

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ
УПРАВЛІННЯ ЗАДАЧАМИ З РОЗПОДІЛЕНОЮ
АРХІТЕКТУРОЮ
DESIGN AND IMPLEMENTATION OF A WEB-BASED
TASK MANAGEMENT SYSTEM WITH DISTRIBUTED
ARCHITECTURE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-122

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ШАПОВАЛ Д.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ДЬЯЧУК Т.С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н., проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШАПОВАЛА Данила Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація вебсистеми управління задачами з розподіленою архітектурою.

Design and Implementation of a Web-based Task Management System with Distributed Architecture

керівник проєкту (роботи) к.т.н., ГОФМАН Євгеній Олександрович ,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є. О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 20 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсового проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5 тиждень	Розділ 3
5	Тестування та експериментальне дослідження програми.	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування.	7 тиждень	Додатки
7	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Данило ШАПОВАЛ

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Євгеній ГОФМАН

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра:
131 с., 10 табл., 41 рис., 3 дод., 14 джерел.

ПРОЄКТНИЙ МЕНЕДЖМЕНТ, ВЕБЗАСТОСУНОК, КОМАНДНА СПІВПРАЦЯ, КАНБАН-ДОШКА, СПРИНТИ, РЕАЛЬНИЙ ЧАС, ВЕБСОКЕТИ.

Об'єкт дослідження – процес автоматизації управління проєктами та командної взаємодії в межах розробки програмного забезпечення.

Предмет дослідження – методи та інструменти побудови клієнт-серверних систем для синхронізації проєктних даних у реальному часі.

Мета роботи – підвищення ефективності управління робочими процесами в командах за рахунок створення централізованої платформи з миттєвою синхронізацією даних та гнучким налаштуванням прав доступу.

Матеріали, методи та технічні засоби: використано методи об'єктно-орієнтованого програмування. Серверну частину (Backend) реалізовано на фреймворку NestJS із використанням TypeScript, Prisma ORM та бази даних PostgreSQL. Клієнтську частину (Frontend) побудовано на базі Next.js (React) з використанням Tailwind CSS та компонентів Shadcn UI. Для забезпечення миттєвої синхронізації стану дошки між користувачами використано технологію WebSockets (Socket.io). Документування API виконано за допомогою Swagger (OpenAPI). Розроблення проводилося в середовищі Visual Studio Code на персональному комп'ютері з операційною системою macOS.

Результати. Розроблено повнофункціональний вебзастосунок «Planner», який забезпечує створення проєктів, управління беклогом та спринтами, а також відстеження завдань різних типів (Task, Bug, Story, Epic). Реалізовано інтерактивну канбан-дошку з підтримкою drag-and-drop, систему вкладених коментарів, миттєві сповіщення (внутрішні та електронною

поштою) та систему ролей (Admin, Member, Viewer). Впроваджено механізм пошуку завдань у реальному часі та інтегрований центр підтримки користувачів.

Висновки. Досягнуто мети роботи: створено інструмент, який автоматизує робочі процеси команди, забезпечує цілісність даних при одночасній роботі багатьох користувачів та надає прозору візуалізацію прогресу проєкту. Застосунок підтримує адаптивний дизайн, темну та світлу теми, а також безпечну автентифікацію за допомогою JWT та HTTP-only cookies.

Галузь використання – управління проєктами в IT-командах, розробка програмного забезпечення, а також координація спільних завдань у малих та середніх робочих групах.

Економічна ефективність. Використання застосунку скорочує витрати часу на синхронізаційні наради завдяки візуалізації завдань у реальному часі, мінімізує ризики втрати інформації та підвищує загальну продуктивність команди за рахунок чіткого розподілу відповідальності та термінів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 131 pages, 10 tables, 41 figures, 3 appendixes, 14 sources.

PROJECT MANAGEMENT, WEB APPLICATION, TEAM COLLABORATION, KANBAN BOARD, SPRINTS, REAL-TIME, WEBSOCKETS.

