг тіла запиту), далі через authenticateToken (перевірка валідності токена), і лише потім потрапляє до контролера.
3. Легковажність: Express не нав'язує жорсткої структури папок чи ORM, що надає гнучкість у виборі супутніх бібліотек.

Взаємодія з даними та безпека Для забезпечення зв'язку між Node.js та базою даних MySQL обрано драйвер mysql2 [33]. Його перевагою над стандартним драйвером є вбудована підтримка промісів (promise wrapper), що дозволяє використовувати сучасний синтаксис async/await для написання чистого та читабельного асинхронного коду [25, 26]. Крім того, бібліотека підтримує механізм Connection Pooling (пул з'єднань), що значно підвищує продуктивність шляхом повторного використання вже відкритих з'єднань з БД.

Для забезпечення безпеки на серверній стороні використовуються спеціалізовані бібліотеки:

- bcryptjs: Для криптографічного хешування паролів перед збереженням у базу даних, що унеможливлює їх викрадення навіть при компрометації БД.
- jsonwebtoken (JWT): Для реалізації механізму stateless-автентифікації [15], що є стандартом для REST API.

Висновок Проведений аналіз підтверджує, що обраний технологічний стек Node.js + Express + MySQL є оптимальним для вирішення поставлених задач. Асинхронна модель Node.js забезпечує високу продуктивність при роботі з численними запитами, фреймворк Express.js гарантує швидку розробку та гнучку архітектуру маршрутизації, а використання спеціалізованих бібліотек безпеки дозволяє створити захищений та надійний програмний продукт, що відповідає сучасним вимогам до веб-застосунків.

2.3.2 Архітектура клієнтського застосунку

Клієнтська частина системи проєктується як основне середовище взаємодії користувача з програмним комплексом. Враховуючи функціональні вимоги до системи, яка призначена для координації командної діяльності в режимі реального часу, інтерфейс повинен відповідати критеріям високої продуктивності, інтерактивності (responsiveness) та інтуїтивної зрозумілості. Проєктування цього рівня вимагає переходу від статичних веб-сторінок до побудови повноцінного програмного застосунку, що виконується у браузері.

Для реалізації поставлених вимог архітектуру клієнтського застосунку побудовано за моделлю SPA (Single Page Application) [8]. Такий підхід передбачає, що при першому зверненні до системи завантажується єдиний HTML-шаблон та пакет скриптів, а подальша взаємодія (навігація, фільтрація, оновлення даних) відбувається шляхом динамічної зміни вмісту DOM-дерева без повного перезавантаження сторінки. Це дозволяє забезпечити користувацький досвід, максимально наближений до роботи зі стаціонарним програмним забезпеченням, зменшуючи навантаження на мережу та прискорюючи відгук інтерфейсу.

Для реалізації рівня представлення обрано бібліотеку React.js [12]. Вибір цієї технології обумовлений використанням компонентного підходу [9], що дозволяє декомпозувати складний інтерфейс на набір незалежних, ізольованих модулів (компонентів), які можна повторно використовувати.

Ключовим архітектурним рішенням для оптимізації продуктивності є використання Virtual DOM. Проєктування механізму рендерингу базується на мінімізації прямих звернень до повільної об'єктної моделі браузера. Система спочатку вносить зміни у віртуальну копію структури сторінки в пам'яті, обчислює різницю станів (процес Reconciliation) і лише після цього точково оновлює реальний інтерфейс. Це є критично важливим для забезпечення плавності роботи інтерактивної дошки задач, де кількість елементів може бути значною.

Логічну структуру клієнтського застосунку організовано у вигляді деревоподібної ієрархії функціональних модулів, кожен з яких відповідає за окремий бізнес-процес:

1. Модуль ініціалізації та маршрутизації: Кореневий елемент архітектури, який відповідає за керування життєвим циклом застосунку. Його функціями є перевірка наявності токена авторизації при завантаженні та розподіл потоків навігації між публічною зоною (вхід/реєстрація) та захищеним робочим простором (Protected Routes).

2. Модуль управління проєктами: Проєктується для візуалізації портфеля проєктів. Архітектура модуля передбачає використання адаптивної сітки для відображення карток проєктів [36]. Також тут реалізується логіка модальних вікон для створення нових сутностей, що дозволяє користувачеві виконувати дії без втрати контексту.

3. Модуль візуалізації задач: Центральний компонент системи, архітектура якого базується на концепції сортувальних контейнерів. Передбачається створення динамічних колонок, що відповідають статусам життєвого циклу задачі (Planned, In Progress, Review, Completed). Кожна картка задачі є інтерактивним об'єктом, що містить метадані та елементи управління.

4. Модуль аналітики та звітності: Набір компонентів для графічної інтерпретації даних. Проєктується як ізольований шар, що працює в режимі відображення ("read-only"). Для реалізації діаграм використовується бібліотека Chart.js, яка інтегрується в React-компоненти через Canvas API, забезпечуючи високу швидкість рендерингу графіки.

Управління станом та потоками даних Для забезпечення синхронізації даних між різними модулями системи (наприклад, між дошкою задач та графіками аналітики) спроєктовано дворівневу модель управління даними:

Локальний стан: Для управління даними, що є специфічними для окремих форм (введення тексту, відкриття меню).

Глобальний контекст: Для передачі даних крізь ієрархію компонентів без створення жорстких зв'язків (уникнення проблеми "prop drilling"). Заплановано використання KanbanContext як єдиної "шини подій", що дозволяє оновлювати аналітичні віджети миттєво після зміни статусу задачі на дошці.

Архітектура Drag-and-Drop взаємодії Для реалізації функції переміщення задач проєктується підсистема на базі бібліотеки @dnd-kit, яка забезпечує абстракцію над нативними подіями браузера. Архітектура взаємодії включає:

- Сенсорний шар (Sensors): Для уніфікованої обробки подій від миші, клавіатури та сенсорних екранів.
- Алгоритми детекції колізій: Використання алгоритму closestCorners для точного визначення позиції картки відносно колонок у реальному часі.
- Оптимістичне оновлення інтерфейсу (Optimistic UI): Архітектурний патерн, що передбачає миттєву візуальну зміну стану системи (переміщення картки) до отримання підтвердження від сервера. Це дозволяє нівелювати затримки мережі та забезпечити відчуття миттєвої реакції інтерфейсу.

Проєктування мережевого шару Обмін даними з серверною частиною реалізується через асинхронні HTTP-запити з використанням бібліотеки Axios. Для підвищення безпеки та зручності підтримки, мережевий шар клієнта спроєктовано з використанням механізму перехоплення запитів (Interceptors). Цей патерн дозволяє:

1. Автоматично додавати JWT-токен авторизації до заголовків кожного вихідного запиту.

2. Централізовано обробляти помилки (наприклад, автоматично перенаправляти користувача на сторінку входу при отриманні статусу 401 Unauthorized).

Запропонована архітектура клієнтської частини поєднує гнучкість компонентного підходу React, ефективність глобального управління станом та надійність типізованих HTTP-запитів. Це створює міцну основу для реалізації функціональних вимог системи та забезпечує можливість подальшого масштабування інтерфейсу.

2.3.3 Огляд рішень для реалізації інтерактивної дошки та аналітики

При проєктуванні клієнтської частини системи управління проєктами виникає об'єктивна потреба в реалізації специфічних інтерфейсних рішень, складність яких виходить за межі стандартного функціоналу базових веб-фреймворків. Ключовими елементами взаємодії користувача з системою є інтерактивна Kanban-дошка, що передбачає складну логіку переміщення об'єктів, та панель аналітики для візуалізації статистичних даних.

Самостійна реалізація цих механізмів «з нуля» на чистому JavaScript (Vanilla JS) є нераціональною з точки зору витрат часу на розробку, тестування та забезпечення кросбраузерності. Тому в рамках проєктування інформаційного забезпечення необхідно провести аналіз існуючих Open Source рішень та обрати бібліотеки, що найкраще відповідають вимогам продуктивності, доступності та зручності підтримки коду.

Функціонал Drag-and-Drop («перетягни й кинь») є фундаментом користувацького досвіду (UX) при роботі з Kanban-дошкою. Технічно це комплексний процес, який включає відстеження координат вказівника, обробку подій натискання (як миші, так і сенсорних екранів), виявлення перетинів між

переміщуваною карткою та цільовою колонкою (алгоритми детекції колізій), а також анімацію переміщення об'єктів у DOM-дереві.

В екосистемі React на сьогоднішній день існує три домінуючі бібліотеки для вирішення цієї задачі: React-beautiful-dnd, React DnD та dnd-kit. Для обґрунтованого вибору було проведено їх порівняння, результати якого наведено в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз бібліотек Drag-and-Drop

Характеристика	React-beautiful-dnd	React DnD	dnd-kit
Розмір (minzipped)	~30 KB	~16 KB	~10 KB
Архітектура	Високорівнева (готова логіка списків)	Низькорівнева (потребує багато коду)	Модульна (хуки та контекст)
Підтримка сіток	Обмежена (тільки вертикальні/горизонтальні списки)	Повна (але складна в реалізації)	Повна (підтримує будь-яку верстку)
Доступність (A11y)	Хороша	Потребує ручного налаштування	Відмінна (вбудована підтримка клавіатури)
Сенсорні пристрої	Так	Потребує додаткового бекенду	Так (уніфіковані сенсори)
Продуктивність	Середня (важкий рендеринг)	Висока	Висока (мінімізація перемалювань)

На основі проведеного аналізу для проєктування системи обрано бібліотеку dnd-kit [16]. Вибір на користь dnd-kit зумовлений тим, що це найбільш сучасне та легковажне рішення, яке використовує актуальні можливості React (Hooks API).

- Гнучкість: На відміну від React-beautiful-dnd, яка накладає жорсткі обмеження на структуру DOM (працює лише зі списками), dnd-kit не диктує правил верстки. Вона надає лише "примітиви" (сенсори, модифікатори), дозволяючи розробнику створювати Kanban-дошки будь-якої складності, включаючи адаптивний дизайн.
- Продуктивність: Модульна архітектура дозволяє підключати лише необхідні алгоритми (наприклад, стратегію сортування rectSortingStrategy), що позитивно впливає на розмір кінцевого бандлу (Bundle size) та швидкість завантаження застосунку.
- Доступність: Бібліотека "з коробки" підтримує управління з клавіатури, що є важливою вимогою сучасних веб-стандартів (WAI-ARIA).

Вибір інструменту для візуалізації даних

Для реалізації модуля аналітики, який повинен відображати продуктивність команди (наприклад, розподіл статусів задач, динаміку виконання), необхідно обрати бібліотеку для побудови графіків. Головними вимогами є: легкість інтеграції з React, адаптивність (Responsive Design) та висока швидкість рендерингу при частому оновленні даних.

Розглянуто два концептуальні підходи до візуалізації у веб-середовищі:

1. SVG-бібліотеки (наприклад, Recharts). Вони рендерять графіки як набір векторних тегів <svg> у HTML. Це зручно для стилізації через CSS і забезпечує чіткість при масштабуванні. Однак, при великій кількості точок даних (тисячі елементів) це створює значне навантаження на DOM-дерево, що може призвести до "гальмування" сторінки.

2. Canvas-бібліотеки (наприклад, Chart.js). Вони малюють графік як єдине растрове зображення на елементі <canvas>. Цей підхід є набагато продуктивнішим, оскільки браузеру не потрібно відстежувати події для тисяч окремих елементів.

Для реалізації аналітичного модуля пропонується обрати Chart.js (разом з бібліотекою-обгорткою react-chartjs-2) [17]. Цей вибір обґрунтовується наступними факторами:

- Продуктивність: Використання технології HTML5 Canvas дозволяє миттєво перемальовувати графіки при надходженні нових даних через WebSocket або API, що критично важливо для системи реального часу.
- Простота впровадження: Бібліотека надає готові, естетично привабливі типи діаграм (Pie Chart для статусів, Bar Chart для історії), які не потребують від розробника глибоких знань математики чи комп'ютерної графіки, на відміну від низькорівневих інструментів на кшталт D3.js.
- Екосистема React: Наявність офіційного компонента-обгортки дозволяє працювати з графіками в декларативному стилі React, передаючи дані через пропси (Props), що відповідає загальній архітектурі проєкту.

Проведений аналіз дозволив сформувати оптимальний набір інструментів для вирішення специфічних інтерфейсних задач. Інтеграція dnd-kit забезпечить створення сучасної, швидкої та доступної Kanban-дошки, а використання Chart.js дозволить реалізувати потужну систему звітності без шкоди для продуктивності клієнтського застосунку. Обрані технології гармонійно вписуються в компонентну архітектуру React, забезпечуючи модульність, масштабованість та зручність подальшої підтримки коду.

2.4 Проєктування бази даних

2.4.1 Аналіз та вибір моделі організації даних

Ефективність функціонування будь-якої інформаційної системи значною мірою залежить від обраного способу зберігання та організації даних. База даних є фундаментом, на якому будується вся бізнес-логіка застосунку. Помилка на етапі вибору типу бази даних може призвести до проблем з цілісністю інформації, складнощів при масштабуванні системи та зниження швидкодії при виконанні пошукових запитів.

Для розробки системи планування та координації командної діяльності необхідно визначити модель даних, яка найкраще відповідає предметній області. Специфіка системи полягає в наявності чітко визначених сутностей (користувачі, проєкти, задачі) та, що найважливіше, наявності складних і жорстких зв'язків між ними.

У сучасній розробці веб-застосунків домінують два основні підходи до організації баз даних: нереляційний (NoSQL) та реляційний (SQL).

1. Нереляційна модель (NoSQL) Цей підхід (представлений такими системами як MongoDB, Cassandra) передбачає зберігання даних у вигляді документів (JSON-об'єктів), графів або пар «ключ-значення».

- Переваги: Висока гнучкість структури (можна додавати нові поля "на льоту"), легкість горизонтального масштабування.
- Недоліки для даного проєкту: Слабкі механізми забезпечення цілісності зв'язків. У системі управління проєктами критично важливо, щоб задача не могла існувати без проєкту, а користувач — без ролі. У NoSQL контроль таких зв'язків доводиться реалізовувати на рівні програмного коду, що збільшує ризик помилок.

2. Реляційна модель базується на представленні даних у вигляді двовимірних таблиць, які взаємодіють між собою за допомогою механізму зв'язків (ключів).

- Переваги: Сувора схема даних, підтримка стандарту SQL (Structured Query Language), гарантія цілісності даних, підтримка транзакцій (ACID).

Проаналізувавши вимоги до системи координації командної діяльності, для реалізації рівня зберігання даних обрано реляційну модель. Цей вибір обумовлений природою даних, що циркулюють у системі:

По-перше, структурованість даних. Інформація про задачу (назва, опис, дедлайн, статус) має чітку, незмінну структуру. На відміну від соціальних мереж або систем збору логів, де структура даних хаотична, у системі Kanban кожен

атрибут має фіксований тип. Реляційна модель дозволяє зафіксувати цю структуру на рівні схеми БД, що запобігає запису некоректних даних.

