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)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня
Бакалавр

(назва освітнього ступеня)

на тему: **Розробка веб-платформи ProjectPulse для управління задачами та аналізу продуктивності**

Виконав(ла): студент(ка) 4 курсу, групи СПс-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Смик А.О.
	(підпис)	(прізвище та ініціали)
Керівник		Бачинський М.В.
	(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
2026

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

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавра
(назва освітнього ступеня)

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

студенту Смику Андрію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-платформи ProjectPulse для управління задачами та аналізу продуктивності

Керівник роботи Бачинський Михайло Володимирович, канд. техн. наук, доц.
(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, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області
Проєктування системи та архітектура
Реалізація та тестування системи
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації та діаграми процесів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Мариненко С.Ю.		
основи охорони праці			
Нормоконтроль	Стоянов Ю.М.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз завдання		
3	Виконання розділу 1		
4	Виконання розділу 2		
5	Виконання розділу 3		
6	Оформлення пояснювальної записки		
7	Оформлення графічного та презентаційного матеріалу		
8	Попередній захист		
9	Захист		

Студент _____ Смик А.О
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконав Смик Андрій Олегович, студент групи СПс-41 Тернопільського національного технічного університету імені Івана Пулюя, на тему «Розробка веб-платформи ProjectPulse для управління задачами та аналізу продуктивності». Робота має обсяг 75 сторінок, включає 23 рисунки, 20 таблиць, 4 додатки та бібліографію з 29 джерел.

Ключові слова: управління проєктами, Kanban, Angular, Node.js, MongoDB, REST API, Socket.io, веб-застосунок, продуктивність команди, спринт.

Кваліфікаційна робота присвячена розробці веб-платформи ProjectPulse – інструменту для управління проєктами, завданнями та аналізу продуктивності команди. У роботі проведено аналіз предметної галузі управління проєктами, розглянуто існуючі системи (Jira, Trello, Asana, Notion) та виявлено їхні недоліки з точки зору складності, вартості та надмірної функціональності для невеликих команд.

На основі аналізу сформульовано вимоги до системи та обрано технологічний стек: Angular 14 з бібліотекою компонентів PrimeNG для клієнтської частини, Node.js / Express.js для серверної частини, MongoDB як базу даних, Socket.io для реального часу, JWT для автентифікації.

Розроблено архітектуру системи у вигляді клієнт-серверного застосунку з REST API. Спроєктовано структуру бази даних та інтерфейс користувача. Реалізовано основні модулі: управління проєктами та завданнями (Kanban-дошка з перетягуванням), спринти з відстеженням прогресу, аналітика продуктивності (графіки, діаграми), управління командою, реальні сповіщення, функції експорту у PDF та Excel.

Проведено тестування функціональності та перевірено адаптивність інтерфейсу для мобільних пристроїв. Отримано працездатний прототип веб-платформи, придатний для використання малими та середніми командами розробників.

ABSTRACT

Bachelor’s qualification thesis, completed by Smyk Andrii, a student of group SPs-41 at Ternopil Ivan Puluj National Technical University, is devoted to «Development of a Web Platform ProjectPulse for Project Management and Team Productivity Analysis». The thesis comprises 75 pages, includes 23 figures, 20 appendices, a bibliography of 29 sources.

Keywords: project management, Kanban, Angular, Node.js, MongoDB, REST API, Socket.io, web application, team productivity, sprint.

This qualification work is dedicated to the development of ProjectPulse – a web platform for project and task management with built-in team productivity analysis. The work includes a comprehensive analysis of the project management domain, a review of existing tools (Jira, Trello, Asana, Notion), and identification of their limitations in terms of complexity, cost, and feature overload for small teams.

Based on the domain analysis, system requirements were defined and the technology stack was selected: Angular 14 with PrimeNG component library for the client side, Node.js / Express.js for the server side, MongoDB as the database, Socket.io for real-time communication, and JWT for authentication.

A client-server architecture with a RESTful API was designed. The database schema and user interface were planned prior to implementation. The following modules were implemented: project and task management with a drag-and-drop Kanban board, sprint lifecycle management with progress tracking, productivity analytics with charts and diagrams, team member management, real-time notifications, and export functionality to PDF and Excel formats.

Functional testing was conducted and the interface was verified for responsiveness on mobile devices. A working prototype of the web platform was produced, suitable for use by small and medium-sized development teams.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface – інтерфейс програмування застосунків

CDK – Component Dev Kit – інструментарій розробки компонентів (Angular)

CORS – Cross-Origin Resource Sharing – спільне використання ресурсів між різними джерелами

CRUD – Create, Read, Update, Delete – базові операції зі збереженими даними

CSS – Cascading Style Sheets – каскадні таблиці стилів

ER – Entity-Relationship – сутність-зв'язок (тип діаграми бази даних)

HTML – HyperText Markup Language – мова гіпертекстової розмітки

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

HTTPS – HyperText Transfer Protocol Secure – захищений протокол передачі гіпертексту

IDE – Integrated Development Environment – інтегроване середовище розробки

JSON – JavaScript Object Notation – текстовий формат обміну даними

JWT – JSON Web Token – токен автентифікації у форматі JSON

MVC – Model-View-Controller – архітектурний патерн

MVP – Minimum Viable Product – мінімально життєздатний продукт

LGBM – LightGBM (Light Gradient Boosting Machine)

NoSQL – Not Only SQL – нереляційні бази даних

ODM – Object Document Mapper – об'єктно-документний маппер

ORM – Object-Relational Mapping – об'єктно-реляційне відображення

REST – Representational State Transfer – архітектурний стиль взаємодії

SCSS – Sassy CSS – розширена версія CSS з додатковими можливостями

SPA – Single Page Application – односторінковий застосунок

SQL – Structured Query Language – мова структурованих запитів

UI – User Interface – інтерфейс користувача

UX – User Experience – досвід взаємодії користувача

WS – WebSocket – протокол двостороннього зв'язку

XSS – Cross-Site Scripting – міжсайтовий скриптинг (тип атаки)

ВСТУП

В умовах стрімкого розвитку ринку програмного забезпечення та дистанційних форм роботи ефективне управління проєктами стає критично важливим чинником успіху будь-якої команди розробників. Спільна робота над програмними продуктами вимагає чіткого планування завдань, відстеження прогресу виконання, аналізу продуктивності учасників та своєчасного обміну інформацією.

Сучасний ринок пропонує широкий спектр інструментів управління проєктами: від простих Kanban-дошок (Trello) до повнофункціональних корпоративних систем (Jira, Azure DevOps). Проте більшість існуючих рішень мають суттєві обмеження для малих та середніх команд: надмірна складність налаштування, висока вартість ліцензій, надлишкова функціональність, яка ускладнює щоденне використання. Водночас спрощені безкоштовні інструменти часто позбавлені аналітичних можливостей та спринтового планування.

Актуальність розробки власної веб-платформи ProjectPulse обумовлена потребою у збалансованому рішенні, яке поєднує зручний Kanban-інтерфейс, спринтове планування за методологією Agile/Scrum, вбудовану аналітику продуктивності та реальні сповіщення – у вигляді легкого у розгортанні та адаптивного веб-застосунку.

Мета роботи – розробка веб-платформи для управління проєктами та аналізу продуктивності команди з використанням сучасного стеку технологій на основі Angular та Node.js.

Задачі дослідження:

1. Провести аналіз предметної галузі управління проєктами та існуючих програмних рішень, визначити їх переваги та недоліки.
2. Сформулювати функціональні та нефункціональні вимоги до розроблюваної системи.
3. Обрати та обґрунтувати технологічний стек для реалізації клієнтської та серверної частин застосунку.

4. Спроєктувати архітектуру системи, структуру бази даних та REST API.

5. Реалізувати модулі управління проєктами, завданнями, спринтами та командою.

6. Реалізувати аналітичний модуль з візуалізацією даних продуктивності.

7. Реалізувати систему реальних сповіщень на основі WebSocket.

8. Реалізувати функціонал експорту даних у формати PDF та Excel.

9. Провести тестування розробленої системи та виправити виявлені дефекти.

Об'єкт дослідження – процес управління програмними проєктами в розподілених командах розробників.

Предмет дослідження – методи та засоби розробки веб-платформ для управління проєктами з використанням клієнт-серверної архітектури та сучасних JavaScript-технологій.

У роботі використано такі методи дослідження:

– аналіз та синтез – для дослідження існуючих систем управління проєктами та формування вимог до розроблюваного застосунку;

– структурний аналіз – для проєктування архітектури системи та структури бази даних;

– об'єктно-орієнтований підхід – для розробки серверної логіки та структури Angular-модулів;

– Agile/Scrum-методологія – як теоретична основа для реалізації функціоналу спринтів;

– тестування – для перевірки коректності роботи функціональних модулів.

У роботі вперше для навчального проєкту поєднано:

– kanban-дошку з підтримкою drag-and-drop на базі Angular CDK;

– спринтове планування з автоматичним переміщенням завдань у беклог після завершення;

– вбудовану аналітику продуктивності з інтерактивними графіками (ApexCharts);

– реальні WebSocket-сповіщення про зміну статусу завдань;

– адаптивний інтерфейс, оптимізований для мобільних пристроїв.

Розроблений прототип веб-платформи ProjectPulse може бути використаний:

– малими командами розробників (2–10 осіб) для організації робочого процесу;

– як основа для подальшого розвитку та комерціалізації продукту;

– як навчальний приклад побудови повнофункціонального SPA-застосунку з REST API.

Результати роботи представлено у вигляді функціонального веб-застосунку, задеплоєного локально та протестованого на різних типах пристроїв і браузерів.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної галузі управління проєктами, розглянуто методології Agile/Scrum/Kanban, здійснено огляд та порівняльний аналіз існуючих систем, сформульовано вимоги до розроблюваного застосунку, обрано та обґрунтовано технологічний стек.

У другому розділі описано архітектуру системи, спроєктовано структуру бази даних, REST API та компонентну структуру Angular-застосунку. Наведено UML-діаграми: діаграма варіантів використання, діаграма компонентів, діаграми послідовності.

У третьому розділі описано процес реалізації основних функціональних модулів системи, наведено ключові фрагменти коду, описано результати тестування та виявлені й виправлені дефекти.

У висновках підсумовано результати роботи, оцінено ступінь виконання поставлених задач та визначено перспективи подальшого розвитку платформи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Управління проєктами у сфері програмної інженерії є складним міждисциплінарним процесом, що охоплює планування, організацію, контроль та координацію ресурсів з метою досягнення визначених цілей у встановлені терміни та в рамках бюджету. На відміну від традиційних галузей, розробка програмного забезпечення характеризується високою невизначеністю вимог, частими змінами та необхідністю тісної взаємодії між членами команди.

1.1 Управління проєктами в розробці програмного забезпечення

Серед складових процесу управління проєктами можна виділити: визначення обсягу робіт та розбиття проєкту на задачі, моніторинг стану виконання задач та виявлення відхилення від запланованого, розподіл завдань між учасниками та відстеження завантаженості, обмін інформацією між учасниками, збір та аналіз даних про продуктивність.

1.1.1 Методологія Agile

Agile – це сукупність принципів гнучкої розробки програмного забезпечення, задекларованих у Маніфесті Agile (2001) [1]. Ключові принципи:

- Пріоритет працюючого програмного забезпечення над вичерпною документацією.
- Взаємодія з замовником важливіша за узгодження контракту.
- Готовність до змін вимог протягом усього проєкту.
- Регулярна поставка працюючого продукту (ітерації) [17].

1.1.2 Методологія Scrum

Scrum є найпоширенішим фреймворком у рамках Agile[2]. Його ключові елементи:

– Спринт (Sprint) – фіксований часовий відрізок (1–4 тижні), протягом якого команда виконує визначений обсяг роботи.

– Беклог продукту (Product Backlog) – впорядкований перелік усіх вимог до продукту.

– Беклог спринту (Sprint Backlog) – підмножина задач з продуктового беклогу, які команда зобов'язується виконати протягом спринту.

– Ретроспектива – зустріч після завершення спринту для аналізу результатів та покращення процесів.

– Velocity – показник продуктивності команди (кількість story points за спринт).

1.1.3 Методологія Kanban

Kanban – метод управління потоком роботи, що базується на візуалізації завдань на дошці з колонками, що відображають стадії виконання[3]. Основні принципи:

– Візуалізація робочого процесу.

– Обмеження незавершеної роботи (WIP-ліміти).

– Управління потоком та вимірювання ефективності.

– Постійне покращення.

У системі ProjectPulse Kanban реалізовано у вигляді дошки з чотирма колонками: To Do – In Progress – Review – Done.

1.2 Огляд існуючих систем управління проєктами

Ринок систем управління проєктами є насиченим. Розглянемо основних представників та їхні характеристики. Серед них виділяються і займають переважну частину ринку Jira і Trello від Atlassian, Asana та Notion. Кожна з перерахованих систем має як свої плюси так і значні мінуси, які впливають, як на вибір системи для конкретного проєкту, так і на досвід користування.

Серед плюсів у користуванні системою Jira можна виділити повнофункціональну підтримку для Scrum та Kanban, велика варіативність налаштувань та інтеграцій і детальна аналітика та звіти. Проте звідси випливають і мінуси. Дана система має надмірний функціонал для невеликих команд і досить відчутну ціну за користування. Найкраще вона підходить для великих команд.

В свою чергу Trello навпаки, підходить для малих команд. Даний сервіс виділяється простим і зрозумілим інтерфейсом, є безкоштовним та швидко налаштованим. Але на відміну від Jira тут значно звужений функціонал стосовно аналітики та відсутня можливість будувати детальні звіти.

Наступним застосунком описаним буде Asana. Серед її плюсів виділяються зручний інтерфейс з різними представленнями (список, дошка, часова шкала), підтримка залежностей між задачами та можливість розбиття задачі на підзадачі. З іншої сторони, даний є платним для відкриття повного функціоналу та в ньому відсутня нативна підтримка спринтів.

Останньою системою що можна згадати є Notion. Плюсами даної системи також є зручність та гнучкість і доступна ціна. Проте, управління проєктами не є безпосереднім призначенням даного сервісу. Оскільки він в першу чергу призначений для ведення нотаток та документації для проєкту, тому в ньому відсутній необхідний функціонал для управління.

Оскільки було визначено основні переваги та недоліки популярних систем управління проєктами то можна скласти таблицю в якій будуть наведені ці ж переваги та недоліки. Результати порівняльного аналізу зображені в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих систем

Критерій	**Jira**	**Trello**	**Asana**	**Notion**	**ProjectPulse**
Kanban-дошка	+	+	+	Частково	+
Спринти	+	-	Частково	-	+
Аналітика	+	-	Частково	-	+
Реальні сповіщення	+	+	+	-	+
Відстеження часу	+	Через плагін	+	-	+
Безкоштовно	Частково	Частково	Частково	Частково	+(self-hosted)
Простота	-	+	+	+	+
Мобільна адаптація	+	+	+	+	+
Відкритий код	-	-	-	-	+

1.3 Формулювання вимог до системи

На основі проведеного аналізу сформульовано вимоги до розроблюваної платформи ProjectPulse.

1.3.1 Функціональні вимоги

Функціональні вимоги визначають, що система повинна вміти робити. Вони згруповані за функціональними областями та наведені у таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги до системи ProjectPulse

Група	**Коди**	**Вимоги**
Автентифікація	FR-1.1-1.4	Реєстрація користувача; вхід з видачею JWT-токена; захищені маршрути; розрізнення ролей власник / учасник
Управління проєктами	FR-2.1-2.4	CRUD проєктів; пріоритет та статус; колір для ідентифікації; статистика (задачі, виконання)
Управління завданнями	FR-3.1-3.7	CRUD задач; пріоритет і статус; призначення виконавців; оцінки часу; дедлайни з виділенням прострочених; Kanban drag-and-drop; фільтрація та пошук
Управління спринтами	FR-4.1-4.5	Створення спринту з датами та ціллю; додавання задач на етапі planning; запуск спринту; завершення з переміщенням задач у беклог; аналітика спринту
Управління командою	FR-5.1-5.3	Запрошення учасників за email; видалення учасника власником; відображення аватарів та імен
Аналітика	FR-6.1-6.4	Дашборд з ключовими метриками; графіки за статусом та пріоритетом; Gantt-графік; аналітика швидкості команди
Сповіщення	FR-7.1-7.4	WebSocket-сповіщення при призначенні та зміні статусу задачі; лічильник непрочитаних; журнал активності
Експорт	FR-8.1-8.2	Експорт даних проєкту у форматах PDF та Excel

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи – продуктивність, безпеку, зручність і підтримуваність. Повний перелік наведено у таблиці 1.3.

Таблиця 1.3 - Нефункціональні вимоги до системи ProjectPulse

Категорія	**Коди**	**Вимоги**
Продуктивність	NFR-1.1-1.2	Час відповіді API ≤ 500 мс; Kanban-дошка відображає до 100 задач без затримок
Безпека	NFR-2.1-2.4	Хешування паролів bcrypt (salt = 10) [22]; JWT-токени з терміном дії 7 діб; захист маршрутів middleware; захист від XSS [23] та CORS
Зручність використання	NFR-3.1-3.3	Адаптивний інтерфейс desktop/tablet/mobile [25]; підтримка Chrome, Firefox, Edge, Safari; час завантаження ≤ 3 с
Масштабованість	NFR-4.1-4.2	Горизонтальне масштабування серверної частини; MongoDB з підтримкою sharding
Підтримуваність	NFR-5.1-5.2	Feature-based структура Angular; серверний код за принципом MVC

1.4 Вибір та обґрунтування технологічного стеку

Для реалізації платформи ProjectPulse обрано стек технологій, що забезпечує єдину мову програмування (JavaScript / TypeScript) на клієнті та сервері, що суттєво спрощує розробку та підтримку коду. Клієнтська частина побудована на Angular 14 [4] з бібліотекою UI-компонентів PrimeNG [5], модулем drag-and-drop Angular CDK [13], бібліотекою графіків ng-apexcharts [12] та препроцесором стилів SCSS [28]. Серверна частина реалізована на Node.js [6] з фреймворком Express.js [7]; для реальних сповіщень використовується Socket.io [10]. Дані зберігаються у документо-орієнтованій базі MongoDB [8] з ODM-бібліотекою Mongoose [9]. Автентифікація реалізована на основі JWT [11] (RFC 7519) з хешуванням паролів через bcryptjs [22].

Повний перелік технологій та інструментів з їх призначенням наведено у таблиці 1.4.

