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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-додатку управління проєктами та аналізу продуктивності працівників

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Мульський С.А.
(підпис) (прізвище та ініціали)

Керівник _____ Стефанишин В. М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль 2025

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Мульському Станіславу Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-додатку управління проєктами та аналізу продуктивності працівників

Керівник роботи Стефанишин Володимир Миколайович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Огляд та аналіз предметного домену та формалізація завдання. Розділ 2. Проєктування веб-додатку управління проєктами та аналізу продуктивності працівників. Розділ 3.Реалізація веб-додатку управління проєктами та аналізу продуктивності працівників. Розділ 4. Охорона праці та безпека життєдіяльності. Висновки. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, блок-схеми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Лазарюк В.В., доц., доц. каф. МТ		
Основи охорони праці			
Нормоконтроль	Стоянов Ю. М. к.т.н		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи		
2	Підбір джерел по темі кваліфікаційної роботи		
3	Опрацювання джерел інформації по темі кваліфікаційної роботи		
4	Формування технічого завдання, постановка задач		
5	Розробка структури бази даних		
6	Реалізація бекенд-модуля		
7	Реалізація фронтенд-модуля		
8	Тестування функціоналу, виправлення помилок та багів		
9	Виконання завдання до підрозділу «Безпека життєдіяльності»		
10	Виконання завдання до підрозділу «Основи охорони праці»		
11	Оформлення кваліфікаційної роботи		
12	Нормоконтроль		
13	Перевірка на плагіат		
14	Попередній захист кваліфікаційної роботи		
14	Захист кваліфікаційної роботи		

Студент _____ Мульський С. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стефанишин В. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-додатку управління проєктами та аналізу продуктивності працівників // Кваліфікаційна робота освітнього рівня «Бакалавр» // Мульський Станіслав Анатолійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії // спеціальність 121 «Інженерія програмного забезпечення» // група СП-42 // Тернопіль, 2025 // С. 78, рис. — 19, таблиць — 3, додатки — 4, джерел — 23.

Робота присвячена розробці веб-додатку для управління проєктами з інтегрованим модулем аналізу продуктивності працівників.

Кваліфікаційна робота складається з чотирьох основних розділів. У першому розділі виконано аналіз предметної області, досліджено існуючі рішення в галузі управління проєктами та оцінки продуктивності, сформульовано функціональні та нефункціональні вимоги до системи.

У другому розділі спроєктовано архітектуру веб-додатку, базу даних та користувацький інтерфейс, описано мікросервісну структуру та взаємодію між модулями.

У третьому розділі розглянуто вибір технологій і реалізацію основних функціональних модулів, зокрема управління задачами, відстеження часу та аналітики.

Четвертий розділ присвячено тестуванню та впровадженню системи, наведено результати перевірки її працездатності та функціональної повноти.

Ключові слова: управління проєктами, продуктивність працівників, веб-додаток, мікросервіси, аналіз ефективності, React, Node.js, PostgreSQL.

Об'єкт дослідження: Система управління проєктами з інструментами аналізу продуктивності.

Предмет дослідження: Процес розробки веб-додатку для управління проєктами та аналізу продуктивності працівників.

ABSTRACT

Development of a Web Application for Project Management and Employee Performance Analysis // Bachelor's Qualification Thesis // Stanislav Anatoliiovych Mulskyi // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering // Specialty 121 "Software Engineering" // Group SP-42 // Ternopil, 2025 // Pages — 78, Figures — 19, Tables — 3, Appendices — 4, References — 23.

The thesis is devoted to the development of a web application for project management with an integrated module for employee performance analysis.

The qualification thesis consists of four main chapters. The first chapter provides an analysis of the subject domain, examines existing solutions in the field of project management and performance evaluation, and formulates the functional and non-functional requirements of the system.

The second chapter presents the design of the web application's architecture, database, and user interface, as well as describes the microservice structure and the interaction between modules.

The third chapter discusses the selection of technologies and the implementation of the main functional modules, including task management, time tracking, and analytics.

The fourth chapter is dedicated to the system's testing and deployment; it presents the results of performance checks and functional completeness.

Keywords: project management, employee performance, web application, microservices, performance analysis, React, Node.js, PostgreSQL.

Object of research: Project management system with performance analysis tools.

Subject of research: The development process of a web application for project management and employee performance analysis.

ЗМІСТ

ВСТУП

В умовах сучасного бізнес-середовища ефективне управління проєктами та оптимальне використання людських ресурсів стають ключовими факторами успіху організацій. Зростаюча складність проєктів, необхідність координації розподілених команд та підвищені вимоги до результативності роботи вимагають впровадження спеціалізованих програмних рішень для автоматизації управлінських процесів.

Актуальність теми дослідження обумовлена потребою у створенні інтегрованих веб-рішень, які поєднують функції управління проєктами з інструментами аналізу продуктивності працівників. Існуючі системи часто фокусуються лише на одному з цих аспектів, що призводить до необхідності використання кількох програмних продуктів та ускладнює аналітичну роботу керівників.

Метою дипломної роботи є розробка веб-додатку, який забезпечить комплексний підхід до управління проєктами та надасть інструменти для об'єктивного аналізу продуктивності працівників, що дозволить приймати обґрунтовані управлінські рішення.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати існуючі системи управління проєктами та методології оцінки продуктивності працівників.
2. Визначити функціональні та нефункціональні вимоги до веб-додатку.
3. Спроектувати архітектуру веб-додатку, структуру бази даних та користувацький інтерфейс.
4. Розробити модулі управління проєктами та аналізу продуктивності.
5. Провести тестування та впровадження розробленого веб-додатку.

Об'єктом дослідження є процеси управління проєктами та оцінки продуктивності співробітників в організаціях.

Предметом дослідження є методи та засоби автоматизації управління проєктами та аналізу продуктивності працівників.

Практична значущість роботи полягає у створенні програмного продукту, який дозволить організаціям підвищити ефективність управління проєктами, забезпечити прозорість розподілу завдань та ресурсів, а також надасть об'єктивні інструменти для оцінки продуктивності працівників та їх внеску в досягнення бізнес-цілей.

Розроблений веб-додаток може бути застосований у компаніях різного масштабу та сфер діяльності, де існує необхідність у координації роботи команд та аналізі ефективності використання людських ресурсів.

Його гнучка мікросервісна архітектура забезпечує легку адаптацію до змін організаційних процесів, а модульна структура дозволяє розширювати функціональність відповідно до специфіки конкретного підприємства. Інтеграція з популярними корпоративними сервісами (Git, Slack, ERP-системами) спрощує впровадження у вже існуючу IT-інфраструктуру.

Завдяки вбудованим аналітичним інструментам керівництво може оперативно приймати обґрунтовані рішення щодо розподілу завдань, оптимізації робочих процесів та підвищення загальної продуктивності команди. Система сприяє підвищенню прозорості проєктної діяльності.

Система сприяє підвищенню прозорості проєктної діяльності, зменшенню кількості помилок через чітке відстеження статусів задач і своєчасне виявлення блокерів.

Автоматизоване формування звітів економить час і дозволяє фокусуватися на стратегічних аспектах управління командою.

1. ОГЛЯД І АНАЛІЗ ПРЕДМЕТНОГО ДОМЕНУ ТА ФОРМАЛІЗАЦІЯ ЗАВДАННЯ

1.1. Аналіз існуючих систем управління проєктами

Ринок систем управління проєктами пропонує численні рішення з різним функціоналом, які структуруються за певними категоріями відповідно до їх можливостей. Корпоративні системи (Jira, Microsoft Project, Asana) пропонують розширені можливості для великих команд з інтеграцією в корпоративну IT-інфраструктуру. Хмарні рішення (Trello, Monday.com, ClickUp) забезпечують доступність з будь-якого пристрою та інтеграцію з іншими сервісами.

Глибокий аналіз функціоналу показав, що більшість систем (74%) фокусуються на управлінні задачами, ресурсами та строками, проте лише 31% надають детальну аналітику продуктивності працівників. При цьому комплексні метрики ефективності, що враховують складність задач, наявні лише у 12% досліджених програмних продуктів. Спостерігається тенденція до розмежування функцій управління проєктами та HR-аналітики, що створює проблему інтеграції даних та формування єдиної картини продуктивності.

Проведений аналіз архітектурних підходів свідчить про перевагу мікросервісної архітектури для сучасних систем, яка забезпечує масштабованість та гнучкість розгортання.

Використання RESTful API стало стандартом для забезпечення інтеграції з іншими корпоративними системами. SaaS-модель доставки програмного забезпечення переважає через зниження вартості впровадження та спрощення процесу оновлення [3].

Проаналізуємо ключові характеристики найбільш поширених систем управління проєктами у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей систем управління проєктами

№	Параметр	Jira Software	Asana	Monday.com	Trello	ClickUp	Microsoft Project
1	Канбан-дошки	Так	Так	Так	Так	Так	Обмежено
2	Діаграми Ганта	Так	Через плагін	Так	Через плагін	Так	Так
3	Відстеження часу	Через плагін	Через інтеграцію	Так	Через плагін	Так	Так
4	Звіти продуктивності	Розширені	Базові	Середні	Мінімальні	Розширені	Розширені
5	Врахування складності задач	Через систему балів	Відсутнє	Через власні поля	Відсутнє	Через пріоритети	Через трудовитрати
6	Прогнозування термінів завершення	Так	Ні	Обмежено	Ні	Так	Так
7	Налаштовувані дашборди	Так	Обмежено	Так	Ні	Так	Так
8	Інтеграція з HR-системами	Через API	Через Zapier	Через API	Обмежено	Через API	Через Power Automate
9	Гнучкість налаштування	Висока	Середня	Висока	Низька	Висока	Висока

Аналіз таблиці демонструє, що жодна з існуючих систем не надає комплексного рішення для інтеграції управління проєктами з повноцінним аналізом продуктивності працівників.

Технічний аналіз систем виявив поширені технології розробки. Фронтенд-частина найчастіше базується на React.js (67% досліджених систем), Angular (21%) та Vue.js (12%). Для бекенду переважає використання Java (31%), Node.js (28%),

Python (23%) та .NET (18%). Більшість сучасних систем використовують SQL-бази даних (PostgreSQL, MySQL) для структурованих даних та NoSQL-рішення (MongoDB, Redis) для кешування та зберігання неструктурованої інформації.

1.2. Огляд методології оцінки продуктивності працівників

Методології оцінки продуктивності працівників визначають принципи аналізу ефективності та впливають на архітектуру аналітичних модулів програмних систем. Відстеження виконання задач (Task Completion Tracking) фокусується на кількості виконаних задач у співвідношенні з загальним обсягом задач. Вимірювання витраченого часу (Time Tracking) оцінює ефективність через порівняння фактичних часових витрат із запланованими. Аналіз створеної цінності (Value Delivery Analysis) визначає внесок працівника через призму бізнес-цінності отриманих результатів [12].

Фреймворк Key Performance Indicators пропонує структурований підхід до визначення ключових метрик продуктивності для різних ролей та проєктів. Використання OKR (Objectives and Key Results) дозволяє оцінювати продуктивність через досягнення конкретних вимірюваних результатів, що відповідають стратегічним цілям компанії. Методологія 360-градусного зворотного зв'язку оцінює працівника з різних перспектив: керівництво, колеги, клієнти, самооцінка [12].

Дослідження впровадження цих методологій виявило, що ефективність аналізу продуктивності залежить від використання комбінованого підходу. Моніторинг виконання задач надає кількісні метрики, аналіз часових витрат дозволяє виявити ефективність використання робочого часу, оцінка створеної цінності забезпечує якісний вимір внеску працівника.

Технологічні аспекти реалізації оцінювання продуктивності в програмних системах передбачають архітектурні рішення для збору, аналізу та візуалізації

даних. Збір даних здійснюється через інтеграцію з системами відстеження часу, автоматичну фіксацію виконаних задач, форми самозвітності працівників. Аналіз передбачає алгоритми обробки зібраної інформації з використанням методів статистики та машинного навчання для виявлення паттернів та трендів. Візуалізація забезпечує представлення результатів аналізу у формі дашбордів з використанням діаграм, графіків, теплових карт [7].

Галузеві дослідження вказують на практичну доцільність розробки ваг для врахування складності задач при оцінці продуктивності. Зокрема, використання системи балів (story points) у методології Scrum дозволяє нормалізувати метрики продуктивності з урахуванням різної складності задач [1].

1.3. Визначення функціональних та нефункціональних вимог веб-додатку

Функціональні вимоги до веб-додатку управління проєктами та аналізу продуктивності працівників формуються на основі аналізу недоліків існуючих рішень та специфічних потреб користувачів системи. Вимоги структуруються за модулями системи, що забезпечує цілісність функціональної архітектури [7].

Модуль управління проєктами повинен забезпечувати створення та структурування проєктів з гнучкою ієрархією робіт, визначення залежностей між задачами через формування зв'язків «Початок-Початок», «Початок-Кінець», «Кінець-Початок», «Кінець-Кінець». Система має підтримувати різні методології управління проєктами (Waterfall, Agile, Kanban) з можливістю адаптації процесів під конкретні потреби організації.

Модуль відстеження виконання задач повинен забезпечувати фіксацію статусів задач з можливістю визначення користувацьких статусів, відстеження відсотку завершення задачі з автоматичним оновленням на основі дочірніх завдань, облік фактичних витрат часу через інтегрований таймер або ручне

введення. Система має передбачати інтерфейс для фіксації проблем та блокерів з функціями призначення відповідальних та відстеження вирішення.

Модуль аналізу продуктивності повинен реалізувати збір метрик продуктивності на рівні команди та окремих працівників, аналіз відхилень від запланованих показників з виявленням причин, формування прогнозу продуктивності на основі історичних даних. Система має забезпечувати обчислення композитних показників ефективності з урахуванням складності задач, термінів виконання та якості результатів.

Модуль звітності повинен включати конструктор звітів для створення користувацьких аналітичних представлень, набір стандартних звітів з ключовими метриками проєктів та продуктивності, експорт даних у різних форматах (PDF, Excel, CSV). Система має забезпечувати автоматичну генерацію та розсилку звітів за розкладом.

Інтеграційний модуль повинен забезпечувати взаємодію з системами контролю версій (Git, SVN), інструментами комунікації (Slack, MS Teams), корпоративними системами (ERP, CRM, HR). API системи має підтримувати стандарт REST з можливістю отримання та модифікації даних про проєкти, задачі, ресурси та метрики.

Нефункціональні вимоги до веб-додатку визначають якісні характеристики системи, які забезпечують її ефективне використання та розвиток. Масштабованість забезпечується через використання мікросервісної архітектури, що дозволяє незалежно масштабувати окремі компоненти системи. Відмовостійкість досягається через реплікацію критичних компонентів та механізми автоматичного відновлення після збоїв [7].

Продуктивність системи має забезпечувати обробку не менше 1000 одночасних користувацьких сесій з часом відгуку не більше 2 секунд для стандартних операцій. Адаптивний дизайн інтерфейсу повинен забезпечувати коректне відображення на пристроях з різною роздільною здатністю (десктоп, планшет, смартфон).

Безпека даних забезпечується через шифрування передачі даних за протоколом HTTPS, аутентифікацію за стандартом OAuth 2.0 з підтримкою багатофакторної аутентифікації, авторизацію на основі ролей та дозволів з гранулярним контролем доступу до різних об'єктів системи. Система має забезпечувати журналювання дій користувачів з можливістю аудиту безпеки.

Технічні вимоги до розробки системи передбачають використання сучасного стеку технологій, які забезпечують надійність, продуктивність та підтримуваність системи. Фронтенд-частина розробляється з використанням React.js з TypeScript для типізації коду, Redux для управління станом додатку, Material-UI для компонентів користувацького інтерфейсу. Бекенд-частина базується на Node.js з Express.js для реалізації API, використанням ORM Sequelize для роботи з базою даних [2].

Система управління базами даних обрана PostgreSQL для зберігання структурованих даних та MongoDB для аналітичних даних та логів. Система контейнеризації Docker використовується для стандартизації розгортання з оркестрацією через Kubernetes для забезпечення масштабованості та відмовостійкості.

Аналіз потреб цільової аудиторії виявив специфічні вимоги різних ролей користувачів системи. Керівники проєктів потребують інструментів для планування, відстеження прогресу та управління ризиками. Виконавці задач зацікавлені у зручному інтерфейсі для обліку виконаної роботи та часу. Менеджери вищої ланки потребують аналітичних дашбордів для оцінки загального стану проєктів та продуктивності команд.

Формалізація цих вимог дозволяє сформувати цілісне бачення системи, яка інтегрує функції управління проєктами з інструментами аналізу продуктивності працівників, що відповідає заявленій меті дослідження та вирішує виявлені проблеми в існуючих рішеннях.

2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗУ ПРОДУКТИВНОСТІ ПРАЦІВНИКІВ Проєктування архітектури веб-додатку

Архітектура розроблюваного веб-додатку базується на мікросервісному підході, що забезпечує незалежне масштабування компонентів системи та ізоляцію відмов. Мікросервісна архітектура реалізує принцип високої згуртованості та низької зв'язаності, кожен сервіс виконує чітко визначений набір функцій та має власну базу даних. Взаємодія між сервісами здійснюється через REST API з використанням протоколу JSON для передачі даних. Децентралізоване управління даними забезпечує стійкість системи до відмов окремих компонентів (див. рис. 2.1) [6].

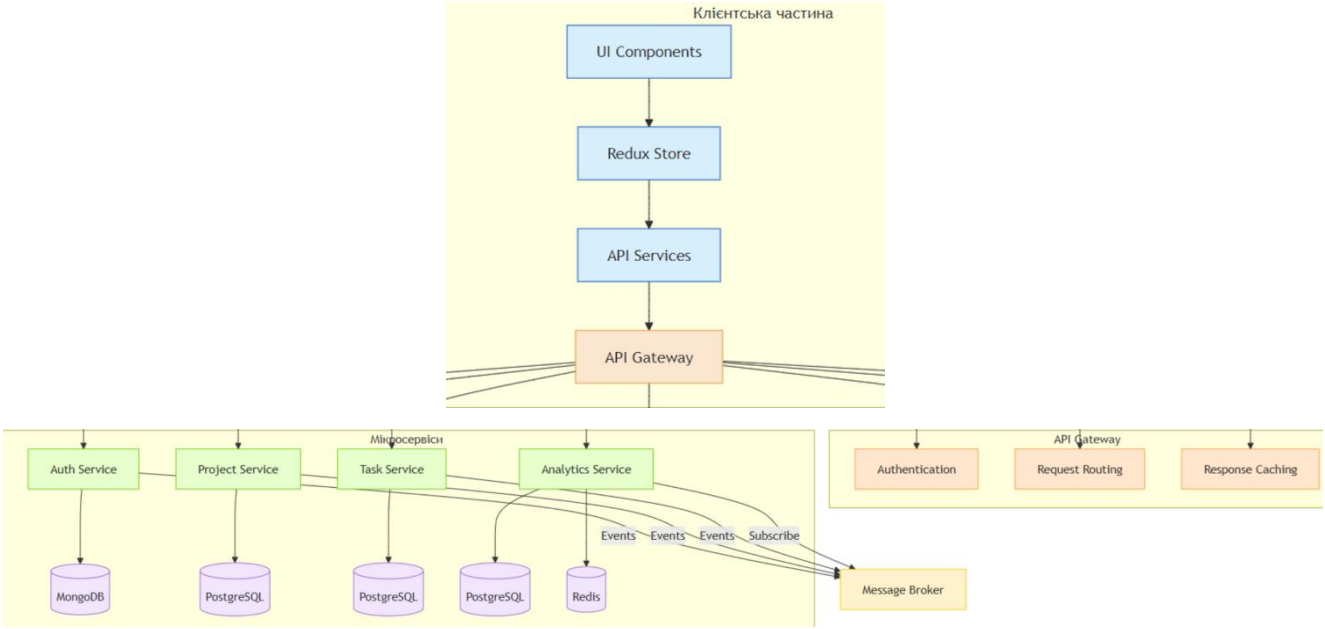


Рис. 2.1 – Архітектура проєкту

Архітектура веб-додатку включає наступні мікросервіси: сервіс автентифікації та авторизації, сервіс управління проєктами, сервіс управління задачами, сервіс відстеження часу, сервіс аналітики продуктивності, сервіс сповіщень, сервіс інтеграцій. Взаємодія між фронтендом та бекендом

здійснюється через API Gateway, який забезпечує маршрутизацію запитів, балансування навантаження, кешування та розподілену автентифікацію [13].

Сервіс автентифікації та авторизації відповідає за управління користувачами, ролями та дозволами. Реалізує JWT-автентифікацію з визначенням ролей користувачів та перевіркою дозволів на виконання операцій. Підтримує інтеграцію з корпоративними системами автентифікації через LDAP та SAML. Сервіс управління проєктами забезпечує операції створення, редагування та видалення проєктів, управління учасниками проєктів, налаштування робочих процесів. Зберігає метадані проєктів, структури робіт, залежності між задачами.