По-друге, складність зв'язків. Систему пронизують численні залежності:

- Один проєкт містить багато задач (зв'язок «Один-до-багатьох»).
- Один користувач може бути учасником багатьох проєктів, і в одному проєкті багато учасників (зв'язок «Багато-до-багатьох»).
- Задача має історію змін статусів (зв'язок «Один-до-багатьох»).

Реляційні бази даних оптимізовані для роботи з такими зв'язками. Використання зовнішніх ключів (Foreign Keys) дозволяє базі даних автоматично контролювати цілісність. Наприклад, якщо адміністратор видалить проєкт, СУБД може автоматично видалити (або заборонити видалення) всі пов'язані з ним задачі, запобігаючи появі «сирітських» записів.

По-третє, потреба в аналітичних вибірках. Однією з функцій системи є аналіз діяльності (побудова графіків). Мова SQL дозволяє виконувати складні операції агрегації (групування, підрахунок середнього, фільтрація) безпосередньо на стороні бази даних, що значно швидше, ніж обробка масивів даних у коді програми.

В якості конкретної реалізації реляційної моделі пропонується обрати СУБД MySQL [14]. Це рішення ґрунтується на наступних факторах:

1. Сумісність: MySQL має відмінну підтримку драйверів для платформи Node.js (обраної для серверної частини).
2. Продуктивність: Двигун зберігання InnoDB, що використовується в MySQL за замовчуванням, забезпечує надійну підтримку транзакцій та блокування на рівні рядків, що важливо для багатокористувацької роботи (коли кілька людей редагують дошку одночасно).
3. Поширеність: Це найпопулярніша Open Source база даних у світі, що гарантує наявність великої кількості документації та спільноти для вирішення потенційних проблем.

Таким чином, вибір реляційної моделі та СУБД MySQL є стратегічно правильним рішенням для системи з чіткою бізнес-логікою та високими вимогами

до узгодженості даних. Це забезпечить надійний фундамент для реалізації функціоналу управління проєктами, задачами та користувачами.

2.4.2 Концептуальна та логічна моделі даних

Проєктування бази даних є одним із найбільш критичних та ітераційних етапів розробки інформаційної системи, оскільки від якості спроєктованої структури даних залежить швидкодія застосунку, складність написання серверного коду та можливість подальшого масштабування системи. Цей процес складається з послідовного створення моделей різного рівня абстракції, що дозволяє плавно перейти від загального розуміння предметної області до конкретної технічної реалізації у середовищі СУБД MySQL.

У рамках розробки системи планування та координації діяльності передбачається створення двох рівнів моделей: концептуальної (інфологічної) та логічної (датологічної). Головною метою цього етапу є створення нормалізованої схеми, яка забезпечує цілісність даних, мінімізує їх надлишковість та підтримує виконання складних аналітичних запитів.

Концептуальна модель представляє собою високорівневий опис предметної області, незалежний від конкретної програмної реалізації. Її метою є визначення основних сутностей (об'єктів), якими оперує система, та встановлення семантичних зв'язків між ними, без деталізації атрибутів чи типів даних.

Для системи командної роботи було виділено наступні ключові сутності:

1. Користувач (User): Головний суб'єкт системи, який виконує дії, створює контент та взаємодіє з іншими учасниками.
2. Роль (Role): Сутність, що визначає набір повноважень та прав доступу (наприклад: Адміністратор, Менеджер, Виконавець).
3. Проєкт (Project): Ізольований робочий простір, що об'єднує групу користувачів та пул задач для досягнення спільної мети.
4. Задача (Task): Базова одиниця роботи, яка має життєвий цикл (статуси), пріоритет та часові обмеження (дедлайни).

На етапі концептуального проєктування встановлюються такі зв'язки між сутностями, що відображають бізнес-правила системи:

- Користувач — Роль: Зв'язок, що дозволяє динамічно призначати права. У даній системі реалізовано гнучкий підхід, де користувач може мати певну глобальну роль, а в межах проєктів — специфічні права.
- Користувач — Проєкт (M:N): Зв'язок типу «Багато-до-багатьох». Один користувач може бути учасником декількох проєктів одночасно, а кожен проєкт складається з команди учасників.
- Проєкт — Задача (1:N): Зв'язок типу «Один-до-багатьох». Проєкт виступає контейнером, що містить безліч задач, проте кожна конкретна задача належить виключно одному проєкту.
- Задача — Користувач (N:1): Зв'язок типу «Багато-до-одного». Задача може бути призначена конкретному виконавцю. Це забезпечує персональну відповідальність за виконання роботи.

На рисунку 2.2 графічно зображено концептуальну модель, що демонструє взаємозв'язки між основними інформаційними об'єктами.

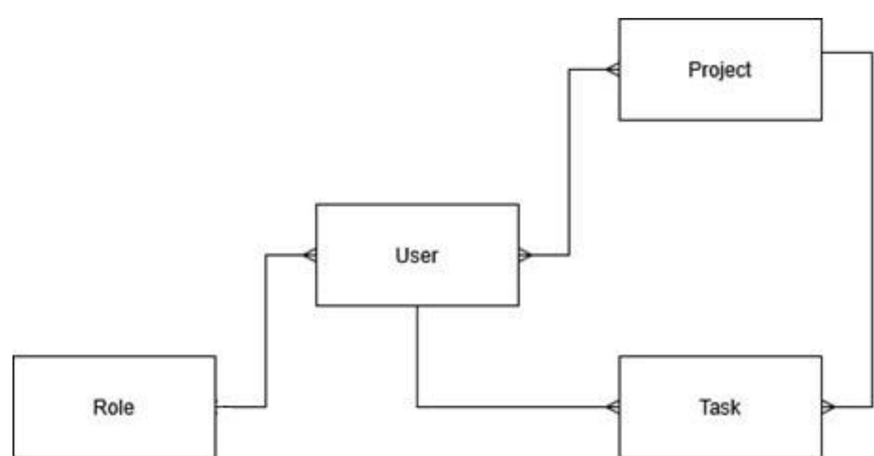

Рисунок 2.2 - Концептуальна модель представлення бази даних

Логічна модель є технічною деталізацією концептуальної схеми з урахуванням специфіки реляційної моделі даних та правил нормалізації [40]. На цьому етапі абстрактні зв'язки «Багато-до-багатьох» розкриваються через проміжні таблиці,

визначаються первинні та зовнішні ключі, а також формується повний перелік атрибутів для кожної сутності з визначенням типів даних. В результаті нормалізації було спроєктовано наступну структуру таблиць бази даних:

1. Основні таблиці:

- Users: Зберігає ідентифікаційні дані (username, email) та хеш пароля (password_hash) для забезпечення безпеки.

- Projects: Містить метадані проєкту (name, description, deadline) та посилання на творця (created_by).

- Tasks: Центральна таблиця, що зберігає зміст задачі, її пріоритет, дедлайн та посилання на виконавця (assigned_to) і проєкт (project_id). Поле status реалізовано як ENUM для фіксації етапів життєвого циклу.

2. Таблиці зв'язків (Linking Tables):

- UserRoles: Асоціативна таблиця, що розкриває зв'язок між користувачами та ролями. Вона дозволяє призначати користувачу роль як глобально, так і в межах конкретного проєкту (через поле project_id), що забезпечує гнучку модель доступу (RBAC).

- ProjectMembers: Таблиця для реалізації зв'язку «Багато-до-багатьох» між проєктами та користувачами, що формує склад команди.

3. Спеціалізовані технічні таблиці:

- KanbanOrder: Критично важлива таблиця для функціонування інтерфейсу Kanban-дошки. Вона зберігає позицію (position) кожної задачі в колонці. Це дозволяє реалізувати функціонал Drag-and-Drop без необхідності змінювати порядок записів у головній таблиці Tasks, що значно підвищує продуктивність.

- TaskStatusHistory: Таблиця для накопичення історичних даних. Вона фіксує кожну зміну статусу задачі, дату зміни (changed_at) та ініціатора. Саме на основі даних цієї таблиці будуються аналітичні графіки продуктивності команди.

На рисунку 2.3 наведено розгорнуту логічну схему бази даних (ER-діаграму) з відображенням усіх атрибутів та типів зв'язків.

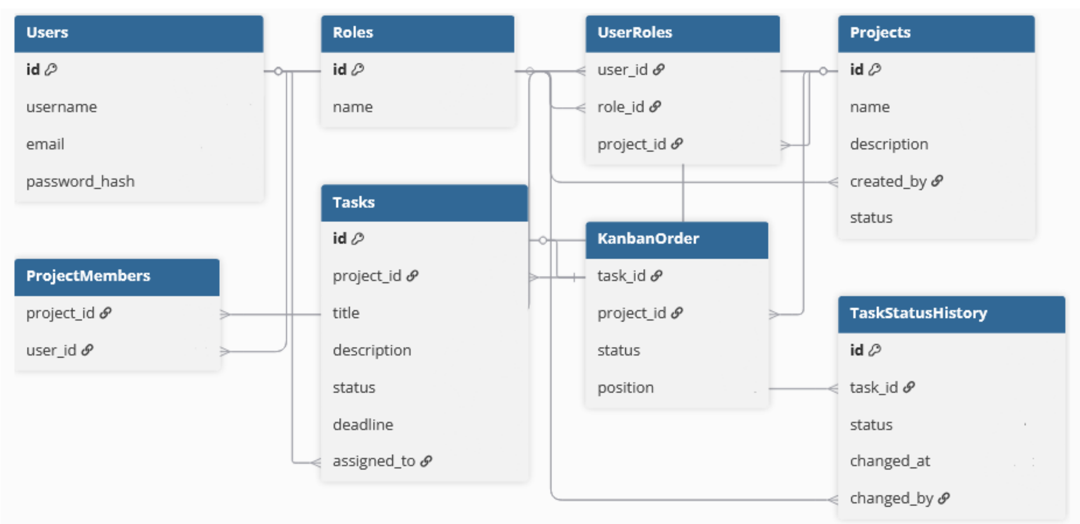

Рисунок 2.3 - Концептуальна модель представлення бази даних

Розроблена модель бази даних повністю відповідає функціональним вимогам системи. Вона забезпечує надійне зберігання даних без надлишковості завдяки нормалізації, підтримує складні механізми розмежування доступу та надає необхідну структуру для реалізації бізнес-логіки, включаючи інтерактивну Kanban-дошку та модуль аналітики. Спроектована схема є гнучкою та дозволяє масштабувати систему в майбутньому без необхідності докорінної перебудови архітектури даних.

2.4.3 Фізична модель та типи даних

Фізична модель даних є фінальним етапом проєктування бази даних, на якому абстрактні зв'язки логічної моделі трансформуються у конкретні технічні рішення, адаптовані під обрану систему управління базами даних (СУБД). Для реалізації системи планування та координації командної діяльності обрано СУБД MySQL з використанням рушія зберігання даних InnoDB, який забезпечує підтримку транзакцій та контроль цілісності зовнішніх ключів.

Фізична модель визначає типи даних для кожного атрибута, правила валідації (NOT NULL, DEFAULT), налаштування автоінкременту для первинних ключів та індексацію для оптимізації пошукових запитів.

Графічне представлення фізичної моделі, що відображає структуру таблиць, типи даних та зв'язки між ними, наведено на рисунку 2.3.

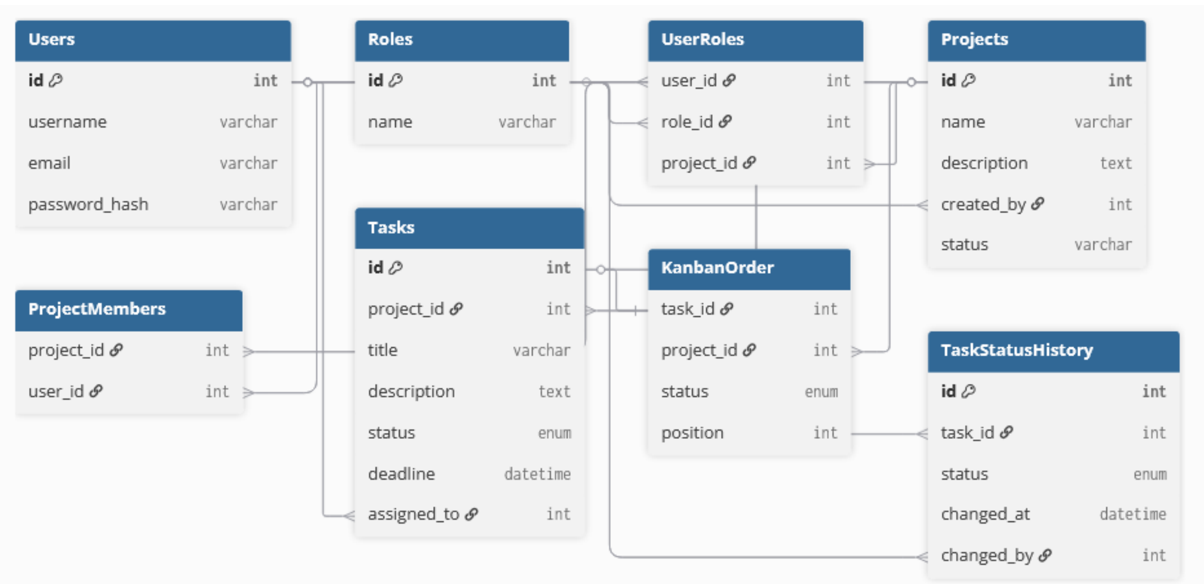

Рисунок 2.3 — Фізична модель бази даних системи

База даних системи складається з 8 взаємопов'язаних таблиць. Нижче наведено детальний опис кожної таблиці, призначення її полів та обраних типів даних.

1. Таблиця Users (Користувачі) Призначена для зберігання облікових даних користувачів системи.

- id (INT, PK, AUTO_INCREMENT) — унікальний ідентифікатор користувача.
- username (VARCHAR(255), NOT NULL) — ім'я користувача для відображення в інтерфейсі.
- email (VARCHAR(255), UNIQUE, NOT NULL) — електронна пошта, що використовується як логін. Унікальний індекс запобігає повторній реєстрації.

- password_hash (VARCHAR(255), NOT NULL) — хеш пароля (замість відкритого пароля), що забезпечує безпеку.

2. Таблиця Roles (Ролі) Довідник доступних ролей у системі.

- id (INT, PK, AUTO_INCREMENT) — ідентифікатор ролі.
- name (VARCHAR(50), NOT NULL) — назва ролі (наприклад, 'Administrator', 'Project Manager', 'Executor').

3. Таблиця UserRoles (Ролі користувачів) Проміжна таблиця для реалізації зв'язку «Багато-до-багатьох» між користувачами та ролями. Дозволяє налаштовувати права доступу.

- user_id (INT, FK) — посилання на користувача.
- role_id (INT, FK) — посилання на роль.
- project_id (INT, FK, NULLABLE) — посилання на проєкт. Якщо поле містить NULL, роль вважається глобальною (наприклад, адміністратор всієї системи). Якщо вказано ID проєкту — роль діє лише в межах цього проєкту.