Таблиця 1.4 – Перелік використаних технологій та інструментів

Технологія/Інструмент	**Призначення**
Angular 14	Основний фреймворк клієнтської частини (SPA)
TypeScript	Мова розробки з суворою статичною типізацією
PrimeNG	Бібліотека готових UI-компонентів для Angular
Angular CDK	Drag-and-drop для Kanban-дошки
ng-apexcharts	Інтерактивні графіки (donut, bar, Gantt)
SCSS	Препроцесор стилів (змінні, вкладення, міксини)
Node.js	Середовище виконання серверної частини
Express.js	Веб-фреймворк для побудови REST API
Socket.io	WebSocket-сповіщення в реальному часі
MongoDB	Документо-орієнтована NoSQL база даних
Mongoose	ODM для роботи з MongoDB (схеми, валідація)
JWT	Stateless-автентифікація на основі токенів
bcryptjs	Хешування паролів користувачів
jsPDF	Генерація PDF-звітів на стороні клієнта
xlsx (SheetJS)	Генерація Excel-файлів на стороні клієнта
Visual Studio Code	IDE для розробки
Postman	Тестування REST API
Git	Система контролю версій

1.5 Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної галузі управління проєктами. Розглянуто методології Agile, Scrum та Kanban як теоретичну основу для розроблюваної системи. Здійснено порівняльний аналіз чотирьох провідних систем управління проєктами (Jira, Trello, Asana, Notion), визначено їхні переваги та недоліки, що дозволило обґрунтувати потребу у розробці ProjectPulse.

Сформульовано 8 груп функціональних вимог та 5 груп нефункціональних вимог до системи. Обрано та обґрунтовано технологічний стек: Angular 14 + PrimeNG для клієнтської частини, Node.js + Express.js + Socket.io для серверної, MongoDB + Mongoose як база даних, JWT для автентифікації.

2. ПРОЄКТУВАННЯ СИСТЕМИ ТА АРХІТЕКТУРА

У другому розділі розглянуто проєктні рішення, що лежать в основі платформи ProjectPulse. Визначено загальну архітектуру системи та обґрунтовано поділ на три рівні: клієнтський, серверний та рівень даних. Описано feature-based структуру Angular-застосунку, компонентну ієрархію та маршрутизацію. Спроєктовано структуру бази даних MongoDB з повним описом колекцій та зв'язків між ними. Сформовано специфікацію REST API з переліком усіх маршрутів. Побудовано UML-діаграми: варіантів використання та три діаграми послідовності для ключових сценаріїв взаємодії. Завершує розділ опис підходів до проєктування інтерфейсу – дизайн-система та адаптивний дизайн.

2.1 Загальна архітектура системи

Система ProjectPulse реалізована за класичною клієнт-серверною архітектурою з розподілом на три рівні [15]:

1. Клієнтський рівень (Presentation Layer) – Angular SPA, що виконується у браузері користувача.

2. Серверний рівень (Application Layer) – Node.js / Express REST API, що обробляє бізнес-логіку та запити від клієнта.

3. Рівень даних (Data Layer) – MongoDB, що зберігає всі дані застосунку.

Взаємодія між клієнтом та сервером відбувається двома каналами: HTTP REST API [14] – для стандартних CRUD-операцій (синхронний запит-відповідь). WebSocket (Socket.io) [10] – для реальних сповіщень (асинхронний двосторонній канал).

Діаграму архітектури системи подано на рисунку 2.1.

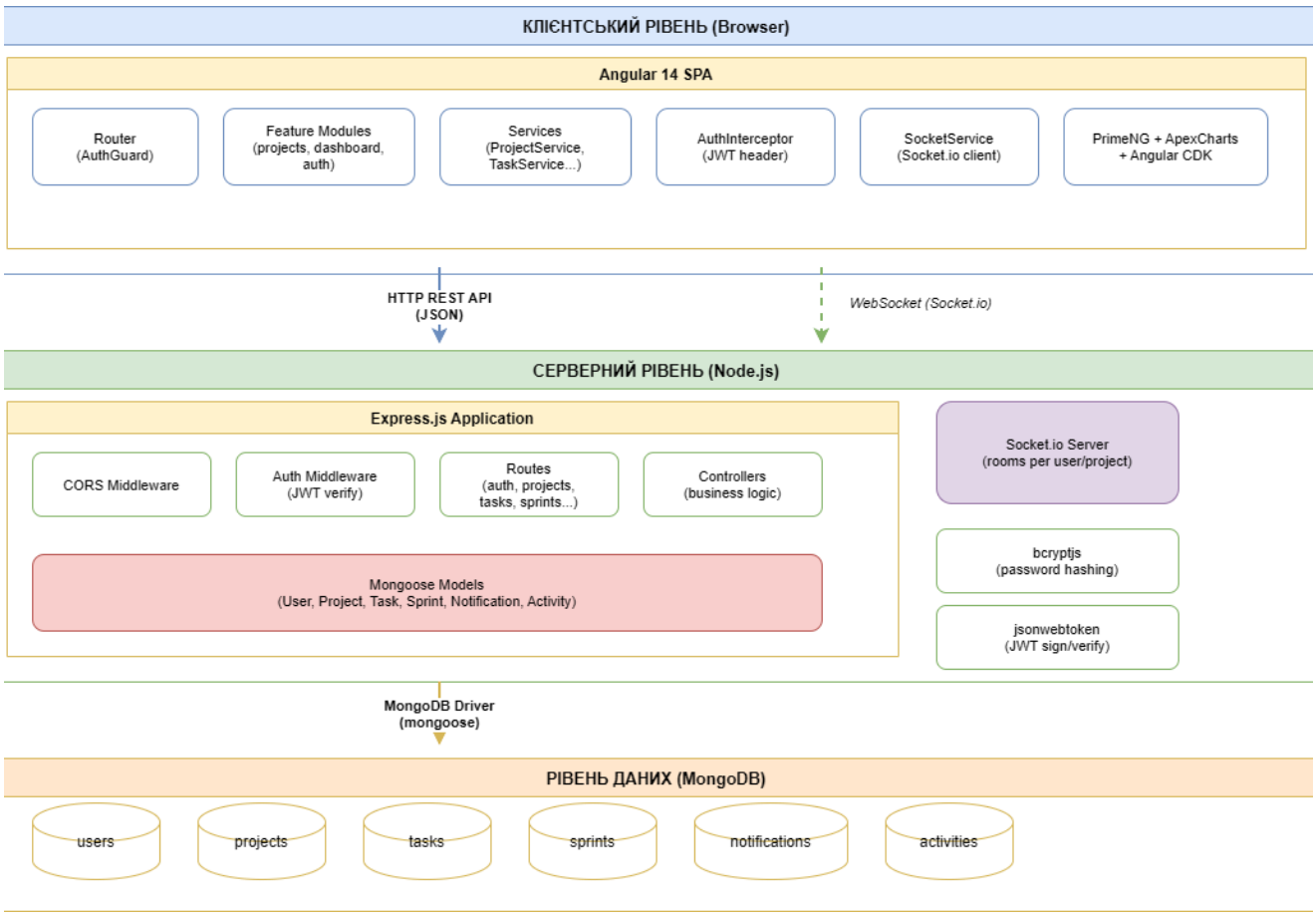

Рисунок 2.1 – Загальна архітектура системи ProjectPulse

2.2 Структура Angular-застосунку

Angular-застосунок організовано за feature-based принципом [4]: кожна функціональна область (projects, tasks, dashboard тощо) є окремим модулем з власними компонентами, сервісами та маршрутами.

2.2.1 Структура директорій

Angular-застосунок організовано за feature-based принципом структурування, при якому весь код поділяється на чотири незалежні зони відповідальності.

Директорія core/ містить singleton-сервіси та guards, що існують в єдиному екземплярі протягом усього життєвого циклу застосунку.

Директорія features/ – серце застосунку. Кожен підмодуль відповідає за одну функціональну область: auth/ за логін та реєстрацію, dashboard/ за головну сторінку, projects/ за головну сторінку з деталями проєктів.

Директорія shared/ містить компоненти та pipes, що використовуються у кількох feature-модулях одночасно.

Директорія layout/ визначає загальний макет застосунку: HeaderComponent, SidebarComponent та LayoutComponent.

Така структура забезпечує чіткий поділ відповідальності, спрощує навігацію по проєкту та дозволяє розробляти кожен feature-модуль незалежно (див. рис. 2.2).

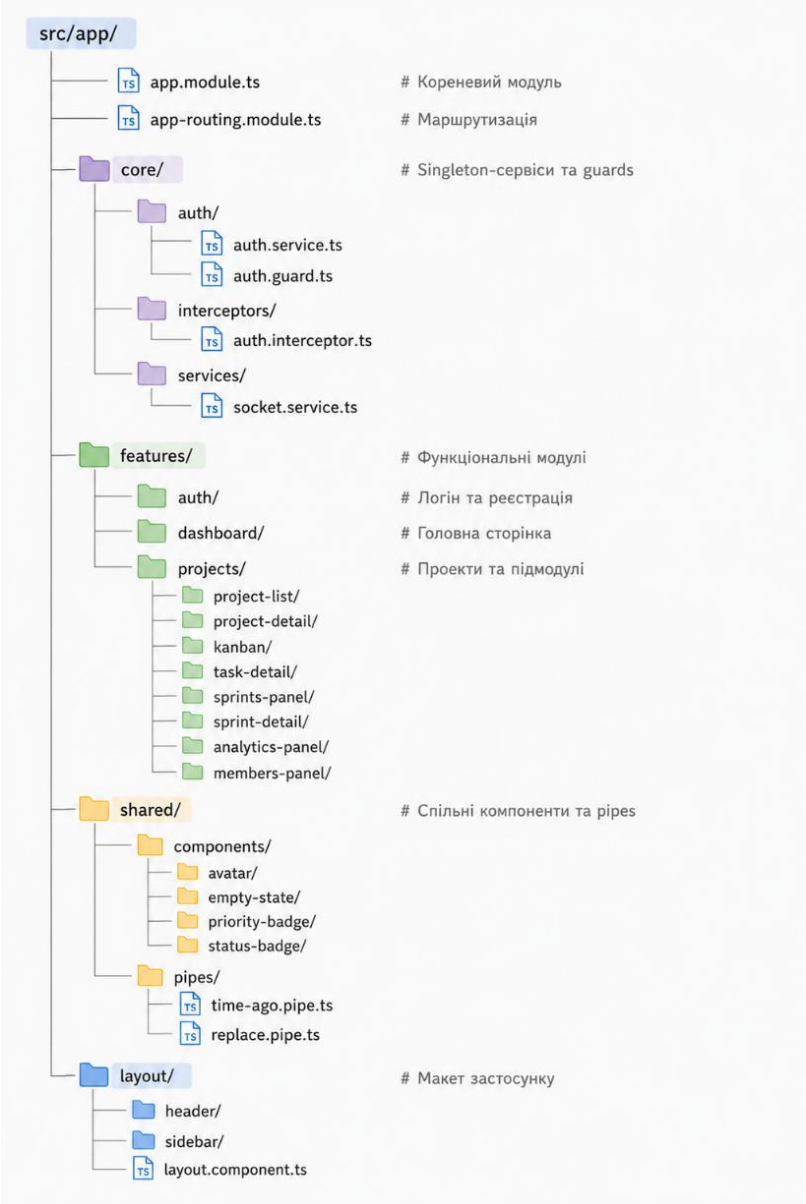

Рисунок 2.2 – Структура директорій

2.2.2 Компонентна ієрархія

Кореневим компонентом застосунку є AppComponent, який підключає модуль маршрутизації. Для всіх захищених сторінок дочірнім є LayoutComponent – він формує макет із двох зон: фіксованої навігаційної панелі та змінної зони контенту.

HeaderComponent та SidebarComponent присутні в макеті постійно, незалежно від маршруту. HeaderComponent відповідає за три елементи: hamburger-кнопку мобільного drawer, панель сповіщень з лічильником та меню користувача. SidebarComponent отримує через @Input() стан відкриття і перемикає CSS-клас для анімації на мобільних пристроях.

Зона контенту реалізована через <router-outlet>. На маршруті /dashboard відображається DashboardComponent із зведеною статистикою. На маршруті /projects/:id – ProjectDetailComponent, який містить p-tabView з п'ятьма вкладками, кожна з яких рендерить свій дочірній компонент.

KanbanComponent при кліку на картку відкриває TaskDetailSidebar як p-sidebar поверх дошки, не перериваючи її стан. AnalyticsPanelComponent будує графіки через ng-apexcharts. SprintsPanelComponent відображає список спринтів із вбудованими списками задач. SprintDetailComponent є окремим маршрутом і рендериться безпосередньо у router-outlet, а не як дочірній компонент ProjectDetailComponent.

Спільні компоненти (AvatarComponent, PriorityBadgeComponent, StatusBadgeComponent, EmptyStateComponent) використовуються на будь-якому рівні ієрархії через SharedModule і не відображені на діаграмі для збереження читабельності.

Діаграма компонентів для застосунку ProjectPulse зображена на рисунку 2.3.

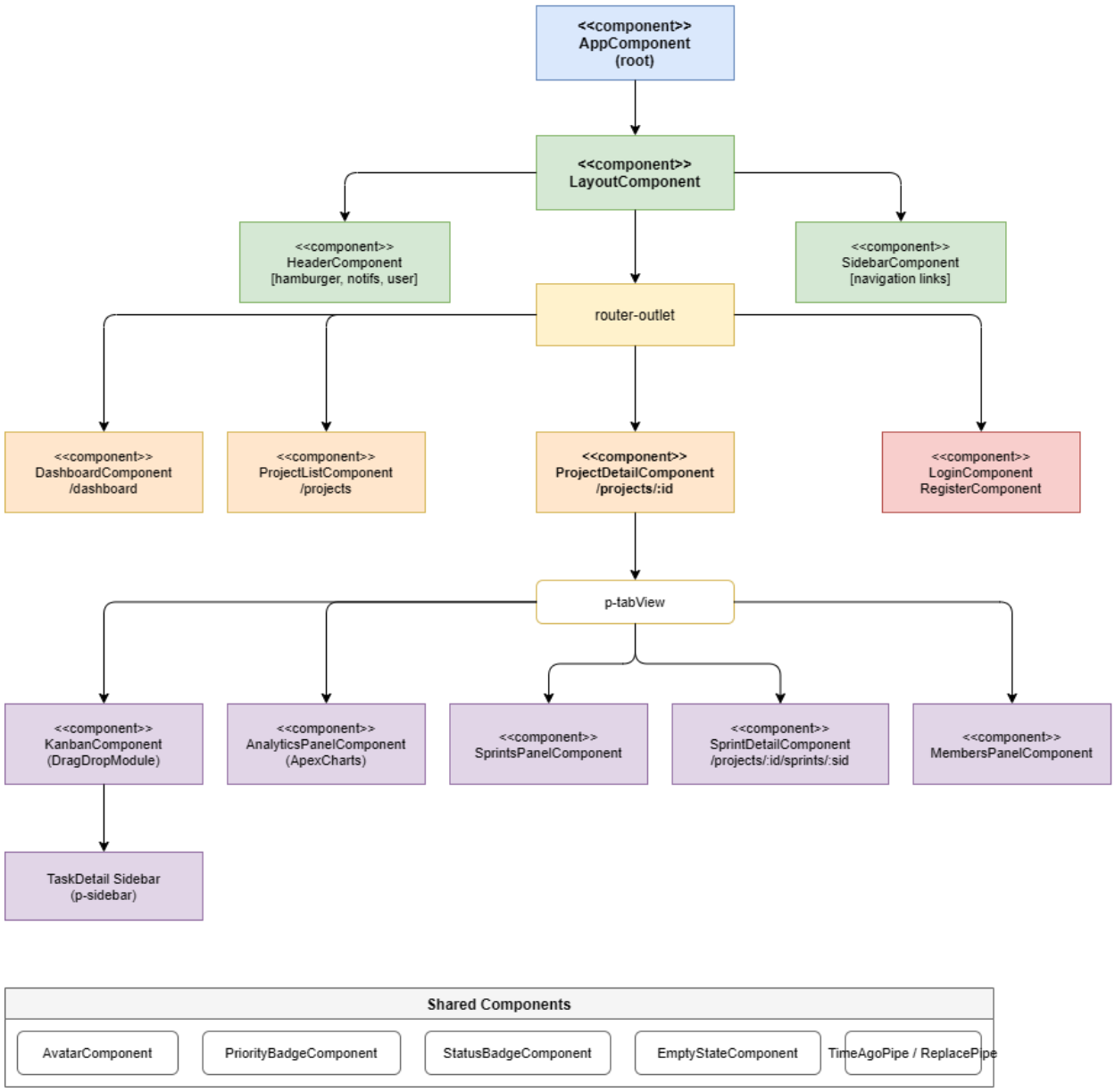

Рисунок 2.3 – Компонентна ієрархія Angular-застосунку

2.3 Проєктування структури бази даних

MongoDB зберігає дані у вигляді документів (BSON) [8, 19]. Розроблено такі колекції: Центральною колекцією системи є tasks – вона містить найбільшу кількість полів і зв'язана з усіма іншими колекціями (див. лістинг 2.1).

Лістинг 2.1 – Схема колекції tasks

```
// Колекція tasks – Mongoose schema
{
    _id: ObjectId,
```

```
    title: String,          // назва задачі (max 300 символів)
    description: String,    // опис (max 5000 символів)
    status: String,      // 'todo' | 'in-progress' | 'review' | 'done'
    priority: String,    // 'critical' | 'high' | 'medium' | 'low'
    project:  ObjectId → projects,
    parent: ObjectId → tasks,
    assignees: [ObjectId] → users,
    createdBy: ObjectId → users,
    dueDate: Date,
    startDate: Date,
    completedAt: Date,
    estimatedHours: Number,
    storyPoints: Number,  // оцінка в story points
    tags:[String],
    attachments: [{ filename, originalName, mimetype, size, url,
                    uploadedBy → users }],
    timeEntries: [{ user → users, startTime, endTime,
                    duration, description }],
    order: Number, // порядок картки у колонці Kanban
    timestamps: { createdAt, updatedAt }
}
```

Зв'язок між спринтом і задачами зберігається в колекції sprints (масив tasks), а не навпаки – задача не має поля sprint. Решта колекцій наведена у зведеній таблиці 2.1.