Research object – the process of automating project management and team interaction within software development workflows.

Research subject – methods and tools for developing the “Planner” web application for task tracking and sprint planning.

The purpose of the work – to increase the efficiency of workflow management in teams by creating a centralized platform with instant data synchronization and flexible access rights configuration.

Materials, methods, and technical means: object-oriented programming methods were used. The backend is implemented on the NestJS framework using TypeScript, Prisma ORM, and a PostgreSQL database. The frontend is built using Next.js (React), Tailwind CSS, and Shadcn UI components. To ensure instant board state synchronization between users, WebSockets (Socket.io) technology was employed. API documentation was performed using Swagger (OpenAPI). Development was carried out in the Visual Studio Code environment on a personal computer running macOS.

Results. A full-featured web application “Planner” was developed, providing project creation, backlog and sprint management, and tracking of various issue types (Task, Bug, Story, Epic). An interactive Kanban board with drag-and-drop support, a nested comment system, instant notifications (both internal and via email), and a role-based system (Admin, Member, Viewer) were implemented. A real-time task search mechanism and an integrated user help center were introduced.

Conclusions. The work's objective was achieved: a tool was created that automates team workflows, ensures data integrity during simultaneous multi-user operation, and provides transparent visualization of project progress. The application supports responsive design, dark and light themes, and secure authentication using JWT and HTTP-only cookies.

Field of application – project management in IT teams, software development, and coordination of collaborative tasks in small and medium-sized working groups.

Economic efficiency. The application reduces time spent on synchronization meetings due to real-time task visualization, minimizes the risks of information loss, and increases overall team productivity through clear distribution of responsibilities and deadlines.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	11
Вступ	12
1 Аналіз предметної області	15
1.1 Короткий опис досліджуваної предметної області.....	15
1.2 Огляд існуючих рішень проблеми.....	17
1.2.1 Застосунок Trello	17
1.2.2 Застосунок Jira	18
1.2.3 Застосунок Linear.....	19
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів.....	20
1.3 Опис основних вимог до програми	21
1.3.1 Визначення функціональних вимог	21
1.3.2 Нефункціональні вимоги.....	23
1.4 Постановка завдання роботи.....	24
1.5 Висновки за розділом 1	25
2 Розробка архітектури програми	26
2.1 Структура системи	26
2.1.1 Основні компоненти системи	26
2.1.2 Взаємодія компонентів системи	27
2.2 Функціонування системи	28
2.2.1 Основні процеси системи.....	28
2.2.3 Логіка роботи системи	30
2.3 Вибір мови програмування та інструментарію для розробки програми	30
2.3.1 Мова програмування	30
2.3.2 Середовище розробки.....	32
2.3.3 База даних	34
2.4 Вимоги до технічних і програмних засобів	36

2.4.1 Необхідні технічні засобів для розробки та експлуатації	36
2.4.2 Вимоги до апаратного та програмного забезпечення.....	37
2.5 Висновки за розділом 2	37
3 Розробка програми	39
3.1 Архітектура системи	39
3.1.1 Структурна схема системи.....	41
3.2 Функціонування системи	42
3.2.1 Функціональна схема	42
3.2.2 Опис процесу обміну даними у системі	42
3.3 Проєктування бази даних.....	44
3.3.1 Структура бази даних.....	44
3.3.2 Взаємозв'язки між таблицями	44
3.4 Висновки за розділом 3	45
4 Експлуатація, тестування та експериментальне дослідження програми	48
4.1 Опис застосування програми	48
4.1.1 Огляд планової цільової аудиторії.....	48
4.1.2 Візуалізація планованої цільової аудиторії.....	50
4.2 Інструкція по роботі з програмою	51
4.2.1 Встановлення програми.....	51
4.2.2 Запуск та початкове налаштування програми	53
4.2.3 Методика роботи з програмою	55
4.3 Приклад роботи користувача з програмою	56
4.4 Тестування роботи програми	59
4.4.1 Функціональне тестування.....	59
4.4.2 Аналіз функціонального тестування	61
4.4.3 Тестування продуктивності	62
4.4.4 Аналіз продуктивності	63
4.4.5 Тестування безпеки	65
4.4.6 Аналіз безпеки	66
4.4.7 Аналіз користувацького досвіду	67