4. Таблиця Projects (Проєкти) Зберігає метадані про робочі простори.

- id (INT, PK, AUTO_INCREMENT) — ідентифікатор проєкту.
- name (VARCHAR(100), NOT NULL) — назва проєкту.
- description (TEXT) — детальний опис мети та задач проєкту. Використано тип TEXT для зберігання великих обсягів інформації.
- created_by (INT, FK) — посилання на користувача, який створив проєкт (власника).
- status (ENUM('Active', 'Archived'), DEFAULT 'Active') — стан проєкту.

5. Таблиця ProjectMembers (Учасники проєкту) Визначає склад команди для конкретного проєкту.

- project_id (INT, FK) — посилання на проєкт.
- user_id (INT, FK) — посилання на користувача.
- Обмеження: Композитний унікальний ключ (project_id, user_id) гарантує, що користувач не може бути доданий в один проєкт двічі.

6. Таблиця Tasks (Задачі) Основна сутність системи, що містить інформацію про заплановані роботи.

- id (INT, PK, AUTO_INCREMENT) — ідентифікатор задачі.
- project_id (INT, FK) — проєкт, до якого належить задача.
- title (VARCHAR(255), NOT NULL) — короткий зміст задачі.
- description (TEXT) — повний опис завдання.
- status (ENUM('Planned', 'In Progress', 'Review', 'Completed'), DEFAULT 'Planned') — поточний етап виконання. Використання ENUM забезпечує цілісність даних, обмежуючи набір можливих значень.
- deadline (DATETIME) — граничний термін виконання.
- assigned_to (INT, FK, NULLABLE) — виконавець задачі. Може бути NULL, якщо задача ще нікому не призначена.

7. Таблиця KanbanOrder (Порядок відображення) Технічна таблиця для збереження позицій карток на дошці (функціонал Drag-and-Drop).

- task_id (INT, FK) — посилання на задачу.
- project_id (INT, FK) — посилання на проєкт (для швидкої фільтрації).
- status (ENUM) — колонка, в якій знаходиться задача.
- position (INT) — порядковий номер задачі у колонці.

8. Таблиця TaskStatusHistory (Історія змін) Журнальна таблиця для побудови аналітичних звітів та графіків продуктивності.

- id (INT, PK, AUTO_INCREMENT) — ідентифікатор запису історії.
- task_id (INT, FK) — посилання на задачу.
- status (ENUM) — новий статус, який отримала задача.
- changed_at (DATETIME, DEFAULT CURRENT_TIMESTAMP) — час зміни статусу.
- changed_by (INT, FK) — користувач, який здійснив зміну.

Спроєктована фізична модель даних повністю відповідає вимогам системи. Використання типів ENUM забезпечує валідацію даних на рівні СУБД, а система зовнішніх ключів гарантує посилальну цілісність, запобігаючи появі некоректних записів. Така структура бази даних є оптимальною для забезпечення швидкодії веб-застосунку та коректної роботи алгоритмів аналітики.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Ініціалізація та конфігурація середовища розробки

3.1.1 Створення проєкту та встановлення залежностей

Процес розробки програмного забезпечення починається з формування надійного фундаменту — ініціалізації робочого середовища. Враховуючи, що проєктована система планування та координації командної діяльності є комплексним веб-застосунком, який складається з клієнтської та серверної частин, процес ініціалізації вимагав створення двох незалежних програмних оточень, об'єднаних у єдину структуру (монорепозиторій).

Першим етапом стало розгортання серверної частини, яка відповідає за бізнес-логіку та взаємодію з базою даних. Для цього було створено директорію server. Оскільки в якості платформи виконання обрано Node.js, ініціалізація проєкту виконувалася за допомогою стандартної утиліти пакетного менеджера командою npm init.

В результаті виконання цієї команди було згенеровано файл конфігурації package.json. Цей файл відіграє роль центрального реєстру проєкту: він містить метадані застосунку, скрипти для автоматизації запуску, а також список залежностей. Саме через цей механізм забезпечується стабільність роботи системи на різних машинах — розробнику достатньо виконати команду встановлення, щоб отримати ідентичне середовище.

Для реалізації функціональних вимог, визначених у другому розділі, до серверного проєкту було підключено ряд ключових бібліотек:

1. Express.js: Для обробки HTTP-запитів було встановлено цей мікрофреймворк. Вибір на його користь зроблено через необхідність побудови REST API. Express бере на себе низькорівневу роботу з мережевими протоколами, дозволяючи зосередитися на маршрутизації — визначенні того, як система має реагувати на запити типу «отримати список задач» або «створити проєкт».

2. mysql2: Для взаємодії з системою управління базами даних MySQL було обрано саме цю версію драйвера. Ключовою причиною вибору стала вбудована

підтримка промісів. Це дозволило використовувати в коді сучасні конструкції async/await, що робить логіку складних транзакцій (наприклад, при автоматичному розподілі задач) лінійною, зрозумілою та захищеною від помилок асинхронності («Callback Hell»).

3. Cors, Dotenv, Bcryptjs: Для забезпечення безпеки та налаштування оточення було підключено допоміжні модулі. cors вирішує проблему політики єдиного походження браузерів [32], дозволяючи клієнту з одного порту звертатися до сервера на іншому. dotenv забезпечує захист конфіденційних даних (паролів БД), виносячи їх у змінні середовища. bcryptjs реалізує криптографічне хешування паролів користувачів [21], що є обов'язковим стандартом безпеки .

Наступним кроком стало розгортання клієнтської частини в директорії client. Оскільки ручне налаштування сучасного JavaScript-оточення (Webpack, Babel, лінтери) є трудомістким процесом [41], було використано інструмент автоматизації create-react-app.

Ця утиліта розгорнула готову до роботи структуру Single Page Application (SPA). Вона автоматично налаштувала "гаряче перезавантаження" (Hot Reloading), що дозволяє бачити зміни в коді миттєво без оновлення сторінки, та створила оптимізований конвеєр збірки проєкту для продакшну.

Для реалізації специфічного інтерфейсу системи командної роботи базового функціоналу React було недостатньо, тому було інтегровано спеціалізовані бібліотеки:

1. Axios: Для організації обміну даними з сервером [18]. На відміну від стандартного методу fetch, Axios дозволяє налаштувати глобальні перехоплювачі. У проєкті це використовується для автоматичного додавання токена авторизації до кожного запиту, що позбавляє необхідності дублювати код аутентифікації в кожному компоненті.

2. @dnd-kit/core: Для реалізації інтерактивної дошки задач. Ця бібліотека забезпечує математичну базу для обробки подій перетягування (Drag-and-Drop) [16]. Вона розраховує координати курсора, визначає зони колізій та забезпечує доступність інтерфейсу для сенсорних екранів.

3. Chart.js та react-chartjs-2: Для модуля аналітики. Бібліотека використовує HTML5 Canvas для швидкого малювання графіків [17]. Це дозволяє візуалізувати дані про продуктивність команди (кругові діаграми статусів, гістограми виконаних задач) без суттєвого навантаження на браузер.

Таким чином, на етапі ініціалізації було сформовано повноцінну екосистему розробки, де серверна частина оптимізована для швидкої обробки запитів, а клієнтська — для побудови насиченого інтерактивного інтерфейсу.

3.1.2 Опис файлової архітектури проєкту

У сучасному процесі розробки програмного забезпечення ефективність подальшої підтримки, тестування та масштабування програмного продукту перебуває у прямій залежності від якості та логічності організації його файлової структури. Хаотичне розміщення файлів неминуче призводить до зростання технічного боргу та ускладнює введення нових розробників у курс справ. Саме тому в розробленій системі було застосовано модульний архітектурний підхід, який передбачає суворе розмежування програмного коду за його функціональним призначенням та зонами відповідальності.

Проєкт організовано як єдиний репозиторій (Monorepo-like structure) [35], внутрішній простір якого логічно розділений на дві незалежні підсистеми: server (серверна частина, що відповідає за бізнес-логіку та роботу з даними) та client (клієнтська частина, що відповідає за інтерфейс користувача). Такий поділ дозволяє чітко відокремити фронтенд від бекенду, спрощуючи процеси розгортання та адміністрування.

Файлова структура серверу спроєктована з дотриманням принципів «чистої архітектури» [2, 22], головною метою якої є відокремлення налаштувань інфраструктури від безпосередньої бізнес-логіки застосунку. Це дозволяє змінювати конфігурацію бази даних або налаштування сервера, не зачіпаючи основні алгоритми обробки даних.

На рисунку 3.1 схематично зображено ієрархію файлової структури серверної частини проєкту.

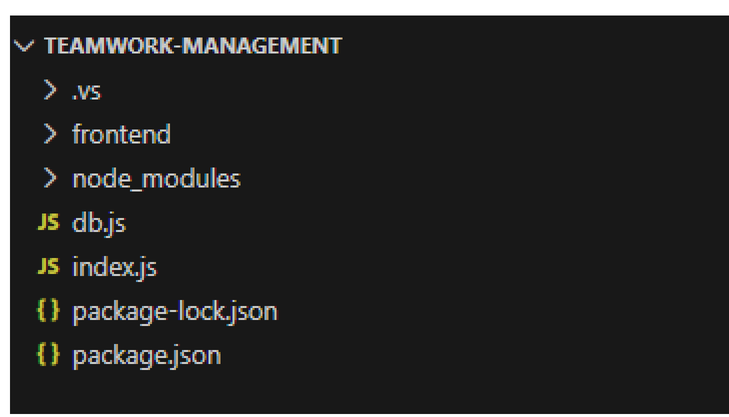

Рисунок 3.1 — Файлова структура серверної частини

Детальний опис ключових елементів архітектури сервера:

- Точка входу (index.js): Цей файл виступає центральним вузлом (Entry Point) усього бекенд-застосунку. Саме тут відбувається ініціалізація екземпляра Express-сервера, налаштування глобальних обробників помилок, запуск прослуховування мережевого порту та, що найважливіше, підключення проміжного програмного забезпечення (Middleware), такого як cors для налаштування політики доступу та парсерів JSON.
- Конфігурація даних (db.js): Окремий, ізольований модуль, що відповідає виключно за встановлення та підтримку з'єднання з реляційною базою даних. У ньому реалізовано програмний патерн "Singleton" для пулу з'єднань MySQL. Таке рішення гарантує ефективне використання системних ресурсів, запобігаючи створенню зайвих підключень при кожному запиті, що є критичним для високоінтерактивних систем.
- Маршрутизація та контролери: Логіка обробки вхідних HTTP-запитів не зосереджена в одному місці, а згрупована за відповідними сутностями предметної області. Наприклад, маршрути, що стосуються процесів автентифікації (вхід, реєстрація), відокремлені від маршрутів, що відповідають за управління

життєвим циклом задач. Така структуризація дозволяє розробнику легко орієнтуватися в кодовій базі [23, 24]: у разі необхідності зміни алгоритму створення задачі, спеціаліст точно знає, що відповідний блок коду знаходиться у контролері задач, а не в загальному файлі.

Структура клієнтського застосунку, розташована в директорії src, є більш розгалуженою та складною, оскільки вона відповідає за візуалізацію інтерфейсу та управління станом користувача. Вона побудована відповідно до кращих практик екосистеми React.

На рисунку 3.2 наведено детальну схему файлової структури клієнтської частини проєкту.

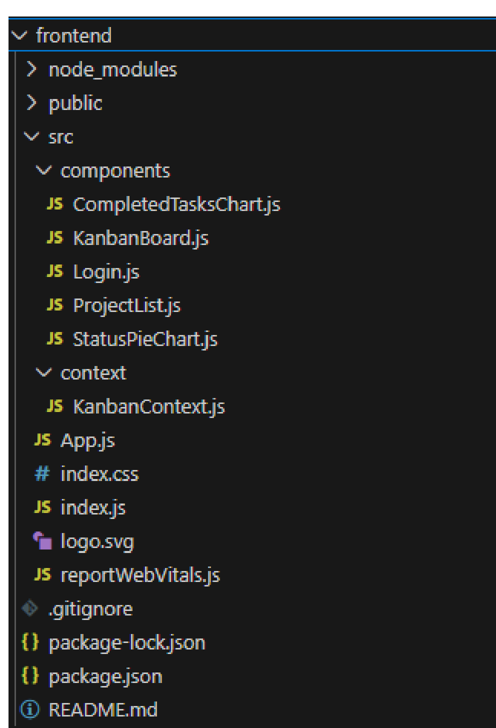

Рисунок 3.2 — Файлова структура клієнтської частин

Ключові директорії та їх функціональне призначення:

1. src/components (Бібліотека компонентів): У цій директорії зберігаються всі візуальні елементи системи (UI-компоненти). Архітектурно вони поділені на

логічні блоки, кожен з яких відповідає за відображення конкретної частини інтерфейсу:

- KanbanBoard.js: Найбільш складний компонент, що інкапсулює логіку інтерактивної дошки, управління станами колонок, реалізацію Drag-and-Drop взаємодії та візуалізацію карток завдань.
- ProjectList.js: Відповідає за відображення дашборду з переліком доступних проєктів, а також містить логіку модальних вікон для створення нових сутностей.
- Login.js: Ізольований модуль, що реалізує інтерфейс форми авторизації та обробку вхідних даних користувача.
- StatusPieChart.js, CompletedTasksChart.js: Спеціалізовані компоненти для візуалізації статистичних даних у вигляді графіків та діаграм.

2. src/context (Управління глобальним станом): Директорія, призначена для реалізації механізмів React Context API. Файл KanbanContext.js забезпечує створення глобального сховища даних, що дозволяє організувати обмін інформацією між глибоко вкладеними компонентами (наприклад, між дошкою задач та графіками) без необхідності передавати пропси через усю ієрархію компонентів (уникнення проблеми "Prop Drilling").

3. src/App.js (Маршрутизація та композиція): Кореневий компонент вищого рівня, який визначає загальну структуру навігації застосунку. Він відповідає за умовний рендеринг сторінок залежно від стану авторизації та захищає приватні маршрути від несанкціонованого доступу.

Підсумовуючи вищевикладене, можна стверджувати, що обрана стратегія організації файлової системи створює надійний фундамент для всього проєкту. Модульність та чітке розділення відповідальності дозволяють команді розробників працювати над різними частинами проєкту паралельно, мінімізуючи ризики виникнення конфліктів при злитті коду (Merge Conflicts). Крім того, така архітектура забезпечує високу гнучкість системи, гарантуючи легкість інтеграції нового функціоналу та масштабування в майбутньому без необхідності суттєвого рефакторингу існуючої кодової бази.

3.2 Реалізація серверної частини

3.2.1 Розробка механізмів авторизації та Middleware

Серверна частина системи виступає центральним елементом архітектури, що забезпечує не лише обробку бізнес-логіки та взаємодію з базою даних, але й гарантує цілісність та конфіденційність інформації. У контексті розробки системи для командної роботи питання безпеки набувають критичного значення, адже компрометація даних може призвести до витоку комерційної інформації про проєкти. Реалізація серверної частини виконана на базі платформи Node.js з використанням мікрофреймворку Express, що дозволило створити гнучку та масштабовану структуру API. Основна логіка маршрутизації та конфігурації сервера зосереджена у файлі index.js.