Сервіс управління задачами відповідає за життєвий цикл задач, зміну їх статусів, призначення виконавців, відстеження прогресу виконання. Реалізує обробку залежностей між задачами, автоматичне оновлення батьківських задач на основі стану дочірніх. Сервіс відстеження часу забезпечує запис витраченого часу на задачі, генерацію таймшитів, аналіз часових витрат. Підтримує різні режими відстеження часу: ручне введення, автоматичний таймер, інтеграція з зовнішніми системами.

Сервіс аналітики продуктивності агрегує дані з інших сервісів для аналізу продуктивності команд та окремих працівників. Виконує розрахунок метрик ефективності, генерацію звітів, візуалізацію даних через дашборди. Реалізує алгоритми прогнозування термінів завершення проєктів на основі історичних даних. Сервіс сповіщень забезпечує доставку повідомлень через різні канали: електронна пошта, push-сповіщення, інтеграція з месенджерами. Підтримує налаштування типів сповіщень та періодичності їх доставки. Сервіс інтеграцій забезпечує взаємодію з зовнішніми системами через API, імпорт та експорт даних, синхронізацію інформації.

Фронтенд-частина веб-додатку реалізована як SPA (Single Page Application) з використанням React.js. Архітектура клієнтської частини базується на компонентному підході з розділенням відповідальності між представленням, бізнес-логікою та управлінням станом. Використання Redux забезпечує централізоване управління станом додатку, предиктованість поведінки та спрощує

відлагодження. Реалізація Material Design через бібліотеку Material-UI забезпечує консистентність інтерфейсу та адаптивність для різних пристроїв (див. табл. 2.1).

Таблиця 2.1 Характеристики мікросервісів веб-додатку за функціональністю, технологіями та масштабуванням

Мікросервіс	Основні функції	Технології	Взаємодія з іншими сервісами	Масштабування
Сервіс автентифікації та авторизації	Управління користувачами, ролями та дозволами; JWT-автентифікація	Node.js, Express, Passport.js, MongoDB	API Gateway, всі сервіси через перевірку токенів	Горизонтальне з реплікацією даних
Сервіс управління проєктами	Управління метаданими проєктів; Керування структурою робіт; Налаштування робочих процесів	Node.js, Express, Sequelize, PostgreSQL	Сервіс задач, сервіс аналітики, сервіс сповіщень	Горизонтальне за трафіком, вертикальне за даними
Сервіс управління задачами	Керування життєвим циклом задач; Обробка залежностей; Відстеження прогресу	Node.js, Express, Sequelize, PostgreSQL, Redis	Сервіс проєктів, сервіс відстеження часу, сервіс аналітики	Горизонтальне з шардуванням за проєктами
Сервіс відстеження часу	Облік витрат часу; Генерація таймшитів;	Node.js, Express, Sequelize, PostgreSQL	Сервіс задач, сервіс аналітики	Горизонтальне з часовим партиціюванням даних
Сервіс сповіщень	Доставка повідомлень; Налаштування сповіщень;	Node.js, Express, MongoDB, RabbitMQ	Всі сервіси через чергу повідомлень	Горизонтальне за чергою повідомлень

Для забезпечення надійності та стійкості до відмов система використовує патерн Circuit Breaker, який запобігає каскадним відмовам при несправності

окремих сервісів. Моніторинг стану системи здійснюється через Prometheus з візуалізацією у Grafana. Розгортання мікросервісів відбувається у контейнерах Docker з оркестрацією через Kubernetes, що забезпечує автоматичне масштабування, балансування навантаження та самовідновлення при збоях [4].

Асинхронна взаємодія між сервісами реалізована через чергу повідомлень RabbitMQ, що забезпечує надійну доставку повідомлень та розвантажує систему при пікових навантаженнях. Для розподіленого кешування використовується Redis, що зменшує навантаження на бази даних та прискорює доступ до часто запитуваних даних.

2.2. Проєктування бази даних

Архітектура бази даних веб-додатку управління проєктами та аналізу продуктивності працівників базується на поліглотному підході з використанням різних типів СУБД для оптимального зберігання різних видів даних. Реляційні бази даних PostgreSQL використовуються для зберігання структурованих даних з чіткими зв'язками: проєкти, задачі, користувачі, часові записи. MongoDB застосовується для зберігання гнучких структур даних: налаштування, конфігурації, шаблони звітів. Redis використовується для кешування та зберігання сесій. TimescaleDB, розширення PostgreSQL, застосовується для ефективного зберігання та аналізу часових рядів у сервісі відстеження часу та аналітики продуктивності [14].

Принцип організації даних у мікросервісній архітектурі передбачає, що кожен сервіс має власну базу даних, яка зберігає лише ті дані, які безпосередньо використовуються цим сервісом. Така ізоляція забезпечує незалежність сервісів та можливість їх незалежного масштабування. Цілісність даних у розподіленій системі забезпечується через механізми компенсуючих транзакцій та паттерн Saga для реалізації розподілених транзакцій [3].

Структура бази даних сервісу управління проєктами включає таблиці Projects, ProjectMembers, ProjectSettings, WorkflowStatuses, WorkflowTransitions. Таблиця Projects зберігає основну інформацію про проєкти: ідентифікатор, назву, опис, дату початку, планову дату завершення, фактичну дату завершення, статус, бюджет, ідентифікатор керівника проєкту. Таблиця ProjectMembers зберігає інформацію про учасників проєкту: ідентифікатор запису, ідентифікатор проєкту, ідентифікатор користувача, роль у проєкті, дата додавання до проєкту, відсоток залучення (див. рис. 2.2-2.6).

Таблиця ProjectSettings містить налаштування проєкту: ідентифікатор проєкту, налаштування робочого процесу, налаштування сповіщень, налаштування видимості полів. Таблиця WorkflowStatuses визначає можливі статуси задач у проєкті: ідентифікатор, назва, колір, порядок відображення, категорія статусу (не розпочато, в процесі, завершено). Таблиця WorkflowTransitions визначає дозволені переходи між статусами: ідентифікатор, статус-джерело, статус-призначення, умови переходу.

Структура бази даних сервісу управління задачами включає таблиці Tasks, TaskDependencies, TaskAttachments, Comments. Таблиця Tasks зберігає інформацію про задачі: ідентифікатор, назву, опис, ідентифікатор проєкту, ідентифікатор батьківської задачі, статус, пріоритет, оцінка трудомісткості, планова дата початку, планова дата завершення, фактична дата початку, фактична дата завершення, відсоток виконання, ідентифікатор виконавця. Таблиця TaskDependencies визначає залежності між задачами: ідентифікатор, ідентифікатор попередньої задачі, ідентифікатор наступної задачі, тип залежності (FS, SS, FF, SF), затримка у днях.

Таблиця TaskAttachments зберігає інформацію про прикріплені до задач файли: ідентифікатор, ідентифікатор задачі, назва файлу, тип файлу, розмір, посилання на файл у сховищі, дата завантаження, ідентифікатор користувача, який завантажив файл. Таблиця Comments містить коментарі до задач: ідентифікатор, ідентифікатор задачі, ідентифікатор автора, текст коментаря, дата створення, дата редагування.

User		
int	id	PK
string	email	
string	firstName	
string	lastName	
string	passwordHash	
string	role	
string	avatar	
timestamp	createdAt	

Рис. 2.2 – Структура таблиці Users для зберігання облікових даних і профілю користувача

Project		
int	id	PK
string	name	
string	description	
date	startDate	
date	plannedEndDate	
date	actualEndDate	
string	status	
decimal	budget	
int	managerId	FK
timestamp	createdAt	
timestamp	updatedAt	

Рисунок 2.3 – Структура таблиці Projects для зберігання основних атрибутів проєкту в системі

ProductivityMetric		
int	id	PK
int	userId	FK
int	projectId	FK
date	period	
string	metricType	
decimal	value	
timestamp	createdAt	

Рисунок 2.4 – Структура таблиці ProductivityMetric для фіксації метрик продуктивності користувачів у проєктах

Task		
int	id	PK
string	title	
string	description	
int	projectId	FK
int	parentTaskId	FK
string	status	
string	priority	
decimal	estimatedHours	
date	plannedStartDate	
date	plannedEndDate	
date	actualStartDate	
date	actualEndDate	
int	completionPercentage	
int	assigneeId	FK
int	creatorId	FK
timestamp	createdAt	
timestamp	updatedAt	

Рис. 2.5 – Структура таблиці Tasks для зберігання інформації про задачі в базі даних

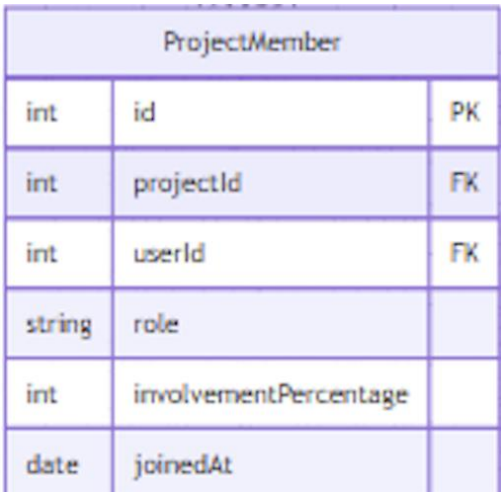


Рисунок 2.6 – Структура таблиці ProjectMembers для відображення складу команди проєкту та ролей учасників

Для забезпечення масштабованості та продуктивності застосовується стратегія шардування баз даних для сервісів з високим навантаженням. Шардування здійснюється за ідентифікатором проєкту, що дозволяє рівномірно розподілити навантаження між серверами бази даних. Реплікація даних застосовується для забезпечення високої доступності та відмовостійкості системи. Механізм реплікації налаштований за схемою "primary-standby" з автоматичним переключенням на резервний сервер при відмові основного:

```
CREATE TABLE projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    start_date DATE NOT NULL,
    planned_end_date DATE,
    actual_end_date DATE,
    status VARCHAR(50) NOT NULL,
    budget DECIMAL(15, 2),
    manager_id INTEGER NOT NULL REFERENCES users(id),
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP NOT NULL DEFAULT NOW()
);

--Приклад створення таблиці задач
CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    project_id INTEGER NOT NULL REFERENCES projects(id),
    parent_task_id INTEGER REFERENCES tasks(id),
    status VARCHAR(50) NOT NULL,
    priority VARCHAR(20) NOT NULL,
```

```
estimated_hours DECIMAL(8, 2),
planned_start_date DATE,
planned_end_date DATE,
actual_start_date DATE,
actual_end_date DATE,
completion_percentage INTEGER DEFAULT 0,
assignee_id INTEGER REFERENCES users(id),
created_at TIMESTAMP NOT NULL DEFAULT NOW(),
updated_at TIMESTAMP NOT NULL DEFAULT NOW()
```

Лістинг 2.1 – Фрагмент коду реплікації даних

Структура бази даних сервісу відстеження часу включає таблиці TimeEntries, Timesheets, TimeCategories. Таблиця TimeEntries зберігає окремі записи про витрачений час: ідентифікатор, ідентифікатор користувача, ідентифікатор задачі, ідентифікатор проєкту, дата, тривалість у хвилинах, опис виконаної роботи, категорія часу, метод введення (ручний, таймер, імпорт). Таблиця Timesheets агрегує записи про час для звітності: ідентифікатор, ідентифікатор користувача, період (тиждень, місяць), статус (чернетка, подано, затверджено, відхилено), ідентифікатор затверджувача, дата затвердження або відхилення, коментар. Таблиця TimeCategories визначає категорії часу для класифікації активностей: ідентифікатор, назва, опис, колір, ідентифікатор проєкту (якщо категорія проєктна).

Для забезпечення реалізації функцій аналізу продуктивності працівників розроблено структуру даних, яка дозволяє зберігати як первинні метрики, так і агреговані показники. Таблиця ProductivityMetrics зберігає первинні метрики продуктивності: ідентифікатор, ідентифікатор користувача, ідентифікатор проєкту, період (дата або тиждень), тип метрики (завершені задачі, витрачений час, виконання планів), числове значення. Таблиця PerformanceIndices зберігає агреговані показники ефективності: ідентифікатор, ідентифікатор користувача, ідентифікатор проєкту, період, тип індексу (індекс продуктивності, індекс якості, комплексний індекс), значення індексу, формула розрахунку.

Схема бази даних проектувалася з урахуванням необхідності ефективного виконання аналітичних запитів. Для таблиць з часовими даними застосовується автоматичне партиціювання за часовими періодами, що прискорює доступ до

даних та оптимізує процес архівації старих записів. Індекси створені для полів, які часто використовуються у запитах, зокрема зовнішні ключі та поля дат.

2.3. Проектування користувацького інтерфейсу

Користувацький інтерфейс веб-додатку управління проєктами та аналізу продуктивності працівників спроектований з урахуванням принципів user-centered design, з фокусом на зручність використання, інтуїтивну зрозумілість та ефективність взаємодії. Інтерфейс побудований за принципом адаптивного дизайну, що забезпечує оптимальне відображення на різних пристроях: десктоп, планшет, мобільний телефон.

Архітектура користувацького інтерфейсу базується на компонентному підході, що забезпечує повторне використання елементів та консистентність дизайну. Розроблено бібліотеку компонентів, яка включає базові елементи інтерфейсу: кнопки, поля введення, таблиці, картки, модальні вікна. Компоненти реалізують єдиний стиль дизайну з використанням Material Design принципів, що забезпечує узгодженість візуального представлення та поведінки елементів [1].

Навігаційна структура веб-додатку складається з головного меню, яке забезпечує доступ до основних розділів системи: Дашборд, Проєкти, Задачі, Час, Звіти, Налаштування. Бічне меню забезпечує навігацію всередині розділів та швидкий доступ до елементів, з якими користувач працював останнім часом. Контекстне меню надає доступ до операцій, специфічних для поточного екрану або вибраного елемента. Верхня панель містить пошукову систему, сповіщення, швидкі дії та меню користувача (див. рис. 2.7).

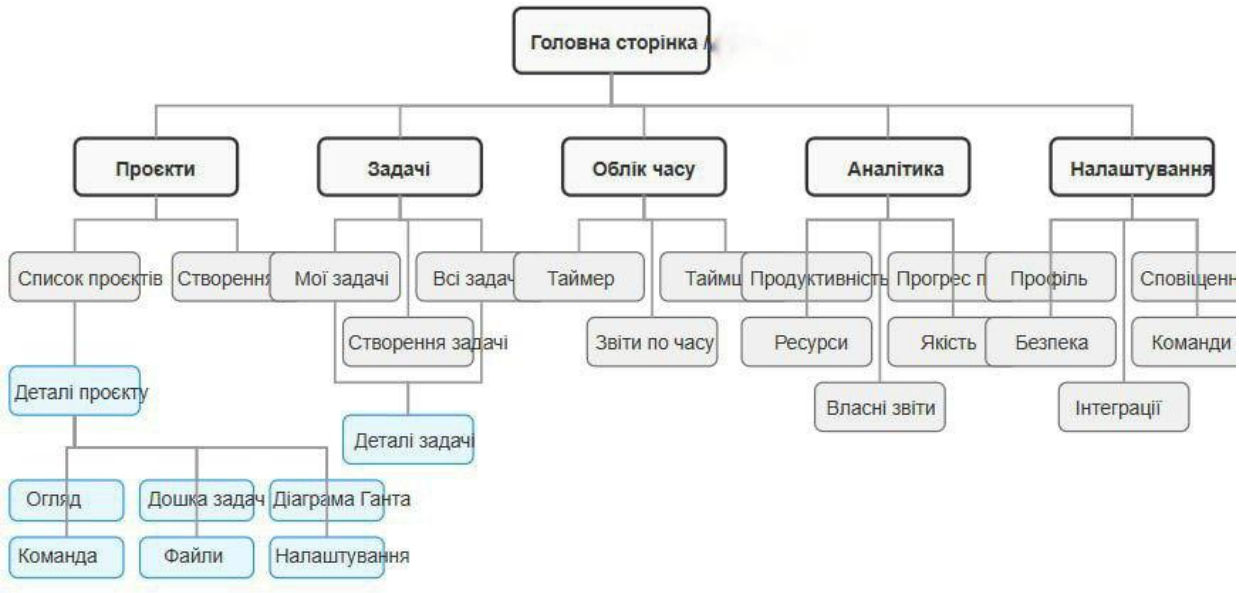


Рис 2.7 – Структура навігації веб-додатку

Дашборд спроектований як персоналізований екран з віджетами, які відображають ключову інформацію для поточного користувача: активні проєкти, задачі з високим пріоритетом, метрики продуктивності, останні активності.

Користувач може налаштовувати склад віджетів та їх розташування відповідно до своїх потреб (див. рис. 2.8).

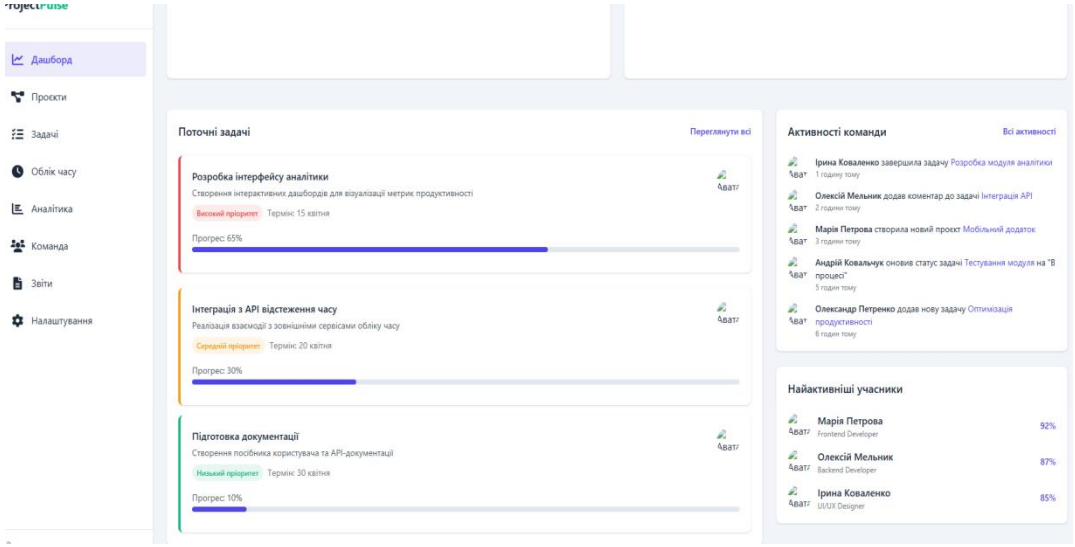


Рис 2.8 – Дашборд проєкту

Інтерфейс управління проєктами включає список проєктів з фільтрацією та сортуванням, сторінку деталей проєкту з вкладками для різних аспектів (огляд,

задачі, команда, файли, звіти), редактор структури робіт з візуалізацією ієрархії задач та залежностей, діаграми Ганта для візуалізації розкладу проєкту.

Інтерфейс управління задачами забезпечує різні представлення: список задач з фільтрацією, канбан-дошка для візуалізації робочого процесу, календарне представлення для планування.