Таблиця 2.1 – Колекції бази даних веб-застосунку ProjectPulse

Колекція	**Ключові поля**	**Зв’язки**
users	name, email, password(bcrypt), avatar	-
projects	name, description, color, status, priority	owner → users; members → [users]
sprints	name, goal, status, startDate, endDate, completedAt	project → projects; tasks → [tasks]
notifications	type, message, isRead	user, task, project → відповідні колекції
activities	action (enum), createdAt	project, user, task → відповідні колекції

Також наведена ER-діаграма бази даних на рисунку 2.4.

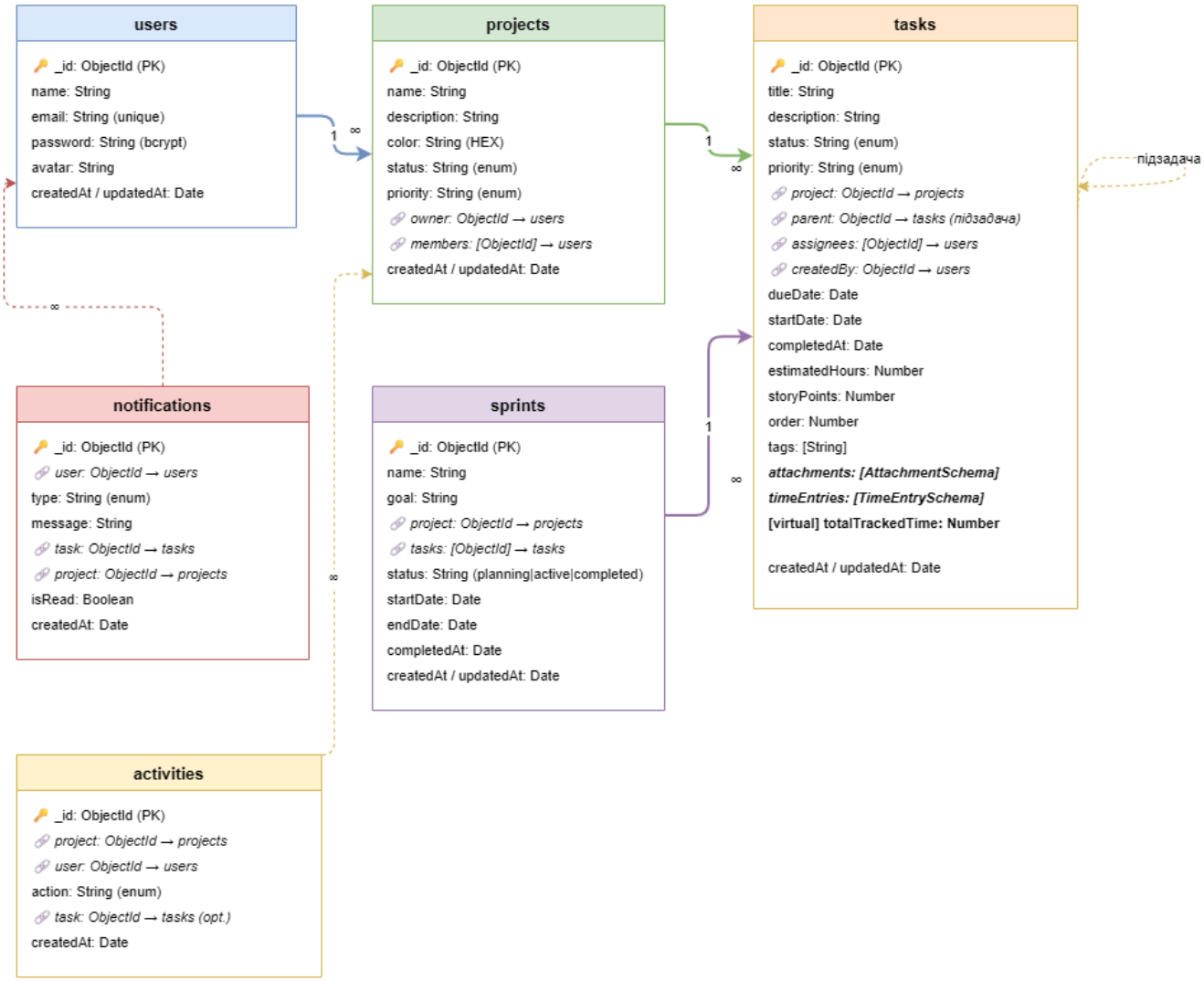

Рисунок 2.4 – ER-діаграма бази даних

2.4 Проєктування REST API

Серверне API побудовано за принципами REST [14, 18] з такими стандартами:

– ресурси іменуються у множині (projects, tasks, sprints);

– використовуються стандартні HTTP-методи: GET, POST, PUT, PATCH, DELETE;

– відповіді повертаються у форматі JSON;

– HTTP-статуси відповідають семантиці операцій (200, 201, 400, 401, 403, 404, 500);

– усі маршрути (крім /auth) захищені middleware verifyToken/.

2.4.1 Маршрути автентифікації

Перелік маршрутів автентифікації подано в таблиці 2.2.

Таблиця 2.2 – Маршрути автентифікації

Метод	**URL**	**Дія**
POST	/api/auth/register	Реєстрація нового користувача
POST	/api/auth/login	Вхід, повертає JWT-токен
GET	/api/auth/me	Отримання профілю поточного користувача

2.4.2 Маршрути проєктів

Маршрути проєктів перелічені в таблиці 2.3.

Таблиця 2.3 – Маршрути проєктів

Метод	**URL**	**Дія**
GET	/api/projects	Список проєктів поточного користувача
POST	/api/projects	Створення проєкту
GET	/api/projects/:id	Деталі проєкту
PUT	/api/projects/:id	Оновлення проєкту
DELETE	/api/projects/:id	Видалення проєкту
GET	/api/projects/:id/stats	Статистика проєкту
POST	/api/projects/:id/members	Додавання учасника
DELETE	/api/projects/:id/members/:userId	Видалення учасника
GET	/api/projects/:id/activities	Журнал активності

2.4.3 Маршрути задач

Маршрути необхідні для роботи з задачами наведені в таблиці 2.4.

Таблиця 2.4 – Маршрути задач

Метод	**URL**	**Дія**
GET	/api/tasks?project=:id	Задачі проєкту
POST	/api/tasks	Створення задачі
GET	/api/tasks/:id	Деталі задачі
PUT	/api/tasks/:id	Оновлення задачі
DELETE	/api/tasks/:id	Видалення задачі
PATCH	/api/tasks/:id/status	Зміна статусу (Kanban drag)
POST	/api/tasks/reorder	Зміна порядку задач у колонці

2.4.4 Маршрути задач

В таблиці 2.5 перераховано маршрути спринтів.

Таблиця 2.5 – Маршрути спринтів

Метод	**URL**	**Дія**
GET	/api/sprints?project=:id	Список спринтів проєкту
POST	/api/sprints	Створення спринту
GET	/api/sprints/:id	Деталі спринту
PUT	/api/sprints/:id	Оновлення спринту
DELETE	/api/sprints/:id	Видалення спринту
POST	/api/sprints/:id/tasks	Додавання задач до спринту
POST	/api/sprints/:id/start	Запуск спринту
POST	/api/sprints/:id/complete	Завершення спринту
GET	/api/sprints/:id/stats	Статистика спринту

2.4.5 Маршрути сповіщень

Всі маршрути, які використовуються для сповіщень наведені в таблиці 2.6.

Таблиця 2.6 – Маршрути сповіщень

Метод	**URL**	**Дія**
GET	/api/notifications	Список сповіщень користувача
PATCH	/api/notifications/:id/read	Позначення як прочитане
PATCH	/api/notifications/read-all	Позначення всіх як прочитані

2.5 Варіанти використання

Система має двох акторів: Гість (неавтентифікований користувач) та Учасник (автентифікований користувач). Власник проєкту є спеціалізацією Учасника з розширеними правами.

Повна UML-діаграма варіантів використання зображена на рисунку 2.5.

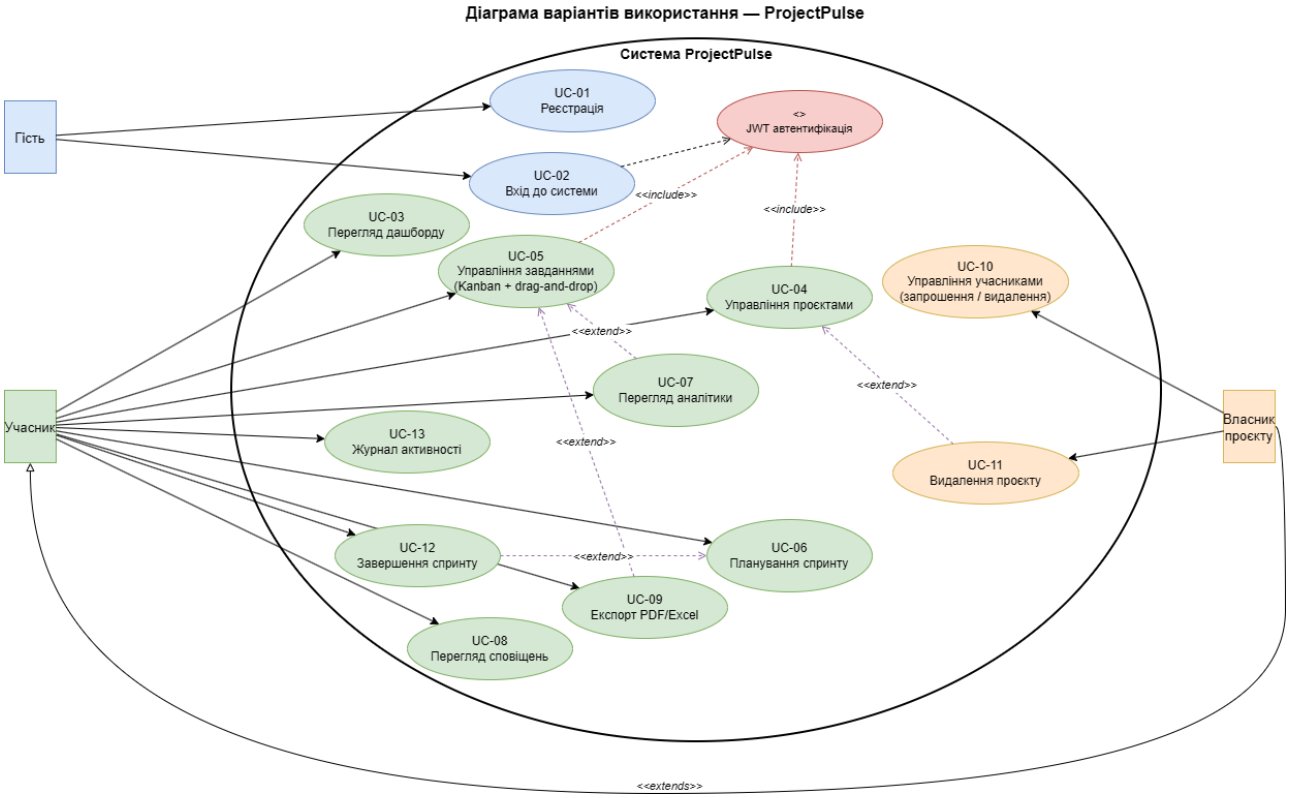

Рисунок 2.5 – UML-діаграма варіантів використання веб-застосунку ProjectPulse

2.6 Діаграми послідовності

Для ілюстрації ключових сценаріїв взаємодії між компонентами розроблено три UML-діаграми послідовності. Кожна діаграма відображає учасників (LifeLine), активаційні смуги (вертикальні прямокутники) та два типи повідомлень: синхронні виклики (суцільна стрілка із заповненим наконечником) та повернення результатів (пунктирна стрілка).

2.6.1 Послідовність: Автентифікація користувача

Діаграма ілюструє процес входу до системи від введення облікових даних до отримання JWT-токена та навігації на дашборд. Реалізовано блок alt, що показує дві гілки: успішну автентифікацію та повернення HTTP 401 Unauthorized (див. рис. 2.6).

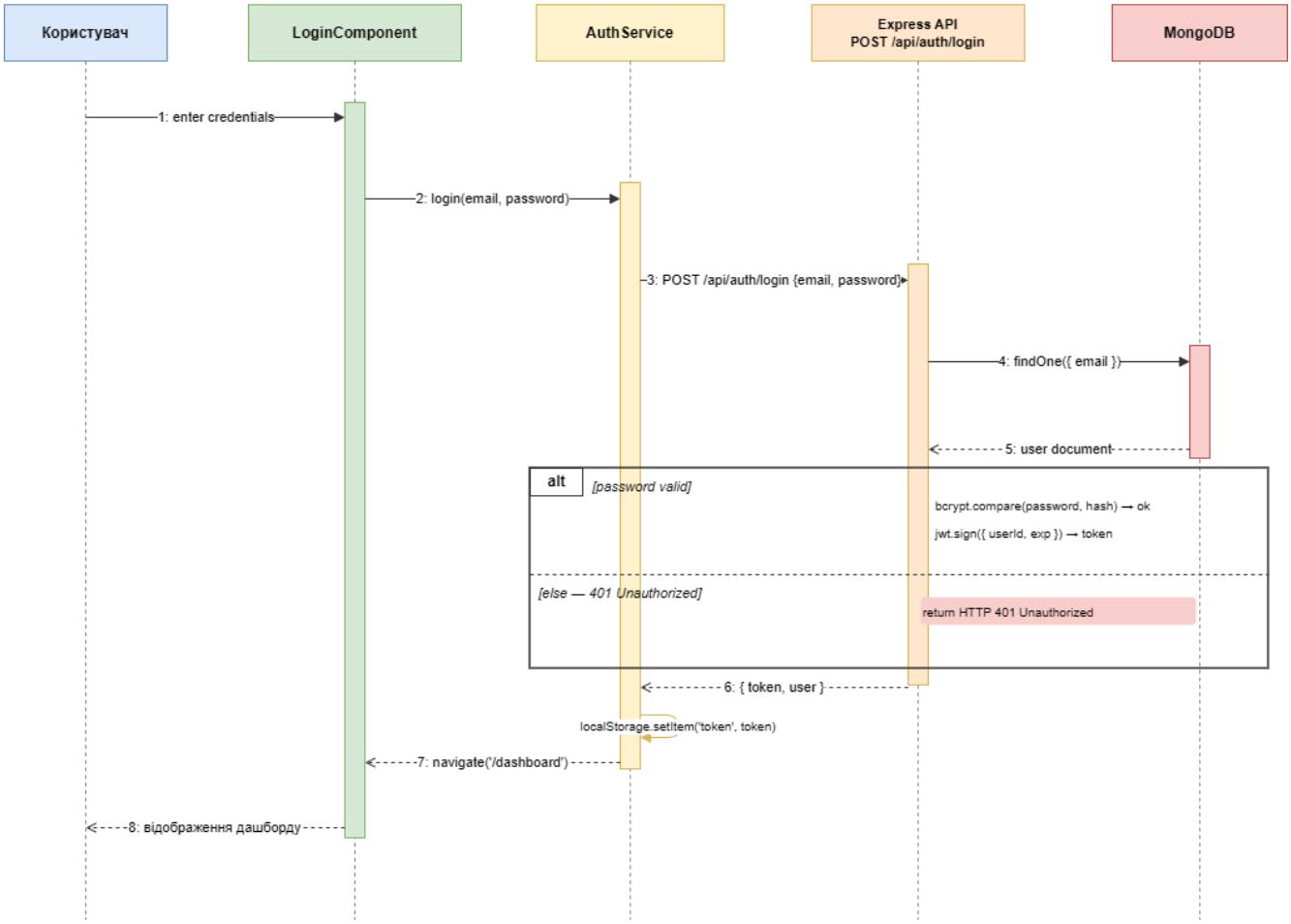

Рисунок 2.6 – Діаграма послідовності: Автентифікація користувача

Учасники: Користувач, LoginComponent, AuthService, Express API (POST /api/auth/login), MongoDB.

Основні кроки:

1. Користувач вводить email і пароль – LoginComponent отримує подію форми.

2. AuthService надсилає HTTP POST-запит на сервер із даними облікового запису.

3. Express API шукає користувача в MongoDB за полем email.

4. У блоці alt [password valid] – bcrypt.compare() [22] підтверджує пароль, jwt.sign() [11] формує токен, сервер повертає { token, user }; гілка [else] – HTTP 401.

5. AuthService зберігає токен у localStorage та ініціює навігацію на /dashboard

2.6.2 Послідовність: Автентифікація користувача

Діаграма показує завершення активного спринту з підтвердженням через PrimeNG-діалог. Блок alt відображає дві гілки: підтверджене завершення (серверне оновлення + статистика) та скасування (без змін) (див. рис. 2.7).

Учасники: Користувач, SprintsPanelComponent, ConfirmationService (PrimeNG), SprintService, Express API (POST /api/sprints/:id/complete).

Основні кроки:

1. Власник проєкту натискає «Complete Sprint» – SprintsPanelComponent викликає ConfirmationService.confirm(), яка показує p-ConfirmDialog.

2. У блоці alt [confirmed]: SprintService надсилає POST-запит; сервер виставляє sprint.status = 'completed', незавершені задачі виключаються зі спринту, формуються сповіщення для учасників.

3. Сервер повертає статистику.

4. Гілка [else – cancelled]: reject() закриває діалог без жодних змін.