Фундаментом безпеки будь-якої інформаційної системи є два взаємопов'язані процеси:

1. Автентифікація (Authentication): Процедура перевірки справжності користувача шляхом порівняння введених облікових даних (логіна та пароля) з даними, що зберігаються в системі. Фактично, це відповідь на запитання: «Ким є цей користувач?».
2. Авторизація (Authorization): Процес визначення прав доступу вже автентифікованого користувача до певних ресурсів або дій. Це відповідь на запитання: «Що цьому користувачеві дозволено робити?».

Оскільки розроблювана система базується на архітектурі REST API, яка за своєю природою є Stateless (без збереження стану), сервер не зберігає інформацію про сесію користувача між запитами в оперативній пам'яті. Замість класичних сесій (cookies + session ID), для забезпечення безпеки та підтримки стану "залогіненого" користувача було реалізовано механізм на основі JSON Web Tokens (JWT). Цей стандарт дозволяє передавати інформацію про користувача в зашифрованому вигляді безпосередньо в HTTP-запитах [15], що забезпечує високу масштабованість системи.

Маршрут /login, визначений у файлі index.js, слугує єдиною точкою входу для процедури автентифікації. Алгоритм обробки запиту на вхід спроєктовано з урахуванням сучасних вимог до кібербезпеки:

1. Отримання та валідація даних: Сервер приймає POST-запит, тіло якого містить email та password. На цьому етапі відбувається первинна перевірка наявності обох полів, що запобігає обробці некоректних запитів.

2. Безпечний пошук користувача: Виконується SQL-запит до таблиці Users. Критично важливим аспектом є використання параметризованих запитів (через бібліотеку mysql2), де вхідні дані передаються окремо від тіла запиту (через знак ?). Це повністю нівелює загрозу SQL-ін'єкцій — одного з найпоширеніших типів атак на веб-застосунки [19]. Якщо користувача з вказаною поштою не знайдено, сервер негайно повертає статус 401 Unauthorized [31].

3. Криптографічна перевірка пароля: Паролі в базі даних ніколи не зберігаються у відкритому вигляді. Натомість зберігається їх хеш — незворотне перетворення рядка фіксованої довжини. Для перевірки використовується бібліотека bcryptjs (метод compare). Цей алгоритм автоматично додає до пароля випадковий рядок ("сіль") перед хешуванням, що робить неможливим використання райдужних таблиць (rainbow tables) для злому. Метод порівнює хеш введеного пароля зі збереженим у БД хешем.

4. Визначення контексту безпеки (Ролі): Після успішної верифікації пароля система виконує запит до таблиць UserRoles та Roles (з використанням JOIN), щоб визначити поточний рівень привілеїв користувача (наприклад, 'Administrator', 'Project Manager' або 'Executor'). Ця інформація є необхідною для подальшої генерації токена.

5. Генерація JWT-токена: За допомогою бібліотеки jsonwebtoken створюється цифровий ключ доступу. Структура токена включає Header (тип алгоритму), Payload (корисне навантаження) та Signature (цифровий підпис). У Payload зашиваються id, email та role. Токен підписується секретним ключем (JWT_SECRET), який зберігається виключно в змінних середовища сервера (.env) і недоступний ззовні. Це гарантує, що клієнт не зможе підробити токен.

6. Відповідь клієнту: Сервер повертає HTTP-відповідь зі статусом 200 OK, що містить згенерований токен та об'єкт користувача (без пароля). Клієнтська частина зобов'язана зберегти цей токен (зазвичай у localStorage) для подальших запитів.

Графічно процес автентифікації зображено на діаграмі послідовності (рисунок 3.3).

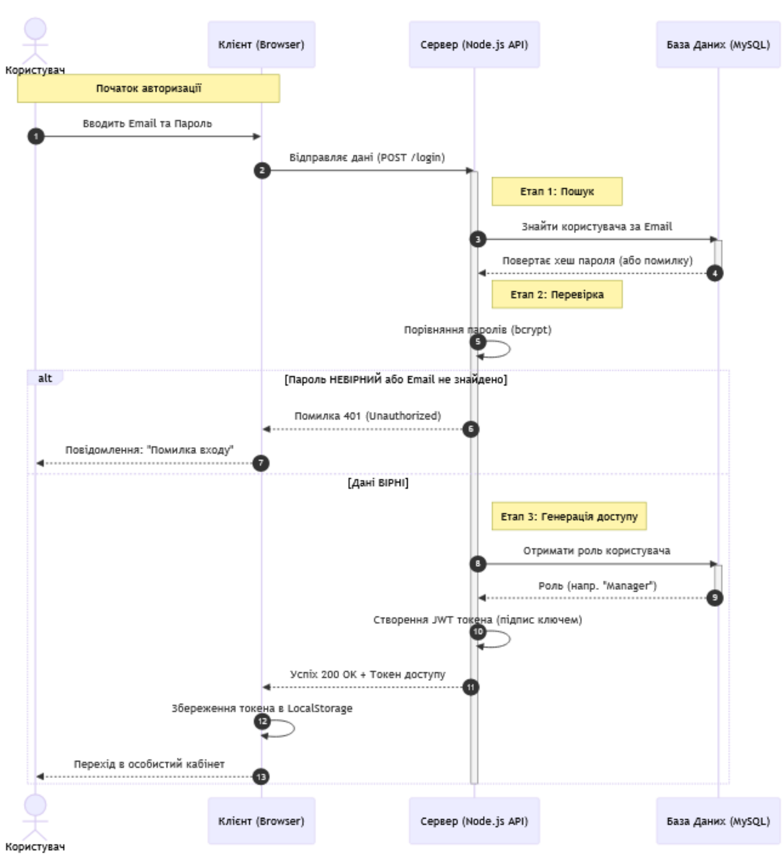

Рисунок 3.3 — Діаграма послідовності процесу аутентифікації користувача

Для захисту приватних маршрутів API та дотримання принципу DRY (Don't Repeat Yourself) було розроблено функції проміжної обробки запитів — Middleware. Ці функції перехоплюють HTTP-запит до того, як він досягне контролера бізнес-логіки, дозволяючи централізовано керувати безпекою (ДОДАТОК А).

Функція authenticateToken Ця функція є головним бар'єром безпеки, що фільтрує всі захищені запити. Алгоритм її роботи:

1. Вилучення токена: Функція зчитує заголовок Authorization з вхідного запиту. Згідно зі стандартами, токен передається у форматі Bearer [TOKEN].

2. Валідація наявності: Якщо заголовок відсутній, запит миттєво відхиляється зі статусом 401 (Неавторизовано), оскільки анонімний доступ до API заборонено.

3. Верифікація підпису: За допомогою методу jwt.verify сервер перевіряє валідність цифрового підпису токена, використовуючи секретний ключ. Цей етап гарантує цілісність даних (токен не було змінено третьою стороною) та перевіряє термін його дії (expiration time).

4. Ін'єкція контексту користувача: У разі успішної перевірки, розшифровані дані з Payload (ID, роль) записуються в об'єкт запиту req.user. Це критично важливо, оскільки дозволяє будь-якому наступному контролеру точно знати, хто ініціював запит, без повторних звернень до БД.

5. Передача управління: Виклик функції next() передає оброблений запит далі по ланцюжку middleware до кінцевого обробника маршруту.

Функція requireRole Ця функція реалізує модель управління доступом на основі ролей (RBAC — Role-Based Access Control). Технічно вона є функцією вищого порядку (Higher-Order Function), що приймає масив дозволених ролей і повертає middleware-функцію. Логіка роботи:

1. Перевіряє об'єкт req.user, який був сформований на попередньому етапі функцією authenticateToken.

2. Порівнює роль поточного користувача зі списком дозволених ролей, переданим як аргумент (наприклад, ['Administrator', 'Project Manager']).

3. Якщо роль користувача не знайдена у списку дозволених, запит блокується зі статусом 403 Forbidden (Доступ заборонено). Важливо розрізняти помилки: 401 означає "ми не знаємо, хто ви", а 403 — "ми знаємо, хто ви, але вам сюди не можна". Це гарантує, що навіть авторизований виконавець не зможе

виконати адміністративні дії, такі як видалення проєкту чи зміна ролей інших користувачів.

Реалізована схема авторизації забезпечує високий рівень безпеки та гнучкості системи. Використання JWT дозволило створити stateless-архітектуру, що легко масштабується, оскільки серверу не потрібно витрачати ресурси на зберігання сесій. Впровадження Middleware дозволило відокремити логіку безпеки від бізнес-логіки, зробивши код чистішим та простішим у підтримці. Комбінація authenticateToken та requireRole створює надійний ешелонований захист, гарантуючи, що кожен запит до API проходить сувору перевірку автентичності та повноважень перед виконанням.

3.2.2 Реалізація REST API для управління проєктами та задачами

Взаємодія між клієнтським інтерфейсом (Frontend) та сервером (Backend) побудована за принципами архітектури REST (Representational State Transfer) [5, 6]. Сервер надає набір уніфікованих кінцевих точок (endpoints), кожна з яких відповідає за виконання певної операції над ресурсами системи: проєктами, задачами або користувачами. Обмін даними здійснюється у форматі JSON, що є стандартом для сучасних веб-застосунків. Використання семантичних методів HTTP (GET для отримання, POST для створення, PUT для оновлення, DELETE для видалення) [30] забезпечує передбачуваність поведінки API та спрощує його інтеграцію та тестування [34].

Для захисту інформаційного периметра всі маршрути API захищені проміжним програмним забезпеченням (Middleware) authenticateToken. Його робота полягає у перехопленні HTTP-запиту, вилученні JWT-токена із заголовка Authorization та його верифікації. Це гарантує доступ до даних лише ідентифікованим користувачам. Для операцій, що змінюють стан системи або є критичними для бізнес-процесу (створення проєктів, видалення задач, примусовий перерозподіл), додатково застосовується механізм перевірки ролей requireRole. Він діє як фільтр, що обмежує права доступу виконавців («Executor»), дозволяючи

виконання адміністративних дій лише користувачам із ролями «Administrator» або «Project Manager».

Одним із центральних елементів серверної логіки є маршрут отримання списку проєктів (GET /projects), який відповідає за формування головного дашборду користувача. Логіка обробки цього запиту є адаптивною та залежить від контексту безпеки:

- Для адміністраторів формується запит на вибірку всіх наявних у системі проєктів без обмежень.
- Для менеджерів та виконавців застосовується фільтрація через таблицю зв'язку ProjectMembers (оператор INNER JOIN), що повертає лише ті проєкти, до яких користувач був явно доданий.

З метою оптимізації продуктивності та уникнення класичної проблеми «N+1 запитів» (коли для кожного проєкту довелося б робити окремий запит для підрахунку задач), в основний SQL-запит інтегровано підзапити (Subqueries). Вони на рівні бази даних виконують агрегацію статистики, автоматично підраховуючи загальну кількість задач (COUNT(*)) та кількість завершених завдань для кожного проєкту. Це дозволяє клієнту отримати готовий об'єкт із метаданими для візуалізації прогресу (Progress Bar) за одну мережеву транзакцію.

Для ініціалізації робочого простору реалізовано маршрут отримання даних дошки (GET /kanban/:projectId). Його технічною особливістю є комплексний SQL-запит, який об'єднує дані з трьох таблиць:

1. Tasks — змістовна частина (назва, опис, дедлайн).
2. KanbanOrder — презентаційна частина (статус колонки, порядковий номер).
3. Users — інформація про виконавця (ім'я користувача для відображення на картці).

Критично важливим є сортування результатів на рівні СУБД спочатку за статусом, а потім за позицією (ORDER BY k.status, k.position). Це гарантує, що клієнт отримує вже впорядкований масив даних, що знімає навантаження з браузера при первинному рендерингу.

Процес створення нової задачі (POST /create-task) реалізовано як багатоступеневу операцію, що вимагає забезпечення атомарності. Для цього використовується механізм транзакцій бази даних. Після успішної валідації вхідних даних сервер виконує послідовність дій:

1. Відкриття транзакції (BEGIN).
2. Створення запису в основній таблиці Tasks зі статусом «Planned».
3. Автоматичне обчислення позиції нової картки шляхом знаходження максимального поточного індексу у відповідній колонці (MAX(position) + 1).
4. Запис отриманої інформації в технічну таблицю KanbanOrder.
5. Фіксація факту створення задачі в історичній таблиці TaskStatusHistory.
6. Закриття транзакції (COMMIT).

Такий підхід гарантує цілісність даних (Data Integrity): якщо на будь-якому етапі виникне помилка, спрацьовує механізм ROLLBACK, який скасовує всі зміни, запобігаючи появі «фантомних» записів [14].

Для забезпечення роботи Drag-and-Drop інтерфейсу розроблено маршрут оновлення стану (POST /update-kanban). Алгоритм роботи цього ендпоінту включає не лише оновлення координат, а й перевірку специфічних бізнес-правил. Зокрема, реалізовано програмну заборону для виконавців самостійно повертати задачі зі статусу «Completed» назад у роботу без погодження з менеджером, що забезпечує контроль якості виконання.

Технічна реалізація збереження позиції базується на використанні SQL-конструкції INSERT ... ON DUPLICATE KEY UPDATE (Upsert) для таблиці KanbanOrder. Це дозволяє оновлювати координати задачі або створювати запис про позицію при його відсутності одним запитом, забезпечуючи стійкість до розсинхронізації даних. Паралельно з переміщенням відбувається оновлення статусу в основній таблиці та додавання запису в журнал змін з поточною часовою міткою NOW().

Для підтримки модуля звітності розроблено спеціалізовані маршрути агрегації даних. Маршрут /task-status/:projectId використовує групування (GROUP BY status) для підрахунку розподілу задач, а /completed-tasks/:projectId аналізує

історію змін статусів, фільтруючи записи за датою завершення. Така архітектура переносить обчислювальне навантаження з клієнта на оптимізований рушій бази даних, що значно прискорює побудову графіків.

Важливим елементом архітектури є налаштування CORS (Cross-Origin Resource Sharing) [32], що дозволяє клієнтському SPA-застосунку безпечно звертатися до API з іншого домену або порту. Також реалізовано централізовану обробку помилок: сервер повертає стандартизовані HTTP-коди (400 Bad Request — при помилках валідації, 401 Unauthorized — при проблемах з токеном, 403 Forbidden — при відсутності прав, 500 Internal Server Error — при збоях сервера), що дозволяє клієнту коректно реагувати на нештатні ситуації.

Таким чином, програмна реалізація серверної частини забезпечила створення надійного, масштабованого та гнучкого REST API, який виступає фундаментом для взаємодії з клієнтським застосунком. Використання транзакційного підходу при створенні та розподілі задач гарантує сувору цілісність даних (ACID). Оптимізація SQL-запитів, зокрема перенесення логіки агрегації статистики та сортування на сторону системи управління базами даних, дозволила досягти високої продуктивності та мінімізувати час відгуку сервера. Реалізована архітектура маршрутизації з інтегрованим багаторівневим захистом повністю задовольняє вимоги до безпеки, забезпечуючи розмежування доступу та захист ресурсів від несанкціонованого втручання.