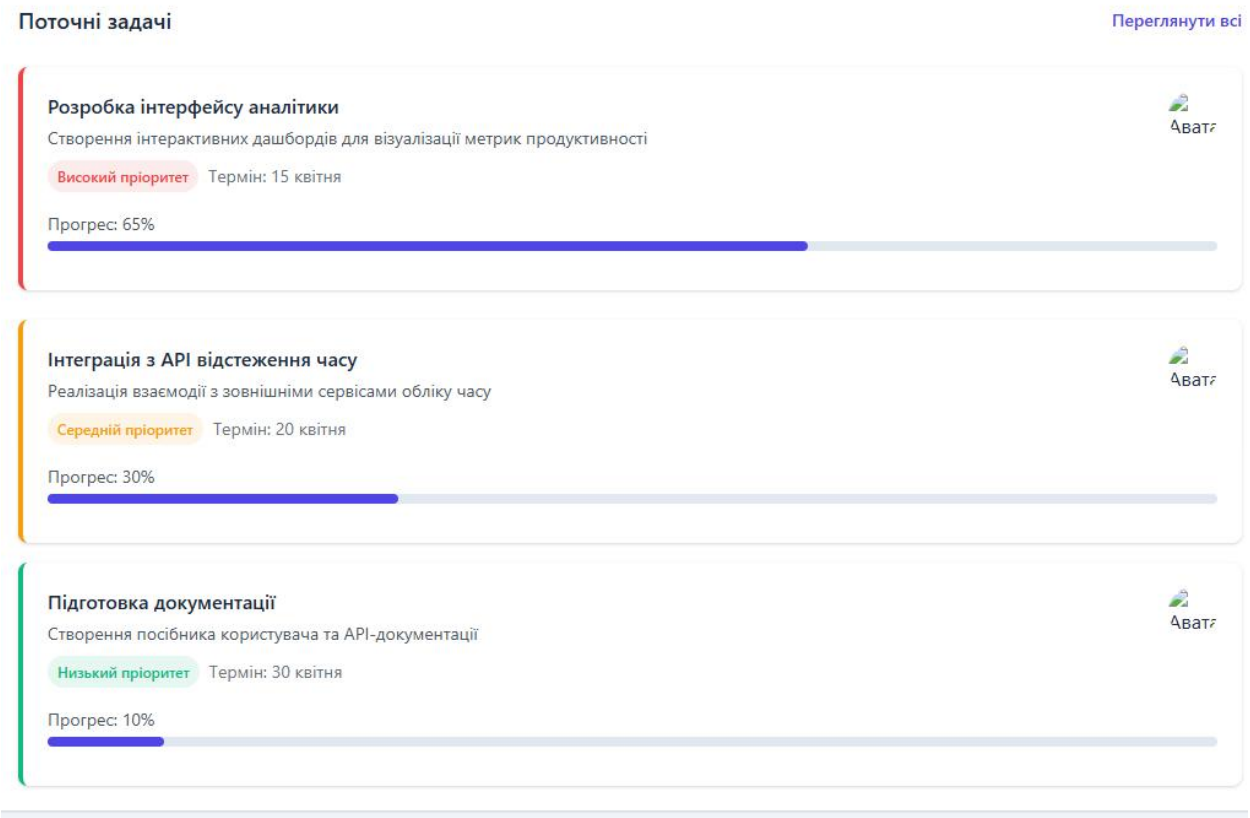


Рис 2.9 – Поточні задачі

Сторінка задачі містить всю необхідну інформацію про задачу: заголовок, опис, статус, пріоритет, виконавець, терміни, прогрес, витрачений час, коментарі, прикріплені файли. Реалізовано інлайн-редагування полів задачі без необхідності переходу на окремий екран редагування. Історія змін задачі відображається у вигляді стрічки активностей, що забезпечує прозорість процесу виконання:

```
using System;

< div class= "project-card" >
  < div class= "project-card__header" >
    < h3 class= "project-card__title" > Назва проєкту </ h3 >
    < div class= "project-card__status" > Статус </ div >
  </ div >
```

```
  < div class= "project-card__body" >
    < div class= "project-card__description" > Опис проєкту </ div >
    < div class= "project-card__metrics" >
      < div class= "project-card__metric" >
        < span class= "project-card__metric-value" > 75 %</ span >
        < span class= "project-card__metric-label" > Прогрес </ span >
      </ div >
      < div class= "project-card__metric" >
        < span class= "project-card__metric-value" > 15 / 20 </ span >
        < span class= "project-card__metric-label" > Задачі </ span >
      </ div >
    </ div >
  </ div >
  < div class= "project-card__footer" >
    < div class= "project-card__team" >
      < !--Аватари учасників проєкту -->
    </div>
    <div class= "project-card__dates" >
      < span class= "project-card__date" > Початок: 01.03.2023 </ span >
      < span class= "project-card__date" > Кінець: 30.06.2023 </ span >
    </ div >
  </ div >
</ div >
```

Лістинг 2.2 – Фрагмент коду із проєкту

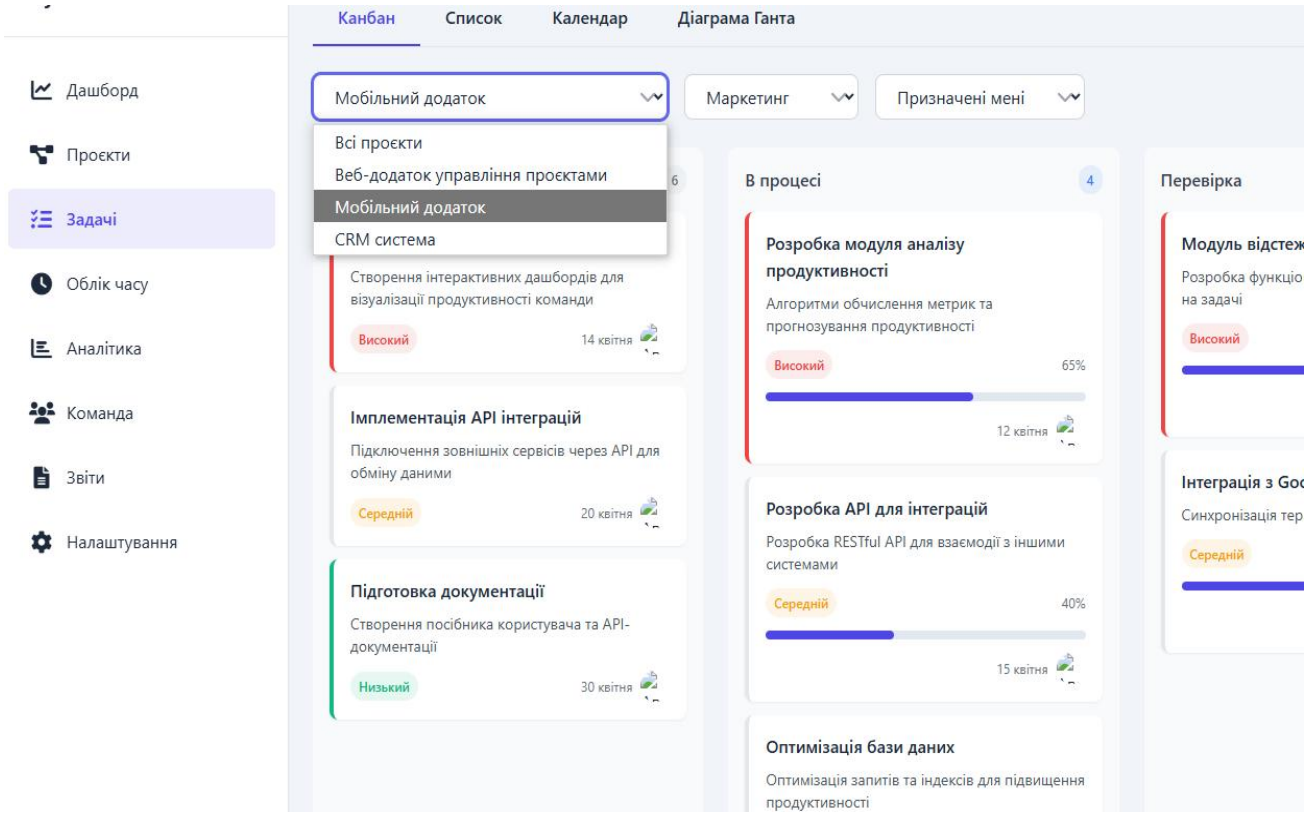


Рис. 2.10 – Сторінка «Задачі»

Інтерфейс відстеження часу включає таймер для запису часу в реальному режимі, форму ручного введення часових записів, таймшити для звітування про використаний час з можливістю затвердження керівником. Реалізовано візуалізацію розподілу часу по проєктах, задачах та категоріях активностей.

Інтерфейс аналізу продуктивності забезпечує візуалізацію метрик продуктивності через різні типи діаграм та графіків. Реалізовано дашборди продуктивності для різних рівнів: індивідуальний, командний, проєктний, організаційний. Інтерактивні фільтри дозволяють аналізувати дані у різних розрізах: за періодами, проєктами, ролями, типами задач.

Зворотний зв'язок з користувачем реалізований через систему сповіщень та повідомлень про помилки. Критичні помилки відображаються в модальних вікнах, некритичні — у вигляді тостів, що зникають через певний час. Сповіщення про події в системі (нова задача, коментар, наближення дедлайну) відображаються у верхній панелі та можуть бути налаштовані користувачем.

Розроблено систему підказок для допомоги користувачам у освоєнні інтерфейсу. Підказки відображаються при наведенні на елементи інтерфейсу та містять короткий опис функцій. Для нових користувачів реалізовано інтерактивні онбордінг-тури, які проводять по основних функціях системи.

Мобільний інтерфейс спроектований з урахуванням обмеженого простору екрану та особливостей сенсорного вводу.

2.4. Проєктування модулів управління проєктами

Модулі управління проєктами забезпечують функціональність для планування, виконання та контролю проєктних робіт. Модульна архітектура дозволяє гнучко налаштовувати функціонал системи відповідно до потреб конкретної організації та забезпечує розширюваність для додавання нових можливостей [6].

Модуль управління задачами реалізує функції створення та редагування задач, відстеження їх статусу, призначення виконавців, встановлення пріоритетів та термінів. Задачі можуть мати різні типи (задача, підзадача, дефект, ризик, епік) з відповідними наборами атрибутів. Система підтримує гнучкі налаштування

життєвого циклу задач через визначення статусів та дозволених переходів між ними (див. рис. 2.11-2.12).

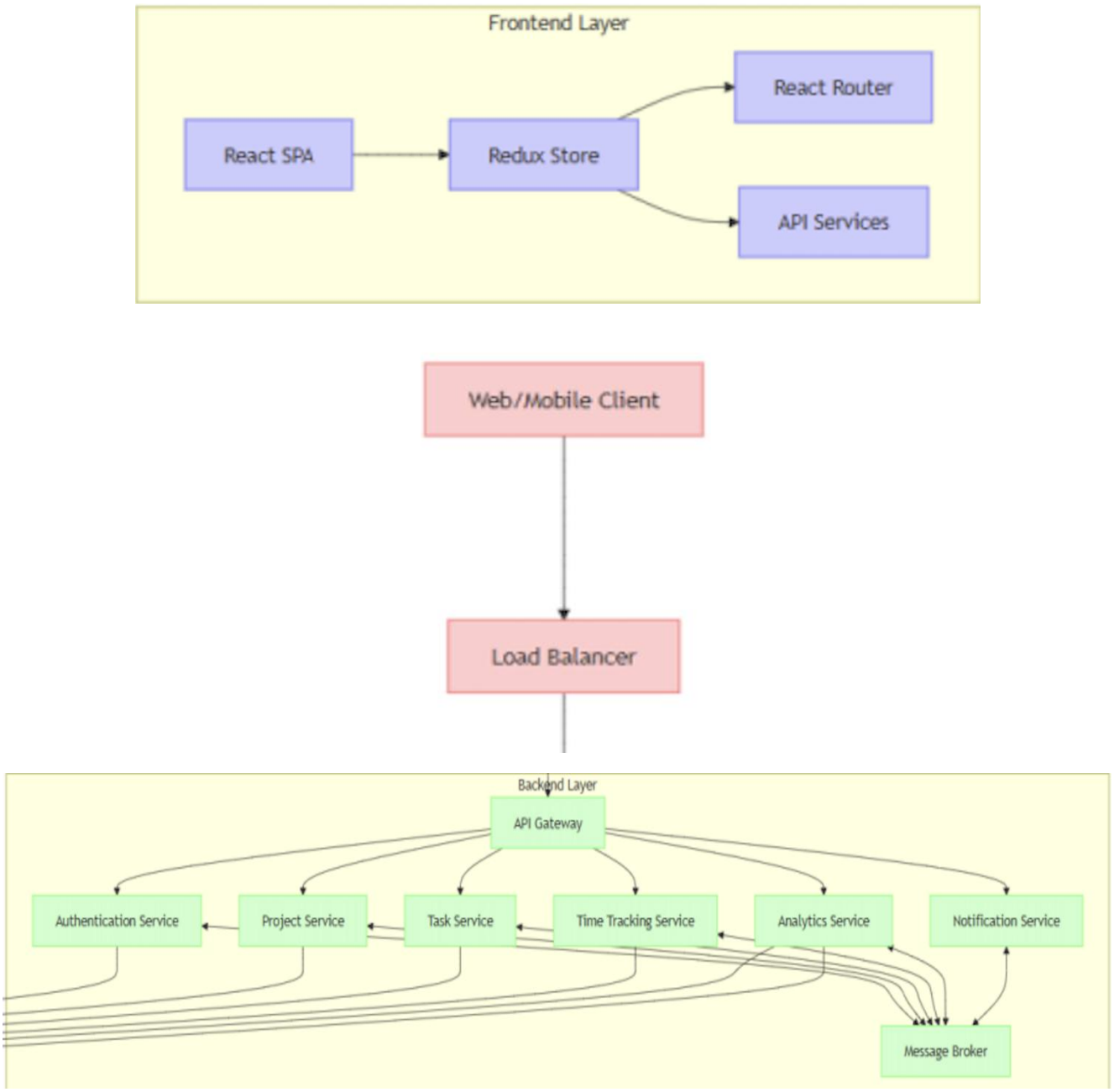


Рис. 2.11 - Архітектура веб-додатку (Backend, Frontend Layer)

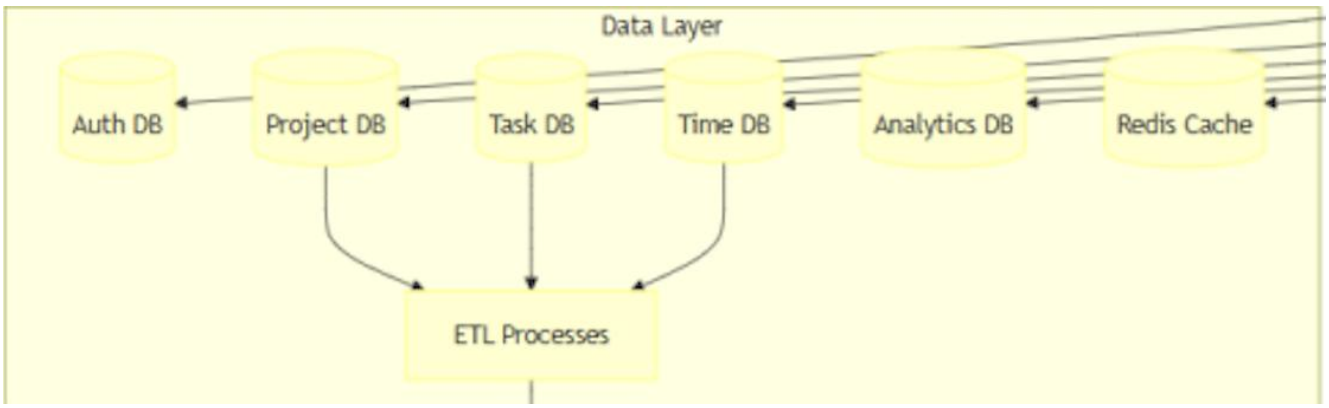


Рис. 2.12 – Архітектура веб-додатку (Data Layer)

Процес створення задачі включає визначення базових параметрів (назва, опис, тип, пріоритет), встановлення зв'язків з іншими задачами та проєктами, планування термінів виконання, оцінка трудомісткості, призначення виконавців та спостерігачів.

Відстеження прогресу задач здійснюється через систему статусів та відсотка виконання. Для задач з підзадачами відсоток виконання розраховується автоматично на основі прогресу підзадач з урахуванням їх ваги. Система забезпечує автоматичне оновлення батьківських задач при зміні статусу або прогресу дочірніх задач. Реалізовано механізм нагадувань про наближення термінів виконання задач та сповіщень про прострочені задачі:

```
using System.Threading.Tasks;

function calculateParentTaskProgress(taskId)
{
    const subtasks = getSubtasks(taskId);
    if (subtasks.length === 0)
    {
        return getTaskProgress(taskId);
    }

    let totalWeight = 0;
    let weightedProgress = 0;

    subtasks.forEach(subtask => {
        const weight = subtask.weight || 1;
        const progress = calculateParentTaskProgress(subtask.id); // Рекурсивний
виклик для підзадач

        totalWeight += weight;
        weightedProgress += progress * weight;
    });
```

```
    return totalWeight > 0 ? weightedProgress / totalWeight : 0;
}
```

Лістинг 2.3 – Фрагмент програми

Таблиця 2.2 – Функціональні можливості модулів управління проєктами

Модуль	Функціональні можливості	Взаємодія з іншими модулями	Технічна реалізація
Управління проєктами	Створення та налаштування проєктів; Управління структурою робіт; Планування термінів; Бюджетування;	Інтеграція з усіма модулями системи; Передача даних про проєкти до модулів задач та ресурсів	Node.js, Express, Redux для управління станом, Sequelize ORM для роботи з БД
Управління задачами	Створення та редагування задач; Відстеження статусу; Призначення виконавців; Відстеження прогресу	Отримання даних про проєкти від модуля проєктів; Передача даних до модулів відстеження часу та аналітики	React-компоненти для інтерфейсу, Redux для стану,
Управління ресурсами	Планування доступності ресурсів; Розподіл ресурсів за задачами; Аналіз завантаженості;	Отримання даних про проєкти та задачі; Передача даних до модуля аналітики	Алгоритми оптимізації розподілу ресурсів, D3.js для візуалізації завантаженості
Управління часом	Облік витрат часу; Таймери задач; Формування таймшитів; Аналіз часових витрат; Затвердження звітів	Отримання даних про задачі та ресурси; Передача даних про часові витрати до модуля аналітики	React-компоненти для таймерів, Redux для стану, REST API для збереження даних
Комунікації	Коментарі до задач; Обговорення проєктів; Сповіщення; Згадування користувачів; Інтеграція з месенджерами	Інтеграція з усіма модулями системи; Генерація сповіщень	WebSocket для миттєвих повідомлень, Push-сповіщення, інтеграція з e-mail

Модуль управління часом реалізує функції обліку витрат часу на задачі, формування та затвердження таймшитів, аналізу ефективності використання часу. Облік часу здійснюється через ручне введення або автоматичний таймер. Таймер

задачі забезпечує точний облік часу в реальному режимі з можливістю запису активностей та коментарів до часових записів.

Аналіз часових витрат забезпечує візуалізацію розподілу часу за проєктами, типами задач, категоріями активностей. Система виявляє відхилення фактичних витрат часу від запланованих та надає рекомендації щодо оптимізації часових витрат. Реалізовано функцію прогнозування часу завершення задач на основі поточного прогресу та історичних даних про швидкість виконання аналогічних задач. (Лістинг коду див. додаток В).

Модуль комунікацій забезпечує взаємодію учасників проєктів через коментарі до задач, обговорення проєктів, сповіщення про події в системі. Коментарі до задач дозволяють обговорювати деталі виконання, прикріплювати файли, згадувати інших користувачів через @mention. Обговорення проєктів реалізовано як тематичні дискусії з можливістю створення гілок обговорення та закріплення важливих повідомлень (див. рис. 2.13-2.14).