	10
4.5 Рекомендації щодо використання.....	69
4.6 Плани з розвитку	70
Висновки.....	72
Перелік джерел посилання	74
Додаток А Технічне завдання.....	76
Додаток Б Текст програми.....	85
Додаток В Слайди презентації	126

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CORS	– Cross-Origin Resource Sharing;
CSS	– Cascading Style Sheets;
DOM	– Document Object Model;
HTML	– HyperText Markup Language;
HTTP	– HyperText Transfer Protocol;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Token;
ORM	– Object-Relational Mapping;
REST	– Representational State Transfer;
SPA	– Single Page Application;
SQL	– Structured Query Language;
TS	– TypeScript;
UI	– User Interface;
UX	– User Experience;
WS	– WebSocket;
JWT	– JSON Web Token;
ЗК (FK)	– зовнішній ключ (Foreign Key);
ПК (PK)	– первинний ключ (Primary Key).

ВСТУП

Стрімка еволюція цифрових технологій диктує нові вимоги до засобів командної взаємодії, де ключовим фактором успіху стає швидкість адаптації до змін. У сучасному циклі розробки програмного забезпечення особливої ваги набуває прозорість робочих процесів: від оперативного відстеження життєвого циклу завдань до чіткого планування ітерацій. Попри велику кількість доступних інструментів, чимало команд досі відчують брак гнучких рішень, які б забезпечували миттєве відображення правок без затримок та перевантаження інтерфейсу зайвими функціями. Актуальність розробки зумовлена необхідністю створення середовища, де візуалізація прогресу та обмін даними відбуваються безперервно та синхронно для всіх учасників.

Глобальні тенденції в ІТ-секторі підтверджують, що ефективна комунікація є фундаментом продуктивності. Аналітичні звіти, зокрема State of Agile, наголошують: розсинхронізація статусів та відсутність єдиного джерела істини в проєкті часто стають головними чинниками порушення термінів [1]. Хоча автоматизація управління проєктами не є новою концепцією, ринок потребує інструментів, що поєднують високу швидкість обміну інформацією із простотою впровадження. Існуючі громіздкі системи часто виявляються занадто складними для швидкого старту малих і середніх груп, створюючи додатковий адміністративний тиск.

Проблема дослідження полягає в необхідності створення легкого та швидкодіючого інструменту, який забезпечує зручне управління беклогом, миттєву синхронізацію станів між учасниками проєкту за допомогою вебсокетів та чітку рольову модель доступу. Огляд ринку показує, що хоча подібні задачі вирішуються в таких системах, як Jira, Trello або ClickUp, вони часто мають надлишкову складність для невеликих команд або потребують постійного оновлення сторінок для відображення змін [2]. Провідні дослідження в галузі веброзробки (зокрема, документація фреймворків Next.js

та NestJS) вказують на перспективність використання архітектури на основі подій (event-driven) та типізованих мов програмування, як-от TypeScript, для створення надійних систем командної співпраці.

Об'єктом дослідження виступають процеси автоматизації командної роботи та координації завдань у межах гнучких методологій розробки.

Предметом дослідження є методи та інструменти побудови клієнт-серверних систем для синхронізації проєктних даних у реальному часі.

Мета роботи полягає у створенні цілісної вебecosистеми для управління проєктами, яка забезпечує безперебійну групову роботу через інтерактивні канбан-дошки, гнучке управління спринтами та автоматизоване сповіщення учасників.