3.2.3 Програмна реалізація алгоритмів бізнес-логіки

Реалізація серверної частини інформаційної системи не обмежується лише стандартними операціями обробки запитів на збереження та зчитування даних (CRUD). Ключовою цінністю розробленого програмного продукту є трансформація системи з пасивного сховища інформації в активний інструмент автоматизації управлінських рішень. Це досягається шляхом впровадження спеціалізованих алгоритмів бізнес-логіки, які мінімізують вплив людського фактора та пришвидшують процеси менеджменту. До найбільш значущих

програмних рішень належать механізм динамічного розрахунку пріоритетів та алгоритм автоматичного розподілу навантаження між виконавцями.

У традиційних системах управління проєктами пріоритет задачі зазвичай є статичним атрибутом (наприклад, поле priority зі значеннями «Low», «Medium», «High»), який зберігається в базі даних. Такий підхід має суттєвий недолік: пріоритет залишається незмінним, поки менеджер не змінить його вручну, навіть якщо до дедлайну залишилася одна година.

У розробленій системі реалізовано підхід динамічної пріоритезації, де статус терміновості є функцією від часу. Серверна частина виступає джерелом «правдивих даних» (Single Source of Truth), забезпечуючи передачу точних часових міток дедлайнів у форматі ISO 8601.

З метою оптимізації продуктивності та зниження навантаження на систему управління базами даних (СУБД), архітектура рішення передбачає делегування обчислень на сторону клієнта (Client-side calculation). Якби актуалізація статусу («Прострочено») відбувалася на сервері, це вимагало б запуску фонових завдань (cron jobs), які б щохвилини сканували всю таблицю задач та виконували масові операції запису (UPDATE), створюючи надмірне навантаження на диск (I/O). API передає лише «сирі» дані про дедлайн [20]. Клієнтський застосунок при отриманні даних у реальному часі обчислює різницю (Δt) між дедлайном та поточним системним часом. Це дозволяє миттєво візуалізувати критичні задачі (наприклад, автоматично змінювати колір індикатора на помаранчевий, якщо до завершення залишилося менше 48 годин, або на червоний, якщо $\Delta t < 0$), не створюючи при цьому жодного зайвого запиту до сервера.

Найбільш складним та інтелектуальним компонентом бізнес-логіки є модуль автоматичного балансування навантаження, реалізований у маршруті /auto-assign (ДОДАТОК Б). Його мета — справедливо розподілити пул накопичених, але ще не призначених задач (зі статусом «Planned») між доступними виконавцями проєкту. Алгоритм вирішує класичну проблему «вузького місця» (bottleneck), запобігаючи ситуаціям, коли один ефективний працівник перевантажений роботою, а інші простоюють.

Технічна реалізація та транзакційна цілісність Оскільки процес розподілу передбачає масове оновлення записів у базі даних, критично важливо забезпечити узгодженість даних. Реалізація алгоритму загорнута в SQL-транзакцію (ACID). Це означає, що процес виконується атомарно: або всі задачі будуть успішно розподілені та збережені, або, у випадку будь-якої помилки (збій мережі, блокування таблиці), жодна зміна не буде застосована.

Логіка роботи алгоритму складається з наступних етапів:

1. Ініціалізація контексту: Сервер відкриває транзакцію (connection.beginTransaction()). Далі виконується вибірка всіх користувачів проєкту, що мають роль «Executor».

2. Розрахунок базового навантаження: Система виконує аналітичний запит до таблиці Tasks, підраховуючи кількість активних завдань для кожного виконавця. Під «активним навантаженням» розуміються задачі зі статусами «In Progress» (В роботі) та «Review» (На перевірці). Завершені задачі («Completed») не враховуються, оскільки вони не потребують подальших ресурсів часу. Результат зберігається у тимчасовій структурі даних (наприклад, хеш-мапі: {userId: taskCount}).

3. Ітеративний жадібний розподіл (Greedy Algorithm): Система обирає всі задачі зі статусу «Planned» і запускає цикл розподілу.

- Крок сортування: На початку кожної ітерації список виконавців сортується за зростанням їхнього поточного навантаження.
- Крок призначення: Перша задача з пулу призначається виконавцю, який опинився на вершині відсортованого списку (тобто має найменшу кількість задач).
- Віртуальний інкремент: Після призначення задачі сервер не робить повторний запит до БД для перерахунку навантаження (це спричинило б проблему продуктивності N+1). Натомість, лічильник навантаження обраного виконавця у локальній пам'яті збільшується на +1.
- Наслідок: На наступній ітерації циклу цей виконавець вже не буде «найвільнішим», і наступна задача перейде до іншого члена команди.

Такий підхід гарантує рівномірний, «хвилеподібний» розподіл роботи навіть при одномоментному надходженні великої кількості нових завдань

4. Фіксація результатів: Після успішного завершення циклу виконується команда COMMIT, яка фізично зберігає зміни в базі даних. У разі виникнення виключної ситуації (Exception) у блоці try-catch, автоматично викликається ROLLBACK, що повертає систему до початкового стабільного стану.

Графічно логіку роботи алгоритму авторозподілу зображено на рисунку 3.4.

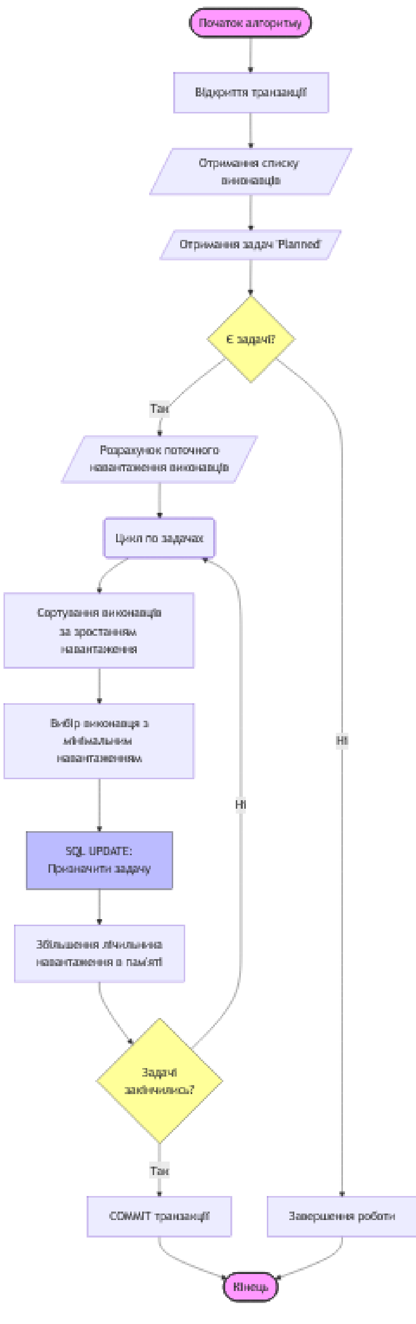

Рисунок 3.4 - Логіка роботи алгоритму авторозподілу

Реалізація описаних алгоритмів дозволила суттєво підвищити автономність системи. Динамічна пріоритезація забезпечує актуальність візуальної інформації без навантаження на сервер, а алгоритм авторозподілу виступає ефективним інструментом менеджменту, що математично точно балансує навантаження в команді, усуваючи суб'єктивність при розподілі завдань.

3.3 Реалізація клієнтської частини

3.3.1 Організація маршрутизації та структури компонентів

Головним контролером та точкою входу клієнтського застосунку виступає кореневий компонент App.js. У межах обраної архітектури Single Page Application (SPA), він виконує роль центрального диспетчера, який не лише керує відображенням інтерфейсу, а й утримує «єдине джерело правди» (Single Source of Truth) для глобального стану системи. Така централізація дозволяє уникнути розсинхронізації даних між різними частинами застосунку.

Замість використання важковагових бібліотек маршрутизації для базової навігації [29], у компоненті App реалізовано власний механізм перемикання представлень (Views) на основі станів. Цей підхід працює за ієрархічним алгоритмом «захисних умов» (Guard Clauses), що забезпечує послідовний доступ до функціоналу:

1. Рівень безпеки (Auth Guard): Першочергово перевіряється змінна стану user.

- Якщо значення null, система блокує рендеринг будь-яких внутрішніх компонентів, відображаючи виключно форму авторизації <Login />. Це гарантує, що захищені дані навіть теоретично не можуть потрапити у DOM-дерево до моменту успішного входу.
- При успішній автентифікації спрацьовує callback-функція handleLogin, яка оновлює глобальний стейт отриманим об'єктом користувача.

2. Рівень навігації (Project Selection): Якщо користувач авторизований, система аналізує змінну selectedProjectId.

- Значення null ініціює рендеринг компонента <ProjectList />. Це проміжний дашборд, який дозволяє користувачеві оглянути доступні проєкти або створити новий.
- Цей рівень діє як «меню вибору», розділяючи логіку управління портфелем проєктів від логіки роботи над конкретними задачами.

3. Рівень робочого простору (Workspace): Лише при наявності валідного об'єкта користувача та ID обраного проєкту завантажується основний інтерфейс системи.

- Він є композицією трьох незалежних блоків: навігаційної панелі, інтерактивної дошки <KanbanBoard /> та модуля аналітики (StatusPieChart, CompletedTasksChart).
- Важливою деталлю є наявність кнопки «Назад», яка просто скидає стан selectedProjectId у null, миттєво повертаючи користувача до списку проєктів без перезавантаження сторінки.

Для покращення користувацького досвіду (UX) реалізовано механізм збереження сесії між перезавантаженнями вкладки браузера. Технічно це реалізовано через побічні ефекти (Side Effects) у хуку useEffect:

- Ініціалізація: При першому монтуванні компонента App (залежність []), скрипт синхронно зчитує дані з localStorage за ключами user та currentProjectId. Якщо дані валідні, вони записуються у стейт React, і користувач миттєво опиняється на тому ж екрані, де завершив роботу, минаючи екран логіну.
- Завершення роботи: Функція handleLogout виконує повне очищення: видаляє токен безпеки та дані користувача з браузера, а також скидає стейт застосунку до початкових значень, що перенаправляє на екран входу.

Однією з ключових проблем компонентного підходу є необхідність передачі даних через проміжні компоненти, які їх не використовують (так зване «Prop Drilling»). У розробленій системі ця проблема вирішена шляхом впровадження патерну «Провайдер» через React Context API [39].

Компонент App створює контекст <KanbanProvider>, який діє як глобальна шина подій для робочого простору. Він транслює два об'єкти:

1. refreshTrigger: Числова змінна-лічильник (версія стану даних).
2. triggerRefresh: Функція-активатор, яка інкрементує цей лічильник.

Потік даних (Data Flow):

1. Дія: Користувач переміщує картку на дошці <KanbanBoard />. Після успішного запиту до API, цей компонент викликає triggerRefresh().
2. Реакція: Зміна лічильника в контексті автоматично детектується компонентами графіків (StatusPieChart, CompletedTasksChart), які підписані на цей контекст.
3. Оновлення: Графіки ініціюють повторний запит даних із сервера, відображаючи актуальну статистику.

Структуру взаємодії компонентів та напрямки потоків даних графічно зображено на рисунку 3.5.

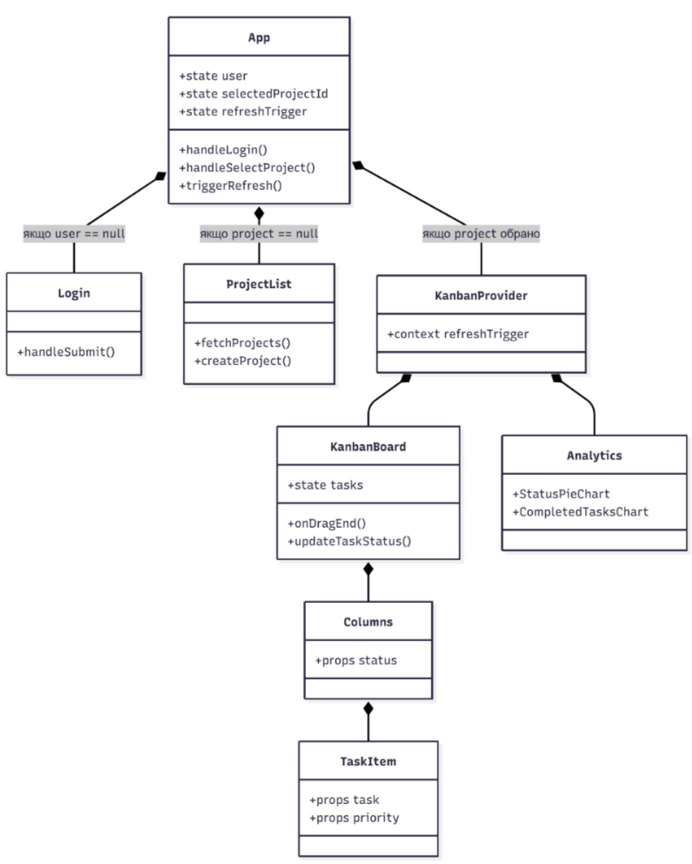

Рисунок 3.5 – Діаграма класів

Така архітектура забезпечує слабку зв'язаність (loose coupling) компонентів: модуль аналітики не знає про існування дошки задач, а дошка не знає про графіки. Їхня синхронізація відбувається виключно через спільний контекст, що значно спрощує підтримку та розширення коду.

3.3.2 Реалізація інтерфейсу Kanban-дошки та механізму Drag-and-Drop

Центральним елементом користувацького інтерфейсу системи є модуль KanbanBoard.js. Цей компонент відповідає за візуалізацію робочого процесу, відображення карток задач та забезпечення інтерактивної взаємодії користувача з системою за допомогою технології Drag-and-Drop. Архітектура модуля побудована за принципом вкладених компонентів, що дозволяє ізолювати логіку управління станом від презентаційної логіки. Інтерфейс дошки складається з чотирьох семантичних колонок («Planned», «In Progress», «Review», «Completed») [4], які рендеряться динамічно на основі масиву статусів за допомогою допоміжного компонента DroppableColumn.

Ключовою особливістю клієнтської реалізації є унікальний алгоритм динамічного розрахунку терміновості, інкапсульований у функції getUrgencyLevel. У традиційних системах пріоритет є статичним атрибутом, що зберігається в базі даних. У розробленій системі реалізовано підхід, де важливість завдання є функцією від часу. При кожному рендерингу картки система порівнює поточну дату (new Date()) із датою дедлайну задачі, отриманою з сервера. На основі різниці часу (diffDays) застосовується наступна логіка візуального кодування:

- Критичний стан: Якщо diffDays < 0 (дедлайн минув), картка отримує яскраво-червоний індикатор та текстову мітку «🔥 Прострочено!». Це миттєво привертає увагу команди до проблемних зон.
- Висока терміновість: Якщо до дедлайну залишається менше 2 днів (diffDays <= 2), відображається мітка «⚡ Терміново» з відповідним кольоровим акцентом.