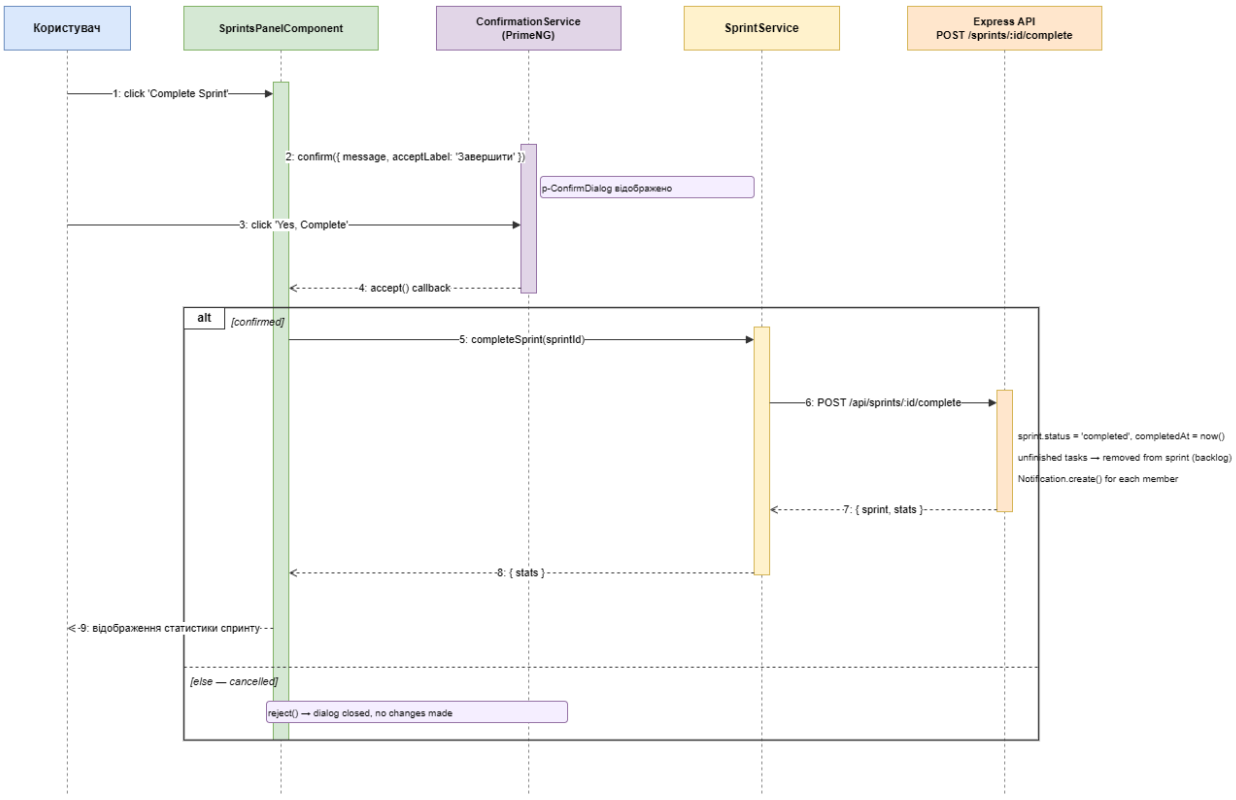

Рисунок 2.7 – Діаграма послідовності: Завершення спринту

2.6.3 Послідовність: Drag-and-drop задачі на Kanban-дошці

Діаграма описує сценарій перетягування картки між колонками. Ключова особливість – оптимістичне оновлення UI: KanbanComponent одразу переміщує картку засобами CDK (self-call transferArrayItem), не чекаючи підтвердження сервера, що забезпечує миттєву реакцію інтерфейсу (див. рис. 2.8).

Учасники: Користувач, KanbanComponent (CDK DragDrop), TaskService, Express API (PATCH /api/tasks/:id/status), Socket.io Server.

Основні кроки:

1. Користувач відпускає картку задачі в нову колонку – спрацьовує подія cdkDropListDropped.

2. KanbanComponent викликає transferArrayItem() для негайного оновлення масивів у пам'яті (optimistic update).

3. TaskService надсилає PATCH-запит; сервер зберігає новий статус у MongoDB та фіксує подію в колекції activities.

4. Express API емітує task_updated через Socket.io у кімнату проєкту – всі підключені учасники отримують оновлення в реальному часі.

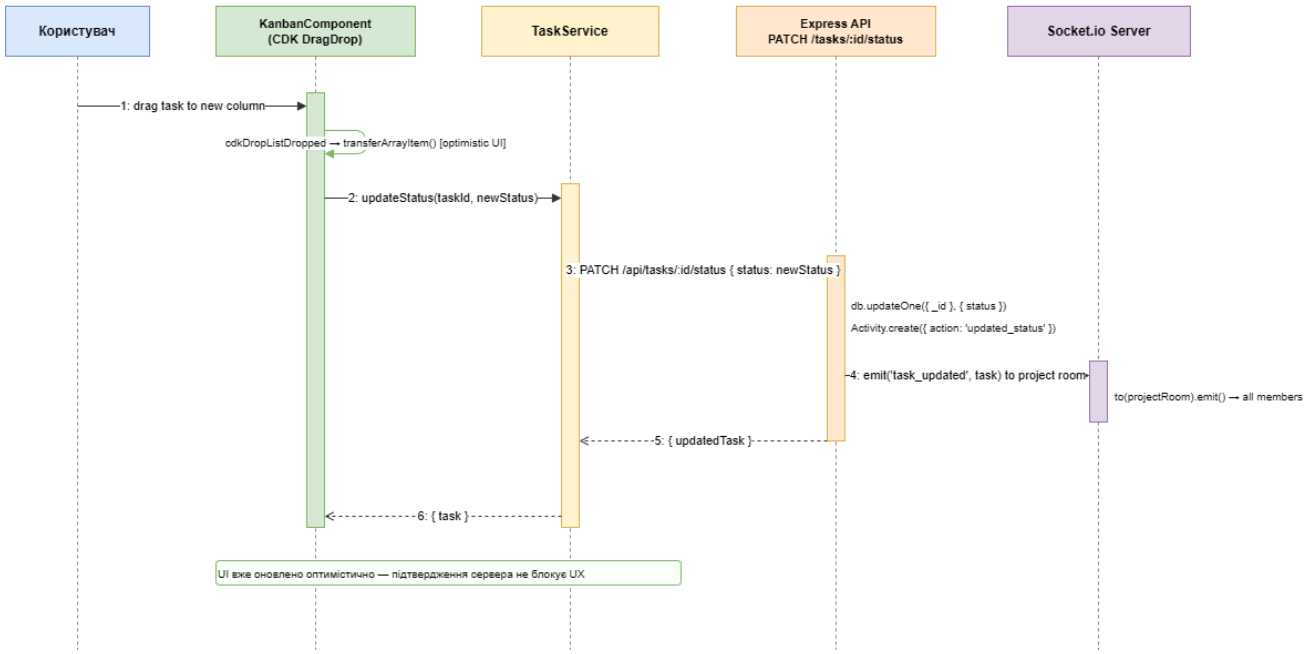

Рисунок 2.8 – Діаграма послідовності: Drag-and-drop задачі на Kanban-дошці

2.7 Проєктування інтерфейсу користувача

Проєктування інтерфейсу спрямоване на забезпечення зручності щоденного використання системи розробниками та менеджерами проєктів. Прийнято рішення будувати UI на єдиній дизайн-системі з фіксованою палітрою кольорів та стандартизованими відступами, що гарантує візуальну узгодженість між різними модулями застосунку. Окрему увагу приділено адаптивності: інтерфейс повинен однаково коректно відображатися як на широкому моніторі, так і на мобільному екрані без деградації функціональності.

2.7.1 Дизайн-система

Для забезпечення візуальної узгодженості в усьому застосунку використовується набір CSS-змінних, визначених у styles.scss (див. лістинг 2.2).

Лістинг 2.2 – styles.css

```
:root {
     --pp-primary: #4F46E5;
     --pp-primary-dark: #4338CA;
     --pp-surface: #ffffff;
     --pp-sidebar-bg: #1E1B4B;
     --pp-border: #E5E7EB;
     --pp-text-muted: #6B7280;
     --pp-radius: 8px;
}
```

Палітра пріоритетів задач:

– critical – червоний (#991B1B на #FEE2E2);

– high – помаранчевий (#9A3412 на #FFEDD5);

– medium – жовтий (#78350F на #FEF3C7);

– low – зелений (#064E3B на #D1FAE5).

2.7.2 Адаптивний дизайн

Реалізовано три ключові breakpoints. Вони зображені в таблиці 2.7.

Таблиця 2.7 – Ключові breakpoints

Breakpoint	**Умова**	**Зміни**
Mobile	max-width: 768px	Sidebar → drawer overlay, hamburger-кнопка
Mobile small	max-width: 540px	Task sidebar → повноекранний
Mobile XS	max-width: 480px	Приховати ім'я користувача в шапці

2.7.3 Ключові сторінки

Сторінка автентифікації – мінімалістична форма входу/реєстрації з логотипом та полями email і пароля. Відображається лише для неавтентифікованих користувачів (див. рис. 2.9).

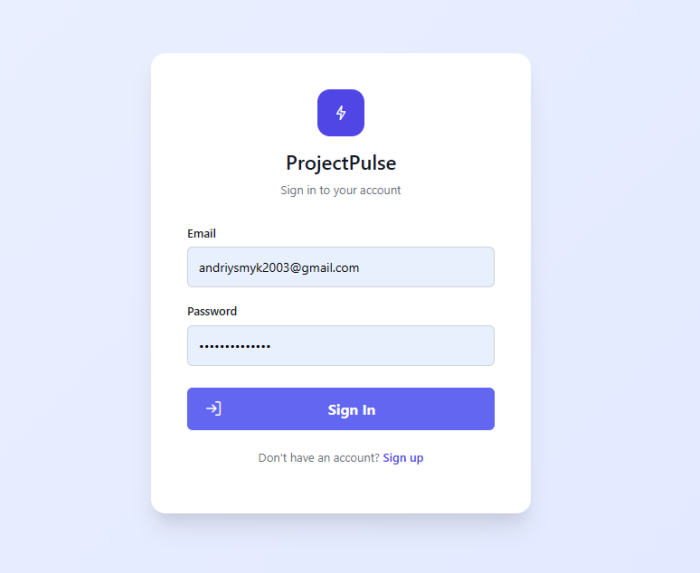

Рисунок 2.9 – Сторінка автентифікації

Дашборд та список проєктів – головна сторінка після входу відображає картки активних проєктів користувача з індикаторами прогресу, пріоритету та кількості задач (див. рис. 2.10).

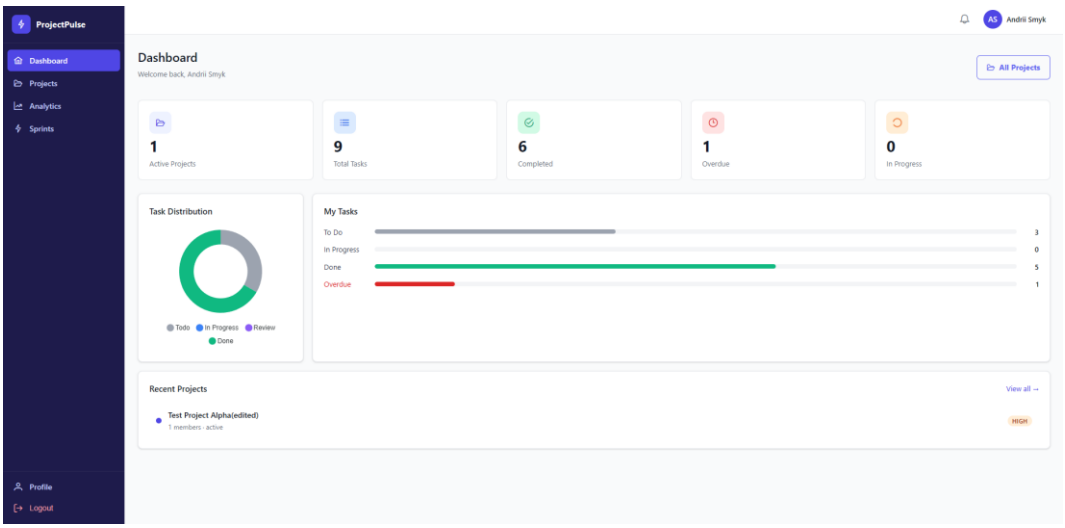

Рисунок 2.10 – Дашборд та список проєктів

Kanban-дошка – основний робочий простір. Чотири колонки (To Do, In Progress, Review, Done) з задачами у вигляді карток. Підтримується drag-and-drop між колонками та всередині колонок (див. рис. 2.11).

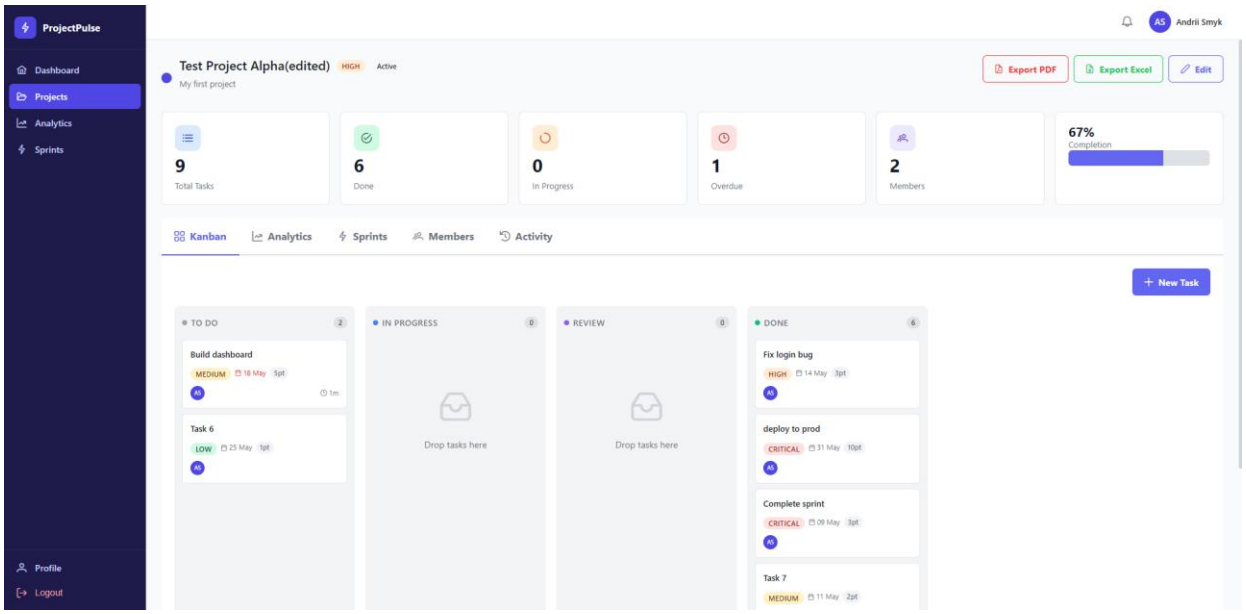

Рисунок 2.11 – Дашборд та список проєктів

Деталі задачі – при кліку на картку відкривається бічна панель (p-sidebar) з повною інформацією про задачу: опис, виконавці, дедлайн, оцінки часу, теги, зміна статусу (див. рис. 2.12).

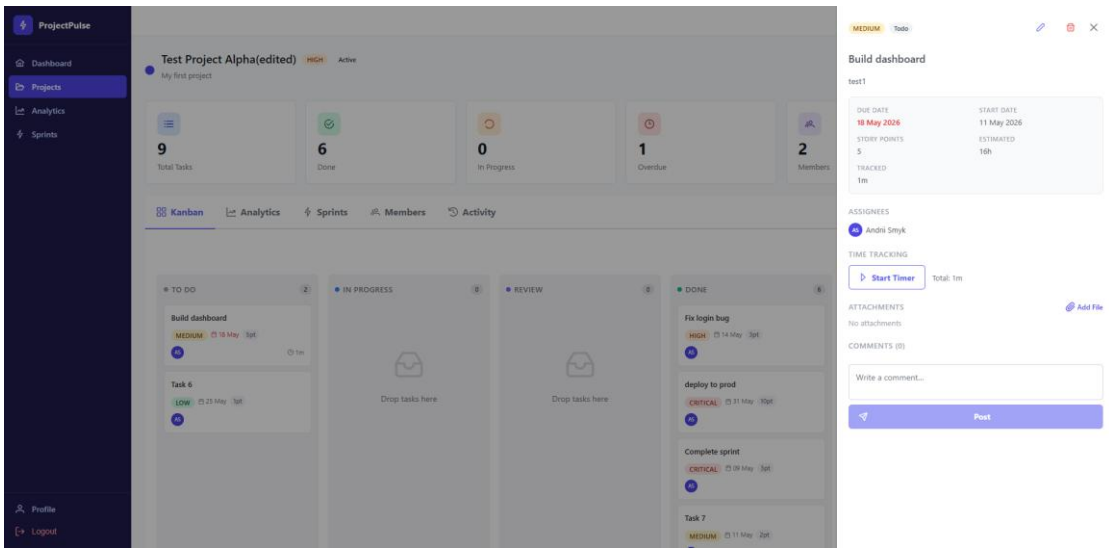

Рисунок 2.12 – Панель з деталями задачі

Управління спринтами – панель зі списком спринтів проєкту: кожен спринт відображає назву, дати, ціль, прогрес-бар і задачі. Кнопки "Start Sprint" та "Complete Sprint" доступні залежно від поточного статусу (див. рис. 2.13).

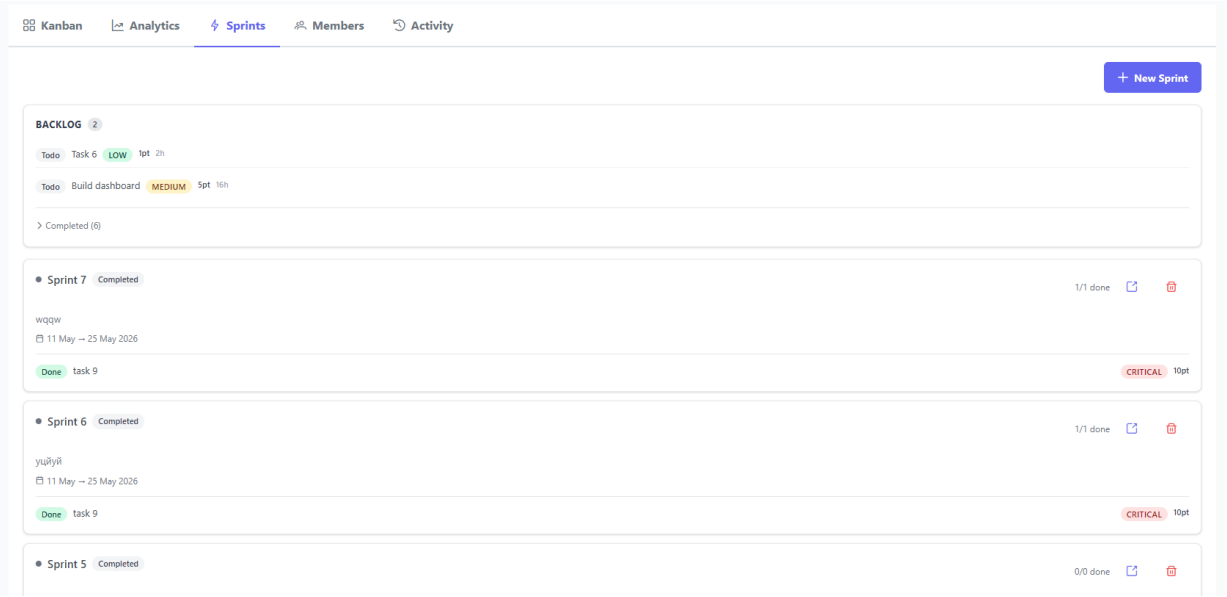

Рисунок 2.13 – Сторінка управління спринтами

Сторінка деталей спринту – деталі спринту з вкладками-фільтрами за статусом, графіками (donut діаграма статусів та bar-графік годин), списком задач.