Для досягнення цієї мети було окреслено та виконано такі етапи дослідження:

- системний аналіз методологій Scrum та Kanban для визначення оптимального набору функцій майбутнього застосунку;
- проєктування архітектури системи на базі вебсокетів для реалізації миттєвого оновлення стану робочого простору;
- розробка та впровадження модулів управління беклогом, коментування завдань та ієрархічної моделі доступу користувачів;
- проведення комплексного тестування системи на предмет швидкодії та коректності синхронізації даних при одночасній роботі групи користувачів.

Методологічну базу дослідження складають принципи об'єктно-орієнтованого проєктування та сучасні стандарти веброзробки. Технічна реалізація базується на використанні Next.js для фронтенд-частини та NestJS для серверної логіки. Роботу з даними організовано за допомогою Prisma ORM та бази PostgreSQL, а динамічний обмін повідомленнями забезпечено через бібліотеку Socket.io.

Розроблений інструментарій має високу практичну цінність для команд, що прагнуть оптимізувати свої внутрішні комунікації та мінімізувати

час на ручне оновлення статусів. Результати впровадження свідчать про значне підвищення прозорості робочих процесів та пришвидшення циклу передачі завдань між етапами розробки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Короткий опис досліджуваної предметної області

Організація командної розробки – це комплексна діяльність, яка базується на структуруванні завдань, моніторингу прогресу та координації зусиль усіх учасників процесу. Чимало ІТ-команд досі намагаються керувати роботою через розрізнені месенджери або статичні таблиці, що неминуче призводить до втрати контексту, плутанини в пріоритетах та порушення термінів. Це зумовлює гостру потребу у впровадженні професійних інструментів, здатних автоматизувати життєвий цикл розробки програмного забезпечення.

Ключові поняття, що формують фундамент сучасного управління проєктами:

- тикет (issue) – атомарна одиниця роботи, що містить опис завдання, призначеного виконавця та статус виконання;
- спринт – чітко обмежений у часі ітераційний цикл, протягом якого команда має реалізувати запланований обсяг функціоналу;
- робоча дошка (grid/board) – інструмент візуалізації потоку задач, що дозволяє в реальному часі відстежувати рух тикетів між етапами розробки;
- беклог – пріоритезований перелік усіх вимог, ідей та помилок, які очікують на свою реалізацію в майбутніх циклах.

На сьогодні галузь управління розробкою стикається з кількома критичними викликами. По-перше, це висока динаміка змін, через яку менеджерам складно підтримувати актуальність планів при інтенсивному ритмі роботи. По-друге, існує проблема миттєвої синхронізації дій між різними відділами, особливо в умовах віддаленої співпраці. По-третє, безпека та розмежування прав доступу: технічна документація та вихідний код є конфіденційними активами, захист яких є пріоритетом. Зрештою, більшість систем лише фіксують факти, але не мають інтелектуальних надбудов, які б допомагали прогнозувати ризики або автоматично розподіляти навантаження.

Серед провідних трендів у цій сфері варто відзначити глибоку інтеграцію штучного інтелекту для автоматизації рутинних описів та оцінки складності завдань, перехід до архітектур з підтримкою оновлень у реальному часі (WebSockets) та розвиток кросплатформених екосистем. Динаміку зростання аудиторії провідних платформ, таких як Jira та Trello, за останній звітний період відображено на рисунках 1.1 та 1.2.