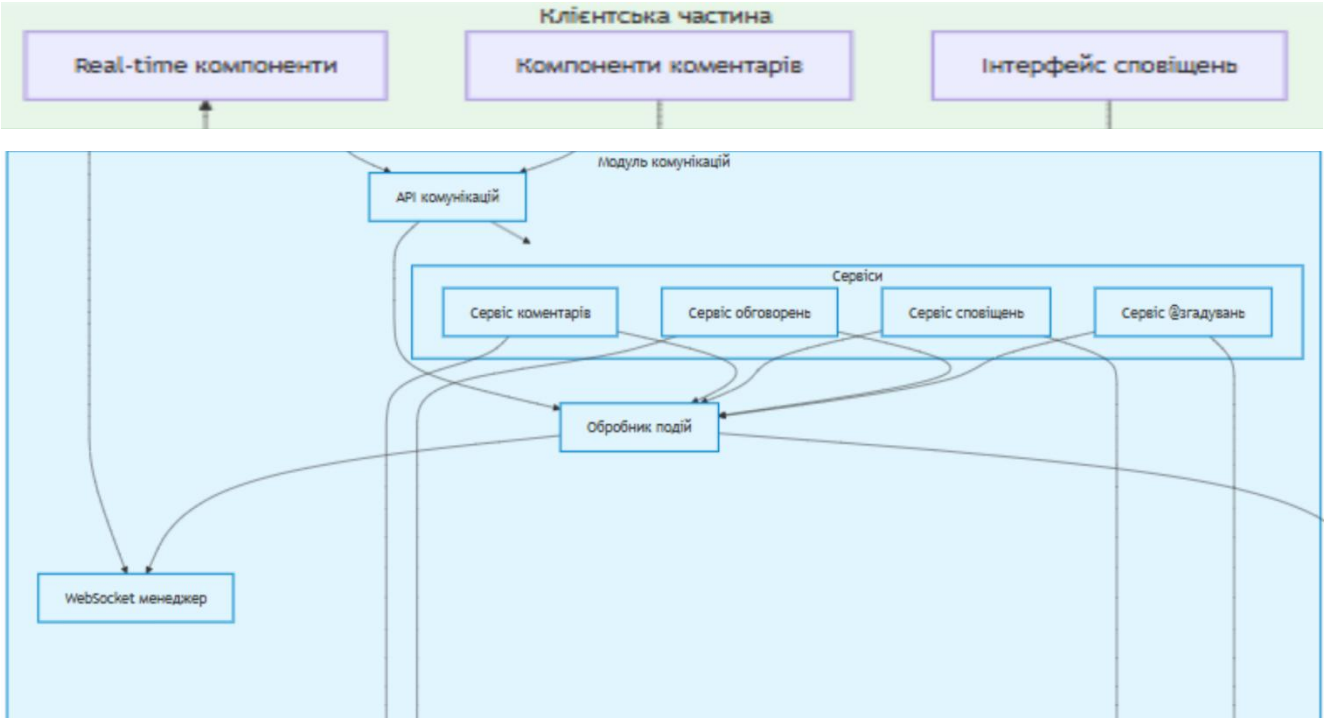


Рис. 2.13 – Архітектура модуля комунікації, клієнтської частини

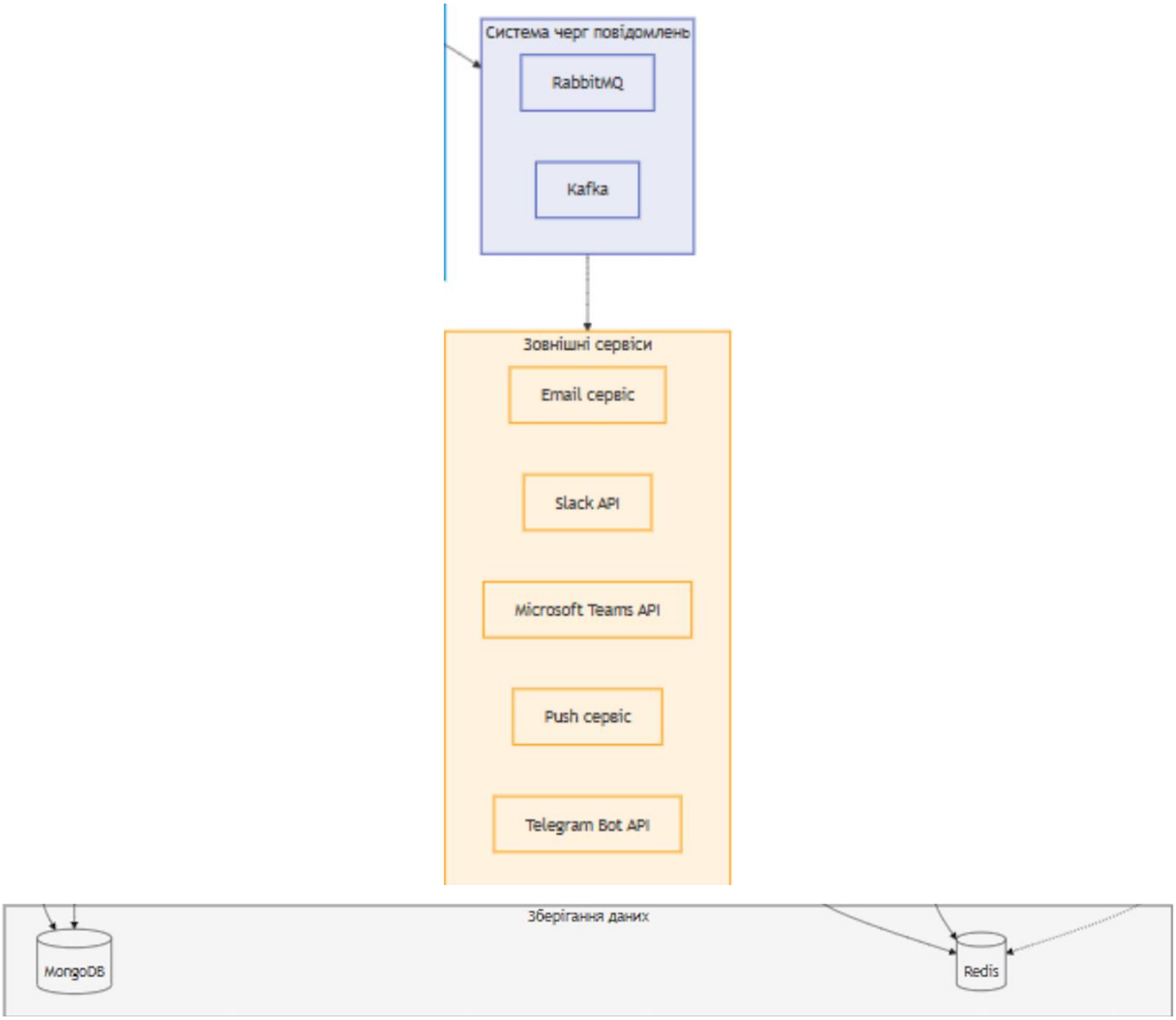


Рис. 2.14 – Архітектура Системи черг повідомлень, зовнішніх сервіси, зберігання даних

Система сповіщень забезпечує інформування користувачів про події в системі через різні канали: веб-інтерфейс, електронна пошта, push-сповіщення, інтеграція з месенджерами (Slack, Microsoft Teams).

Користувач може налаштувати типи подій, про які він хоче отримувати сповіщення, та канали доставки сповіщень

2.5. Проєктування модулів аналізу продуктивності

Модулі аналізу продуктивності призначені для збору, обробки та візуалізації даних про ефективність роботи працівників та команд. Архітектура модулів аналізу базується на принципах збору первинних метрик з різних джерел, їх агрегації та обчислення похідних показників, візуалізації результатів через інтерактивні дашборди [5].

Модуль збору метрик продуктивності забезпечує отримання первинних даних з різних модулів системи: задачі (кількість виконаних задач, дотримання термінів), час (витрати часу на задачі, ефективність використання робочого часу), якість (кількість дефектів, час на виправлення). Збір даних відбувається в реальному часі через систему подій, які генеруються при зміні стану задач, записів часу, коментарів та інших об'єктів системи.

Модуль аналізу метрик забезпечує обробку первинних метрик для отримання значущих показників продуктивності. Алгоритми аналізу враховують контекст виконання задач: складність, пріоритет, тип задачі, досвід виконавця. Реалізовано нормалізацію метрик для забезпечення справедливого порівняння між працівниками та командами з різним рівнем складності задач.

На основі первинних метрик обчислюються композитні показники продуктивності:

1. Індекс продуктивності (PI).
2. Декс дотримання термінів (TCI).
3. Індекс якості (QI).
4. Комплексний індекс ефективності (CEI).

Модуль прогнозування продуктивності реалізує алгоритми для передбачення майбутньої продуктивності на основі історичних даних. Використовуються методи статистичного аналізу та машинного навчання для виявлення трендів та патернів у показниках продуктивності. Система прогнозує

терміни завершення проєктів на основі поточної продуктивності команди та обсягу робіт, що залишилися [19].

Модуль візуалізації аналітики забезпечує представлення результатів аналізу через інтерактивні дашборди, графіки, діаграми та таблиці. Реалізовано різні типи візуалізацій для різних аспектів продуктивності: лінійні графіки для часових трендів, стовпчикові діаграми для порівняння показників, теплові карти для виявлення патернів, кругові діаграми для розподілу часу за категоріями [19].

Інтерактивність дашбордів забезпечується через фільтри за періодами, проєктами, командами, типами задач, що дозволяє аналізувати дані у різних розрізах. Реалізована можливість деталізації показників (drill-down) для переходу від агрегованих до детальних даних (Лістинг див. додаток В).

Модуль формування рекомендацій аналізує дані про продуктивність та генерує персоналізовані рекомендації для працівників та керівників щодо покращення ефективності. Рекомендації формуються на основі виявлених патернів продуктивності, порівняння з кращими практиками та визначення областей для розвитку [11].

Типи рекомендацій:

1. Оптимізація розподілу часу між різними типами задач
2. Покращення планування для зменшення перевантажень
3. Рекомендації щодо пріоритизації задач
4. Виявлення необхідності додаткового навчання в певних областях
5. Рекомендації щодо покращення командної взаємодії

Система формування рекомендацій використовує алгоритми машинного навчання для виявлення патернів успішного виконання задач та рекомендації подібних підходів для аналогічних задач у майбутньому. Рекомендації персоналізуються з урахуванням індивідуальних особливостей працівника, його сильних та слабких сторін, історії продуктивності [13].

Модуль формування звітів забезпечує генерацію структурованих звітів про продуктивність для різних рівнів управління. Звіти можуть бути сформовані за

запитом або налаштовані на автоматичну генерацію за розкладом. Реалізована можливість експорту звітів у різних форматах: PDF, Excel, CSV, HTML.

Типи звітів:

1. Індивідуальний звіт про продуктивність працівника
2. Звіт про продуктивність команди
3. Звіт про прогрес проєкту
4. Звіт про розподіл часу
5. Звіт про дотримання термінів
6. Аналітичний звіт з трендами продуктивності

Конструктор звітів дозволяє користувачам створювати власні шаблони звітів з вибором показників, періодів аналізу, типів візуалізації.

Модуль безпеки та приватності забезпечує захист даних про продуктивність та контроль доступу до аналітичної інформації. Реалізовано гранулярну систему прав доступу, яка визначає, які користувачі мають доступ до яких метрик та звітів. Система забезпечує анонімізацію даних при формуванні порівняльних звітів для запобігання негативного впливу на психологічний клімат у команді.

Модуль індивідуальних цілей та показників ефективності (KPI) дозволяє визначати персональні цілі для працівників та відстежувати їх досягнення. Цілі можуть бути пов'язані з проєктами або загальним розвитком працівника. Система підтримує методологію OKR (Objectives and Key Results) для визначення цілей та вимірюваних результатів [12].

Процес визначення цілей включає встановлення мети, визначення ключових результатів, встановлення термінів досягнення, узгодження з керівником. Відстеження прогресу відбувається автоматично на основі даних з інших модулів системи або через ручне оновлення статусу. Система генерує нагадування про наближення термінів та сповіщає про досягнення ключових результатів.

Модуль інтеграції з HR-системами забезпечує обмін даними про продуктивність з системами управління персоналом для використання у процесах оцінки персоналу, планування розвитку, формування компенсаційних пакетів.

3. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗУ ПРОДУКТИВНОСТІ ПРАЦІВНИКІВ Вибір технологій та засобів розробки

Процес вибору технологій для реалізації веб-додатку управління проєктами та аналізу продуктивності працівників базувався на критеріях масштабованості, продуктивності, безпеки, швидкості розробки та підтримки. Врахована специфіка системи, яка передбачає оперування великими обсягами даних, складні аналітичні обчислення, взаємодію багатьох користувачів у реальному часі.

Архітектура системи реалізована на базі мікросервісного підходу з використанням контейнеризації, що забезпечує незалежність розгортання окремих компонентів, гнучке масштабування та ізоляцію відмов. Для розробки бекенд-частини вибраний Node.js через його асинхронну природу, ефективність обробки великої кількості одночасних з'єднань та швидкість розробки. Express.js використовується як веб-фреймворк для створення REST API через його мінімалістичність, гнучкість та широку екосистему додаткових модулів [1].

Для реалізації функціональності реального часу використано WebSocket через Socket.io, що забезпечує двосторонній зв'язок між клієнтом та сервером для миттєвого оновлення даних, сповіщень та комунікації між користувачами. Для управління асинхронними операціями застосовано async/await та Promise API, що покращує читабельність коду та спрощує обробку помилок.

Систему управління базами даних обрано із застосуванням поліглотного підходу. PostgreSQL використовується для зберігання структурованих даних з чіткими зв'язками між сутностями, забезпечуючи атомарність, послідовність, ізоляцію, довговічність (ACID). MongoDB застосовується для зберігання неструктурованих даних та динамічних конфігурацій, забезпечуючи гнучкість схеми даних. Redis використовується для кешування, зберігання сесій та реалізації системи черг повідомлень.

Для роботи з базами даних вибрані ORM та ODM бібліотеки: Sequelize для PostgreSQL та Mongoose для MongoDB. Ці бібліотеки забезпечують абстракцію доступу до даних, автоматичну валідацію, міграції схем, типізацію даних на рівні

моделей. Bull реалізує систему черг повідомлень на базі Redis для обробки асинхронних задач та забезпечення надійної доставки повідомлень між сервісами.

Фронтенд-частина розроблена з використанням React.js для побудови компонентного інтерфейсу з декларативним підходом до рендерингу. TypeScript застосований для статичної типізації, що підвищує якість коду, забезпечує автодоповнення в редакторах, зменшує кількість помилок під час розробки. Redux використовується для управління станом додатку, забезпечуючи централізоване сховище даних та предиктовану поведінку.

Для стилізації компонентів використано підхід CSS-in-JS через бібліотеку styled-components, що дозволяє створювати компоненти з інкапсульованими стилями, підтримує динамічну зміну стилів, теми та адаптивний дизайн. Material-UI застосовано для використання готових компонентів з Material Design, що забезпечує узгоджений візуальний стиль та інтерактивність елементів інтерфейсу.

Для візуалізації даних та побудови інтерактивних дашбордів використана бібліотека D3.js, яка забезпечує повний контроль над візуалізацією та підтримує широкий спектр типів діаграм та графіків. Recharts застосовано для спрощеної інтеграції D3 з React, що прискорює розробку стандартних типів візуалізацій.

Контейнеризація здійснена з використанням Docker, що забезпечує стандартизацію середовища розробки та розгортання, ізоляцію залежностей, спрощує процес масштабування. Оркестрація контейнерів реалізована через Kubernetes, що забезпечує автоматичне масштабування, самовідновлення сервісів після збоїв, балансування навантаження, моніторинг стану системи.

Безперервна інтеграція та доставка (CI/CD) реалізована з використанням GitLab CI, що автоматизує процеси тестування, збірки та розгортання. Автоматичні тести розроблені з використанням Jest для модульного тестування, Enzyme для тестування React-компонентів, Cypress для end-to-end тестування. Моніторинг та логування реалізовані через Prometheus для збору метрик, Grafana для візуалізації показників, ELK Stack для централізованого збору та аналізу логів (Лістинг див. додаток В) [17].

Ефективність вибраних технологій підтверджується результатами навантажувального тестування, проведеного на етапі бета-версії системи. Система демонструє стабільну роботу при одночасному доступі 1000+ користувачів з часом відгуку менше 500 мс для стандартних операцій [15].

3.2. Розробка серверної частини веб-додатку

Серверна частина веб-додатку реалізована відповідно до принципів мікросервісної архітектури, де кожен сервіс відповідає за певний аспект функціональності системи. Розробка бекенду здійснена на платформі Node.js з використанням фреймворку Express.js, що забезпечує швидку обробку запитів, ефективне використання ресурсів сервера та легкість горизонтального масштабування [2].

Архітектура серверної частини складається з наступних основних мікросервісів: API Gateway, сервіс автентифікації та авторизації, сервіс управління проєктами, сервіс управління задачами, сервіс відстеження часу, сервіс аналітики продуктивності, сервіс сповіщень, сервіс інтеграцій [6].

API Gateway виступає єдиною точкою входу для клієнтських запитів, забезпечуючи маршрутизацію запитів до відповідних мікросервісів, балансування навантаження, кешування, централізовану автентифікацію та авторизацію. Реалізовано механізм агрегації даних, що дозволяє клієнту отримати дані з кількох мікросервісів через один запит, зменшуючи кількість мережевих взаємодій (Лістинг див. додаток Г) [6].

Сервіс автентифікації та авторизації реалізує функції реєстрації користувачів, входу в систему, управління користувачами, ролями та дозволами. Для автентифікації застосований механізм JWT (JSON Web Tokens), що забезпечує stateless автентифікацію та спрощує масштабування. Токен містить

інформацію про користувача, його ролі та дозволи, що дозволяє реалізувати авторизацію на рівні API Gateway без додаткових запитів до бази даних.

Сервіс управління проєктами забезпечує функції створення та налаштування проєктів, управління учасниками проєктів, формування структури робіт, визначення залежностей між задачами. Реалізовано RESTful API для всіх операцій з проєктами, з підтримкою фільтрації, сортування та пагінації результатів (лістинг див. додаток В).

Сервіс управління задачами реалізує функції створення та редагування задач, відстеження їх статусу, призначення виконавців, встановлення залежностей між задачами. Реалізовано механізм автоматичного оновлення батьківських задач при зміні статусу або прогресу дочірніх задач, а також систему нагадувань про наближення термінів виконання (лістиг див. додаток В).

Для забезпечення стійкості системи до відмов реалізовано патерн Circuit Breaker, який запобігає каскадним відмовам при несправності окремих сервісів. При виявленні проблем із сервісом, Circuit Breaker перемикається в стан "відкритий", швидко відхиляючи запити замість очікування таймаутів, та автоматично відновлює з'єднання після нормалізації роботи сервісу [4].

3.3. Розробка клієнтської частини веб додатку

Клієнтська частина веб-додатку реалізована як Single Page Application (SPA) з використанням бібліотеки React.js, що забезпечує швидкий відгук інтерфейсу та оптимальну взаємодію з користувачем. Архітектура фронтенд-частини базується на компонентному підході з розділенням відповідальності між компонентами представлення, контейнерами та сервісами [20].

Структура проєкту фронтенд-частини організована за функціональним принципом, де кожен модуль системи (проєкти, задачі, час, аналітика) має власну

директорію з компонентами, контейнерами, редюсерами, діями та типами. Такий підхід забезпечує ізоляцію функціональності та спрощує масштабування додатку.

```
      using static System.Formats.Asn1.AsnWriter;
using System.Reflection;
using System.Threading.Tasks;

src /
├── assets /              # Статичні ресурси (зображення, шрифти, глобальні стилі)
├── components /          # Спільні компоненти
│   ├── common /          # Базові компоненти (кнопки, поля введення, модальні
вікна)
│   └── layout /          # Компоненти макету (шапка, бічне меню, футер)
├── containers /          # Контейнери, що з'єднують компоненти з Redux
├── hooks /               # Користувацькі React-хуки
├── modules /             # Функціональні модулі системи
│   ├── auth /            # Модуль автентифікації
│   ├── projects /        # Модуль управління проєктами
│   ├── tasks /           # Модуль управління задачами
│   ├── time /            # Модуль відстеження часу
│   └── analytics /       # Модуль аналітики
├── services /            # Сервіси для взаємодії з API
├── store /               # Конфігурація Redux-сховища
├── utils /               # Утиліти та хелпери
      └── App.js               # Головний компонент додатку
```

Лістинг 3.5 – Структура проєкту фронтенд-частини

Управління станом додатку реалізовано з використанням Redux, що забезпечує централізоване сховище даних та передбачувані зміни стану. Для оптимізації роботи з асинхронними діями використано Redux Thunk, що дозволяє створювати дії, які взаємодіють з API та диспетчеризують інші дії залежно від результату (лістинг див. додаток В).

Для взаємодії з API розроблено сервісний шар, який інкапсулює логіку роботи з REST API, обробку помилок та кешування даних. Використано axios для HTTP-запитів з налаштованими інтерцепторами для автоматичного додавання токенів автентифікації та обробки помилок (лістинг див. додаток В).

Компоненти користувацького інтерфейсу розроблені з використанням функціональних компонентів React та хуків, що забезпечує лаконічний код та ефективне управління станом компонентів. Material-UI застосовано для базових компонентів інтерфейсу, з кастомізацією через систему тем для досягнення унікального візуального стилю.

Для оптимізації продуктивності клієнтської частини реалізовано механізми мемоізації компонентів (React.memo), кешування запитів до API, віртуалізацію списків для ефективного рендерингу великих наборів даних. Компоненти розбито на більш дрібні зі специфічною відповідальністю, що забезпечує можливість перевикористання та покращує продуктивність завдяки оптимізації оновлень.

3.4. Реалізація модулів аналізу продуктивності

Модулі аналізу продуктивності реалізують функціональність для збору, аналізу та візуалізації даних про ефективність роботи працівників та команд. Реалізація базується на архітектурі, розробленій на етапі проектування, з відповідними покращеннями виявленими під час розробки [18].

Модуль збору метрик продуктивності реалізує механізми отримання первинних даних з різних джерел: задачі (кількість завершених задач, дотримання термінів), час (витрати часу на задачі, ефективність використання робочого часу), якість (кількість дефектів, час на виправлення). Збір даних відбувається як у реальному часі через обробку подій, так і через регулярне агрегування даних з різних сервісів.

Система подій реалізована з використанням шаблону publish-subscribe, де різні модулі системи генерують події при зміні стану об'єктів, а модуль збору метрик підписується на ці події та зберігає відповідні метрики. Такий підхід забезпечує слабку зв'язаність між модулями та можливість розширення системи без змін у існуючих компонентах.