Аналітична панель – статистика проєкту з графіками розподілу задач, відстеження прогресу, Gantt графік.

2.8 Висновки до розділу 2

У другому розділі проведено комплексне проєктування системи ProjectPulse. Описано трирівневу клієнт серверну архітектуру. Розроблено feature-based структуру Angular-застосунку та компонентну ієрархію. Спроєктовано MongoDB-колекції з необхідними полями та зв'язками між документами. Визначено повний REST API з 30+ маршрутами. Побудовано діаграму варіантів використання та діаграми послідовності для ключових сценаріїв. Описано дизайн-систему та підходи до адаптивного дизайну.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У третьому розділі описано практичну реалізацію платформи ProjectPulse. Розглянуто ключові модулі серверної та клієнтської частин із наведенням найважливіших фрагментів коду. Описано підходи до адаптивного дизайну для мобільних пристроїв та управління спринтами на рівні UI. Завершує розділ опис проведеного тестування: функціональні тест-кейси, виявлені та виправлені дефекти, результати вимірювання продуктивності API.

3.1 Реалізація серверної частини

Серверна частина організована за принципом MVC [16]: маршрути (routes) делегують обробку контролерам (controllers), які взаємодіють з базою даних через Mongoose-моделі (models). Повне дерево директорій наведено у Додатку Б.

Точка входу server.js ініціалізує Express-застосунок, підключає Socket.io та реєструє всі маршрути. Об'єкт io передається у контролери через middleware – це дозволяє будь-якому контролеру надсилати WebSocket-події без глобального стану. Повний лістинг server.js наведено у Додатку Б.

Схеми Mongoose [9] відповідають структурі, спроєктованій у розд. 2.3. Кожна модель має { timestamps: true }, що автоматично додає поля createdAt та updatedAt. Для полів-перерахувань (status, priority) використовується enum – це забезпечує валідацію на рівні бази даних.

Особливість моделі Task – поле sprint допускає значення null: це означає, що задача знаходиться у беклозі і не прив'язана до жодного спринту. Саме це поле використовується при завершенні спринту для масового переміщення незавершених задач назад у беклог.

Повні Mongoose-схеми всіх моделей наведено у Додатку А.

Захист API-маршрутів реалізовано через middleware verifyToken, що перевіряє JWT-токен [11] з заголовка Authorization (див. лістинг 3.1).

Лістинг 3.1 – Middleware автентифікації

```
const jwt = require('jsonwebtoken');
module.exports = (req, res, next) => {
     const token = req.headers.authorization?.split(' ')[1];
     if (!token) return res.status(401).json({ message: 'No token' });
     try {
          req.user = jwt.verify(token, process.env.JWT_SECRET);
          next();
     } catch {
          res.status(401).json({ message: 'Invalid token' });
     }
};
```

Middleware підключається одним рядком на початку кожного захищеного router-файлу: router.use(verifyToken). Таким чином усі маршрути в цьому файлі автоматично вимагають автентифікації без дублювання коду.

Завершення спринту є складною атомарною операцією, що включає три кроки: зміну статусу спринту, масове переміщення незавершених задач у беклог та збір підсумкової статистики для відображення у діалозі. Код реалізації завершення спринту наведено в лістингу 3.2.

Лістинг 3.2 – Код реалізації завершення спринту

```
exports.completeSprint = async (req, res) => {
     const sprint = await Sprint.findById(req.params.id);
     if (!sprint || sprint.status !== 'active')
          return res.status(400).json({ message: 'Sprint is not
     active' });
     // Переміщення незавершених задач у беклог
     await Task.updateMany(
          { sprint: sprint._id, status: { $ne: 'done' } },
          { $set: { sprint: null } }
     );
     sprint.status = 'completed';
     await sprint.save();
     };
     // Збір статистики для діалогу на клієнті
     const tasks = await Task.find({ sprint: sprint._id });
     const doneTasks = tasks.filter(t => t.status === 'done').length;
     const stats = {
          totalTasks: tasks.length,
          doneTasks,
          completionRate: tasks.length
               ? Math.round((doneTasks / tasks.length) * 100) : 0,
          totalPlannedHours: tasks.reduce((s, t) => s +
          (t.estimatedHours || 0), 0),
```

```
            totalTrackedHours:       tasks.reduce((s,    t)    =>    s    +
            (t.trackedHours   || 0), 0),
        };

        res.json({ sprint, stats });
};
```

Після отримання відповіді клієнт відображає діалогове вікно з підсумковою статистикою завершеного спринту (див. рис. 3.1).

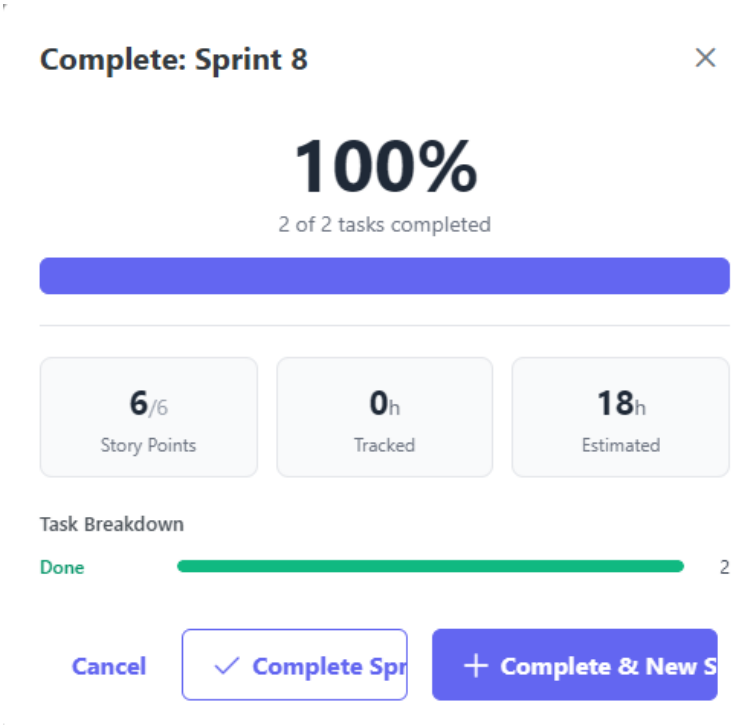

Рисунок 3.1 – Вікно завершення спринту

Реальні сповіщення реалізовано через Socket.io [10]. При призначенні виконавця задачі сервер визначає нових учасників (яких раніше не було), створює записи Notification у базі даних та надсилає їх безпосередньо у персональний профіль кожного користувача. Код реалізації сповіщень подано в лістингу 3.3.

Лістинг 3.3 – Код реалізації сповіщень при призначенні нового користувача

```
for (const userId of newAssignees) {
        const notification = await Notification.create({
                user: userId, type: 'task_assigned',
                message: `You were assigned to "${updatedTask.title}"`,
                task: updatedTask._id, project: updatedTask.project,
```

```
    });
    req.io.to(`user_${userId}`).emit('notification', notification);
}
```

На клієнті SocketService підписується на подію notification і передає її до HeaderComponent через Observable. Лічильник непрочитаних повідомлень у шапці оновлюється миттєво, без перезавантаження сторінки (див. рис. 3.2).

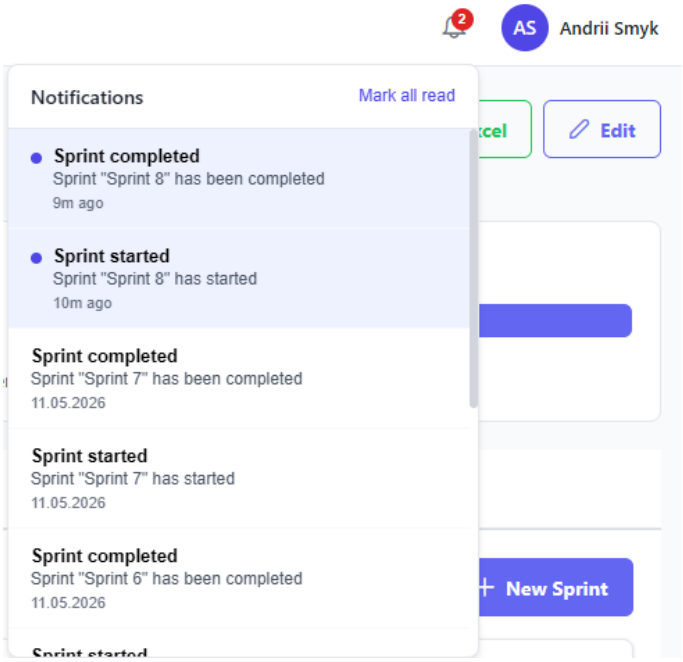

Рисунок 3.2 – Панель сповіщень у шапці

3.2 Реалізація клієнтської частини

Клієнтська частина реалізована як Angular SPA з чітким розподілом на модулі: HTTP-взаємодія з API, Kanban-дошка з drag-and-drop, аналітичні графіки, підтвердження деструктивних дій та експорт даних. Нижче описано ключові з цих модулів із наведенням фрагментів коду.

AuthInterceptor автоматично додає JWT-токен до заголовка кожного вихідного HTTP-запиту [11]. Це дозволяє сервісам звертатися до API без явного передавання токена у кожному виклику (див. лістинг 3.4).

Лістинг 3.4 – Token interceptor

```
intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
     const token = this.auth.token;
     if (!token) return next.handle(req);
     return next.handle(req.clone({
          setHeaders: { Authorization: `Bearer ${token}` }
     }));
}
```

Kanban-компонент використовує Angular CDK DragDropModule [13]. Ключова логіка обробника drop() розрізняє два сценарії: переміщення всередині тієї ж колонки (зміна порядку) та переміщення між колонками (зміна статусу задачі). Код даної реалізації подано в лістингу 3.5.

Лістинг 3.5 – Реалізація drag-and-drop в Kanban-дошці

```
drop(event: CdkDragDrop<Task[]>) {
     if (event.previousContainer === event.container) {
          moveItemInArray(event.container.data,
               event.previousIndex, event.currentIndex);
          this.reorderInColumn(event.container.id,
          event.container.data);
     } else {
          const task =
          event.previousContainer.data[event.previousIndex];
          const newStatus = event.container.id as TaskStatus;
          transferArrayItem(event.previousContainer.data,event.conta
          iner.data,event.previousIndex,event.currentIndex);
          this.taskService.updateStatus(task._id,
          newStatus).subscribe(); }
}
```

UI оновлюється оптимістично (одразу після drop), а HTTP-запит іде асинхронно у фоні. HTML-шаблон компонента з cdkDropList та cdkDrag директивами наведено у Додатку В. При кліку на картку відкривається бічна панель з повними деталями задачі.

AnalyticsPanelComponent завантажує задачі проєкту та будує три типи графіків через ng-apexcharts [12]:

– Donut-діаграма – розподіл задач за статусами;

– Bar-графік – кількість задач за пріоритетами;

– Gantt-рядки – часова шкала задач з дедлайнами відносно поточної дати.

Графіки будуються динамічно на основі поточного стану задач. Якщо задач немає, замість порожніх графіків відображається компонент EmptyState. Вигляд аналітичної панелі наведено на рисунках 3.3-3.5.

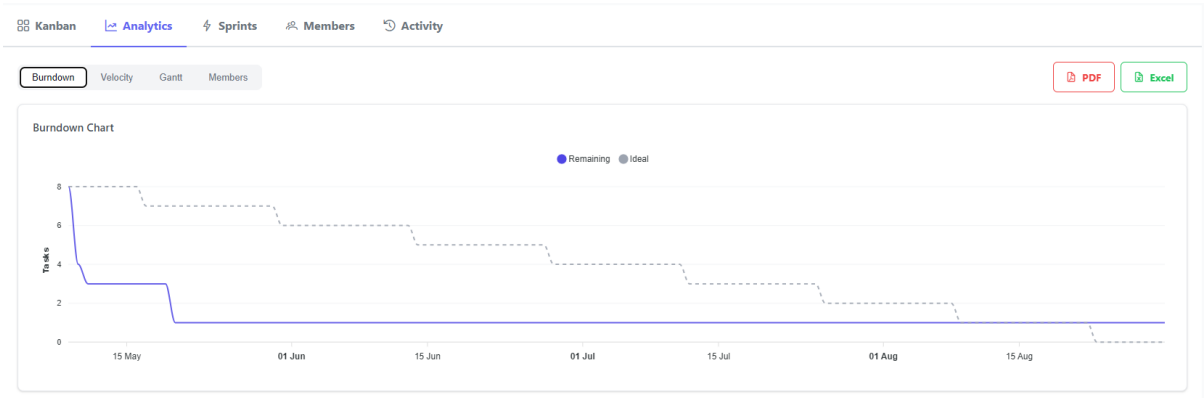

Рисунок 3.3 – Burndown графік

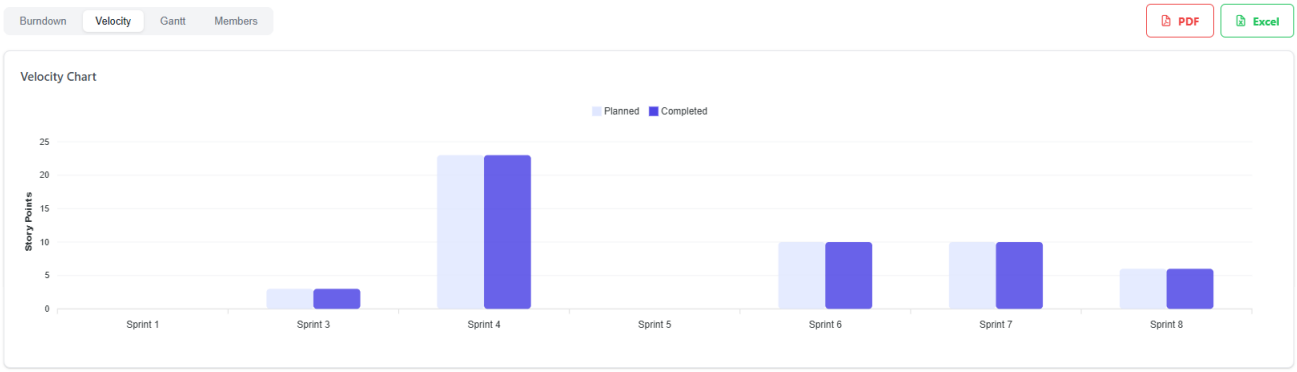

Рисунок 3.4 – Velocity графік

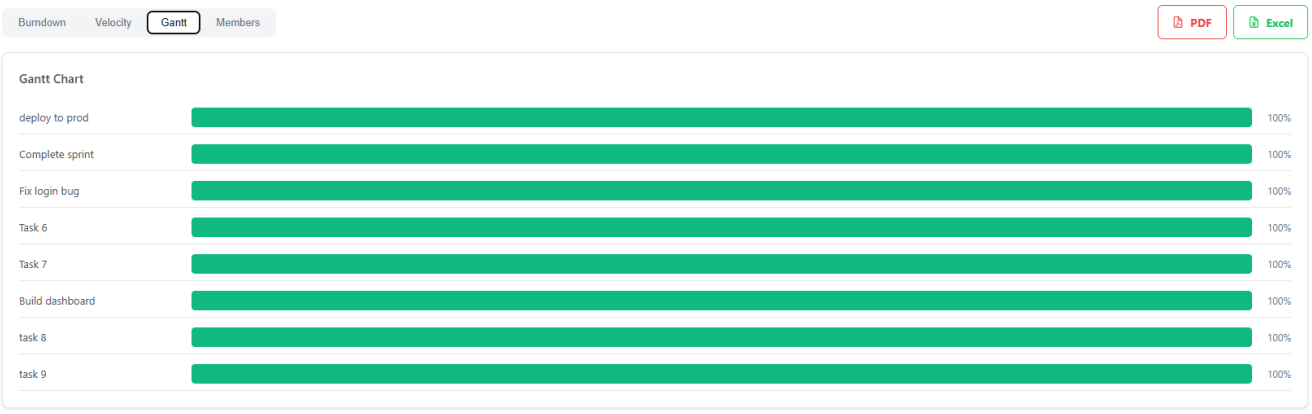

Рисунок 3.5 – Діаграма Gantt

Усі деструктивні операції (видалення проєкту, задачі, учасника, спринту) підтверджуються через PrimeNG ConfirmationService замість нативного window.confirm(). Це забезпечує консистентний UX та відповідає дизайн-системі застосунку. Реалізація вікна підтвердження подана в лістингу 3.6.

Лістинг 3.6 – Реалізація вікна підтвердження при видалені проєкту

```
deleteProject(project: Project) {
     this.confirmService.confirm({
          message:    `Are    you    sure    you    want    to    delete
          "${project.name}"?`,
          header: 'Delete Project',
          icon: 'pi pi-exclamation-triangle',
          acceptButtonStyleClass: 'p-button-danger',
          accept: () => this.projectService.delete(project._id)
                    .subscribe(() => this.load()),
     });
}
```

Вигляд модального вікна підтвердження показано на рис. 3.6.