- Нормальний режим: Якщо часу достатньо, система позначає задачу нейтральним індикатором «☕ Є час».
- Стан завершення: Для виконаних задач (status === 'Completed') алгоритм ігнорує часові рамки, підсвічуючи картку зеленим кольором («✅ Виконано»), що символізує успішний результат.

Такий підхід дозволяє команді фокусуватися на дійсно важливих завданнях без необхідності ручного оновлення пріоритетів менеджером. Інтерфейс Kanban-дошки з прикладом роботи індикації зображено на рисунку 3.6.

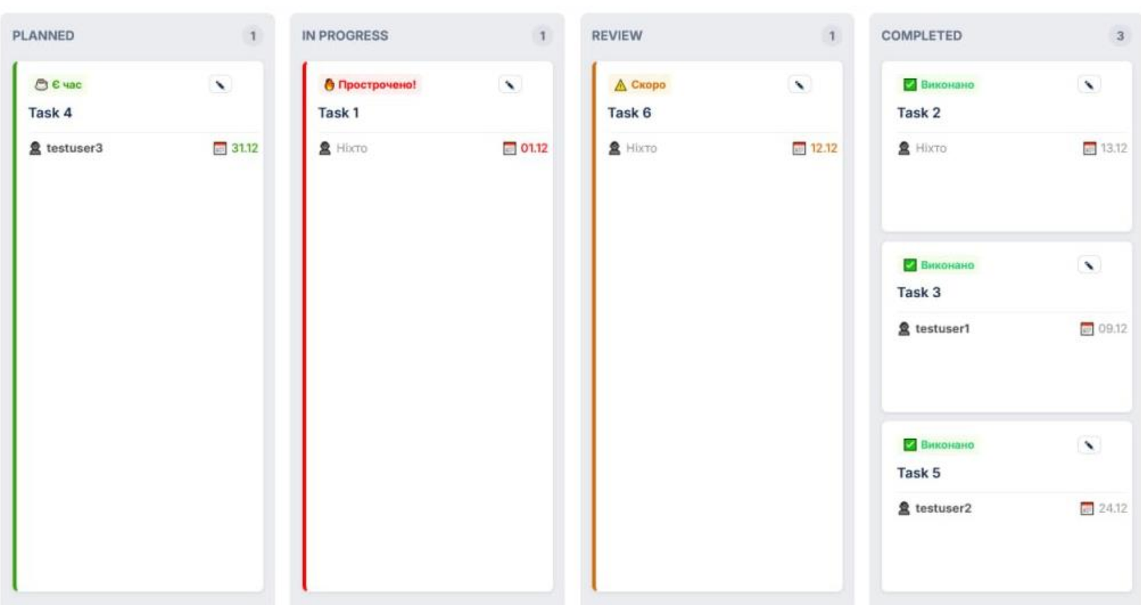

Рисунок 3.6 — Інтерфейс Kanban-дошки з динамічною індикацією пріоритетів

Для реалізації функції переміщення карток використано сучасну бібліотеку @dnd-kit [16]. Весь простір дошки огорнуто в провайдер контексту <DndContext>, який керує глобальним станом перетягування. Для забезпечення точного позиціювання елементів застосовано алгоритм детекції колізій closestCorners. Цей алгоритм обчислює відстань від центру перетягуваної картки до кутів найближчих

контейнерів, що дозволяє коректно визначати цільову колонку навіть при різній висоті стовпчиків або складному макеті сторінки.

Внутрішній простір кожної колонки огорнуто в компонент <SortableContext> зі стратегією сортування rectSortingStrategy. Це надає користувачеві можливість не лише переміщувати задачі між станами, але й змінювати їх порядок всередині однієї колонки (сортування списку), просто перетягуючи картку вгору або вниз.

Найскладніша бізнес-логіка зосереджена у функції-обробнику onDragEnd, яка викликається в момент, коли користувач відпускає картку. Ця функція виконує роль контролера, що реалізує три послідовні етапи обробки події:

1. Ідентифікація контексту переміщення: Система аналізує параметри події (active та over), щоб визначити тип дії: чи була картка переміщена в нову колонку (зміна статусу), чи відбулася лише зміна позиції в межах поточної колонки (reordering).
2. Валідація прав доступу (Guard Logic): Перед застосуванням будь-яких змін виконується перевірка бізнес-правил на основі ролі користувача. Система зчитує права поточного користувача (canManage) і накладає обмеження:

- Блокування завершення: Якщо користувач із роллю «Виконавець» (Executor) намагається перетягнути задачу в колонку «Completed», дія скасовується, а інтерфейс виводить модальне попередження про те, що фіналізація задач є прерогативою менеджера.
- Заборона відкату: Виконавцям також заборонено самостійно повертати задачі зі статусу «Completed» назад у роботу («In Progress»), що запобігає несанкціонованому відновленню вже закритих завдань.

3. Оптимістичне оновлення (Optimistic UI): Якщо перевірки пройдені успішно, система застосовує патерн оптимістичного оновлення інтерфейсу. Це означає, що візуальний стан дошки (локальний стейт tasks) оновлюється миттєво, не чекаючи відповіді від сервера. Картка візуально "стрибає" на нове місце, забезпечуючи відчуття миттєвої реакції застосунку. Паралельно у фоновому режимі відправляється асинхронний запит axios.post на маршрут /update-kanban,

який фіксує зміни в базі даних. Після успішного завершення запиту викликається функція triggerRefresh з глобального контексту KanbanContext, що змушує оновитися аналітичні графіки. У випадку мережевої помилки зміни автоматично відкочуються, повертаючи інтерфейс у попередній узгоджений стан.

3.3.3 Реалізація модулів аналітики та візуалізації даних

В умовах динамічного управління проєктами критично важливу роль відіграють засоби об'єктивного контролю, які дозволяють команді та менеджменту отримувати миттєвий зріз поточного стану робіт. Для вирішення задачі візуалізації статистичних даних у клієнтському застосунку було спроєктовано та імплементовано модуль аналітики, побудований на базі популярної графічної бібліотеки Chart.js. Вибір саме цього інструменту обумовлений його архітектурною особливістю — використанням елемента HTML5 Canvas для малювання графіки [17]. На відміну від бібліотек, що базуються на SVG (Scalable Vector Graphics), де кожен стовпчик або сектор діаграми стає окремим вузлом DOM-дерева, Canvas-підхід генерує єдине растрове зображення. Це забезпечує екстремально високу продуктивність рендерингу та плавність анімацій навіть при візуалізації великих масивів даних, що є критичним для забезпечення швидкодії Single Page Application на пристроях з обмеженими обчислювальними ресурсами.

Аналітична підсистема застосунку структурно розділена на два спеціалізовані компоненти, кожен з яких відповідає за візуалізацію окремого аспекту ефективності проєктної діяльності. Першим ключовим елементом є компонент StatusPieChart, який відповідає за побудову кругової діаграми розподілу завдань за їх поточними статусами. Логіка роботи компонента базується на реактивному підході з використанням хука useEffect. При первинному завантаженні або при надходженні сигналу оновлення компонент ініціює асинхронний HTTP-запит до серверного REST API за маршрутом /task-status. Отримавши «сирі» дані у форматі JSON, система виконує їх трансформацію: масив

об'єктів перетворюється на окремі масиви міток та числових значень, які необхідні для конфігурації екземпляра класу Chart.

З метою покращення когнітивного сприйняття інформації, у компоненті реалізовано сувору синхронізацію кольорової палітри: кольори секторів діаграми чітко відповідають візуальному кодуванню колонок на Kanban-дошці (наприклад, використання відтінку #f6ffed для завершених задач та #e6f7ff для тих, що в роботі). Для підвищення інформативності діаграми було інтегровано додатковий плагін chartjs-plugin-datalabels. Цей модуль автоматично розраховує відсоткове співвідношення кожної категорії задач відносно загальної кількості та рендерить відповідні текстові мітки безпосередньо на секторах графіку, що позбавляє користувача необхідності наводити курсор для отримання точних значень.

Другим аналітичним інструментом є компонент CompletedTasksChart, який слугує для візуалізації динаміки продуктивності команди (Velocity) у часовому розрізі. Компонент будує стовпчасту діаграму, що відображає кількість завершених завдань за певні календарні дати. Технічна реалізація цього модуля включає складну попередню обробку даних на стороні клієнта. Оскільки сервер повертає часові мітки у форматі UTC (ISO 8601), компонент виконує їх конвертацію у локальний формат дати користувача за допомогою методу toLocaleDateString, формуючи зрозумілі підписи для осі абсцис. Конфігурація осей графіка також була адаптована під специфіку дискретних даних: для осі ординат встановлено параметр stepSize: 1, що унеможливлює відображення дробових значень, які є нелогічними для підрахунку кількості задач.

Стилістичне оформлення стовпчастої діаграми наближено до галузевих стандартів корпоративних систем (зокрема, Jira Styling), використовуючи специфічний зелений колір (#36b37e) та заокруглення кутів стовпців (borderRadius: 4) для створення сучасного інтерфейсу. Важливим аспектом користувацького досвіду (UX) є реалізація умовного рендерингу для «порожніх станів»: якщо API повертає порожній масив даних (що свідчить про відсутність виконаних задач у новому проєкті), замість порожнього полотна користувачеві відображається стилізоване текстове повідомлення про відсутність статистики.

З технічної точки зору, критично важливим аспектом реалізації обох графіків є коректне управління життєвим циклом об'єкта Chart у середовищі React. Оскільки React може багаторазово перемальовувати компонент, існує ризик створення нових графіків поверх старих на одному й тому ж елементі Canvas, що призводить до візуальних артефактів («мерехтіння») та витоків пам'яті. Для вирішення цієї проблеми використано функцію очищення (cleanup function) у хуку useEffect, яка викликає метод .destroy() для поточного екземпляра графіка перед кожним новим рендерингом. Це гарантує, що в будь-який момент часу на полотні існує лише один активний графік.

Інтеграція ізольованих аналітичних модулів у загальну екосистему застосунку реалізована через патерн «Спостерігач» (Observer) з використанням React Context API. Обидва графічні компоненти підписані на оновлення глобальної змінної refreshTrigger, яка передається через KanbanContext. Це забезпечує миттєву реактивність системи: будь-яка значуща дія користувача на дошці задач, будь то переміщення картки в колонку «Completed» або створення нового завдання, автоматично змінює стан тригера. Це, в свою чергу, ініціює перезапуск ефектів у компонентах графіків та завантаження актуальних даних із сервера. Такий підхід дозволяє менеджеру спостерігати за змінами ключових метрик проєкту в режимі реального часу без необхідності ручного оновлення сторінки браузера, забезпечуючи цілісність та актуальність відображуваної інформації.

Візуалізацію роботи аналітичних модулів наведено на рисунку 3.7.

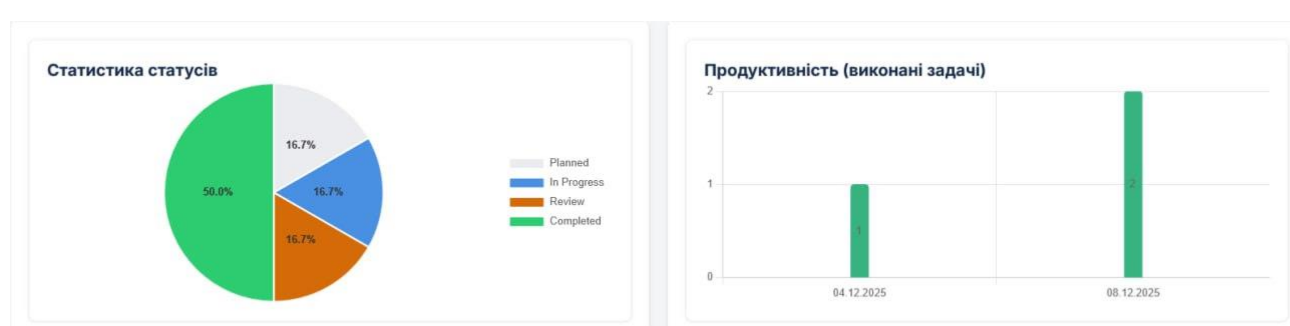

Рисунок 3.7 — Візуалізація аналітичних даних проєкту

3.4 Інтерфейс користувача та сценарії взаємодії

Результатом програмної реалізації проєкту є комплексний веб-застосунок, що надає користувачам зручний графічний інтерфейс для керування командною діяльністю. Візуальна складова системи розроблена з дотриманням принципів Single Page Application (SPA) [8], що забезпечує миттєву реакцію інтерфейсу на дії користувача без необхідності повного перезавантаження сторінки. Дизайн системи виконано в мінімалістичному стилі з використанням контрастних кольорів для виділення ключових елементів, таких як статуси задач та пріоритети, що дозволяє користувачам швидко орієнтуватися у великих обсягах інформації [11].

Взаємодія з системою розпочинається з екрану аутентифікації, який слугує єдиною точкою входу для всіх типів користувачів. Інтерфейс цієї сторінки спроєктовано таким чином, щоб забезпечити швидкий доступ як до форми входу, так і до реєстрації нового облікового запису. При виборі опції реєстрації користувачеві необхідно заповнити анкетні дані: ім'я, унікальну електронну адресу та пароль. На цьому етапі реалізовано інтерактивну валідацію: система в реальному часі аналізує введені дані і, у випадку невідповідності формату (наприклад, занадто короткий пароль або некоректний email). Після успішного введення облікових даних та натискання кнопки входу, система генерує токен доступу, зберігає його в локальному сховищі браузера та автоматично перенаправляє користувача до головного меню. Інтерфейс входу в систему представлено на рисунку 3.8.

Рисунок 3.8 — Інтерфейс входу в систему

Після успішної авторизації відкривається головна панель управління (Dashboard), яка відображає портфель доступних проєктів, що позначено на рисунку 3.9. Візуалізація реалізована у вигляді адаптивної сітки карток [36], що дозволяє комфортно працювати як на широких моніторах, так і на екранах ноутбуків. Кожна картка проєкту містить ключову інформацію: назву, текстовий опис мети та динамічний індикатор прогресу (Progress Bar), який автоматично розраховується на основі співвідношення виконаних та загальних задач. Для менеджерів та адміністратора передбачено функціонал створення нових проєктів: при натисканні відповідної кнопки основний фон затемнюється, і з'являється модальне вікно. У ньому користувач вказує назву та опис проєкту, після чого нова картка миттєво з'являється у сітці без оновлення сторінки. Сторінка зі активними проєктами представлена на рисунку 3.9. Модальне вікно створення проєкту позначено на рисунку 3.10.