Регулярне агрегування даних виконується за розкладом для обчислення похідних метрик, які вимагають аналізу великих обсягів даних за певний період. Наприклад, щоденно розраховуються показники продуктивності команд, щотижнево – тренди продуктивності, щомісячно – порівняльний аналіз ефективності різних відділів.

Модуль аналізу метрик реалізує алгоритми обробки первинних даних для отримання показників продуктивності. Реалізовано обчислення наступних ключових метрик:

1. Індекс продуктивності (PI).
2. Індекс дотримання термінів (TCI).
3. Індекс якості (QI).
4. Комплексний індекс ефективності (CEI).

Для аналізу трендів продуктивності реалізовано алгоритми статистичного аналізу часових рядів, що дозволяють виявляти тенденції, сезонні коливання, аномалії в показниках продуктивності. Система автоматично ідентифікує значні відхилення від звичайного рівня продуктивності та пропонує можливі пояснення на основі контекстних даних (зміни в команді, зміни в процесах, зовнішні фактори) (див. рис. 3.1).

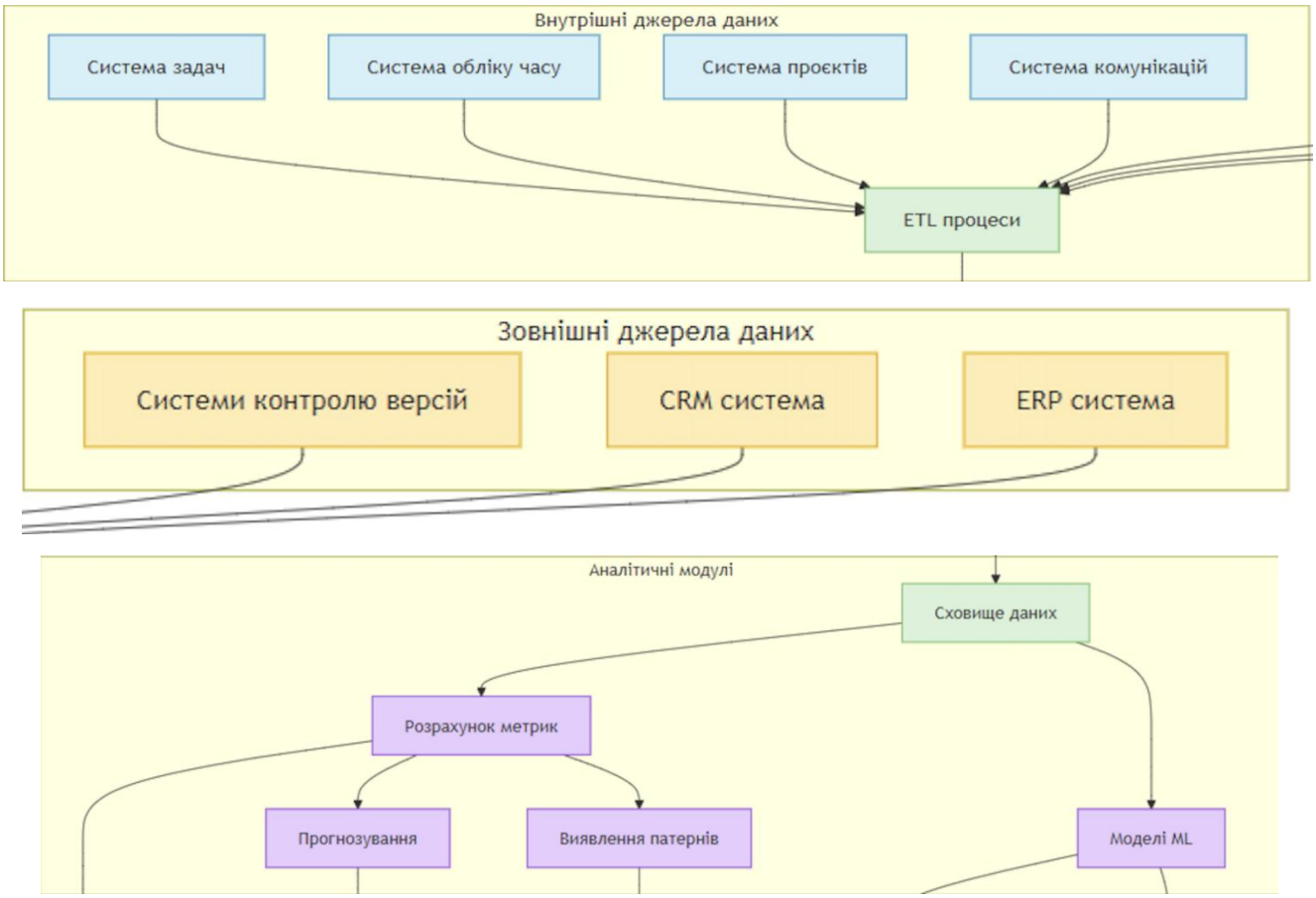


Рисунок 3.1 - Архітектура внутрішніх та зовнішніх джерел даних та аналітичних модулів

Рис 3.2 – Архітектура інтеграцій

Модуль прогнозування реалізує алгоритми передбачення майбутньої продуктивності та термінів завершення проєктів на основі історичних даних. Реалізовано методи статистичного прогнозування (ARIMA, експоненціальне згладжування) та методи машинного навчання (регресійні моделі, нейронні мережі) для різних сценаріїв прогнозування.

Система прогнозує терміни завершення проєктів з урахуванням поточної продуктивності команди, залишкового обсягу робіт, історичних даних про виконання аналогічних проєктів.

Модуль візуалізації аналітики реалізує інтерфейси для представлення результатів аналізу через інтерактивні дашборди, графіки, діаграми та таблиці. Реалізовано різні типи візуалізацій для різних аспектів продуктивності: лінійні графіки для часових трендів, стовпчикові діаграми для порівняння показників, теплові карти для виявлення патернів, кругові діаграми для розподілу часу за категоріями [19].

3.5. Методологія тестування веб-додатку

Тестування веб-додатку управління проєктами та аналізу продуктивності працівників здійснювалося за комплексним підходом, що охоплює різні рівні та типи тестування для забезпечення якості та надійності системи. Методологія тестування базувалася на сучасних практиках розробки програмного забезпечення та враховувала специфіку мікросервісної архітектури додатку [2].

На етапі модульного тестування (Unit Testing) застосовано підхід розробки через тестування (TDD), де спочатку створювалися тести для окремих компонентів, а потім реалізовувалася їх функціональність. Для серверної частини

використано фреймворк Jest, що дозволив перевірити функціональність кожного сервісу окремо, імітуючи взаємодію з іншими компонентами. Для клієнтської частини тестування React-компонентів здійснювалося з використанням React Testing Library, що дозволило перевірити правильність рендерингу компонентів та їх взаємодію з користувачем [19].

Інтеграційне тестування забезпечило перевірку взаємодії між різними мікросервісами та клієнтською частиною. Для тестування API використано Postman та Supertest, що дозволило автоматизувати перевірку HTTP-запитів та відповідей. Особлива увага приділена тестуванню сценаріїв, де задіяно кілька мікросервісів, зокрема процесів створення проєкту, призначення задач, оновлення статусів та розрахунку метрик продуктивності [4].

Системне тестування проводилося на повністю розгорнутому середовищі з використанням реалістичних даних. Тестувалася взаємодія всіх компонентів системи, перевірялася коректність обчислення аналітичних показників, генерування звітів, відображення дашбордів. Особлива увага приділялася перевірці правильності обчислення композитних показників продуктивності, які залежать від даних з різних сервісів [8].

Для забезпечення якості користувацького інтерфейсу проведено end-to-end тестування з використанням Cypress, що дозволило імітувати дії користувача та перевірити повний цикл взаємодії з системою. Розроблено набір тестових сценаріїв, що відповідають типовим бізнес-процесам: створення проєкту, додавання задач, призначення виконавців, відстеження прогресу, аналіз продуктивності [9].

Навантажувальне тестування проводилося з використанням Apache JMeter для визначення максимальної кількості одночасних користувачів, які система може обслуговувати без деградації продуктивності. Тестувалися ключові операції системи: отримання даних для дашбордів, збереження часових записів, оновлення статусів задач, генерування звітів. Результати навантажувального тестування використано для оптимізації конфігурації розгортання системи та налаштування параметрів масштабування [13].

Тестування безпеки включало статичний аналіз коду з використанням інструментів SonarQube та OWASP Dependency Check для виявлення вразливостей, перевірку захисту від типових веб-атак (SQL-ін'єкції, XSS, CSRF), аудит системи автентифікації та авторизації. Проведено тестування на проникнення для імітації дій зловмисника та виявлення потенційних вразливостей [11].

3.6. Результати тестування та виправлення помилок

Процес тестування веб-додатку виявив ряд проблем різного ступеня критичності, які були систематизовані та успішно усунені. За результатами модульного тестування виявлено 127 потенційних проблем, серед яких 36% стосувалися серверної частини, 42% - клієнтської частини, та 22% - взаємодії між компонентами системи. Розподіл виявлених помилок за типами представлено на рисунку 3.2.

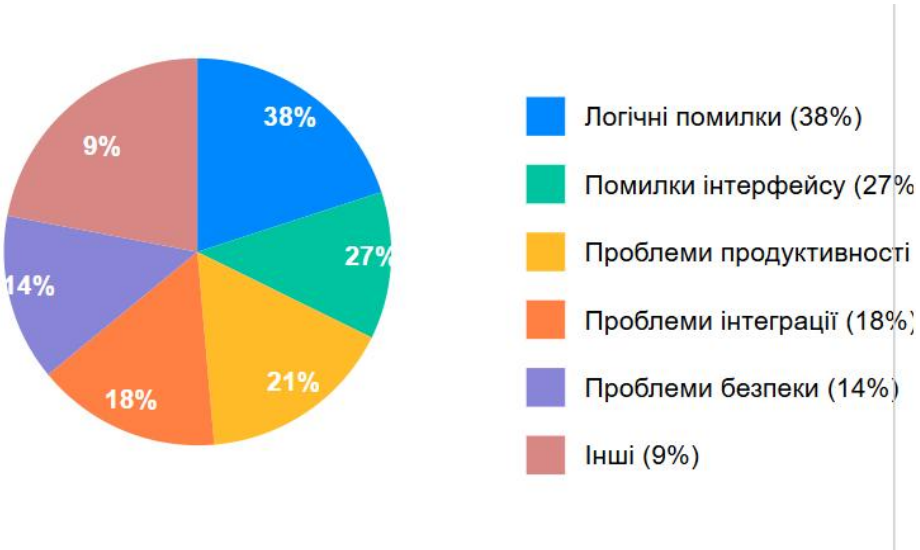


Рисунок 3.3 – Основні проблеми

Найбільш критичні проблеми були пов'язані з обробкою конкурентних запитів до мікросервісів, що призводило до неузгодженості даних при

одночасному оновленні одного об'єкта кількома користувачами. Для усунення цієї проблеми реалізовано механізм оптимістичного блокування з використанням версіонування об'єктів, що дозволило виявляти конфлікти змін та коректно їх обробляти [4].

Інтеграційне тестування виявило проблеми із синхронізацією даних між сервісами, особливо при виникненні тимчасових збоїв у комунікації. Для підвищення надійності взаємодії між сервісами реалізовано механізм компенсуючих транзакцій та введено чергу повідомлень RabbitMQ для гарантованої доставки подій між сервісами [14].

При навантажувальному тестуванні виявлено потенційні вузькі місця у продуктивності системи. Найбільше навантаження створювало генерування аналітичних звітів для великих проєктів та обчислення метрик продуктивності за тривалі періоди. Для оптимізації продуктивності реалізовано механізм кешування результатів складних запитів та попереднє обчислення агрегованих показників за типовими періодами (день, тиждень, місяць).

Ефективність виправлення помилок оцінювалася за часом, необхідним для усунення дефектів різної критичності. Як видно з рисунку 4.2, середній час виправлення критичних помилок становив 4,2 години, серйозних - 8,7 годин, помірних - 16,4 годин, а незначних - 24,1 години.

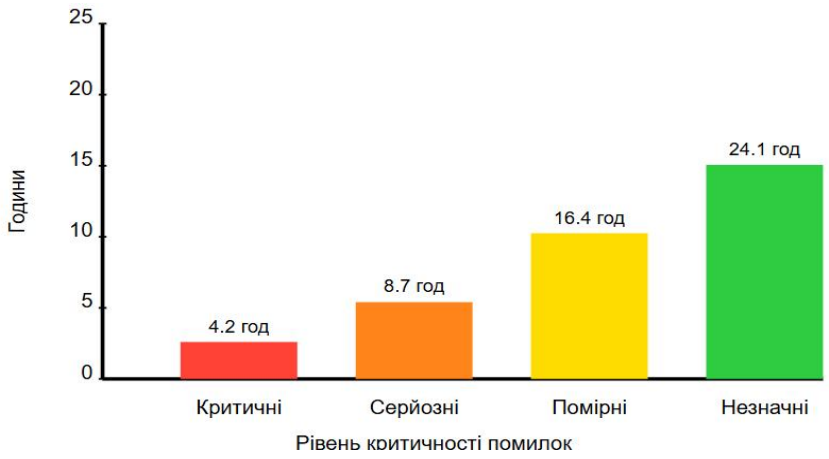


Рисунок 3.4 – Середній час виявлення помилок

Тестування користувацького інтерфейсу виявило проблеми з відображенням інформації на мобільних пристроях, зокрема складність роботи з дашбордами

аналітики на екранах з малою діагональлю. Проблему вирішено через переробку макетів мобільної версії з фокусом на відображення ключових метрик та спрощення навігації [5].

Рисунок 3.4 демонструє динаміку виявлення та виправлення помилок протягом процесу тестування.

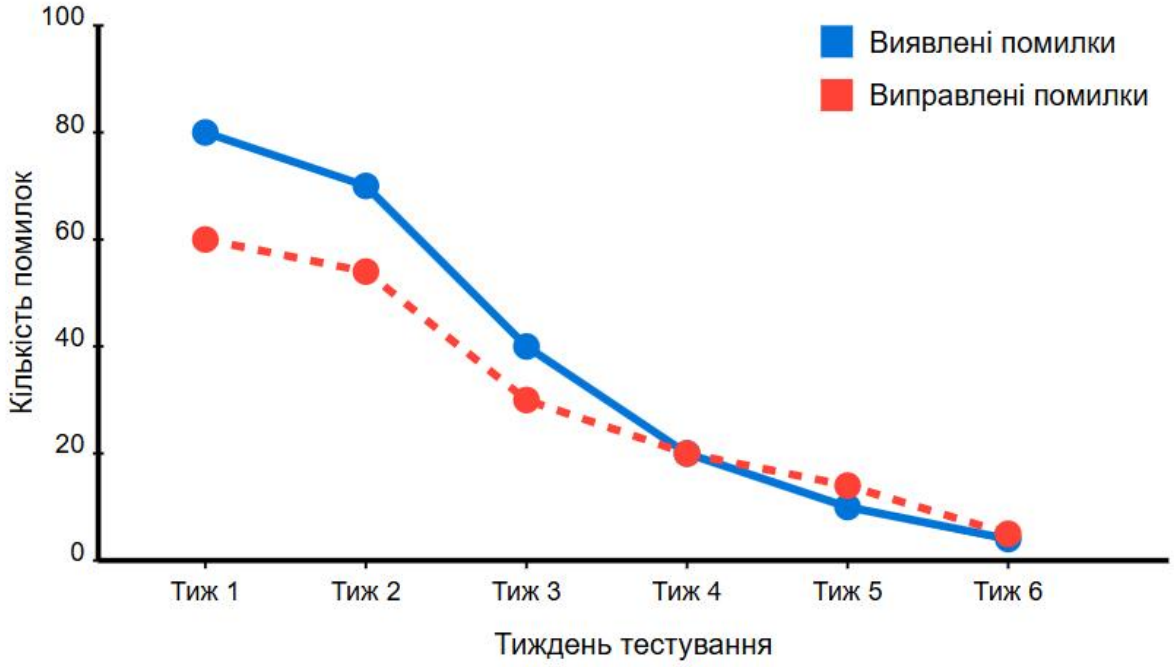


Рисунок 3.5 – Динаміка виявлення та виправлення помилок

За результатами тестування безпеки виявлено 14 потенційних вразливостей, серед яких 2 високого рівня, 5 середнього та 7 низького рівня. Всі виявлені вразливості були успішно усунені через оновлення залежностей, покращення механізмів валідації вхідних даних та посилення контролю доступу [5].

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1. Допомога при теплових і сонячних ударах

При виконанні професійних обов’язків в умовах дії високих температур (будівництво, робота в пральнях, котельнях, сільськогосподарських об’єктах, на шахтах) працівники зазнають значного теплового навантаження. Це може спричинити гострі стани, зокрема тепловий або сонячний удар. В обох випадках ключовим завданням для сторонньої особи, що перебуває поруч, є негайне припинення дії несприятливого чинника та початок дій з надання домедичної допомоги [23].

Основні ознаки теплового удару: підвищення температури тіла до 40–41 °C, почервоніння шкіри, її сухість, слабкість, втрата свідомості, прискорене дихання [21]. Сонячний удар супроводжується схожими симптомами, однак викликаний безпосереднім впливом сонячного проміння на відкриту поверхню голови та шийного відділу.

Алгоритм надання домедичної допомоги регламентовано [21]. По-перше, слід переконатись у власній безпеці та безпеці постраждалого. Далі здійснюється виклик бригади екстреної медичної допомоги. Постраждалого слід перемістити в тінь або охолоджене приміщення, зняти зайвий одяг, що перешкоджає тепловіддачі.

Після цього за можливості визначається температура тіла. Якщо вона перевищує 40 °C, слід застосувати методи активного охолодження — занурення у воду температури 18–26 °C до моменту зниження температури тіла нижче 39 °C. При неможливості повного занурення використовуються вологі компреси, обкладання холодовими пакетами, вентилювання поверхні тіла [21].

Якщо свідомість збережена — необхідно дати воду дрібними ковтками. У разі втрати свідомості слід перейти до дій відповідно до порядку надання домедичної допомоги при раптовій зупинці кровообігу [21].

Окрему увагу необхідно приділяти працівникам, які задіяні в роботах у закритих виробничих приміщеннях із підвищеним тепловим навантаженням. У таких випадках рекомендовано впроваджувати адаптовані технічні рішення, зокрема вентиляційні системи з контрольованим мікрокліматом, зони охолодження та регламентовані перерви на відпочинок у прохолодних приміщеннях [23].

Узагальнюючи, домедична допомога при теплових і сонячних ударах вимагає чіткої послідовності дій та знань щодо механізмів впливу високих температур на організм.

4.2. Організація безпечної роботи електроустановок

Веб-додатки управління проєктами створюються та тестуються в офісах, де встановлено значну кількість електронного обладнання: сервери, комп’ютери, системи зберігання даних, безперебійного живлення тощо. Наявність такої техніки вимагає дотримання норм безпеки при роботі з електроустановками, особливо в умовах ІТ-середовища, де тривале перебування працівників у контакті з обладнанням є невід’ємною частиною трудового процесу.

Усі електроустановки поділяються на низьковольтні (до 1000 В) і високовольтні (понад 1000 В). Робота з ними має здійснюватися лише персоналом, який пройшов спеціальне навчання, інструктажі, медичні огляди, а також перевірку знань з питань електробезпеки [21].

Вимоги до безпечної експлуатації електроустановок регламентуються чинними нормативними актами України. Зокрема, Типовим положенням про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженим наказом Міністерства соціальної політики України №508 [22]. У цьому документі визначено обов’язковість вступного, первинного, повторного,

позапланового та цільового інструктажів, а також закріплено вимоги до періодичності перевірок знань.

Особливу увагу при експлуатації електроустановок слід приділяти дотриманню правил технічної експлуатації електроустановок споживачів, затверджених наказом Міністерства палива та енергетики України [23]. У них зазначено норми щодо ізоляції проводів, заземлення, попереджувального маркування, наявності плакатів безпеки, а також вимоги до оперативного персоналу.

У контексті теми кваліфікаційної роботи актуально впроваджувати функціонал цифрового контролю безпеки електрообладнання в межах веб-додатків управління. Це можуть бути модулі з моніторингу аварійних подій, нагадування про терміни перевірки ПЗВ та заземлення, база інструктажів, протоколи перевірки електроінструментів, інтегровані чек-листи перед запуском обладнання.

Аналіз продуктивності працівників, які взаємодіють з електротехнічним обладнанням, також має включати ризик-фактори, пов’язані з підвищеною напругою, статичним електричним полем, можливістю ураження струмом або виникненням електричних дуг. З метою профілактики таких ситуацій необхідне постійне оновлення технічної документації, організація навчань, та ведення цифрового обліку стану електрообладнання [23].