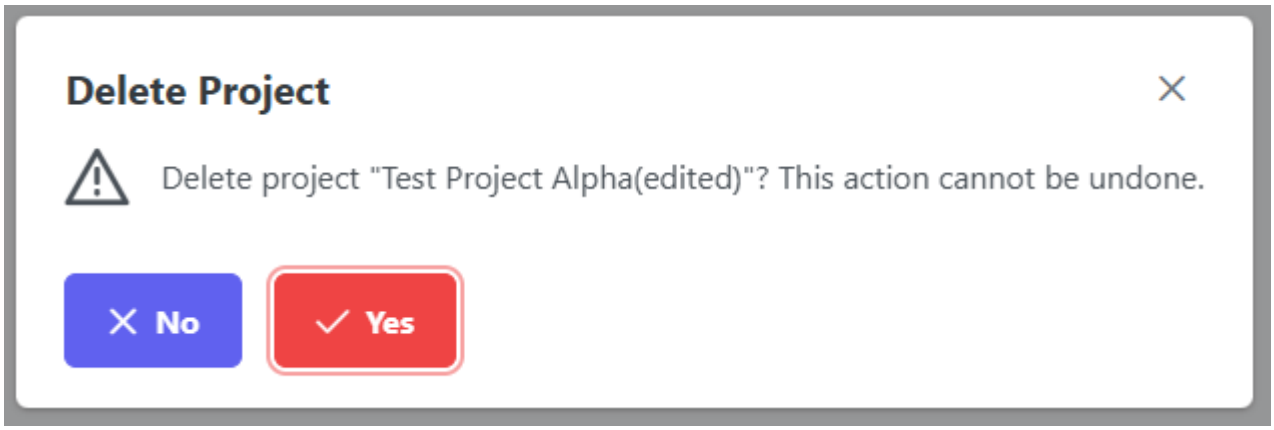

Рисунок 3.6 – Вікно підтвердження при видалені проєкту

Реалізовано два формати експорту даних проєкту:

– PDF – за допомогою бібліотеки jsPDF [20] + jspdf-autotable: заголовок з назвою проєкту та таблиця задач з полями Title, Status, Priority, Assignees, Due Date;

– Excel – за допомогою бібліотеки xlsx [21]: аркуш з розширеним набором колонок, включаючи Estimated та Tracked Hours.

Обидві функції виконуються повністю на стороні клієнта – без звернення до сервера. Повний код функцій exportPdf() та exportExcel() наведено у Додатку В.

3.3 Управління спринтами – клієнтська частина

Панель спринтів (SprintsPanelComponent) відображає список усіх спринтів проєкту, відсортованих за датою створення (новіші – вгорі). Для кожного спринту показується прогрес виконання задач у вигляді breakdown-барів за статусами (див. рис. 3.7).

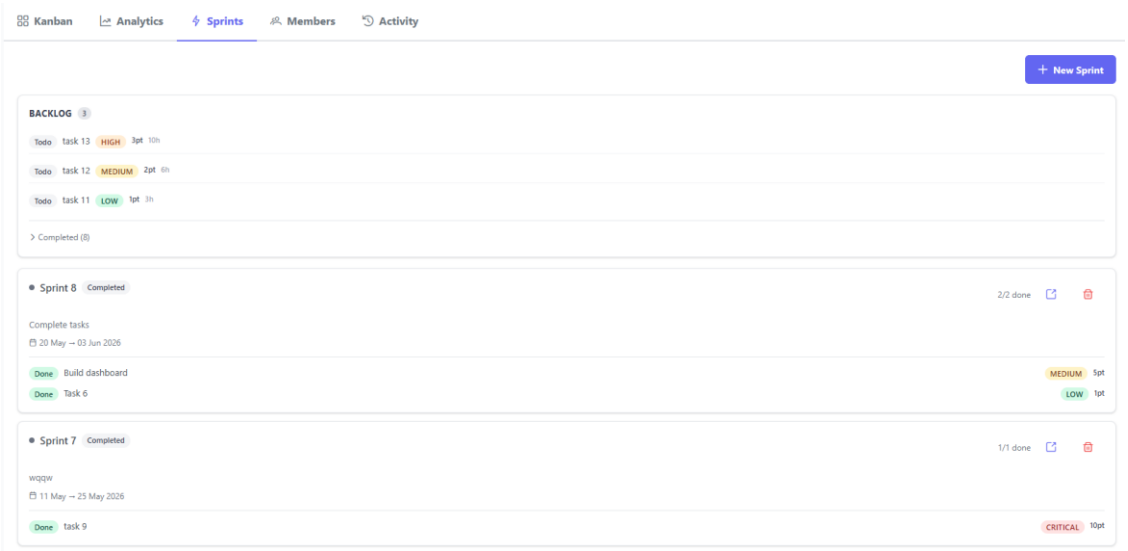

Рисунок 3.7 – Панель спринтів

Доступність кнопок керування визначається статусом спринту (див. таб. 3.1).

Таблиця 3.1 – Доступність кнопок керування спринтом

Статус спринту	**Доступні дії**
planning	Start Sprint, Edit, Delete, Add Tasks, Remove Task
active	Complete Sprint, (задачі заблоковані)
completed	Перегляд статистики (тільки читання)

Блокування задач реалізовано умовою *ngIf="sprint.status === 'planning'" замість !== 'completed' – це запобігає модифікації задач і в активному спринті.

Сторінка деталей спринту (SprintDetailComponent) надає повну картину прогресу: вкладки-фільтри за статусом, графіки (donut-статусів та bar-годин) та посортований список задач (див. рис. 3.8).

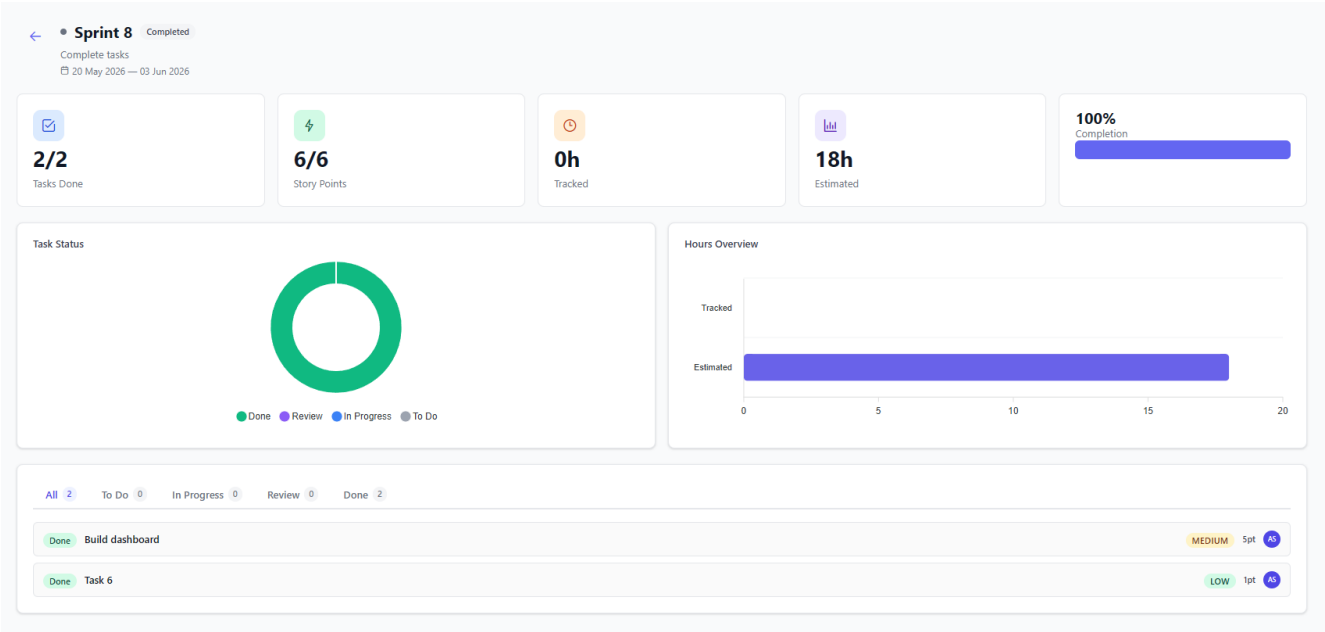

Рисунок 3.8 – Сторінка деталей спринту

3.4 Адаптивний дизайн для мобільних пристроїв

Оскільки платформа орієнтована на щоденне використання, забезпечення коректного відображення на мобільних пристроях є невід'ємною частиною реалізації [24]. Адаптивність досягається виключно засобами CSS – медіа-запитами та flexbox-компонуванням, без підключення сторонніх бібліотек. Нижче описано найбільш характерні рішення, що усувають проблеми, виявлені під час тестування адаптивності.

3.4.1 Мобільна навігація (drawer-overlay)

На екранах шириною до 768px навігаційна панель прихована за межами екрана через transform: translateX(-100%) [25]. При натисканні hamburger-кнопки

клас sidebar-open додається до елемента, що анімовано зсуває панель у видиму зону. В лістингу 3.7 подана реалізація інтерфейсу на мобільному екрані.

Лістинг 3.7 – Реалізація інтерфейсу на мобільному екрані

```
@media (max-width: 768px) {
      .sidebar {
            position: fixed; top: 0; left: 0; height: 100vh;
            z-index: 200; transform: translateX(-100%);
            transition: transform 0.25s ease;
            &.sidebar-open { transform: translateX(0); }
      }
}
```

LayoutComponent підписується на NavigationEnd-події Router і автоматично закриває drawer при переході між сторінками – без жодного додаткового коду у дочірніх компонентах. На рисунку 3.9 показано мобільний вигляд застосунку: drawer відкритий поверх контенту з напівпрозорим backdrop.

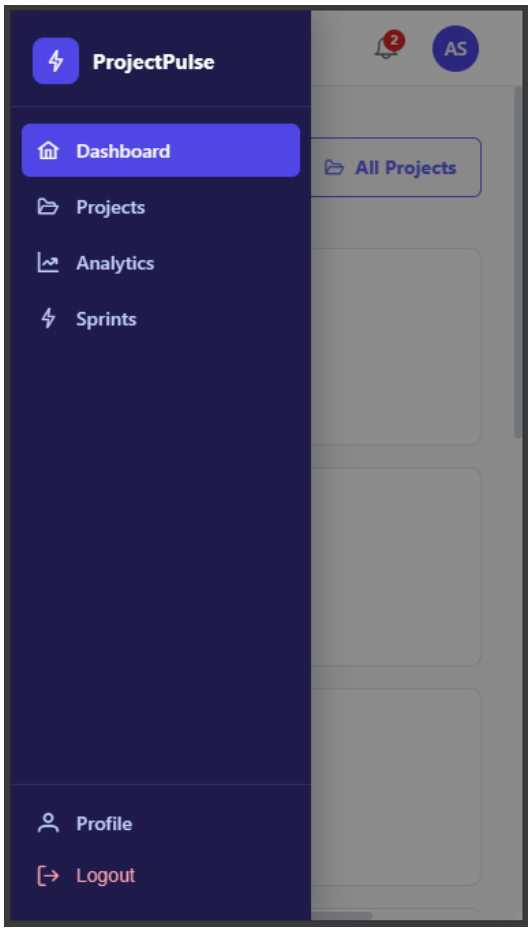

Рисунок 3.9 – Вигляд застосунку на мобільному пристрої

3.4.2 Інші адаптивні рішення

В таблиці 3.2 наведено перелік інших рішень які було впроваджено задля вирішення проблеми, які було помічено в процесі розробки.

Таблиця 3.2 – Проблеми адаптивності і їх вирішення

Проблема	**Рішення**	**Breakpoint**
Кнопки експорту виходять за межі	flex-wrap: wrap у контейнері	≤ 600px
Бейдж перекриває назву спринту	overflow: hidden; text-overflow: ellipsis; min-width: 0	будь-який
Вкладки спринту виходять за межі	overflow-x: auto; scrollbar-width: none	≤ 480px
Task detail sidebar непридатний до використання	width: 100vw !important	≤ 540px
Ім'я користувача в шапці	display: none	≤ 480px
Дії спринту накладаються	flex-direction: column; width: 100%	≤ 520px

3.5 Тестування системи

Після завершення реалізації основних модулів проведено комплексне тестування системи. Метою тестування було підтвердження коректності роботи функціональних вимог, визначення проблем з адаптивністю та вимірювання продуктивності серверних ендпоінтів. Усі виявлені дефекти були задокументовані та виправлені до фінальної версії застосунку.

3.5.1 Методологія тестування

Тестування проводилось у чотири рівні:

1. Ручне функціональне тестування – перевірка кожного варіанту використання за сценарієм.

2. Тестування API – за допомогою Postman: набори запитів для кожної групи ендпоінтів з перевіркою статус-кодів і структури відповіді.

3. Кросбраузерне тестування – Chrome 124, Firefox 125, Edge 124.

4. Тестування адаптивності – DevTools Responsive Mode для ширин 360px, 480px, 540px, 768px, 1280px, 1920px.

На рисунку 3.10 показано процес тестування API в Postman.

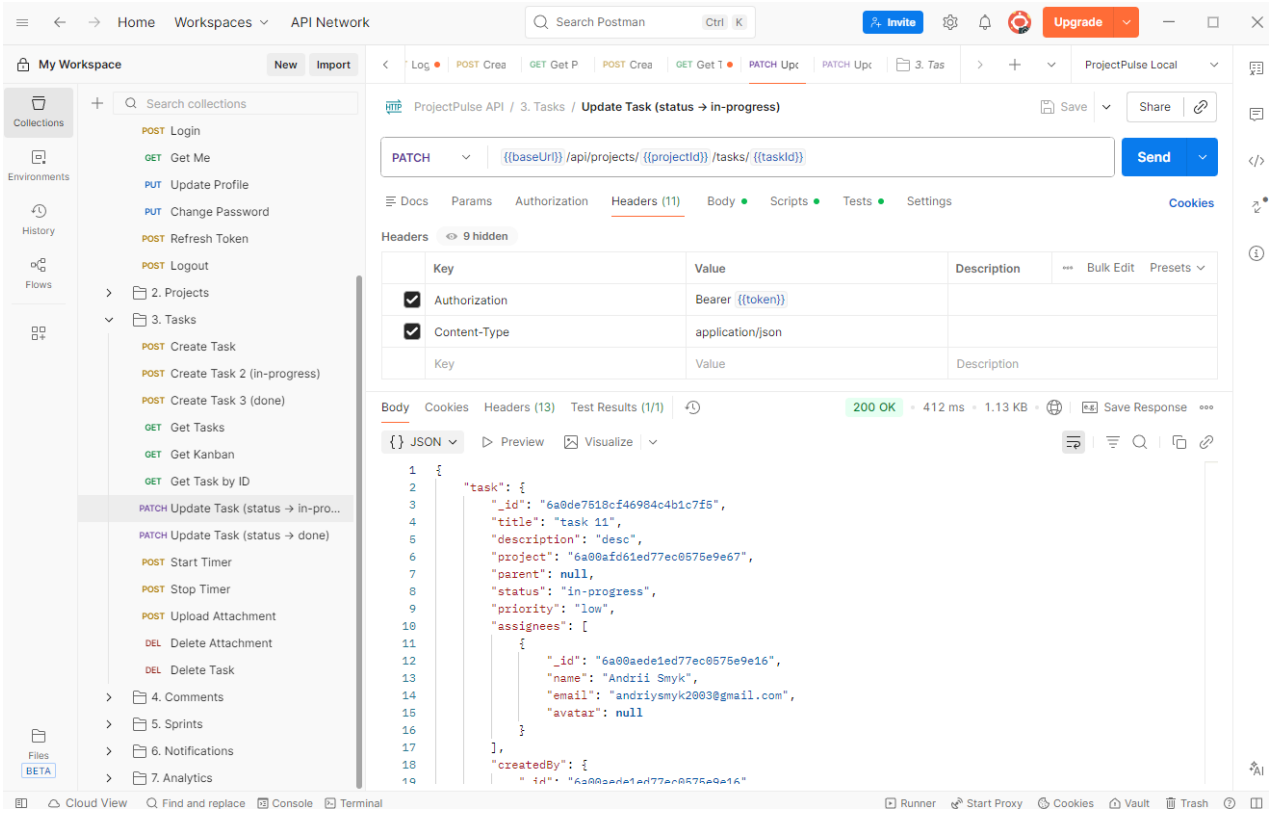

Рисунок 3.10 – Тестування REST API в Postman: запит PATCH /api/tasks/:id/status і відповідь сервера з оновленим об'єктом задачі

3.5.2 Тест-кейси для ключових функцій

Кроки та результати тестування для реєстрації та входу подані у таблиці 3.3.

Таблиця 3.3 – Тест-кейси для реєстрації та входу

Крок	Очікуваний результат	Результат
Заповнити форму реєстрації та натиснути Register	Перенаправлення на /dashboard, JWT збережено	+
Вийти та увійти з правильними даними	Перенаправлення на /dashboard	+
Ввести неправильний пароль	Повідомлення про помилку	+
Відкрити /dashboard без входу	Перенаправлення на /login	+

Наступним було проведено тестування drag-and-drop. Його результати наведено в таблиці 3.4.

Таблиця 3.4 – Тест-кейси для drag-and-drop

Крок	**Очікуваний результат**	**Результат**
Перетягнути задачу в наступну колонку	Статус оновлюється, UI відображає нову колонку	+
Перетягнути задачу всередині колонки	Порядок задач оновлюється	+
Перетягнути задачу назад	Статус повертається	+

Далі було протестовано функціонал спринтів. Результати даного тестування подані в таблиці 3.5.

Таблиця 3.5 – Тест-кейси для спринтів

Крок	**Очікуваний результат**	**Результат**
Створити спринт	Спринт зі статусом planning у списку	+
Додати задачі в статусі planning	Задачі прив'язуються до спринту	+
Спробувати додати задачі до активного спринту	Кнопка Add Tasks відсутня	+
Завершити спринт	Незавершені задачі → беклог, діалог статистики	+

Після цього було проведено тестування правильної роботи панелі сповіщень в різних ситуаціях. Результати показані в таблиці 3.6.

Таблиця 3.6 – Тест-кейси для панелі сповіщень

Крок	**Очікуваний результат**	**Результат**
Призначити задачу іншому користувачу	Сповіщення з'являється без перезавантаження	+
Відкрити панель сповіщень	Список з unread-лічильником	+
Натиснути Mark all as read	Лічильник обнулюється	+

Останнім було проведено тестування мобільної адаптивності. Його результати подані в таблиці 3.7.

Таблиця 3.7 – Тест-кейси для мобільної адаптивності

Крок	**Очікуваний результат**	**Результат**
Відкрити застосунок	Sidebar прихований, hamburger видимий	+
Натиснути hamburger	Sidebar відкривається як overlay	+
Перейти на іншу сторінку	Sidebar закривається автоматично	+
Відкрити task detail sidebar	Відображається на повну ширину	+

3.5.3 Виявлені та виправлені дефекти

У процесі розробки було виявлено низку дефектів які могли погіршити досвід користування застосунком та навіть місцями робили це неможливим. В більшості, ці баги були пов’язані з відображенням кнопок та панелей у мобільній версії веб-застосунку. Виправлені дефекти а також їх причини і суть виправлення подано у таблиці 3.8.