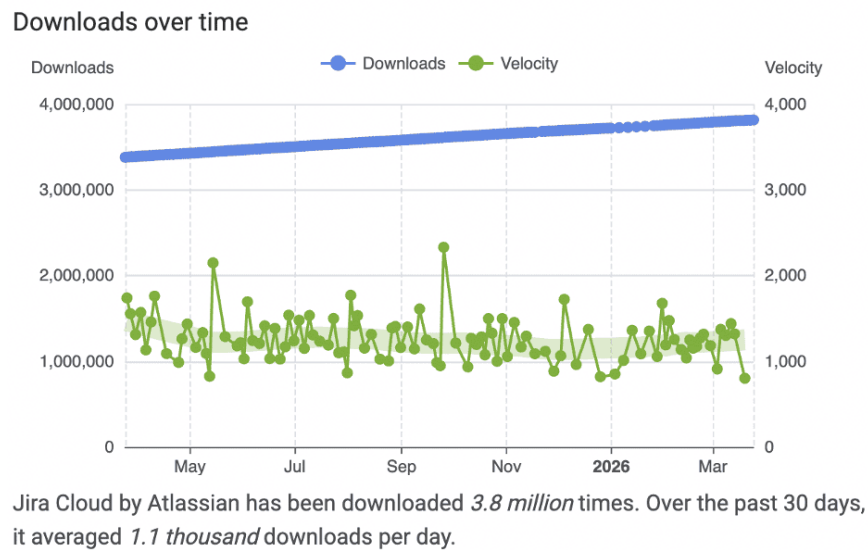


Рисунок 1.1 – Динаміка зростання аудиторії платформи Jira за останній звітний період [3]

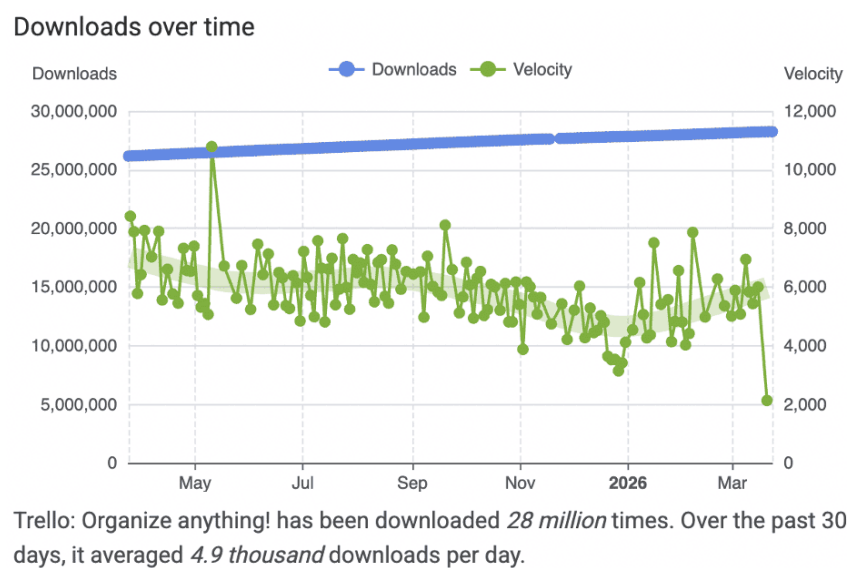


Рисунок 1.2 – Динаміка зростання аудиторії платформи Trello за останній звітний період [4]

1.2 Огляд існуючих рішень проблеми

Проаналізуємо функціональні можливості трьох відомих систем для координації роботи: Trello, Jira та Linear. Кожна з цих платформ пропонує унікальний набір інструментів, проте жодна з них не може в повному обсязі задовольнити специфічні вимоги щодо гнучкої та прозорої організації внутрішніх процесів у невеликих командах розробки.

1.2.1 Застосунок Trello

Trello – популярний кросплатформний сервіс на базі канбан-дошок для візуалізації робочих процесів. Забезпечує хмарну синхронізацію, підтримує дедлайни, чек-лісти та інтеграції (Power-Ups) [5].

Переваги:

- інтуїтивний інтерфейс та легкий старт;
- надійна синхронізація між пристроями;
- гнучка кастомізація через плагіни.

Недоліки:

- слабке налаштування ролей для великих команд;
- платний доступ до аналітики та автоматизації;
- відсутність повноцінного офлайн-режиму;
- брак вбудованих інструментів для технічного планування.