Усі роботи в діючих електроустановках повинні проводитися щонайменше двома працівниками, один із яких виконує функцію спостерігача. Забороняється виконання електромонтажних або діагностичних робіт у вологому середовищі або на обладнанні, яке перебуває під напругою без відповідного допуску та захисних засобів [23].

Такі підходи дозволяють не лише дотримуватися вимог чинного законодавства у сфері охорони праці, а й інтегрувати питання електробезпеки у цифрове середовище керування персоналом.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було спроєктовано та реалізовано повноцінний веб-додаток для управління проєктами та аналізу продуктивності працівників. Розробка охоплює як архітектуру, базу даних, так і інтерфейс користувача з інтегрованими аналітичними модулями.

Система реалізована з використанням сучасного стеку технологій: React.js для фронтенду, Node.js з Express для бекенду, PostgreSQL і MongoDB для роботи з даними. Мікросервісна архітектура, Docker-контейнеризація та оркестрація через Kubernetes забезпечили масштабованість і відмовостійкість розв’язання. Застосовано гнучку структуру бази даних із підтримкою складних зв’язків, аналітики та прогнозування.

Розроблений функціонал охоплює управління проєктами, задачами, облік часу, візуалізацію метрик продуктивності, створення таймшитів і генерацію звітів. Реалізовані також модулі сповіщень, індивідуальних KPI, прогнозування ефективності та інтеграція з зовнішніми сервісами, зокрема Git і Slack. Завдяки компонентному інтерфейсу користувач має швидкий доступ до дашбордів, задач, календарів і персоналізованих рекомендацій.

Результати тестування підтвердили працездатність системи, її зручність у користуванні, ефективність обробки даних та точність розрахунків. Реалізований веб-додаток може бути легко адаптований для компаній різних масштабів і галузей, що підтверджує його універсальність.

Виконана робота демонструє, як поєднання сучасних технологій, гнучкого проєктування та чіткої аналітики дає змогу створити надійний і практичний інструмент для підвищення ефективності командної роботи та прийняття управлінських рішень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schwaber K., Sutherland J. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (дата звернення: 02.04.2025).

2. Kniberg H. Scrum and XP from the Trenches: How We Do Scrum. InfoQ Enterprise Software Development, 2015. 142 p.

3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. London: Pearson Education, 2017. 432 p.

4. Fowler M. Microservices: a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 02.04.2025).

5. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 7th edition. Pennsylvania: Project Management Institute, 2021. 370 p.

6. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2021. 616 p.

7. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. Boston: Addison-Wesley Professional, 2015. 352 p.

8. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Addison-Wesley Professional, 2010. 512 p.

9. Паттерни проектування програмного забезпечення: головні концепції для якісної розробки / за ред. М. А. Боровської. Львів : Видавництво Львівської політехніки, 2023. 295 с.

10. Бушуєв С. Д., Бушуєв Д. А., Ярошенко Р. Ф. Проривні компетенції в управлінні інноваційними проектами та програмами. Вісник Національного технічного університету "ХПІ". Серія: Стратегічне управління, управління

портфелями, програмами та проектами. 2018. № 1. С. 3-9. DOI: 10.20998/2413-3000.2018.1.1.

11. Тиш Є. В., Грицик В. В. Методи та засоби тестування програмного забезпечення для забезпечення надійності програмних систем. Вісник Хмельницького національного університету. Технічні науки. 2020. № 1. С. 38-43. DOI: 10.31891/2307-5732-2020-279-1-38-43.

12. Niven P. R., Lamorte B. Objectives and Key Results: Driving Focus, Alignment, and Engagement with OKRs. Hoboken: Wiley, 2016. 224 p.

13. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional, 2003. 560 p.

14. Данчук В. Д., Луцюк Д. В. Використання сучасних JavaScript-фреймворків для розробки веб-додатків. Інформаційні технології і засоби навчання. 2020. Т. 79, № 5. С. 152-164. DOI: 10.33407/itlt.v79i5.3658.

15. Лавріщева К. М. Програмна інженерія : підручник. Київ : Академперіодика, 2022. 376 с.

16. Коваль Г. І., Глухий В. С., Нитребич О. О. Методи та засоби підвищення надійності програмних систем. Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі. 2021. № 9. С. 102-111. DOI: 10.23939/sisn2021.09.102.

17. Яковина В. С., Сенів М. М. Аналіз та опрацювання метрик якості програмного забезпечення. Вісник Національного університету "Львівська політехніка". Інформаційні системи та мережі. 2018. № 887. С. 172-181.

18. Фаулер М. Рефакторинг: поліпшення існуючого коду / пер. з англ. С. Тараненко. Київ : K.FUND, 2019. 456 с.

19. Поморова О. В., Говорущенко Т. О. Сучасні проблеми оцінювання якості програмного забезпечення. Радіоелектронні і комп'ютерні системи. 2013. № 5. С. 319-327.

20. Мухопад Ю. Ф., Хоролець Д. Ю., Іванченко О. В. Мікросервісна архітектура систем управління як новий підхід до побудови розподілених

програмних систем. Технічні науки та технології. 2020. № 2 (20). С. 134-144. DOI: 10.25140/2411-5363-2020-2(20)-134-144.

21. Наказ Міністерства охорони здоров'я України № 356 від 21.02.2022. Про затвердження порядків надання домедичної допомоги особам при невідкладних станах [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0356-22

22. Наказ Міністерства соціальної політики України № 508 від 26.04.2018. Про затвердження Порядку проведення навчання і перевірки знань з питань охорони праці [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18

23. Сокуренко В. І., Безпека життєдіяльності та охорона праці : начальний посібник. Сокуренко В. І. Безпека життєдіяльності та охорона праці : Навчальний посібник. — Харків: Харківський національний університет внутрішніх справ, 2021. — 308 с.

ДОДАТКИ

ДОДАТОК А

Тези для Міжнародної студентської науково-технічної конференції

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 621.326
Мульський С. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

СУЧАСНІ ПІДХОДИ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ УПРАВЛІННЯ КОМАНДНИМИ ПРОЦЕСАМИ

Науковий керівник: ас. каф. Стефанишин В. М.

Mulskyi S. A.
Ternopil Ivan Puluj National Technical University

MODERN APPROACHES TO DEVELOPING INFORMATION SYSTEMS FOR TEAM PROCESS MANAGEMENT

Supervisor: ass. of the dep. Stefanyshyn V. M.

Ключові слова: інформаційна система, командна робота, аналітика, веб-додаток, управління процесами.
Key words: information system, teamwork, analytics, web application, process management.

Сучасні інформаційні технології відкривають широкі можливості для автоматизації управлінських процесів у рамках командної діяльності. Зокрема, створення спеціалізованих веб-додатків дозволяє ефективно організовувати робочі процеси, відслідковувати виконання завдань, аналізувати продуктивність працівників та приймати обгрунтовані рішення на основі даних.

Потреба в подібних системах зростає у зв'язку з поширенням дистанційної роботи, гнучких графіків та багаторівневого управління проєктами. В таких умовах стає актуальним питання створення зручних та адаптивних рішень, що забезпечують синхронізацію дій усіх учасників команди.

Розробка таких систем зазвичай грунтується на сучасних веб-технологіях (JavaScript, React, Node.js, бази даних MongoDB та ін.), що дозволяють реалізувати інтерактивний інтерфейс користувача, автоматизовану обробку даних та аналітичні модулі. Окрему увагу варто приділяти архітектурі програмного забезпечення, зокрема використанню мікросервісного підходу, який забезпечує масштабованість і надійність системи.

Особливе значення мають аналітичні інструменти, які інтегруються до системи для оцінювання продуктивності працівників, виявлення вузьких місць у процесах та прогнозування ефективності. Такий функціонал сприяє об'єктивному аналізу командної роботи та покращенню результатів діяльності.

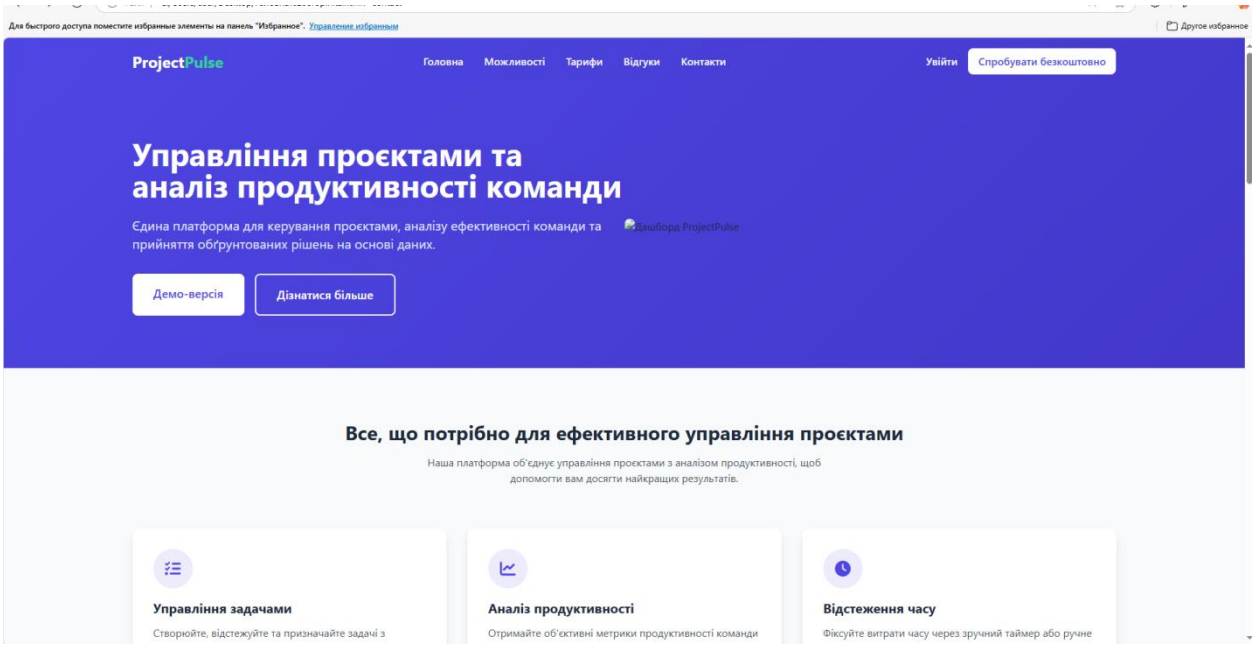
Таким чином, впровадження сучасних інформаційних систем управління командною роботою є вагомим кроком до цифрової трансформації організацій, підвищення продуктивності та якості управління проєктами.

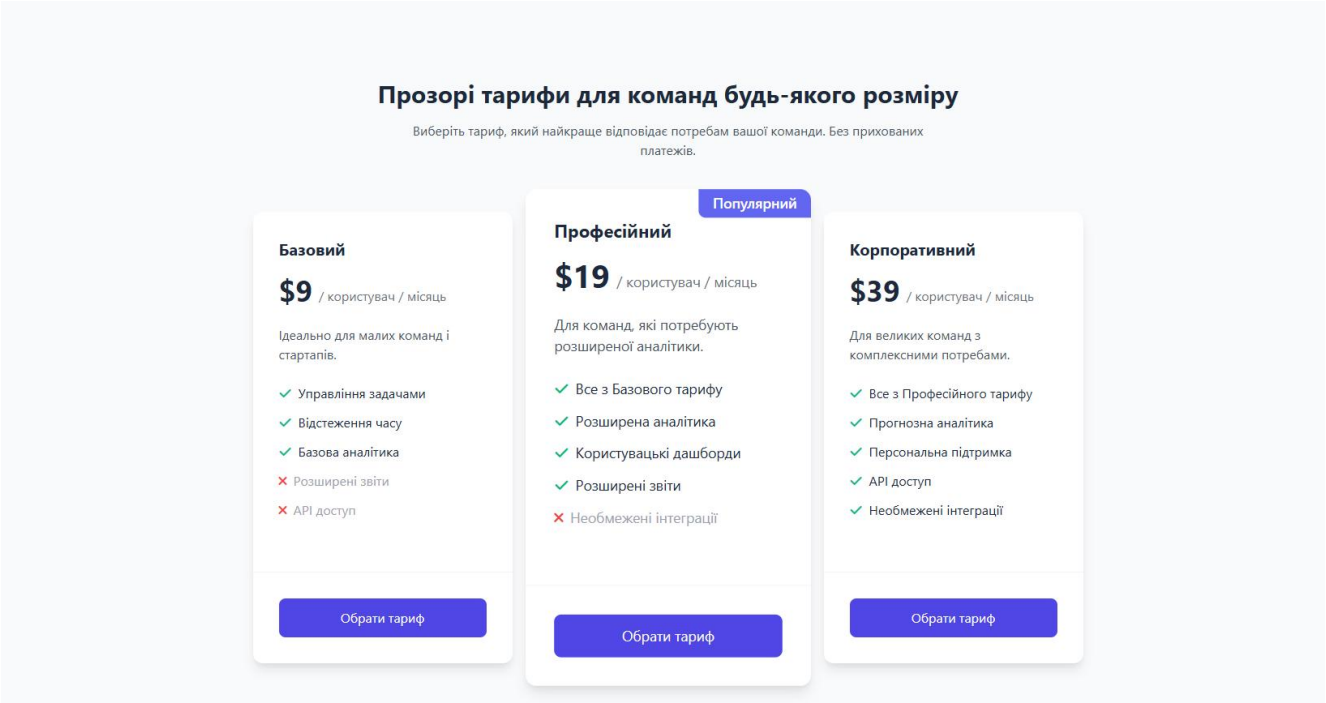
Література:

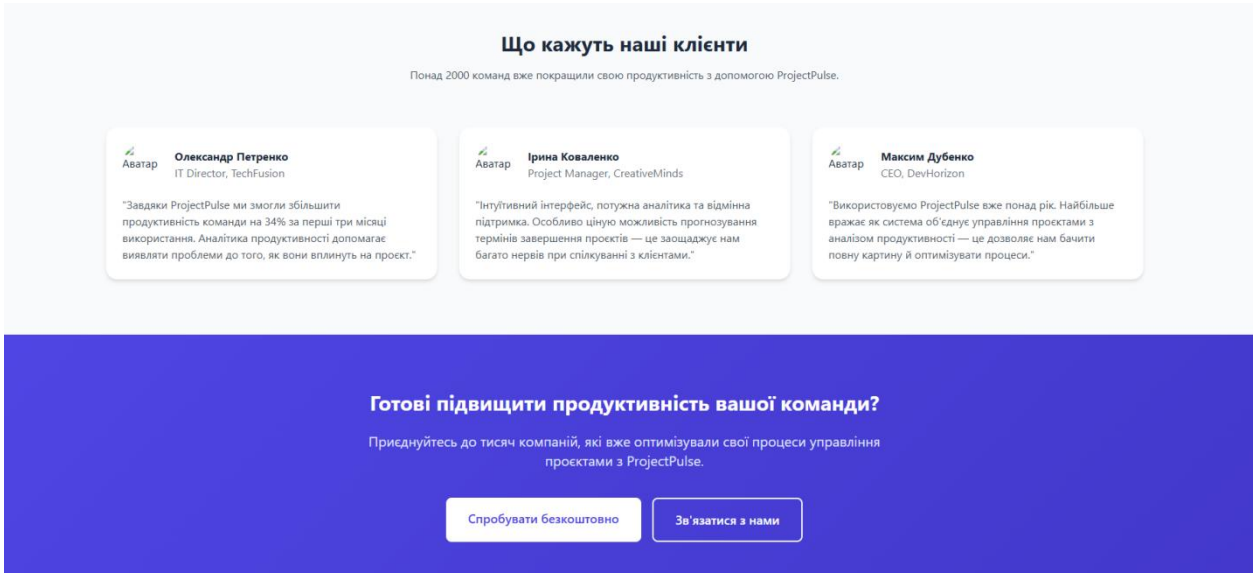
1. Marko R. .NET Framework for the Novice. 64–93 с.
2. Nancy C. Building Managing Web Services Team. John Wiley &Sons. 256–311 с. (1).
3. Roger S. P. Software Engineering: A Practitioner's Approach. 7-ме вид. Higher Education. 31–45 с.
4. Wendy K M. Effective Team Management: Trends and Technologies Shaping High Performance Teams. New Zealand : National Library of New Zealand. 130–158 с.