Таблиця 3.8 – Виявленні та виправлені дефекти

Дефект	**Причина**	**Виправлення**
Кнопки у footer діалогу переносяться	Відсутній flex wrap: nowrap	Глобальне правило у styles.scss
Tooltip переносить слово посередині	Відсутній white space: nowrap	.p-tooltip-text { white-space: nowrap !important }
Задачі можна додавати до активного спринту	Умова !== 'completed'	Змінено на === 'planning'
Бейдж перекриває назву спринту	Відсутній min width: 0	overflow: hidden; text-overflow: ellipsis; min-width: 0
Вкладки спринту виходять за межі	Відсутній overflow x: auto	Scrollable контейнер зі схованим scrollbar
Task sidebar непридатний на мобільному	Фіксована ширина p-sidebar	@media (max-width:540px) { width:100vw !important }
Нативний confirm() замість модалки	Браузерний API	Замінено на PrimeNG ConfirmationService у 4 компонентах
Спринти сортуються за startDate	sort({ startDate:-1 })	Змінено на sort({ createdAt: -1 })

3.5.4 Тестування продуктивності

Виміряно час відповіді ключових API-запитів при 50 задачах у проєкті. Результати тестування відображено в таблиці 3.9.

Таблиця 3.9 – Результати тестування продуктивності

Ендпоінт	**Серердній час відповіді**
GET /api/projects	45 мс
GET /api/tasks?project=:id	62 мс
GET /api/sprints?project=:id	38 мс
GET /api/projects/:id/stats	78 мс
PATCH /api/tasks/:id/status	41 мс

Усі результати значно нижче встановленого порогу у 500 мс.

3.6 Розгортання системи

Для запуску системи у режимі розробки необхідно Node.js ≥ 18.x, MongoDB ≥ 6.x та Angular CLI ≥ 14.x. Серверна частина запускається командою npm run dev (порт 3000), клієнтська – ng serve (порт 4200). Змінні середовища (рядок підключення до MongoDB, JWT-секрет) задаються у файлі .env в кореневій директорії серверного проєкту. Детальна інструкція розгортання наведена у Додатку Д.

3.7 Висновки до розділу 3

У третьому розділі описано реалізацію ключових модулів системи ProjectPulse. Наведено фрагменти коду найважливіших алгоритмів: JWT-middleware, завершення спринту з переміщенням задач у беклог, drag-and-drop логіка Kanban-дошки, WebSocket-сповіщення, підтвердження деструктивних дій через PrimeNG ConfirmationService.

Описано підходи до адаптивного дизайну для мобільних пристроїв. Проведено ручне функціональне тестування за 5 групами тест-кейсів (21 кроком), тестування API в Postman та тестування адаптивності. Виявлено та виправлено 8 дефектів. Час відповіді API для всіх ключових ендпоінтів не перевищує 80 мс.

4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У четвертому розділі розглянуто питання охорони праці, що є обов'язковою складовою кваліфікаційної роботи відповідно до вимог нормативних документів вищої школи. У першому підрозділі розглянуто аварії з викидом радіоактивних речовин як один з найнебезпечніших видів техногенних надзвичайних ситуацій – їх класифікацію, вражаючі фактори та заходи захисту населення і персоналу. У другому підрозділі описано порядок організації служби охорони праці на підприємстві: правову базу, структуру служби залежно від чисельності персоналу, види інструктажів та вимоги до атестації робочих місць.

4.1 Аварії з викидом радіоактивних речовин

Радіаційна аварія – це порушення меж безпечної експлуатації об'єктів ядерної енергетики або інших джерел іонізуючого випромінювання, що призводить до неконтрольованого викиду радіоактивних речовин у навколишнє середовище. Наслідки таких аварій можуть охоплювати значні географічні території – залежно від типу джерела, кількості викинутих речовин та погодних умов [29].

Головна небезпека полягає в тому, що іонізуюче випромінювання не має кольору, запаху і жодними органами чуття не визначається – людина може отримати дозу опромінення, навіть не підозрюючи про це. Саме тому радіаційні аварії відносять до особливо небезпечних видів надзвичайних ситуацій техногенного характеру.

Для класифікації аварій на ядерних об'єктах використовується міжнародна шкала INES (International Nuclear and Radiological Event Scale), розроблена МАГАТЕ. Вона має 7 рівнів: від аномалій (рівень 1) до великих аварій з масовим викидом радіоактивних речовин (рівень 7). Рівень 7 – найвищий – отримали лише дві аварії в історії: Чорнобильська (1986) та Фукусіма-1 (2011). Для України

Чорнобильська катастрофа залишається найбільшою радіаційною трагедією, яка спричинила довгострокове радіоактивне забруднення значної частини території держави.

Захист населення від наслідків радіаційних аварій в Україні регулюється такими основними документами [28]:

– Закон України «Про захист людини від впливу іонізуючих випромінювань» від 14.01.1998 № 15/98-ВР;

– Закон України «Про використання ядерної енергії та радіаційну безпеку» від 08.02.1995 № 39/95 ВР;

– Закон України «Про Цивільний захист» від 02.10.2012 № 5403-VI;

– Норми радіаційної безпеки України (НРБУ-97), затверджені МОЗ у 1997 році;

– Основні санітарні правила забезпечення радіаційної безпеки України (ОСПУ-2005).

Ці документи встановлюють гранично допустимі дози опромінення для населення та персоналу, порядок дій під час аварій, вимоги до планування захисних заходів та контролю радіаційної обстановки.

Потенційні джерела радіаційних аварій поділяються на кілька категорій:

1. Атомні електростанції. Для України це – передусім Запорізька (найбільша в Європі), Рівненська, Хмельницька та Південноукраїнська АЕС. Кожна з них має зону планування захисних заходів радіусом 30 км. Підприємства та населені пункти, що потрапляють у ці зони, мають власні плани евакуації та запаси засобів захисту.

2. Промислові та медичні джерела. Це рентгенологічне обладнання, прилади неруйнівного контролю, медичні гамма-установки для стерилізації, радіоізотопні генератори тощо. Аварії з такими джерелами, як правило, мають локальний характер, проте несуть значну небезпеку для обмеженого кола осіб.

3. Транспортування радіоактивних матеріалів. Дорожньо-транспортні пригоди з автоцистернами або вагонами, що перевозять радіоактивні речовини, також можуть призводити до викиду та локального забруднення.

4. Втрачені або вкрадені джерела. Нерідко є наслідком недбалого поводження з промисловим радіаційним обладнанням. Такі випадки потребують оперативного реагування з боку ДСНС та Держатомрегулювання.

При радіаційній аварії основними вражаючими факторами є:

– зовнішнє опромінення – вплив гамма- та рентгенівського випромінювання від радіоактивної хмари або забрудненої місцевості;

– внутрішнє опромінення – надходження радіонуклідів в організм через органи дихання з повітрям, через шлунково-кишковий тракт із забрудненими продуктами та водою, а також через шкіру;

– бета-опромінення шкіри – при контакті з радіоактивним пилом або рідиною. Ступінь ураження залежить від виду випромінювання, отриманої дози та часу її накопичення.

При дозах понад 1 Гр розвивається гостра променева хвороба – від нудоти і слабкості до пригнічення кровотворення та летального результату. Хронічне опромінення малими дозами підвищує ризик онкологічних захворювань і генетичних змін. Щитоподібна залоза особливо вразлива до радіоактивного йоду-131, тому йодна профілактика є одним із першочергових заходів захисту.

Захист від радіаційних аварій здійснюється за кількома напрямами:

– сповіщення. Населення отримує сигнал через сирени, радіо та телебачення з інструкцією – укритися в приміщеннях, загерметизувати вікна та двері, вимкнути вентиляцію;

– укриття. Капітальна будівля знижує дозу гамма-опромінення в 2–10 разів, підвальні приміщення – у 40–100 разів. Провітрювання під час аварії неприпустиме;

– йодна профілактика. Прийом йодиду калію (125 мг для дорослих) блокує накопичення радіоактивного йоду-131 у щитоподібній залозі. Ефективний лише при прийомі до або в перші кілька годин після надходження радіонукліду в організм;

– евакуація та дезактивація. При значному забрудненні місцевості проводиться евакуація з подальшим дозиметричним контролем. Після перебування

в зоні забруднення необхідно зняти верхній одяг (затримує до 80% пилу), прийняти душ і промити носоглотку водою;

– ЗІЗ. Для захисту органів дихання використовуються респіратори FFP3; за їх відсутності – вологі ватно марлеві пов'язки у кілька шарів. Шкіру закривають одягом, що щільно прилягає.

При отриманні сигналу тривоги персонал діє згідно з планом цивільного захисту підприємства:

1. Припинити роботу, залишитися в приміщенні, закрити вікна та двері, вимкнути вентиляцію.
2. Увімкнути радіо або телебачення, слідувати інструкціям ДСНС.
3. Прийняти йодид калію за вказівкою уповноважених органів.
4. Підготувати документи, воду, герметично упаковані продукти та засоби захисту органів дихання.
5. За розпорядженням про евакуацію – організовано залишити будівлю і рухатися на збірний пункт.

Не слід піддаватися паніці та поширювати неперевірену інформацію. Актуальні відомості про радіаційну обстановку публікує ДСНС та Державна інспекція ядерного регулювання України.

Гранично допустима доза опромінення для населення за НРБУ-97 – 1 мЗв на рік у штатних умовах; для рятувальників під час ліквідації аварії – не більше 100 мЗв за одне втручання. Контроль здійснюється за допомогою індивідуальних дозиметрів та мережі автоматичних постів АСКРО, дані якої в режимі реального часу публікуються на сайті Держатомрегулювання.

4.2 Організація служби охорони праці на підприємстві

Правову основу охорони праці в Україні складає Закон України «Про охорону праці» від 14.10.1992 № 2694-XII (із змінами та доповненнями), який визначає основні положення щодо реалізації конституційного права громадян на безпечні та нешкідливі умови праці, регулює відносини між роботодавцем та

працівником з питань безпеки, гігієни праці та виробничого середовища. Закон поширюється на всіх юридичних та фізичних осіб, які використовують найману працю.

Розробка веб-платформи ProjectPulse здійснюється у сфері інформаційних технологій, де основним видом діяльності є робота з персональними комп'ютерами та оргтехнікою. Трудова діяльність розробників програмного забезпечення регламентується, зокрема:

– Законом України «Про охорону праці» № 2694-XII від 14.10.1992;

– Кодексом законів про працю України;

– НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»;

– ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин»;

– ДБН В.2.2-28:2010 «Будинки адміністративного та побутового призначення».

Відповідно до ст. 15 Закону України «Про охорону праці», роботодавець зобов'язаний створити службу охорони праці на підприємстві [27]. Конкретна форма організації залежить від чисельності персоналу:

– на підприємствах з кількістю працівників 50 і більше осіб – служба охорони праці у вигляді окремого структурного підрозділу;

– від 20 до 49 осіб – призначається штатний спеціаліст з охорони праці;

– до 20 осіб – функції служби може виконувати сам роботодавець або уповноважена особа за сумісництвом, що пройшла відповідне навчання.

Для ІТ-компанії, що займається розробкою та підтримкою платформи ProjectPulse, типовою є організаційна структура малого або середнього підприємства (до 50 осіб), де призначається штатний спеціаліст з охорони праці або ці обов'язки покладаються на відповідального керівника.

Служба охорони праці виконує такі основні функції:

1. Профілактична – розробка заходів щодо попередження нещасних випадків, профзахворювань та аварій; аналіз умов праці на робочих місцях.

2. Контрольна – систематичний контроль стану умов і безпеки праці в підрозділах підприємства; перевірка відповідності обладнання, технологічних процесів та робочих місць вимогам нормативно-правових актів.

3. Інформаційна та навчальна – організація навчання та інструктажів з охорони праці для всіх категорій працівників; ведення необхідної документації.

4. Розслідувальна – участь у розслідуванні нещасних випадків на виробництві та розробці заходів щодо їх запобігання.

5. Координаційна – взаємодія з органами державного нагляду, профспілковими організаціями та іншими зацікавленими сторонами.

Згідно з НПАОП 0.00-4.12-05 «Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці», усі працівники проходять такі види інструктажів:

Вступний інструктаж проводиться спеціалістом з охорони праці (або керівником) з усіма особами, які приймаються на роботу, незалежно від освіти, стажу роботи та посади. Для розробників програмного забезпечення вступний інструктаж охоплює загальні відомості про підприємство, основні небезпечні та шкідливі виробничі фактори (тривала робота за монітором, сидяче положення, електронебезпека), порядок дій у надзвичайних ситуаціях.

Первинний інструктаж проводиться на робочому місці до початку роботи з новим працівником або з працівником, переведеним на інше робоче місце. Включає ознайомлення з конкретним робочим місцем, правилами безпечного використання комп'ютерної техніки, вимогами до організації режиму праці та відпочинку під час роботи з ВДТ (відеодисплейними терміналами).

Повторний інструктаж проводиться не рідше одного разу на рік для всіх працівників, які виконують роботи з підвищеним ризиком, та не рідше одного разу на два роки – для решти. Метою є підтвердження та поглиблення знань з охорони праці.

Позаплановий інструктаж проводиться у разі змін у нормативних документах, введення нового обладнання, виявлення порушень або після нещасного випадку.

Цільовий інструктаж проводиться перед виконанням разових робіт, не пов'язаних з основними обов'язками (наприклад, перестановка обладнання, монтаж нового серверного обладнання тощо). Спеціалісти з охорони праці та керівники підрозділів зобов'язані проходити навчання та перевірку знань не рідше одного разу на три роки в навчальних закладах відповідного профілю або на спеціальних курсах.

Відповідно до Порядку проведення атестації робочих місць за умовами праці (постанова Кабінету Міністрів України від 01.08.1992 № 442), роботодавець зобов'язаний проводити атестацію робочих місць не рідше одного разу на 5 років. Для ІТ-компаній атестація охоплює оцінку:

– рівня освітленості робочих місць (природної та штучної);

– параметрів мікроклімату (температура, відносна вологість, рухливість повітря);

– рівня шуму від комп'ютерного обладнання, систем вентиляції та кондиціювання;

– організації робочого місця (відповідність меблів, розташування монітора, клавіатури, маніпулятора).

За результатами атестації розробляються заходи щодо поліпшення умов праці та встановлюються компенсації для тих категорій працівників, умови праці яких за окремими показниками не відповідають нормативним вимогам.

Попередні та періодичні медичні огляди для розробників програмного забезпечення, що працюють з ВДТ більше 50% робочого часу, регулюються наказом МОЗ України № 246 від 21.05.2007. Медичний огляд проводиться при прийомі на роботу та надалі – один раз на два роки.

За порушення законодавства про охорону праці передбачено дисциплінарну, адміністративну, матеріальну та кримінальну відповідальність згідно з чинним законодавством України. Роботодавець сплачує штраф у розмірі, встановленому ст. 43 Закону «Про охорону праці», якщо за його вини створено загрозу загибелі або каліцтва людини чи стався нещасний випадок або профзахворювання.

4.3 Висновки до розділу 4

У четвертому розділі розглянуто питання охорони праці та питання з безпеки життєдіяльності.

У першому підрозділі розглянуто аварії з викидом радіоактивних речовин як особливо небезпечний вид техногенних надзвичайних ситуацій. Наведено класифікацію аварій за міжнародною шкалою INES, охарактеризовано основні джерела радіаційної небезпеки в Україні, описано вражаючі фактори (зовнішнє та внутрішнє опромінення, бета-опромінення шкіри) та їх вплив на організм людини. Викладено комплекс захисних заходів – сповіщення, укриття, йодна профілактика, евакуація, дезактивація – а також алгоритм дій персоналу при отриманні сигналу про аварію та норми дозиметричного контролю відповідно до НРБУ-97.

У другому підрозділі висвітлено організацію служби охорони праці на підприємстві: визначено правову базу (Закон України «Про охорону праці», галузеві НПАОП та санітарні норми), розкрито форми організації служби залежно від чисельності персоналу, описано всі п'ять видів інструктажів з охорони праці та порядок їх проведення, розглянуто вимоги до атестації робочих місць (один раз на 5 років) і обов'язкових медичних оглядів для осіб, що працюють з ВДТ понад 50% робочого часу.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено веб-платформу ProjectPulse для управління проєктами та аналізу продуктивності команди. Підведемо підсумки виконання поставлених задач.

Було проведено аналіз предметної галузі та існуючих рішень. Розглянуто методології управління проєктами (Agile, Scrum, Kanban) та проаналізовано чотири провідних продукти ринку: Jira, Trello, Asana, Notion. Встановлено, що існуючі рішення або надто складні та дорогі для малих команд (Jira), або позбавлені аналітики та підтримки спринтів (Trello). Це підтвердило актуальність та необхідність розробки ProjectPulse.

Було сформульовано вимоги до системи. Визначено 8 груп функціональних вимог (FR-1..FR-8) та 5 груп нефункціональних вимог (NFR-1..NFR-5), які охоплюють автентифікацію, управління проєктами, завданнями, спринтами, командою, аналітику, сповіщення та експорт.

Було обрано та обґрунтовано технологічний стек. Для клієнтської частини обрано Angular 14 + PrimeNG + Angular CDK + ApexCharts. Для серверної – Node.js + Express.js + Socket.io. Як базу даних – MongoDB + Mongoose. Для автентифікації – JWT + bcrypt. Вибір обґрунтовано відповідністю архітектурним вимогам та зрілістю екосистем.

Було спроєктовано архітектуру системи. Розроблено трирівневу клієнт-серверну архітектуру. Спроєктовано 6 MongoDB-колекцій з необхідними полями та зв'язками. Визначено REST API з 30+ маршрутами. Побудовано діаграму варіантів використання (9 основних UC), компонентну діаграму Angular-застосунку та діаграми послідовності для ключових сценаріїв.

Реалізовано основні функціональні модулі. Розроблено та протестовано: Модуль автентифікації (реєстрація, вхід, захищені маршрути, JWT-interceptor). Kanban-дошку з drag-and-drop між колонками та всередині колонок (Angular CDK). Модуль управління спринтами (life-cycle: planning → active → completed, переміщення задач у беклог). Аналітичну панель з donut-діаграмами, bar-графіками

та Gantt-представленням. Систему реальних WebSocket-сповіщень при призначенні та зміні статусу задачі. Функції експорту даних у форматах PDF (jsPDF + autoTable) та Excel (xlsx).