Зовнішній вигляд головної робочої дошки сервісу Trello зображено на рисунку 1.3.

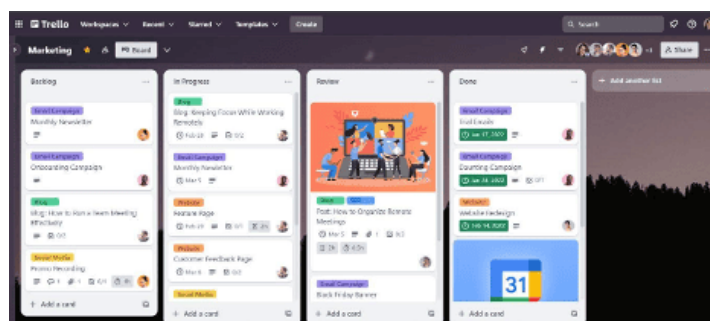


Рисунок 1.3 – Головна робоча дошка сервісу Trello [5]

1.2.2 Застосунок Jira

Jira від компанії Atlassian – професійна платформа для управління софтверними проєктами. Дозволяє налаштовувати складні робочі процеси (workflows), планувати ітерації, будувати дорожні карти (Roadmaps) та інтегруватися з репозиторіями коду (GitHub, Bitbucket) [6].

Переваги:

- глибоке налаштування процесів під будь-яку команду;
- потужна аналітика та детальні звіти (напр. Burndown charts);
- безшовна інтеграція з інструментами розробки та розгортання.

Недоліки:

- складний інтерфейс і високий поріг входження;
- висока вартість ліцензій;
- можливі затримки в роботі при великих обсягах даних;
- відсутність офлайн-режиму.

Зовнішній вигляд головної робочої дошки сервісу Jira зображено на рисунку 1.4.