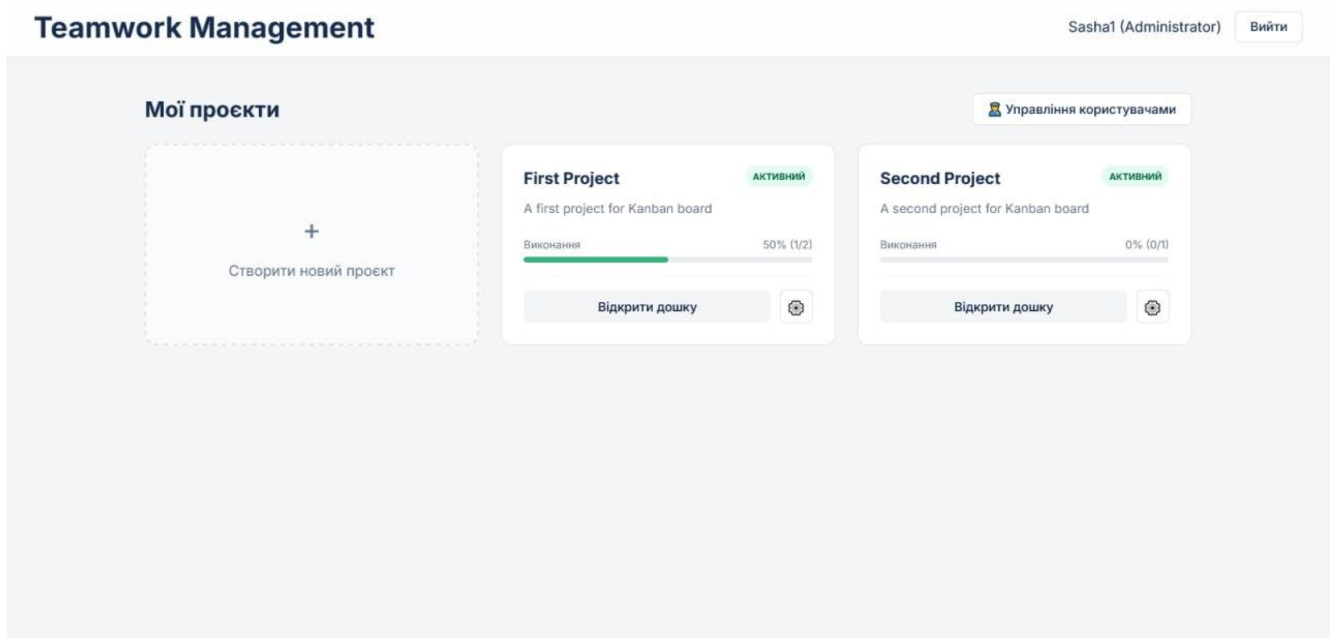

Рисунок 3.9 — Головна панель управління проєктами

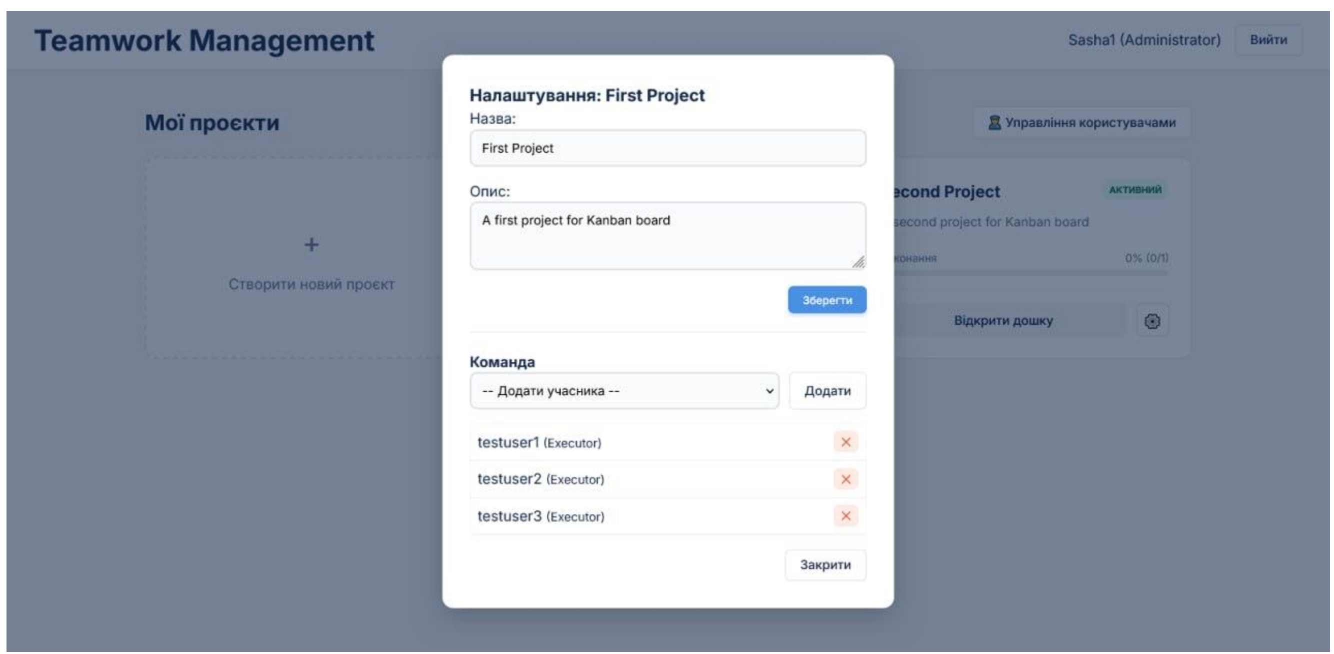

Рисунок 3.10 — Модальне вікно створення проєкту

При переході до конкретного проєкту відкривається основний робочий простір — інтерактивна Kanban-дошка. Інтерфейс дошки розділено на чотири вертикальні колонки, що відповідають етапам життєвого циклу задачі: «Planned», «In Progress», «Review» та «Completed». У шапці кожної колонки відображається лічильник задач, що дозволяє контролювати завантаженість етапів. Задачі представлені у вигляді карток, що містять назву, аватар виконавця та дату дедлайну. Особливістю інтерфейсу є кольорова індикація терміновості: система порівнює поточну дату з дедлайном і маркує картку відповідним кольором (червоний — прострочено, помаранчевий — терміново, зелений — вчасно). Основний сценарій взаємодії — переміщення задач методом Drag-and-Drop. Користувач захоплює картку курсором, при цьому зона приземлення підсвічується, вказуючи, куди буде переміщено задачу. Також на цьому екрані доступна бічна панель зі списком учасників команди та кнопками для швидкого додавання нових завдань через спливаюче вікно.

У нижній частині робочого простору інтегровано модуль візуальної аналітики, який надає менеджеру інструменти для моніторингу ефективності.

Модуль складається з двох графіків, що оновлюються синхронно зі змінами на дошці. Зліва розташовано кругову діаграму, яка демонструє відсотковий розподіл задач за статусами, дозволяючи виявити "вузькі місця" у процесі розробки. Справа відображається стовпчаста діаграма, що показує історію закриття задач по днях. Завдяки використанню технології Canvas, графіки мають плавну анімацію при появі та зміні даних. Цей блок дозволяє оцінити продуктивність команди (Velocity) без необхідності переходу в інші розділи меню.

На рисунку 3.11 представлено робочий простір з інтерактивною дошкою задач та графіками.

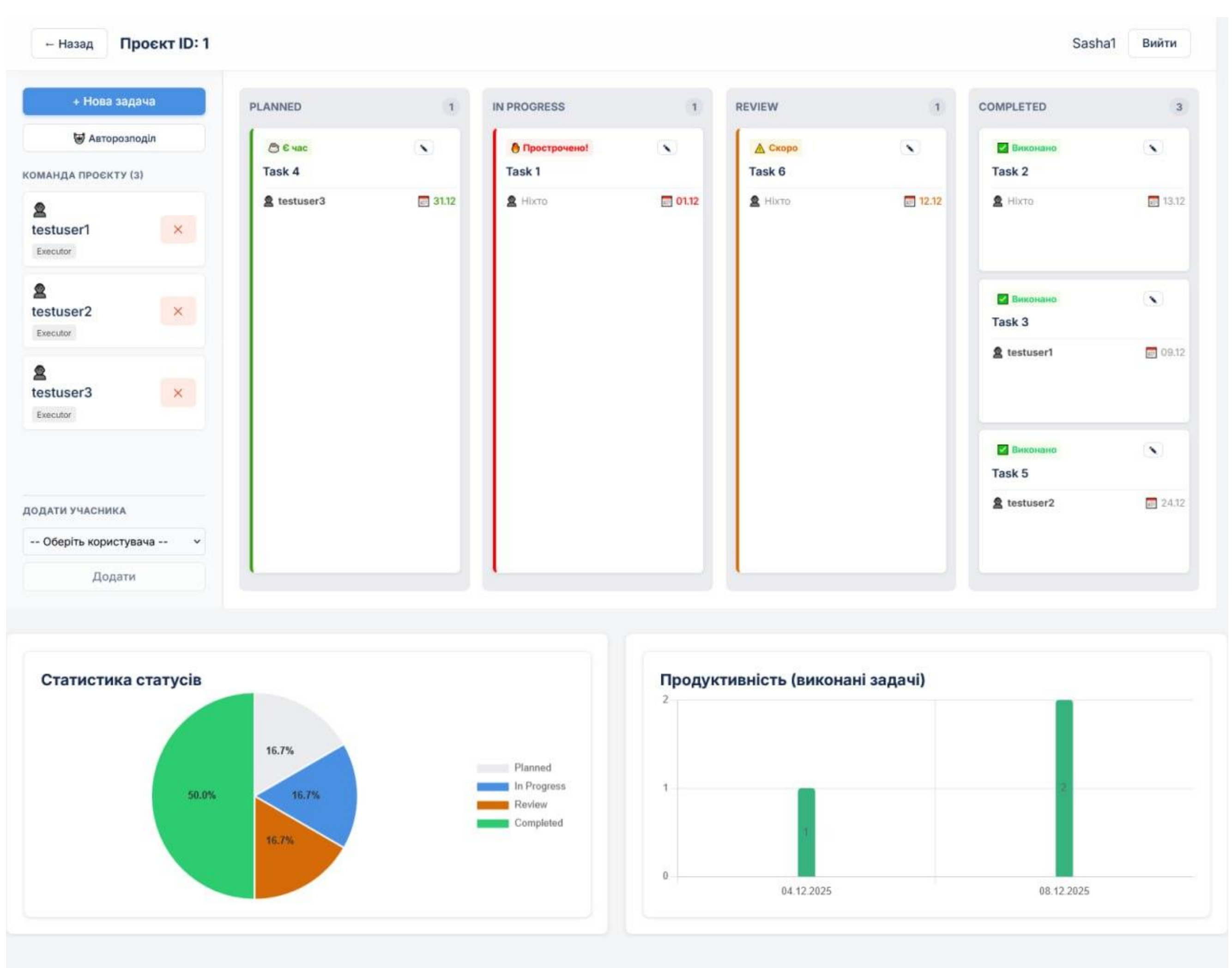

Рисунок 3.11 — Робочий простір з інтерактивною дошкою задач та графіками

Також розроблено спеціалізоване діалогове вікно створення задачі, яке викликається кнопкою на бічній панелі. Інтерфейс цього вікна надає менеджеру розширений набір інструментів: окрім введення назви та текстового опису, користувач обирає дату дедлайну через вбудований календар. Особливістю цього інтерфейсу є випадаючий список виконавців, який автоматично фільтрується, показуючи лише учасників поточної команди. Після збереження, система автоматично розраховує пріоритет задачі на основі введеної дати та розміщує нову картку в колонці «Planned». Модальне вікно створення задачі позначено на рисунку 3.12.

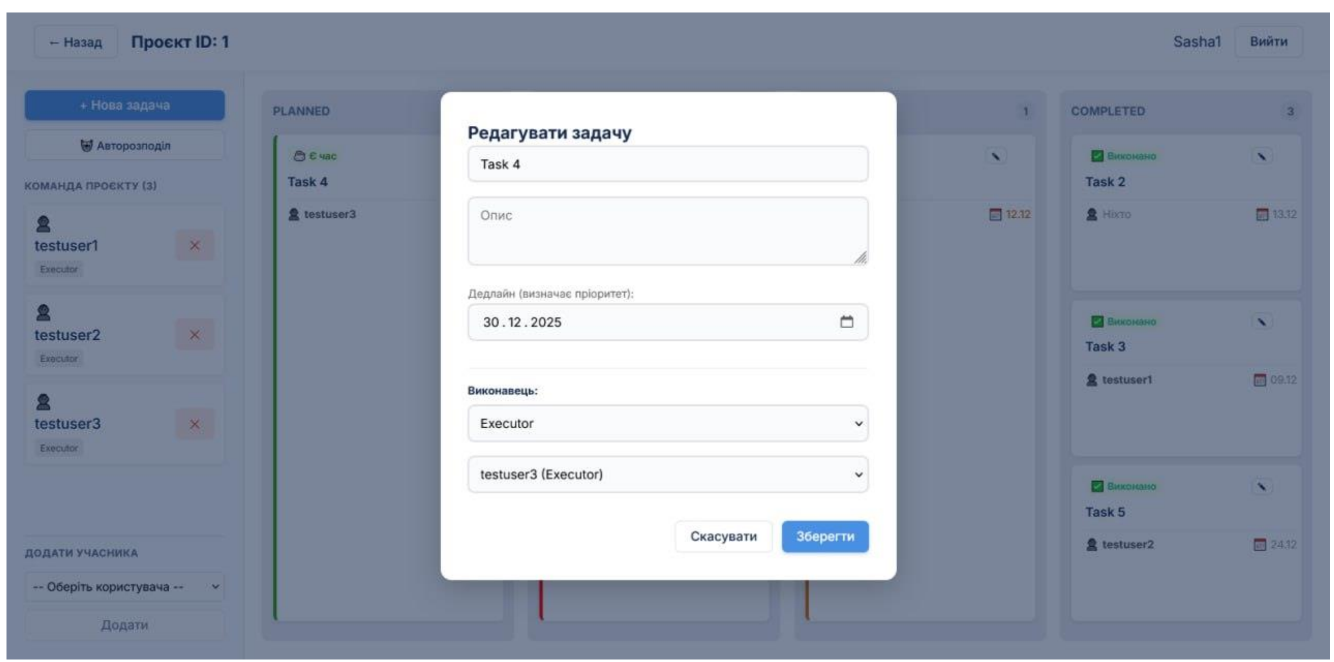

Рисунок 3.12 — Модальне вікно створення задачи

Для забезпечення функцій адміністрування розроблено окремий інтерфейс управління користувачами, доступний лише для ролі «Administrator». Доступ до нього здійснюється через навігаційну панель, яка відкриває спеціалізоване вікно з табличними даними. У таблиці наведено повний перелік користувачів системи з їхніми контактними даними та поточними ролями. Інтерфейс дозволяє змінювати глобальні права доступу в один клік через випадаючий список у кожному рядку.