Забезпечено адаптивний дизайн для мобільних пристроїв. Реалізовано drawer-overlay для sidebar з анімацією (transform: translateX), hamburger-кнопку в шапці, автоматичне закриття drawer при навігації. Виправлено layout-проблеми для ширини 360–768px: переповнення вкладок, перекриття бейджів, непридатність task sidebar.

Проведено тестування та виправлено виявлені дефекти. Виконано ручне функціональне тестування за 5 групами тест-кейсів, тестування API через Postman, кросбраузерне та адаптивне тестування. Виявлено та виправлено 8 дефектів. Час відповіді API для всіх ключових ендпоінтів не перевищує 80 мс.

Перспективи подальшого розвитку платформи:

– інтеграція з GitHub / GitLab для автоматичного оновлення статусу задач при мерджі PR.

– реалізація time-tracking з таймером у реальному часі.

– дошка планування (Planning Poker) для оцінки story points.

– push-сповіщення (PWA Service Worker) для мобільних браузерів.

– мультимовний інтерфейс (i18n) з підтримкою англійської та української.

– CI/CD pipeline для автоматизованого розгортання.

– покриття коду unit-тестами (Jest для Node.js, Karma/Jasmine для Angular).

– Docker-контейнеризація для спрощення розгортання.

Таким чином, усі поставлені задачі виконано. Розроблений прототип веб-платформи ProjectPulse є функціональним, адаптивним та готовим до використання малими та середніми командами розробників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agile Manifesto. Manifesto for Agile Software Development [Електронний ресурс]. – 2001. – Режим доступу: https://agilemanifesto.org/

2. Schwaber K., Sutherland J. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game [Електронний ресурс]. – Scrum.org, 2020. – 13 р. – Режим доступу: https://scrumguides.org/

3. Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. – Blue Hole Press, 2010. – 278 p.

4. Angular Documentation [Електронний ресурс]. – Google, 2026. – Режим доступу: https://angular.io/docs

5. PrimeNG Documentation [Електронний ресурс]. – PrimeTek, 2026. – Режим доступу: https://primeng.org/

6. Node.js Documentation [Електронний ресурс]. – OpenJS Foundation, 2026. – Режим доступу: https://nodejs.org/en/docs

7. Express.js Documentation [Електронний ресурс]. – OpenJS Foundation, 2026. – Режим доступу: https://expressjs.com/

8. MongoDB Manual [Електронний ресурс]. – MongoDB Inc., 2026. – Режим доступу: https://www.mongodb.com/docs/manual/

9. Mongoose Documentation [Електронний ресурс]. – Mongoose, 2026. – Режим доступу: https://mongoosejs.com/docs/

10. Socket.io Documentation [Електронний ресурс]. – Socket.io, 2026. – Режим доступу: https://socket.io/docs/v4/

11. JSON Web Tokens. RFC 7519 [Електронний ресурс]. – IETF, 2015. – Режим доступу: https://tools.ietf.org/html/rfc7519

12. ApexCharts Documentation [Електронний ресурс]. – ApexCharts, 2026. – Режим доступу: https://apexcharts.com/docs/

13. Angular CDK Drag and Drop [Електронний ресурс]. – Google, 2026. – Режим доступу: https://material.angular.io/cdk/drag-drop/overview

14. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures [Текст]: дис. / Roy Thomas Fielding. – University of California, 2000. – 162 p.

15. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley Professional, 2002. – 560 p.

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

17. Highsmith J. Agile Project Management: Creating Innovative Products. – Addison-Wesley, 2009. – 352 p.

18. Richardson L., Ruby S. RESTful Web Services. – O'Reilly Media, 2007. – 454 p.

19. Chodorow K. MongoDB: The Definitive Guide. – O'Reilly Media, 2013. – 432 p.

20. jsPDF Documentation [Електронний ресурс]. – jsPDF, 2026. – Режим доступу: https://rawgit.com/MrRio/jsPDF/master/docs/

21. SheetJS (xlsx) Documentation [Електронний ресурс]. – SheetJS, 2026. – Режим доступу: https://docs.sheetjs.com/

22. bcrypt. Password Hashing Function [Електронний ресурс]. – npm, 2026. – Режим доступу: https://www.npmjs.com/package/bcryptjs

23. OWASP Top Ten [Електронний ресурс]. – OWASP Foundation, 2021. – Режим доступу: https://owasp.org/www-project-top-ten/

24. MDN Web Docs. CSS Flexbox [Електронний ресурс]. – Mozilla, 2026. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout

25. Responsive Web Design. W3C Recommendation [Електронний ресурс]. – W3C, 2012. – Режим доступу: https://www.w3.org/TR/css-mediaqueries-4/

26. TypeScript Documentation [Електронний ресурс]. – Microsoft, 2026. – Режим доступу: https://www.typescriptlang.org/docs/

27. Андрейчук Н.І. Охорона праці : навч. посіб. / Н.І. Андрейчук, Ю.В. Кіт, С.В. Шибанов, О.В. Шерстньова. Львів : Видавництво Львівська політехніка, 2021. 276 с.

28. Норми радіаційної безпеки України. НРБУ – 97 / Д-2000.

29. Атаманчук П.С. Безпека життєдіяльності: навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

ДОДАТКИ

ДОДАТОК А – Mongoose-схеми моделей бази даних

Лістинг А.1 - Модель User

```
const userSchema = new mongoose.Schema({
     name:     { type: String, required: true, trim: true },
     email:    { type: String, required: true, unique: true,
     lowercase: true },
     password: { type: String, required: true },
     // bcrypt hash
     avatar:   { type: String, default: '' },
}, { timestamps: true });
```

Лістинг А.2 - Модель Project

```
const projectSchema = new mongoose.Schema({
     name:        { type: String, required: true, trim: true },
     description: { type: String, default: '' },
     color:       { type: String, default: '#4F46E5' },
     status:   { type: String, enum: ['active','on-
     hold','completed','cancelled'],
     default: 'active' },
     priority: { type: String, enum:
     ['critical','high','medium','low'], default:
     'medium' },
     owner:    { type: mongoose.Schema.Types.ObjectId, ref: 'User',
     required: true },
     members:  [{ type: mongoose.Schema.Types.ObjectId, ref: 'User'
     }],
}, { timestamps: true });
```

Лістинг А.3 - Модель Task

```
const attachmentSchema = new mongoose.Schema({
     filename:     { type: String, required: true },
     originalName: { type: String, required: true },
     mimetype:     { type: String, required: true },
     size:         { type: Number, required: true },
     url:          { type: String, required: true },
     uploadedBy:   { type: mongoose.Schema.Types.ObjectId, ref:
     'User' },
}, { _id: false });
const timeEntrySchema = new mongoose.Schema({
     user:        { type: mongoose.Schema.Types.ObjectId, ref:
'User', required: true
},
     startTime:   { type: Date, required: true },
     endTime:     { type: Date },
     duration:    { type: Number, default: 0 },
     description: { type: String, default: '' },
}, { _id: false });
const taskSchema = new mongoose.Schema({
     title:       { type: String, required: true, trim: true },
```

```
    description: { type: String, default: '' },
    status:      { type: String, enum: ['todo','in-
    progress','review','done'],
    default: 'todo' },
    priority:    { type: String, enum:
    ['critical','high','medium','low'], default:
    'medium' },
    project:     { type: mongoose.Schema.Types.ObjectId, ref:
    'Project', required:
    true },
    parent:      { type: mongoose.Schema.Types.ObjectId, ref:
    'Task', default: null
    },
    assignees:   [{ type: mongoose.Schema.Types.ObjectId, ref:
    'User' }],
    createdBy:   { type: mongoose.Schema.Types.ObjectId, ref:
    'User' },
    dueDate:     { type: Date, default: null },
    startDate:   { type: Date, default: null },
    completedAt: { type: Date, default: null },
    estimatedHours: { type: Number, default: 0 },
    storyPoints:    { type: Number, default: 0 },
    order:   { type: Number, default: 0 },
    tags:    [String],
    attachments: [attachmentSchema],
    timeEntries: [timeEntrySchema],
}, { timestamps: true });
taskSchema.virtual('totalTrackedTime').get(function () {
    return this.timeEntries.reduce((sum, e) => sum + (e.duration ||
0), 0);
});
```

Лістинг А.4 - Модель Sprint

```
const sprintSchema = new mongoose.Schema({
    name:    { type: String, required: true, trim: true },
    goal:    { type: String, default: '' },
    project: { type: mongoose.Schema.Types.ObjectId, ref:
    'Project', required: true
    },
    tasks:   [{ type: mongoose.Schema.Types.ObjectId, ref: 'Task'
    }],
    status:  { type: String, enum:
    ['planning','active','completed'], default:
    'planning' },
    startDate:   { type: Date, default: null },
    endDate:     { type: Date, default: null },
    completedAt: { type: Date, default: null },
}, { timestamps: true });
```

Лістинг А.5 - Модель Notification

```
const notificationSchema = new mongoose.Schema({
     user:    { type: mongoose.Schema.Types.ObjectId, ref: 'User',
     required: true },
     type:    { type: String, enum:
     ['task_assigned','status_changed','sprint_started',
     'sprint_completed','member_added'], required:
     true },
     message: { type: String, required: true },
     task:    { type: mongoose.Schema.Types.ObjectId, ref: 'Task',
     default: null },
     project: { type: mongoose.Schema.Types.ObjectId, ref:
     'Project', default: null
},
     isRead:  { type: Boolean, default: false },
}, { timestamps: true });
```

Лістинг А.6 - Модель Activity

```
const activitySchema = new mongoose.Schema({
     project: { type: mongoose.Schema.Types.ObjectId, ref:
     'Project', required: true
     },
     user:    { type: mongoose.Schema.Types.ObjectId, ref: 'User',
     required: true },
     action:  { type: String, enum:
          ['created_task','updated_status','updated_task',
          'deleted_task','added_member','removed_member',
          'created_sprint','completed_sprint'], required:
     true },
     task:    { type: mongoose.Schema.Types.ObjectId, ref: 'Task',
     default: null },
}, { timestamps: true });
```

ДОДАТОК Б – Структура серверного проєкту та точка входу

Лістинг Б.1 - Структура директорій

```
ProjectPulse-backend/
├── src/
│   ├── config/
│   │   └── db.js
│   ├── middleware/
│   │   └── auth.middleware.js
│   ├── models/
│   │   ├── User.js
│   │   ├── Project.js
│   │   ├── Task.js
│   │   ├── Sprint.js
│   │   ├── Notification.js
│   │   └── Activity.js
│   ├── controllers/
│   │   ├── auth.controller.js
│   │   ├── project.controller.js
│   │   ├── task.controller.js
│   │   ├── sprint.controller.js
│   │   └── notification.controller.js
│   └── routes/
│       ├── auth.routes.js
│       ├── project.routes.js
│       ├── task.routes.js
│       ├── sprint.routes.js
│       └── notification.routes.js
├── server.js
└── package.json
```

Лістинг Б.2 - Точка входу server.js

```
const express  = require('express');
const http     = require('http');
const { Server } = require('socket.io');
const cors     = require('cors');
const connectDB = require('./src/config/db');
const app    = express();
const server = http.createServer(app);
const io     = new Server(server, {
     cors: { origin: process.env.CLIENT_URL ||
'http://localhost:4200',
     credentials: true }
});
app.use(cors({ origin: process.env.CLIENT_URL ||
'http://localhost:4200',
     credentials: true }));
app.use(express.json());
// Передача io до контролерів через request-об'єкт
app.use((req, _res, next) => { req.io = io; next(); });
// Маршрути
app.use('/api/auth', require('./src/routes/auth.routes'));
```

```
app.use('/api/projects', require('./src/routes/project.routes'));
app.use('/api/tasks',  require('./src/routes/task.routes'));
app.use('/api/sprints',    require('./src/routes/sprint.routes'));
app.use('/api/notifications',require('./src/routes/notification.rout
es'));
// Socket.io: користувачі приєднуються до персональної кімнати
io.on('connection', socket => {
     socket.on('join', userId => socket.join(`user_${userId}`));
});
connectDB().then(() => {
     const PORT = process.env.PORT || 3000;
     server.listen(PORT, () => console.log(`Server running on port
     ${PORT}`));
});
```

ДОДАТОК В – Ключові фрагменти клієнтської частини

Лістинг В.1 - HTML-шаблон Kanban-дошки

```
<div class="kanban-board" *ngIf="!loading">
  <div class="kanban-col" *ngFor="let col of columns">
    <div class="col-header">
      <div style="display:flex;align-items:center;gap:6px;">
        <span class="col-dot" [style.background]="col.color"></span>
        <span>{{ col.label }}</span>
      </div>
      <span class="col-count">{{ col.tasks.length }}</span>
    </div>

    <div
      class="col-body"
      cdkDropList
      [id]="col.id"
      [cdkDropListData]="col.tasks"
      [cdkDropListConnectedTo]="columnIds"
      (cdkDropListDropped)="drop($event, col)"
    >
      <div
        *ngFor="let task of col.tasks; trackBy: trackById"
        cdkDrag
        class="task-card"
        (click)="openDetail(task)"
      >
        <div class="task-title">{{ task.title }}</div>
        <div class="task-meta">
          <app-priority-badge [priority]="task.priority"></app-
priority-badge>
          <span *ngIf="task.dueDate" class="due-date"
[class.overdue]="task.dueDate < today && task.status !== 'done'">
            <i class="pi pi-calendar" style="font-size:10px;"></i>
            {{ task.dueDate | date:'dd MMM' }}
          </span>
          <span *ngIf="task.storyPoints" class="story-points">{{
task.storyPoints }}pt</span>
        </div>
        <div class="task-footer" *ngIf="task.assignees.length">
          <div class="assignee-stack">
            <app-avatar *ngFor="let a of task.assignees.slice(0,3)"
[user]="a" [size]="22"></app-avatar>
          </div>
          <span *ngIf="task.totalTrackedTime" class="time-badge">
            <i class="pi pi-clock" style="font-size:10px;"></i>
            {{ task.totalTrackedTime | duration }}
          </span>
        </div>
      </div>
```

```
      <app-empty-state *ngIf="col.tasks.length === 0" icon="pi-
inbox" title="" message="Drop tasks here" style="padding:20px
0;font-size:12px;"></app-empty-state>
    </div>
  </div>
</div>
```

Лістинг В.2 - Функції експорту (project-detail.component.ts)

```
exportPdf(projectId: string): void {
    const token = localStorage.getItem('pp_token');
    const url = `${this.api}/projects/${projectId}/export/pdf`;
    fetch(url, { headers: { Authorization: `Bearer ${token}` } })
      .then(r => r.blob())
      .then(blob => {
        const link = document.createElement('a');
        link.href = URL.createObjectURL(blob);
        link.download = `project_report.pdf`;
        link.click();
      });
 }

 exportExcel(projectId: string): void {
    const token = localStorage.getItem('pp_token');
    const url = `${this.api}/projects/${projectId}/export/excel`;
    fetch(url, { headers: { Authorization: `Bearer ${token}` } })
      .then(r => r.blob())
      .then(blob => {
        const link = document.createElement('a');
        link.href = URL.createObjectURL(blob);
        link.download = `project_report.xlsx`;
        link.click();
      });
 }
```

Лістинг В.3 - SocketService (socket.service.ts)

```
@Injectable({ providedIn: 'root' })
export class SocketService implements OnDestroy {
  private socket: Socket | null = null;

  private _taskCreated$ = new Subject<any>();
  private _taskUpdated$ = new Subject<any>();
  private _taskDeleted$ = new Subject<any>();
  private _commentAdded$ = new Subject<any>();
  private _notification$ = new Subject<any>();
  private _projectUpdated$ = new Subject<any>();

  readonly taskCreated$ = this._taskCreated$.asObservable();
  readonly taskUpdated$ = this._taskUpdated$.asObservable();
  readonly taskDeleted$ = this._taskDeleted$.asObservable();
  readonly commentAdded$ = this._commentAdded$.asObservable();
```

```
  readonly notification$ = this._notification$.asObservable();
  readonly projectUpdated$ = this._projectUpdated$.asObservable();

  constructor(private auth: AuthService) {}

  connect(): void {
    if (this.socket?.connected) return;

    const token = this.auth.getToken();
    if (!token) return;

    this.socket = io(environment.socketUrl, {
      auth: { token },
      transports: ['websocket'],
    });

    this.socket.on('task:created',  (data: any) =>
this._taskCreated$.next(data));
    this.socket.on('task:updated',  (data: any) =>
this._taskUpdated$.next(data));
    this.socket.on('task:deleted',  (data: any) =>
this._taskDeleted$.next(data));
    this.socket.on('comment:added', (data: any) =>
this._commentAdded$.next(data));
    this.socket.on('notification:new', (data: any) =>
this._notification$.next(data));
    this.socket.on('project:updated', (data: any) =>
this._projectUpdated$.next(data));
  }

  joinProject(projectId: string): void {
    this.socket?.emit('join:project', projectId);
  }

  leaveProject(projectId: string): void {
    this.socket?.emit('leave:project', projectId);
  }

  disconnect(): void {
    this.socket?.disconnect();
    this.socket = null;
  }

  ngOnDestroy(): void {
    this.disconnect();
  }
}
```

ДОДАТОК Д – Інструкція з розгортання

Г.1 Вимоги до середовища

Таблиця Г.1 – Вимоги до середовища

Компонент	**Мінімальна версія**
Node.js	18.x LTS
Npm	9.x
MongoDB	6.x
Angular CLI	14.x

Лістинг Г.2 - Запуск у режимі розробки

```
# 1. Налаштувати серверну частину
cd ProjectPulse-backend
npm install
cp .env.example .env   # заповнити MONGODB_URI та JWT_SECRET
npm run dev   # запуск на порті 3000 (nodemon)

# 2. Налаштувати клієнтську частину
cd ../ProjectPulse-frontend
npm install
ng serve         # запуск на порті 4200
```

Лістинг Г.3 - Змінні середовища (.env)

```
MONGODB_URI=mongodb://localhost:27017/projectpulse
JWT_SECRET=replace_with_strong_random_secret
PORT=3000
CLIENT_URL=http://localhost:4200
```

Г.4 Перевірка роботи

Після запуску обох частин:

1. Відкрити браузер за адресою http://localhost:4200

2. Зареєструвати нового користувача

3. Створити тестовий проєкт та кілька задач

4. Переконатися, що Kanban-дошка відображає задачі та drag-and-drop працює коректно