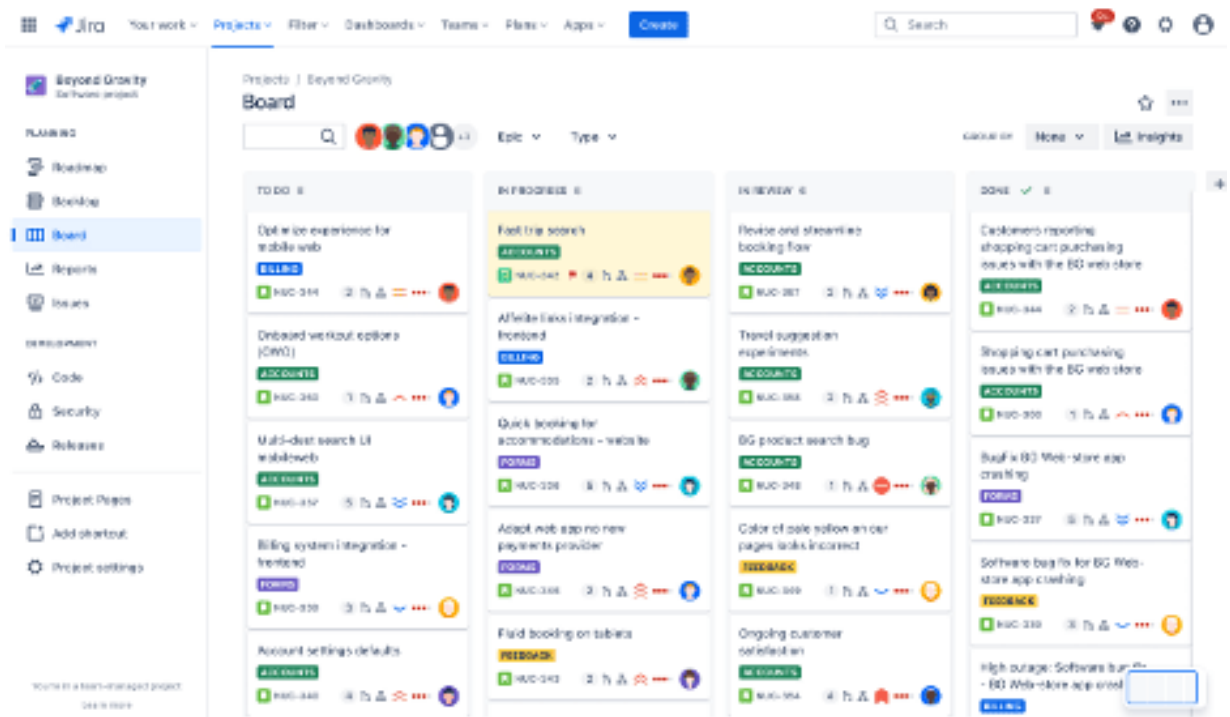


Рисунок 1.4 – Головна робоча дошка сервісу Jira [6]

1.2.3 Застосунок Linear

Linear – мінімалістична платформа для розробників, орієнтована на швидкість та продуктивність. Пропонує зручне керування беклогом, циклами (спринтами) та автоматичну синхронізацію зі статусами в Git [7].

Переваги:

- висока швидкодія та підтримка гарячих клавіш;
- естетичний дизайн, що не відволікає від роботи;
- глибока автоматизація (наприклад, за допомогою GitHub/GitLab).

Недоліки:

- обмежена кастомізація через жорстку внутрішню методологію;
- слабкий функціонал мобільної версії порівняно з десктопом;
- висока вартість для великих команд;
- відсутність складних вбудованих звітів та аналітики.

Зовнішній вигляд головної робочої дошки сервісу Linear зображено на рисунку 1.5.