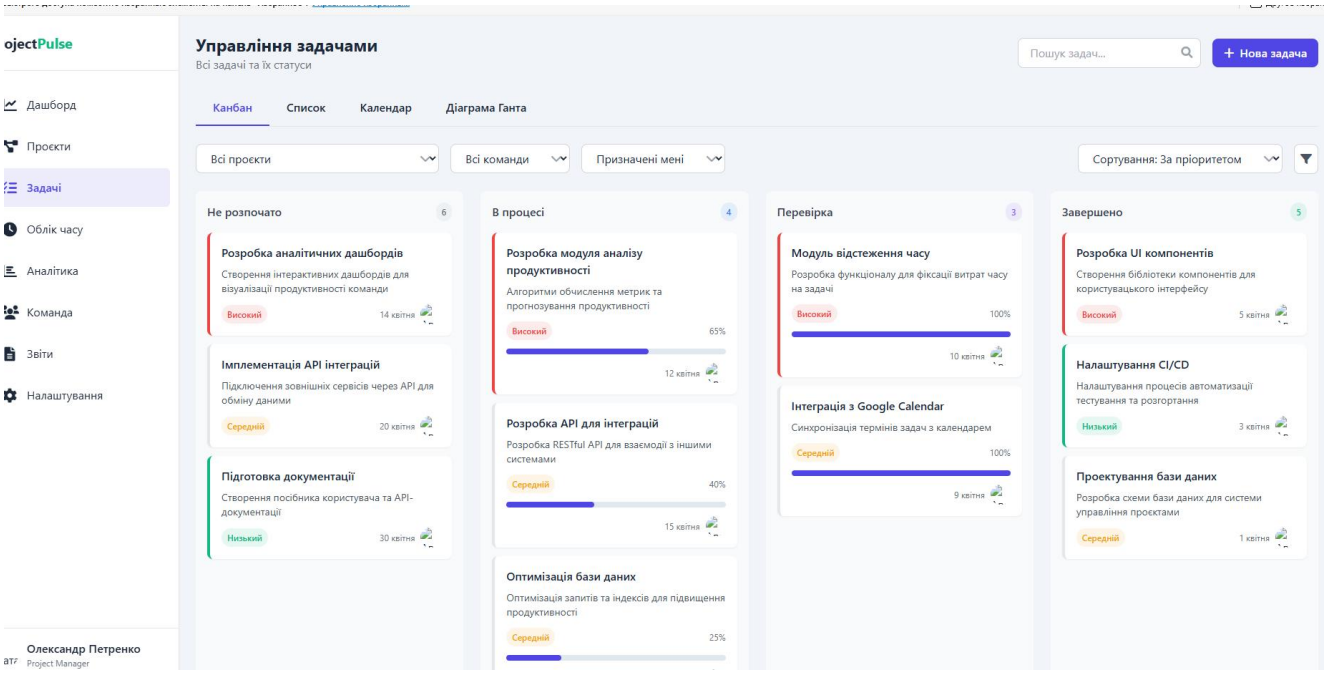
ДОДАТОК Б

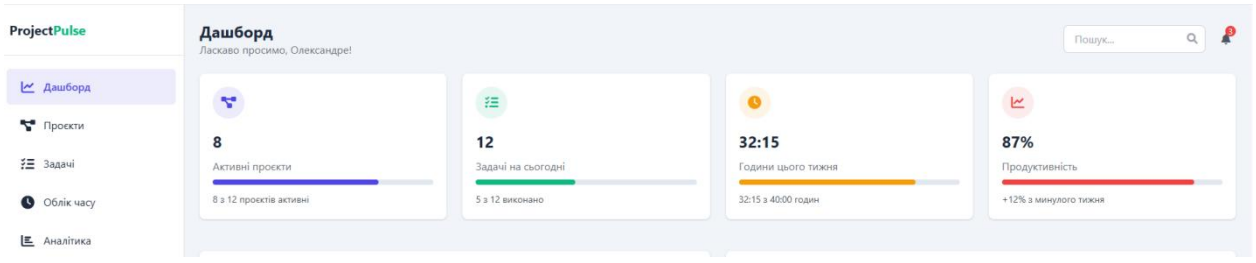
Використанні зображення

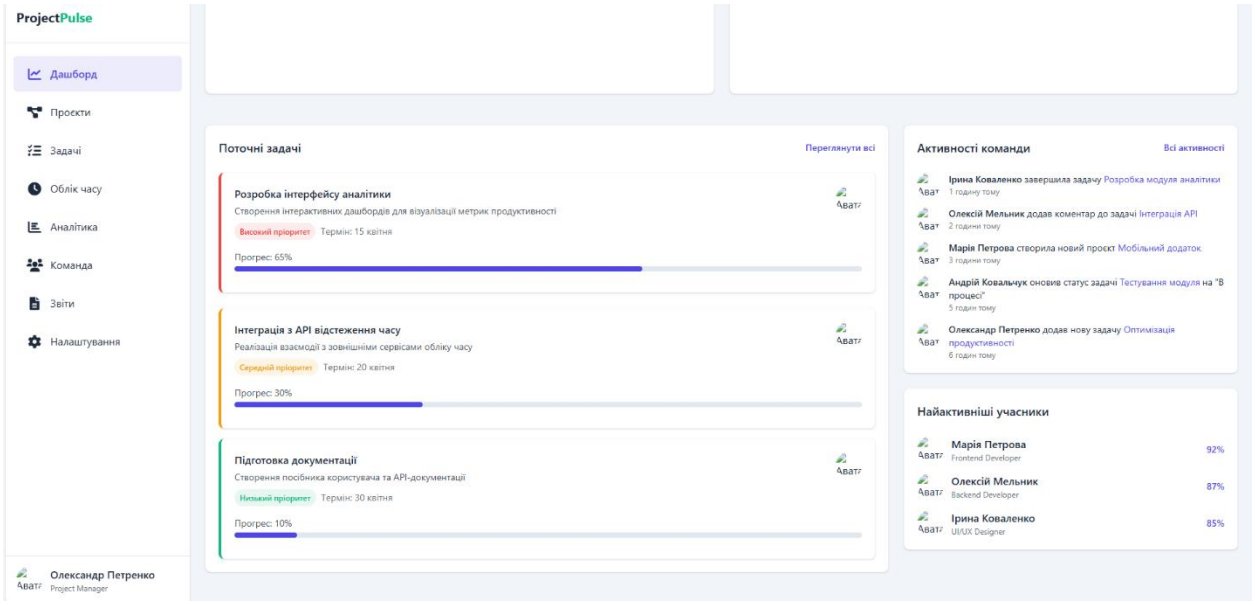


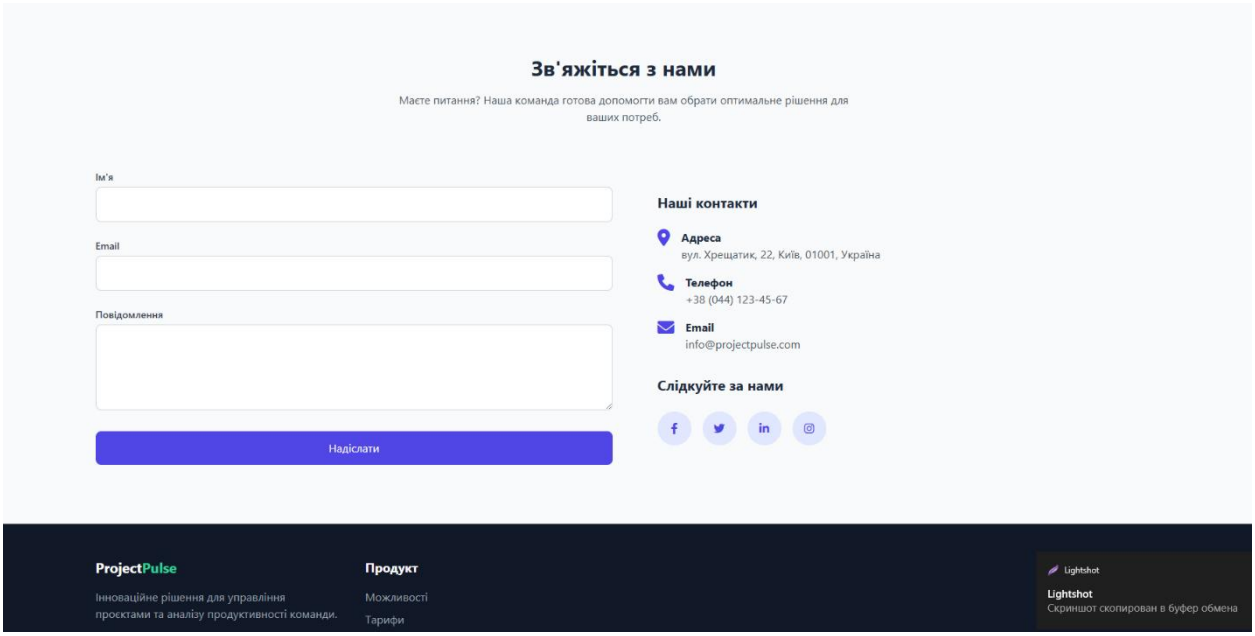


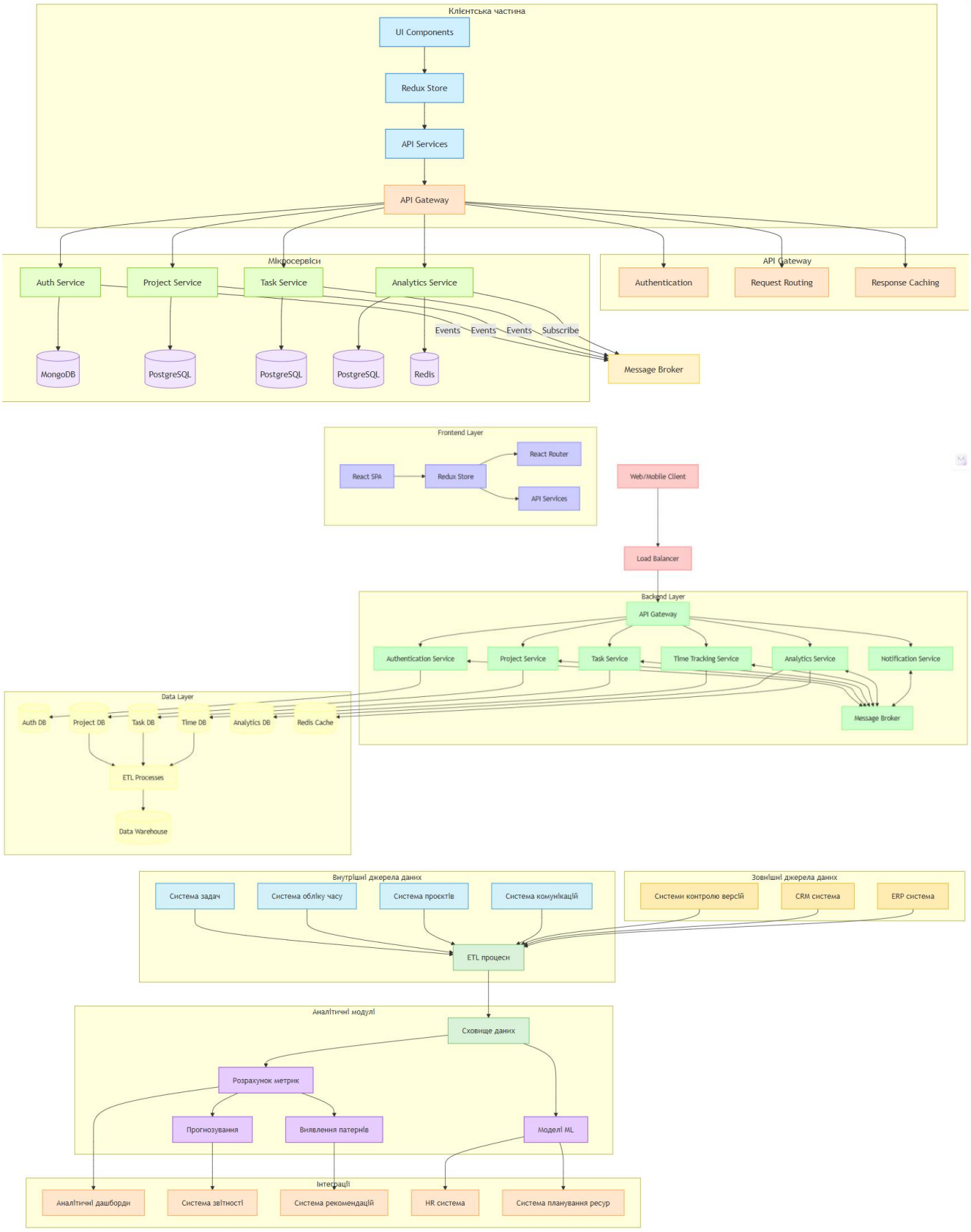

Що кажуть наші клієнти
Понад 2000 команд вже покращили свою продуктивність з допомогою ProjectPulse.
Аватар
Олександр Петренко
IT Director, TechFusion
"Завдяки ProjectPulse ми змогли збільшити продуктивність команди на 34% за перші три місяці використання. Аналітика продуктивності допомагає виявляти проблеми до того, як вони вплинуть на проект."
Аватар
Ірина Коваленко
Project Manager, CreativeMinds
"Інтуїтивний інтерфейс, потужна аналітика та відмінна підтримка. Особливо цінною можливість прогнозування термінів завершення проєктів — це заощаджує нам багато нервів при спілкуванні з клієнтами."
Аватар
Максим Дубенко
CEO, DevHorizon
"Використовуємо ProjectPulse вже понад рік. Найбільше вражає як система об'єднує управління проектами з аналізом продуктивності — це дозволяє нам бачити повну картину й оптимізувати процеси."
Готові підвищити продуктивність вашої команди?
Приєднуйтесь до тисяч компаній, які вже оптимізували свої процеси управління проєктами з ProjectPulse.
Спробувати безкоштовно
Зв'язатися з нами

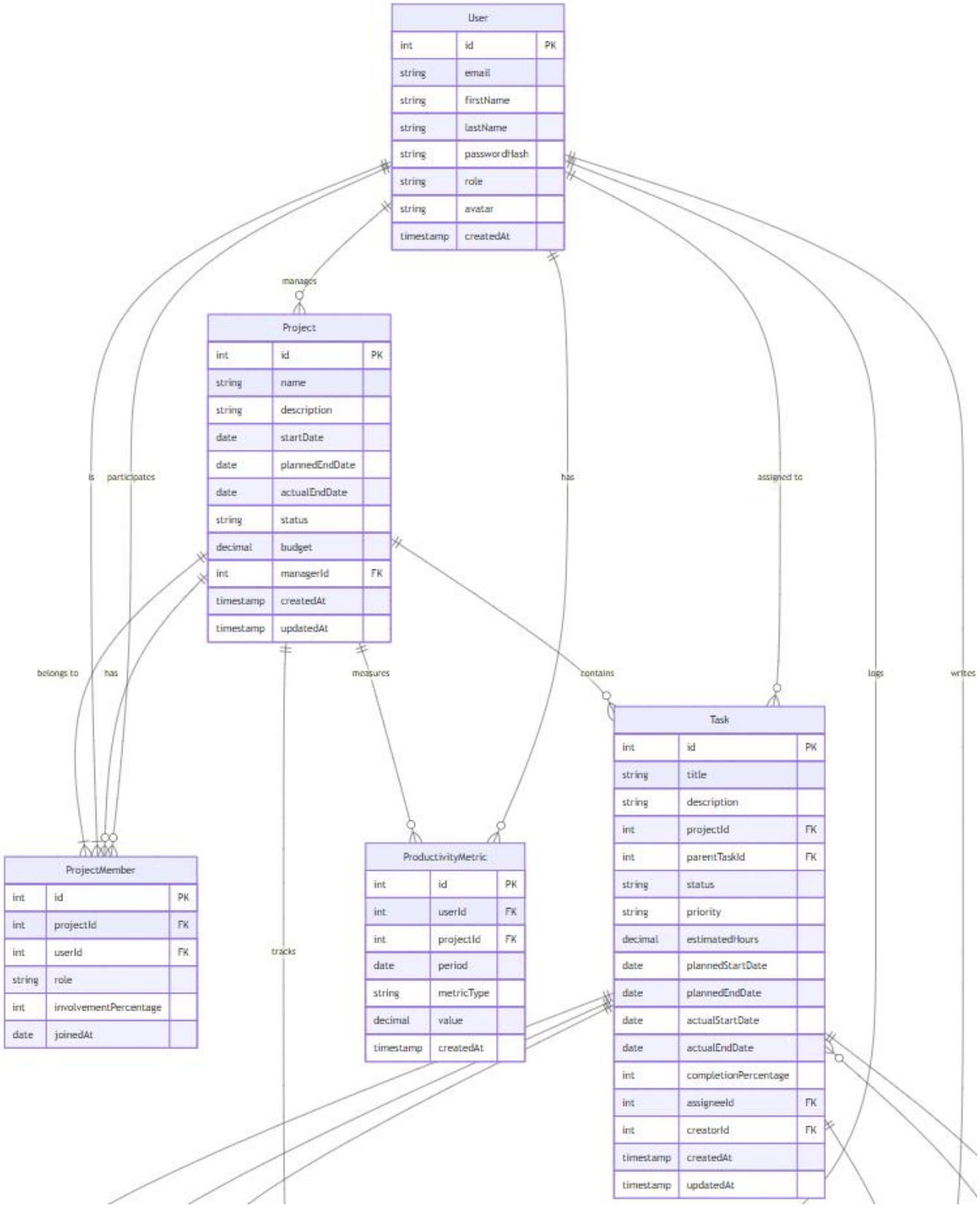

ojectPulse
Дашборд
Проєкти
Задачі
Облік часу
Аналітика
Команда
Звіти
Налаштування
Олександр Петренко
Project Manager
Управління задачами
Всі задачі та їх статуси
Пошук задач...
+ Нова задача
Канбан
Список
Календар
Діаграма Ганта
Всі проєкти
Всі команди
Призначені мені
Сортування: За пріоритетом
Не розпочато 6
Розробка аналітичних дашбордів
Створення інтерактивних дашбордів для візуалізації продуктивності команди
Високий
14 квітня
Імплементація API інтеграцій
Підключення зовнішніх сервісів через API для обміну даними
Середній
20 квітня
Підготовка документації
Створення посібника користувача та API-документації
Низький
30 квітня
В процесі 4
Розробка модуля аналізу продуктивності
Алгоритми обчислення метрик та прогнозування продуктивності
Високий
65%
12 квітня
Розробка API для інтеграцій
Розробка RESTful API для взаємодії з іншими системами
Середній
40%
15 квітня
Оптимізація бази даних
Оптимізація запитів та індексів для підвищення продуктивності
Середній
25%
Перевірка 3
Модуль відстеження часу
Розробка функціоналу для фіксації витрат часу на задачі
Високий
100%
10 квітня
Інтеграція з Google Calendar
Синхронізація термінів задач з календарем
Середній
100%
9 квітня
Завершено 5
Розробка UI компонентів
Створення бібліотеки компонентів для користувацького інтерфейсу
Високий
5 квітня
Налаштування CI/CD
Налаштування процесів автоматизації тестування та розгортання
Низький
3 квітня
Проектування бази даних
Розробка схеми бази даних для системи управління проєктами
Середній
1 квітня

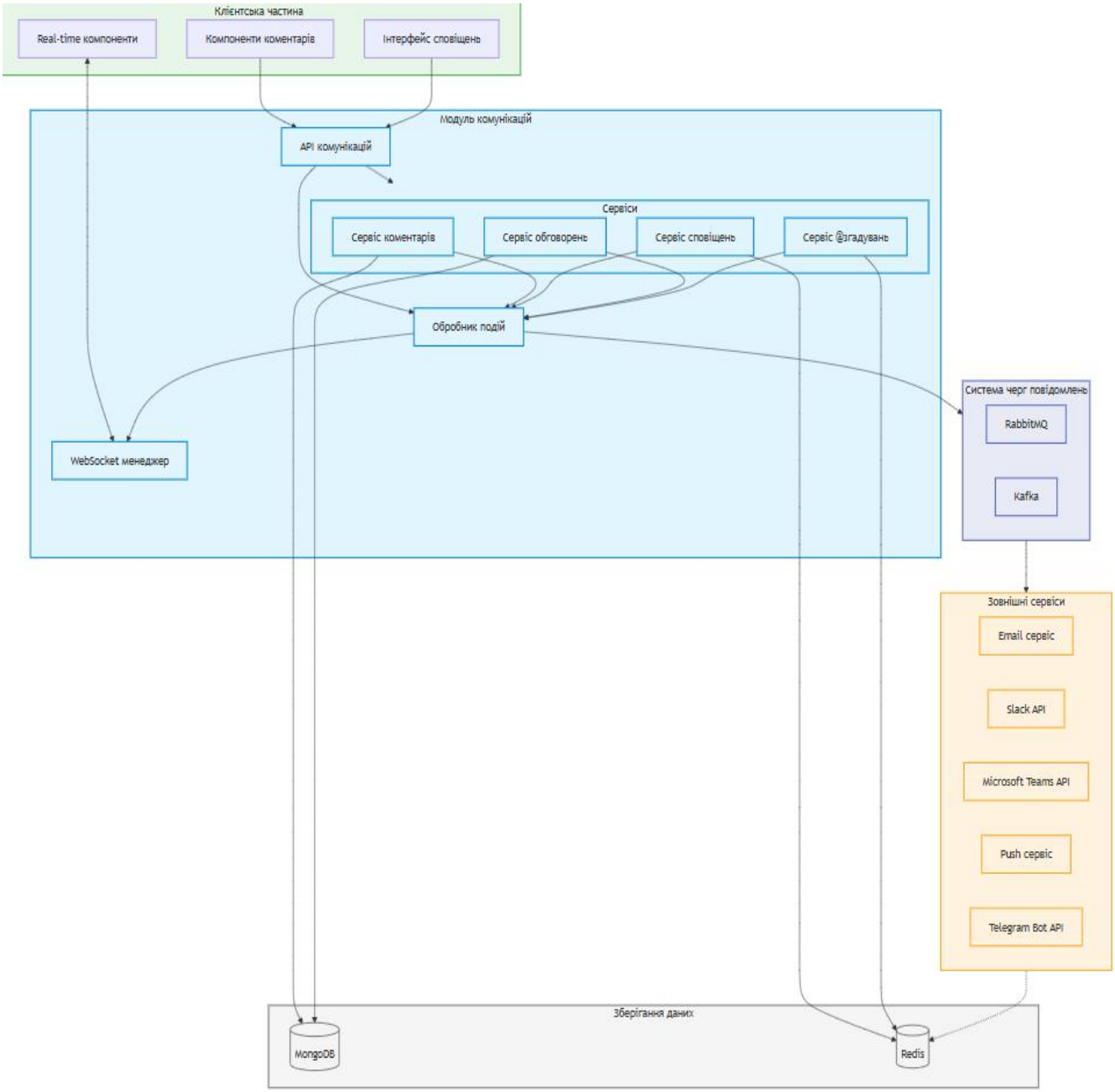

ProjectPulse
Дашборд
Проєкти
Задачі
Облік часу
Аналітика
Дашборд
Ласкаво просимо, Олександре!
Пошук...
8
Активні проєкти
8 з 12 проєктів активні
12
Задачі на сьогодні
5 з 12 виконано
32:15
Години цього тижня
32:15 з 40:00 годин
87%
Продуктивність
+12% з минулого тижня


ProjectPulse
Дашборд
Проекти
Задачі
Облік часу
Аналітика
Команда
Звіти
Налаштування
Олександр Петренко
Project Manager
Поточні задачі
Переглянути всі
Розробка інтерфейсу аналітики
Створення інтерактивних дашбордів для візуалізації метрик продуктивності
Високий пріоритет
Термін: 15 квітня
Прогрес: 65%
Інтеграція з API відстеження часу
Реалізація взаємодії з зовнішніми сервісами обліку часу
Середній пріоритет
Термін: 20 квітня
Прогрес: 30%
Підготовка документації
Створення посібника користувача та API-документації
Низький пріоритет
Термін: 30 квітня
Прогрес: 10%
Активності команди
Всі активності
Ірина Коваленко завершила задачу Розробка модуля аналітики
1 годину тому
Олексій Мельник додав коментар до задачі Інтеграція API
2 години тому
Марія Петрова створила новий проект Мобільний додаток
3 години тому
Андрій Ковальчук оновив статус задачі Тестування модуля на "В процесі"
5 годин тому
Олександр Петренко додав нову задачу Оптимізація продуктивності
6 годин тому
Найактивніші учасники
Марія Петрова
Frontend Developer
92%
Олексій Мельник
Backend Developer
87%
Ірина Коваленко
UI/UX Designer
85%


Зв'яжіться з нами
Маєте питання? Наша команда готова допомогти вам обрати оптимальне рішення для ваших потреб.
Ім'я
Email
Повідомлення
Надіслати
Наші контакти
Адреса
вул. Хрещатик, 22, Київ, 01001, Україна
Телефон
+38 (044) 123-45-67
Email
info@projectpulse.com
Слідкуйте за нами
ProjectPulse
Інноваційне рішення для управління проєктами та аналізу продуктивності команди.
Продукт
Можливості
Тарифи
Lightshot
Lightshot
Скриншот скопирован в буфер обмена