Система містить захисні механізми на рівні інтерфейсу: наприклад, елемент керування власною роллю адміністратора заблоковано, що унеможливлює випадкову втрату прав доступу. Також через цей інтерфейс менеджер може додавати нових учасників до конкретного проєкту, використовуючи пошук по базі користувачів. Інтерфейс зображено на рисунку 3.13.

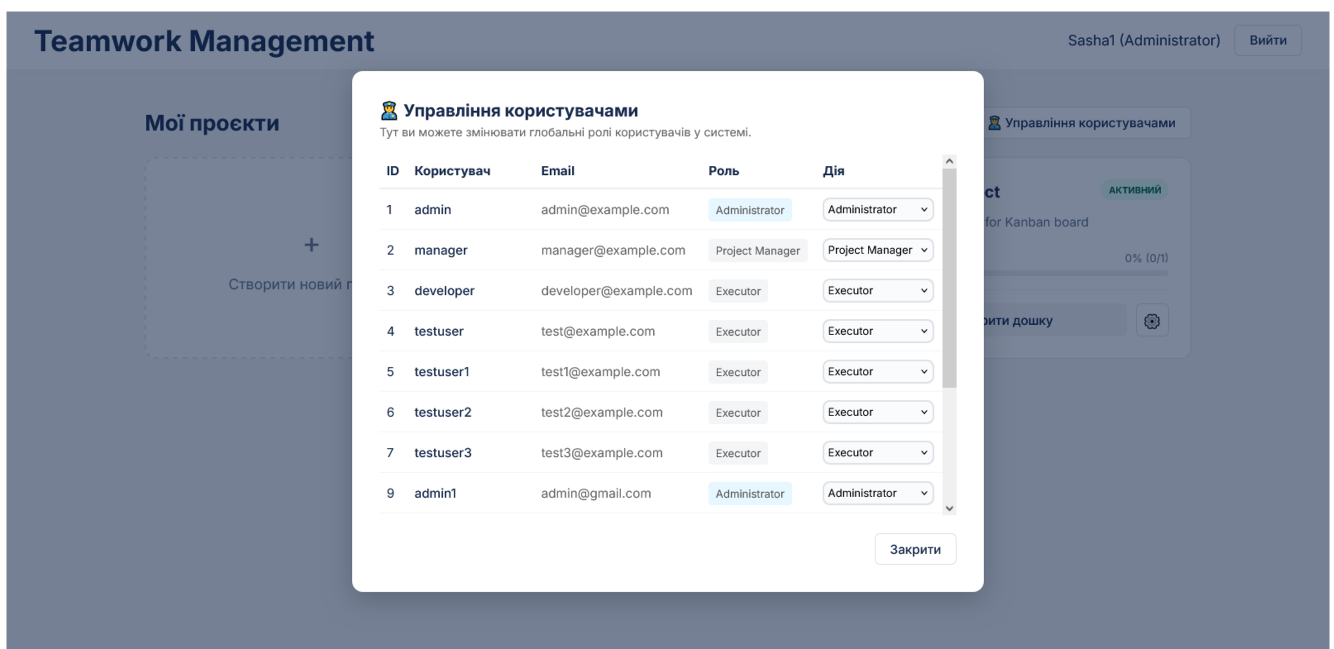

Рисунок 3.13 — Інтерфейс адміністрування та управління ролями

Підсумовуючи, розроблений графічний інтерфейс забезпечує ефективну взаємодію користувачів із програмним комплексом. Використання архітектури SPA та продумана візуальна ієрархія дозволили створити ергономічне та чутливе середовище для командної роботи. Реалізовані сценарії взаємодії повністю покривають ключові бізнес-процеси: від безпечної авторизації та динамічного управління задачами на Kanban-дошці до моніторингу продуктивності через аналітичні панелі. Організація інтерфейсу з чітким розмежуванням доступу для різних ролей гарантує зручність використання системи як для виконавців, так і для адміністраторів.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломної роботи було проведено комплексне дослідження та реалізовано проєкт інформаційної системи, призначеної для планування, координації та аналізу командної діяльності. Основну мету роботи досягнуто в повному обсязі. Створена інформаційна система успішно вирішує поставлені задачі, поєднуючи зручний інтерфейс планування з інструментами оптимізації ресурсів, що дозволяє усунути суб'єктивність в управлінні та підвищити ефективність командної роботи.

На основі сформульованих вимог було спроєктовано та впроваджено надійну клієнт-серверну архітектуру з використанням сучасного технологічного стеку, що включає платформу Node.js та фреймворк Express.js для серверної частини, бібліотеку React.js для клієнтського інтерфейсу та реляційну СУБД MySQL для зберігання даних.

Ключовим результатом роботи стала програмна реалізація унікальних алгоритмів бізнес-логіки, які суттєво розширюють можливості класичної Kanban-методології. Зокрема, було розроблено механізм автоматичного розподілу завдань, який аналізує поточну зайнятість виконавців та призначає нові задачі таким чином, щоб збалансувати навантаження в команді. Додатково впроваджено алгоритм динамічного розрахунку пріоритетів, який автоматично визначає терміновість завдання на основі дедлайну.

Практична цінність роботи підкріплюється створенням повнофункціонального веб-застосунку з інтерактивним інтерфейсом Single Page Application (SPA), що підтримує технологію Drag-and-Drop для управління станами задач та містить модуль візуальної аналітики на базі Chart.js для моніторингу продуктивності команди в реальному часі. Розроблена система забезпечує високий рівень безпеки даних завдяки використанню JWT-автентифікації та рольової моделі доступу, що робить її готовим рішенням для підвищення ефективності роботи малих та середніх команд.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1) Sommerville I. Software Engineering, 10th Edition. Boston : Pearson, 2015. 816 p.
2) Bass L., Clements P., Kazman R. Software Architecture in Practice (SEI Series in Software Engineering), 3rd Edition. Boston : Addison-Wesley Professional, 2012. 640 p.
3) Sommerville I. Software Engineering, 10th Edition. Boston : Pearson, 2015. 816 p.
4) Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim : Blue Hole Press, 2010. 278 p.
5) Richardson L., Ruby S. RESTful Web Services. Sebastopol : O'Reilly Media, 2007. 448 p.
6) Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. Irvine : University of California, 2000.
7) Casciaro M., Mammino L. Node.js Design Patterns. 2nd Edition. Birmingham : Packt Publishing, 2016. 536 p.
8) Mikowski M., Powell J. Single Page Web Applications: JavaScript End-to-End. Shelter Island : Manning Publications, 2013. 408 p.
9) Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps, 2nd Edition. Sebastopol : O'Reilly Media, 2020. 350 p.
10) Date C. J. An Introduction to Database Systems, 8th Edition. Boston : Pearson, 2003. 1024 p.
11) Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. Berkeley : New Riders, 2014. 216 p.
12) React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://react.dev/reference/react (дата звернення: 12.12.2025).

13) Express - fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/ (дата звернення: 12.12.2025).

14) MySQL 8.0 Reference Manual [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 12.12.2025).

15) Introduction to JSON Web Tokens [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction (дата звернення: 12.12.2025).

16) dnd-kit Documentation: Lightweight, modular, performant, accessible and extensible drag & drop toolkit for React [Електронний ресурс]. – Режим доступу: https://docs.dndkit.com/ (дата звернення: 12.12.2025).

17) Chart.js | Open source HTML5 Charts for your website [Електронний ресурс]. – Режим доступу: https://www.chartjs.org/docs/latest/ (дата звернення: 12.12.2025).

18) Axios HTTP Client Documentation [Електронний ресурс]. – Режим доступу: https://axios-http.com/docs/intro (дата звернення: 12.12.2025).

19) OWASP Top 10: The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 12.12.2025).

20) MDN Web Docs: JavaScript [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 12.12.2025).

21) npm: bcryptjs - Optimized bcrypt in JavaScript with zero dependencies [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/bcryptjs (дата звернення: 12.02.2025)

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

23) Hunt A., Thomas D. The Pragmatic Programmer: From Journeyman to Master. Boston : Addison-Wesley Professional, 1999. 352 p.

24) Fowler M. Refactoring: Improving the Design of Existing Code, 2nd Edition. Boston : Addison-Wesley Professional, 2018. 448 p.

25) Simpson K. You Don't Know JS: Scope & Closures. Sebastopol : O'Reilly Media, 2014. 98 p.
26) Crockford D. JavaScript: The Good Parts. Sebastopol : O'Reilly Media, 2008. 176 p.
27) Manifesto for Agile Software Development [Електронний ресурс]. – Режим доступу: https://agilemanifesto.org/ (дата звернення: 12.12.2025).
28) The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game [Електронний ресурс]. – Режим доступу: https://scrumguides.org/scrum-guide.html (дата звернення: 12.12.2025).
29) React Router Documentation: Routing for React apps [Електронний ресурс]. – Режим доступу: https://reactrouter.com/ (дата звернення: 12.12.2025).
30) MDN Web Docs: HTTP request methods [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 12.12.2025).
31) MDN Web Docs: HTTP response status codes [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (дата звернення: 12.12.2025).
32) Cross-Origin Resource Sharing (CORS) - MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 12.12.2025).
33) npm: mysql2 - Fast, simple and compatible MySQL driver for Node.js [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/mysql2 (дата звернення: 12.12.2025).
34) Postman API Platform Documentation [Електронний ресурс]. – Режим доступу: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 12.12.2025).
35) Git - Distributed Version Control System [Електронний ресурс]. – Режим доступу: https://git-scm.com/doc (дата звернення: 12.12.2025).

36) W3C Recommendation: CSS Flexible Box Layout Module [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/css-flexbox-1/ (дата звернення: 12.12.2025).
37) Stack Overflow Developer Survey 2024 [Електронний ресурс]. – Режим доступу: https://survey.stackoverflow.co/2024/ (дата звернення: 12.12.2025).
38) V8 JavaScript Engine [Електронний ресурс]. – Режим доступу: https://v8.dev/ (дата звернення: 12.12.2025).
39) Passing Data Deeply with Context – React Docs [Електронний ресурс]. – Режим доступу: https://react.dev/learn/passing-data-deeply-with-context (дата звернення: 12.12.2025).
40) Hernandez R. The logical model of the relational database. IBM Knowledge Center [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/ (дата звернення: 12.12.2025).
41) Prettier · Opinionated Code Formatter [Електронний ресурс]. – Режим доступу: https://prettier.io/ (дата звернення: 12.12.2025).

ДОДАТОК А

Вихідний код функцій проміжної обробки запитів (Middleware)

```
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'Токен відсутній' });

  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ error: 'Невалідний токен' });
    req.user = user;
    next();
  });
};

const requireRole = (roles) => (req, res, next) => {
  if (!req.user || !roles.includes(req.user.role)) {
    return res.status(403).json({ error: 'Доступ заборонено' });
  }
  next();
};
```

ДОДАТОК Б

Функція авторозподілу задач

```
app.post('/auto-assign/:projectId', authenticateToken,
requireRole(['Administrator', 'Project Manager']), async (req, res) => {
    const { projectId } = req.params;
    const connection = await pool.getConnection();
    try { await connection.beginTransaction();
      const [executors] = await connection.query(`
        SELECT u.id
        FROM ProjectMembers pm
        JOIN UserRoles ur ON pm.user_id = ur.user_id AND pm.project_id =
ur.project_id
        JOIN Roles r ON ur.role_id = r.id
        JOIN Users u ON pm.user_id = u.id
        WHERE pm.project_id = ? AND r.name = 'Executor'
      `, [projectId]);
      if (executors.length === 0) throw new Error('Немає виконавців.');
      const [tasksToAssign] = await connection.query(
        'SELECT id FROM Tasks WHERE project_id = ? AND status =
"Planned"',
        [projectId] );
      if (tasksToAssign.length === 0) {
        await connection.rollback();
        return res.json({ message: 'Немає запланованих задач для розподілу.'
});    }
      const executorLoad = {};
      executors.forEach(e => executorLoad[e.id] = 0);
      const [activeWork] = await connection.query(
        `SELECT assigned_to, COUNT(*) as count
```

```
                FROM Tasks
                WHERE project_id = ? AND status IN ('In Progress', 'Review') AND
assigned_to IS NOT NULL
                GROUP BY assigned_to`,
               [projectId]    );
            activeWork.forEach(row => {
               if (executorLoad[row.assigned_to] !== undefined) {
                  executorLoad[row.assigned_to] = row.count;      }         });
            let assignedCount = 0;
            for (const task of tasksToAssign) {
               const sortedExecutors = executors.sort((a, b) => {
                  return executorLoad[a.id] - executorLoad[b.id];  });
               const chosenOne = sortedExecutors[0];
               await connection.query('UPDATE Tasks SET assigned_to = ? WHERE id
= ?', [chosenOne.id, task.id]);
               executorLoad[chosenOne.id]++;
               assignedCount++; }
            await connection.commit();
            res.json({ success: true, message: `Перерозподілено задач:
${assignedCount}` });
         } catch (error) {
            await connection.rollback();
            console.error('Auto-assign error:', error);
            res.status(500).json({ error: error.message });
         } finally {
            connection.release();
         }
      });
```

ДОДАТОК В

Основні функції і механізми з файлу App.js

useEffect(() => { const storedUser = localStorage.getItem('user'); const storedProjectId = localStorage.getItem('currentProjectId');

if (storedUser) { setUser(JSON.parse(storedUser)); } if (storedProjectId) { setSelectedProjectId(storedProjectId); } }, []);

const handleLogin = (loggedUser) => { setUser(loggedUser); }; const handleLogout = () => { localStorage.removeItem('token'); localStorage.removeItem('user'); localStorage.removeItem('currentProjectId');

setSelectedProjectId(null); };

const [refreshTrigger, setRefreshTrigger] = useState(0);

const triggerRefresh = () => { setRefreshTrigger(prev => prev + 1); };

const handleSelectProject = (projectId) => { setSelectedProjectId(projectId); localStorage.setItem('currentProjectId', projectId); }; const handleBackToProjects = () => { setSelectedProjectId(null); localStorage.removeItem('currentProjectId');

ДОДАТОК Г

Функція розрахунку приорітету по даті

```
const getUrgencyLevel = (deadlineDate) => {
    if (!deadlineDate) return { label: '📅 Без дати', color: '#777', weight: 'None', bgColor: '#f0f0f0' };
    const now = new Date();
    now.setHours(0, 0, 0, 0);
    const deadline = new Date(deadlineDate);
    deadline.setHours(0, 0, 0, 0)
    const diffTime = deadline - now;
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));

    if (diffDays < 0) {
        return { label: '🔥 Прострочено!', color: '#ff0000', weight: 'Critical', bgColor: '#fff1f0' };
    }
    if (diffDays <= 2) {
        return { label: '⚡ Терміново', color: '#cf1322', weight: 'High', bgColor: '#fff1f0' }; // Червоний
    }
    if (diffDays <= 5) {
        return { label: '⚠️ Скоро', color: '#d46b08', weight: 'Medium', bgColor: '#fff7e6' }; // Помаранчевий
    }
    return { label: '☕ Є час', color: '#389e0d', weight: 'Low', bgColor: '#f6ffed' }; // Зелений
};
```