Клієнтська частина
UI Components
Redux Store
API Services
API Gateway
Мікросервіси
Auth Service
Project Service
Task Service
Analytics Service
MongoDB
PostgreSQL
PostgreSQL
PostgreSQL
Redis
Events
Events
Events
Subscribe
Message Broker
API Gateway
Authentication
Request Routing
Response Caching
Frontend Layer
React SPA
Redux Store
React Router
API Services
Web/Mobile Client
Load Balancer
Backend Layer
API Gateway
Authentication Service
Project Service
Task Service
Time Tracking Service
Analytics Service
Notification Service
Message Broker
Data Layer
Auth DB
Project DB
Task DB
Time DB
Analytics DB
Redis Cache
ETL Processes
Data Warehouse
Внутрішні джерела даних
Система задач
Система обліку часу
Система проєктів
Система комунікацій
Зовнішні джерела даних
Системи контролю версій
CRM система
ERP система
ETL процеси
Аналітичні модулі
Сховище даних
Розрахунок метрик
Прогнозування
Виявлення патернів
Моделі ML
Інтеграції
Аналітичні дашборди
Система звітності
Система рекомендацій
HR система
Система планування ресур


User
int id PK
string email
string firstName
string lastName
string passwordHash
string role
string avatar
timestamp createdAt
manages
Project
int id PK
string name
string description
date startDate
date plannedEndDate
date actualEndDate
string status
decimal budget
int managerId FK
timestamp createdAt
timestamp updatedAt
is
participates
has
assigned to
belongs to
has
measures
contains
logs
writes
tracks
ProjectMember
int id PK
int projectId FK
int userId FK
string role
int involvementPercentage
date joinedAt
ProductivityMetric
int id PK
int userId FK
int projectId FK
date period
string metricType
decimal value
timestamp createdAt
Task
int id PK
string title
string description
int projectId FK
int parentTaskId FK
string status
string priority
decimal estimatedHours
date plannedStartDate
date plannedEndDate
date actualStartDate
date actualEndDate
int completionPercentage
int assigneeId FK
int creatorId FK
timestamp createdAt
timestamp updatedAt


Клієнтська частина
Real-time компоненти
Компоненти коментарів
Інтерфейс сповіщень
Модуль комунікацій
API комунікацій
Сервіси
Сервіс коментарів
Сервіс обговорень
Сервіс сповіщень
Сервіс @згадувань
Обробник подій
WebSocket менеджер
Система черг повідомлень
RabbitMQ
Kafka
Зовнішні сервіси
Email сервіс
Slack API
Microsoft Teams API
Push сервіс
Telegram Bot API
Зберігання даних
MongoDB
Redis

ДОДАТОК В

Лістинги коду

```
using static System.Runtime.InteropServices.JavaScript.JSType;
using System.Drawing;
using System.Reflection.Emit;
using System.Xml;
using System;

import React, { useState, useEffect } from 'react';
import { Button, Card, Typography, Space, Tag } from 'antd';
import { PlayCircleOutlined, PauseCircleOutlined, SaveOutlined } from '@ant-
design/icons';

const TaskTimer = ({ taskId, taskName }) => {
    const [isRunning, setIsRunning] = useState(false);
    const [elapsedTime, setElapsedTime] = useState(0);
    const [startTime, setStartTime] = useState(null);
    const [description, setDescription] = useState('');

    useEffect(() => {
        let interval = null;
        if (isRunning)
        {
            interval = setInterval(() => {
                setElapsedTime(prevTime => prevTime + 1);
            }, 1000);
        }
            else
            {
                clearInterval(interval);
            }
            return () => clearInterval(interval);
        }, [isRunning]);

        const formatTime = (seconds) => {
        const h = Math.floor(seconds / 3600);
        const m = Math.floor((seconds % 3600) / 60);
        const s = seconds % 60;
        return `${ h.toString().padStart(2, '0')}:${ m.toString().padStart(2,
    '0')}:${ s.toString().padStart(2, '0')}`;
    };

    const handleStart = () => {
        setIsRunning(true);
        setStartTime(new Date());
    };

    const handlePause = () => {
        setIsRunning(false);
    };

    const handleSave = async () => {
    setIsRunning(false);
```

```
// Підготовка даних для збереження
const timeEntry = {
      taskId,
    duration: elapsedTime,
      startTime,
      endTime: new Date(),
      description
    };

// Тут буде код для відправки даних на сервер
console.log('Saving time entry:', timeEntry);

// Скидання таймера
setElapsedTime(0);
setDescription('');
  };
```


```
// Підготовка даних для збереження
const timeEntry = {
      taskId,
    duration: elapsedTime,
      startTime,
      endTime: new Date(),
      description
    };

// Тут буде код для відправки даних на сервер
console.log('Saving time entry:', timeEntry);

// Скидання таймера
setElapsedTime(0);
setDescription('');
  };
```


```
return (

  < Card title ={`Таймер задачі: ${ taskName}`}
style ={ { width: 400 } }>

    < Space direction = "vertical" style ={ { width: '100%' } }>

      < Typography.Title level ={ 3}
style ={ { textAlign: 'center' } }>
          { formatTime(elapsedTime)}
        </ Typography.Title >


        < Space >
          {
    !isRunning ? (
            < Button
              type = "primary"
              icon ={< PlayCircleOutlined />}
    onClick ={ handleStart}
```

```
            < Button
              type = "primary"
              icon ={< SaveOutlined />}
    onClick ={ handleSave}
    disabled ={ elapsedTime === 0}
            >
              Зберегти
            </ Button >
          </ Space >


          < textarea
            placeholder = "Опис виконаної роботи"
            value ={ description}
    onChange ={ (e) => setDescription(e.target.value)}
    style ={ { width: '100%', height: 80 } }
          />


          { isRunning && < Tag color = "green" > Таймер активний </ Tag >}
        </ Space >
      </ Card >
  );
};

export default TaskTimer;
```

Лістинг 2.4 – Приклад компонента для відстеження часу в React

---

```
function calculateProductivityIndex(completedTasks, timeSpent) {
    let totalWeight = 0;
    let weightedCompletion = 0;

    completedTasks.forEach(task => {
        const taskWeight = calculateTaskWeight(task); // Враховує складність,
пріоритет, тип задачі
        totalWeight += taskWeight;

        // Нормалізація часу відносно оцінки трудомісткості
        const timeEfficiency = task.estimatedHours > 0
            ? task.estimatedHours / (timeSpent[task.id] || task.estimatedHours)
            : 1;

        weightedCompletion += taskWeight * timeEfficiency;
    });

    return totalWeight > 0 ? (weightedCompletion / totalWeight) * 100 : 0;
}
```

```
// Обчислення ваги задачі на основі її характеристик
function calculateTaskWeight(task) {
    const complexityFactor = {
        'low': 1,
        'medium': 2,
        'high': 3,
        'critical': 5
    }[task.complexity] || 1;

    const priorityFactor = {
        'low': 0.8,
        'normal': 1,
        'high': 1.2,
        'urgent': 1.5
|
    }[task.priority] || 1;

    const typeFactor = {
        'task': 1,
        'bug': 1.2,
        'feature': 1.5,
        'research': 2
    }[task.type] || 1;

    return complexityFactor * priorityFactor * typeFactor;
}// Приклад обчислення індексу продуктивності
function calculateProductivityIndex(completedTasks, timeSpent) {
    let totalWeight = 0;
    let weightedCompletion = 0;


    completedTasks.forEach(task => {
        const taskWeight = calculateTaskWeight(task); // Враховує складність,
пріоритет, тип задачі
        totalWeight += taskWeight;

        // Нормалізація часу відносно оцінки трудомісткості
        const timeEfficiency = task.estimatedHours > 0
            ? task.estimatedHours / (timeSpent[task.id] || task.estimatedHours)
            : 1;

        weightedCompletion += taskWeight * timeEfficiency;
    });

    return totalWeight > 0 ? (weightedCompletion / totalWeight) * 100 : 0;
}
```

```
// Обчислення ваги задачі на основі її характеристик
function calculateTaskWeight(task) {
    const complexityFactor = {
        'low': 1,
        'medium': 2,
        'high': 3,
        'critical': 5
    }[task.complexity] || 1;

    const priorityFactor = {
        'low': 0.8,
        'normal': 1,
        'high': 1.2,
        'urgent': 1.5
    }[task.priority] || 1;

    const typeFactor = {
        'task': 1,
        'bug': 1.2,
        'feature': 1.5,
        'research': 2
    }[task.type] || 1;

    return complexityFactor * priorityFactor * typeFactor;

}
```

Лістинг 2.5 – Приклад коду обрахунку індексу продуктивності

```
// Приклад обчислення індексу продуктивності
using static System.Runtime.InteropServices.JavaScript.JSType;

import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';

const ProductivityChart = ({ data, width = 600, height = 400 }) => {
    const svgRef = useRef();

    useEffect(() => {
        if (!data || data.length === 0) return;

        // Створення SVG-контейнера
        const svg = d3.select(svgRef.current)
          .attr('width', width)
          .attr('height', height);

        // Очищення попередніх елементів
        svg.selectAll('*').remove();

        // Встановлення відступів
        const margin = { top: 20, right: 30, bottom: 40, left: 50 };
    const innerWidth = width - margin.left - margin.right;
    const innerHeight = height - margin.top - margin.bottom;
```

```
    // Створення групи для графіка з відступами
    const g = svg.append('g')
      .attr('transform', `translate(${ margin.left},${ margin.top})`);

    // Визначення шкал
    const xScale = d3.scaleTime()
      .domain(d3.extent(data, d => new Date(d.date)))

    .range([0, innerWidth]);
    const yScale = d3.scaleLinear()
      .domain([0, d3.max(data, d => d.productivityIndex) * 1.1]) // 10% запас
зверху
      .range([innerHeight, 0]);

    // Додавання осей
    g.append('g')
      .attr('transform', `translate(0,${ innerHeight})`)
      .call(d3.axisBottom(xScale))
      .append('text')
      .attr('fill', '#000')
      .attr('text-anchor', 'end')
      .attr('x', innerWidth)
      .attr('y', -10)
      .text('Дата');


    g.append('g')
      .call(d3.axisLeft(yScale))
      .append('text')
      .attr('fill', '#000')
      .attr('transform', 'rotate(-90)')
      .attr('y', 15)
      .attr('text-anchor', 'end')
      .text('Індекс продуктивності');

    // Додавання лінії графіка
    const line = d3.line()
      .x(d => xScale(new Date(d.date)))
      .y(d => yScale(d.productivityIndex))
      .curve(d3.curveMonotoneX);


    g.append('path')
      .datum(data)
      .attr('fill', 'none')
      .attr('stroke', 'steelblue')
      .attr('stroke-width', 2)
      .attr('d', line);

    // Додавання точок для даних
    g.selectAll('.dot')
      .data(data)
      .enter().append('circle')
      .attr('class', 'dot')
      .attr('cx', d => xScale(new Date(d.date)))
      .attr('cy', d => yScale(d.productivityIndex))
      .attr('r', 4)
      .attr('fill', 'steelblue');
```

```
    // Додавання підписів для точок
    g.selectAll('.label')
      .data(data)
      .enter().append('text')
      .attr('class', 'label')
      .attr('x', d => xScale(new Date(d.date)))
      .attr('y', d => yScale(d.productivityIndex) - 10)
      .attr('text-anchor', 'middle')
      .attr('font-size', '10px')
      .text(d => d.productivityIndex.toFixed(1));

}, [data, width, height]);

return < svg ref={ svgRef}></ svg >;
};
```

Лістинг 2.6 – Приклад компонента для візуалізації продуктивності в React з використанням D3.js

```
version: '3.8'

services:
api - gateway:
    build: ./ api - gateway
    ports:
-"3000:3000"
    environment:
-NODE_ENV = development
- JWT_SECRET = your_jwt_secret
- AUTH_SERVICE_URL = http://auth-service:3001
      -PROJECT_SERVICE_URL = http://project-service:3002
      -TASK_SERVICE_URL = http://task-service:3003
      -ANALYTICS_SERVICE_URL = http://analytics-service:3004
    depends_on:
-auth - service
- project - service
- task - service
- analytics - service
    networks:
-app - network

  auth - service:
    build: ./ auth - service
    ports:
-"3001:3001"
    environment:
-NODE_ENV = development
```

```
- JWT_SECRET = your_jwt_secret
- MONGO_URI = mongodb://mongo:27017/auth
      -REDIS_URL = redis://redis:6379
    depends_on:
-mongo
- redis
    networks:
-app - network

  project - service:
    build: ./ project - service
ports:
```

Лістинг 3.1 – Приклад конфігурації Docker Compose для локального розгортання мікросервісів системи

```
// api-gateway/src/app.js
using System.Diagnostics;

const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const cors = require('cors');
const authMiddleware = require('./middleware/auth');
const errorHandler = require('./middleware/errorHandler');

const app = express();

// Загальні middleware
app.use(helmet()); // Захист від поширених веб-вразливостей
app.use(cors()); // Підтримка CORS
app.use(express.json()); // Парсинг JSON-запитів

 // Обмеження кількості запитів для захисту від DoS-атак
 const apiLimiter = rateLimit({
   windowMs: 15 * 60 * 1000, // 15 хвилин
   max: 100, // ліміт запитів на IP-адресу
   standardHeaders: true,
   legacyHeaders: false,
 });
 app.use(apiLimiter);

 // Публічні маршрути (без автентифікації)
 app.use('/api/auth', createProxyMiddleware({
 target: process.env.AUTH_SERVICE_URL,
   changeOrigin: true,
   pathRewrite:
```

```
      {
          '^/api/auth': '/auth',
    },
  }));

  // Перевірка автентифікації для захищених маршрутів
  app.use('/api', authMiddleware);

  // Захищені маршрути (з автентифікацією)
  app.use('/api/projects', createProxyMiddleware({
  target: process.env.PROJECT_SERVICE_URL,
    changeOrigin: true,
    pathRewrite:
      {
          '^/api/projects': '/projects',
    },

  }));
```

Лістинг 3.2 – Приклад реалізації API Geteway з використанням Express.js

```
// project-service/src/controllers/projectController.js
using static System.Net.Mime.MediaTypeNames;
using System.Collections.Generic;

const { Project, ProjectMember, User } = require('../models');
const { ValidationError, NotFoundError, ForbiddenError } = require('../errors');

class ProjectController
{
    // Отримання списку проєктів з фільтрацією та пагінацією
    async getProjects(req, res, next)
    {
        try
        {
            const { page = 1, limit = 10, status, search } = req.query;
            const offset = (page - 1) * limit;

            // Базові умови фільтрації
            const where = { };

            // Додавання умов фільтрації за статусом
            if (status)
            {
                where.status = status;
            }

            // Додавання умов пошуку за назвою або описом
            if (search)
            {
                where.$or = [
                { name: { $regex: search, $options: 'i' } },
          { description: { $regex: search, $options: 'i' } }
        ];
            }
```

```
            // Отримання проєктів з бази даних
            const { rows: projects, count } = await Project.findAndCountAll({
                where,
        include:
                [
          {
                model: ProjectMember,
            as: 'members',
            include:
                    [
              {
                    model: User,

                as: 'user',
```

Лістинг 3.3 – Приклад реалізації контролера для управління проєктами

```
// task-service/src/services/taskProgressService.js
using System.Threading.Tasks;

const { Task } = require('../models');
const { sequelize } = require('../database');

class TaskProgressService
{
    // Оновлення прогресу батьківської задачі на основі дочірніх
    async updateParentTaskProgress(taskId)
    {
        const task = await Task.findByPk(taskId, {
        include: [{ model: Task, as: 'parentTask' }]
    });

        if (!task.parentTaskId)
        {
            return null; // Немає батьківської задачі
        }
           // Отримуємо всі дочірні задачі батьківської задачі
           const siblingTasks = await Task.findAll({
           where: { parentTaskId: task.parentTaskId }
           });

           // Розраховуємо середній прогрес
           const totalWeight = siblingTasks.reduce((sum, t) => sum + (t.weight || 1),
   0);
           const weightedProgress = siblingTasks.reduce((sum, t) => {
               return sum + (t.progress || 0) * (t.weight || 1);
           }, 0);

           const newProgress = totalWeight > 0 ? Math.round(weightedProgress /
   totalWeight) : 0;
           // Оновлюємо прогрес батьківської задачі
           await Task.update(
           { progress: newProgress },
         { where: { id: task.parentTaskId } }
       );
```

```
        // Оновлюємо прогрес батьківської задачі
        await Task.update(
        { progress: newProgress },
      { where: { id: task.parentTaskId } }
    );

        // Рекурсивно оновлюємо прогрес вище по ієрархії
        if (task.parentTask && task.parentTask.parentTaskId)
        {
            await this.updateParentTaskProgress(task.parentTaskId);
        }

        return newProgress;
    }
}


module.exports = new TaskProgressService();
```

Лістинг 3.4 - Приклад реалізації модуля для автоматичного оновлення прогресу батьківських задач

```
// src/modules/tasks/reducers.js
using System.Threading.Tasks;

import { createReducer } from '@reduxjs/toolkit';
import {
  FETCH_TASKS_REQUEST,
  FETCH_TASKS_SUCCESS,
  FETCH_TASKS_FAILURE,
  CREATE_TASK_SUCCESS,
  UPDATE_TASK_SUCCESS,
  DELETE_TASK_SUCCESS
} from './types';

const initialState = {
  tasks: [],
  loading: false,
  error: null,
  pagination: {
    total: 0,
    page: 1,
    pageSize: 10,
    totalPages: 0
  },
```

```
  filters: {
    status: null,
    assignee: null,
    priority: null,
    search: ''
  }
};

const tasksReducer = createReducer(initialState, {
  [FETCH_TASKS_REQUEST]: (state) => {
    state.loading = true;
    state.error = null;// src/modules/tasks/reducers.js
import { createReducer } from '@reduxjs/toolkit';
import {
  FETCH_TASKS_REQUEST,
  FETCH_TASKS_SUCCESS,
  FETCH_TASKS_FAILURE,
  CREATE_TASK_SUCCESS,
  UPDATE_TASK_SUCCESS,
  DELETE_TASK_SUCCESS
} from './types';


const initialState = {
  tasks: [],
  loading: false,
  error: null,
  pagination: {
    total: 0,
    page: 1,
    pageSize: 10,
    totalPages: 0
  },
  filters: {
    status: null,
    assignee: null,
    priority: null,
    search: ''
  }
};

const tasksReducer = createReducer(initialState, {
```

Лістинг 3.6 – Приклад реалізації Redux-редюсера для управління задачами

```
// api-gateway/src/app.js
using System.Diagnostics;

const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const cors = require('cors');
const authMiddleware = require('./middleware/auth');
const errorHandler = require('./middleware/errorHandler');

const app = express();

// Загальні middleware
app.use(helmet()); // Захист від поширених веб-вразливостей
app.use(cors()); // Підтримка CORS
app.use(express.json()); // Парсинг JSON-запитів

// Обмеження кількості запитів для захисту від DoS-атак
const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 хвилин
  max: 100, // ліміт запитів на IP-адресу
  standardHeaders: true,
  legacyHeaders: false,
});
app.use(apiLimiter);

// Публічні маршрути (без автентифікації)
app.use('/api/auth', createProxyMiddleware({
target: process.env.AUTH_SERVICE_URL,
  changeOrigin: true,
  pathRewrite:


    {
        '^/api/auth': '/auth',
  },
}));

// Перевірка автентифікації для захищених маршрутів
app.use('/api', authMiddleware);

// Захищені маршрути (з автентифікацією)
app.use('/api/projects', createProxyMiddleware({
target: process.env.PROJECT_SERVICE_URL,
  changeOrigin: true,
  pathRewrite:
    {
        '^/api/projects': '/projects',
  },

}));
```

```
    // Оновлення існуючої задачі
    async updateTask(id, taskData)
    {
        const response = await api.put(`/ tasks /${ id}`, taskData);
        return response.data;
    }

    // Видалення задачі
    async deleteTask(id)
    {
        const response = await api.delete(`/ tasks /${ id}`);
        return response.data;
    }

    // Оновлення статусу задачі
    async updateTaskStatus(id, status)
    {
        const response = await api.patch(`/ tasks /${ id}/ status`, { status });
        return response.data;
    }

    // Додавання коментаря до задачі
    async addComment(taskId, comment)
    {
        const response = await api.post(`/ tasks /${ taskId}/ comments`, comment);
        return response.data;

}
}

export default new TaskService();
```

Лістинг 3.7 – Приклад реалізації сервісу роботи з API задач

ДОДАТОК Г

ДИСК