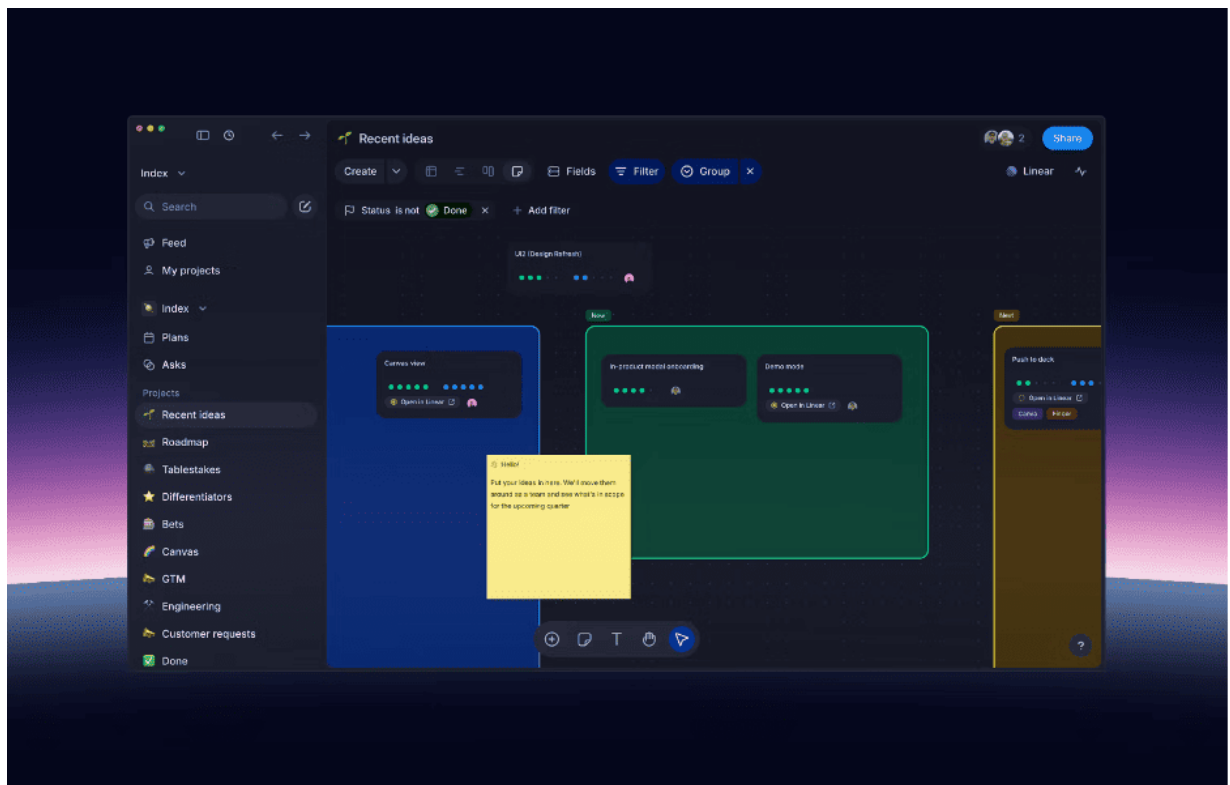


Рисунок 1.5 – Головна робоча дошка сервісу Linear [7]

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Для визначення технічних переваг та функціональних особливостей Planner, проведемо зіставлення цього сервісу з провідними платформами – Trello, Jira та Linear – за такими параметрами: повнота інструментарію, ергономічність інтерфейсу, надійність захисту даних та рівень підтримки командної взаємодії. У таблиці 1.1 відображено результати порівняльного аналізу існуючих рішень та системи, що створюється в межах даної роботи.

Таблиця 1.1 – Функціональні характеристики систем управління проєктами

Критерій	Trello	Jira	Linear	Planner
1	2	3	4	5
Функціональність	Канбан-дошки, чек-лісти, розширення	Гнучкі воркфлоу, звіти, спринти, дорожні карти	Робочі цикли, автоматизація, статусів, беклог	Управління тікетами, спринтами, беклогом
Командна взаємодія	Базове призначення виконавців	Складна ієрархія ролей та прав доступу	Командні простори, спільні проєкти	Чітке розмежування ролей (Admin, Member, Viewer)
Синхронізація	Хмарна, із затримками оновлення	Хмарна, високе навантаження на систему	Оптимізована хмарна синхронізація	Миттєве оновлення через WebSockets

Продовження таблиці 1.1

1	2	3	4	5
Офлайн– режим	Відсутній (лише перегляд у кеші)	Практично відсутній	Підтримується (локальне сховище)	Базова підтримка станів
Безпека	Стандартна хмарна безпека	Корпоративний рівень (SSO, Audit logs)	Високий рівень шифрування	Авторизація JWT, захист API- ендпоінтів
Зручність (UI/UX)	Візуально простий, інтуїтивний	Складний інтерфейс, довге навчання	Мінімалістичний, дуже швидкий	Сучасний дизайн, фокус на швидкодії

1.3 Опис основних вимог до програми

1.3.1 Визначення функціональних вимог

Функціональні вимоги визначають конкретні дії, які користувач може виконувати в системі, та її поведінку у відповідь на запити. Для вебзастосунку «Planner» встановлено такі вимоги:

а) керування доступом та автентифікація:

- 1) реєстрація нових користувачів та вхід до системи за допомогою email та пароля;
- 2) використання безпечних токенів (JWT) для підтримки сесій;
- 3) розмежування прав доступу на основі ролей (адміністратор проєкту, учасник команди);
- 4) можливість редагування профілю та налаштування персональних даних;

б) організація проєктної діяльності:

- 1) створення нових проєктів з визначенням їх назви та опису;

2) керування складом команди: запрошення користувачів до проєкту за їх електронною поштою;

3) призначення ролей учасникам всередині конкретного проєкту;

4) візуалізація списку проєктів, до яких користувач має доступ;

в) робота із завданнями та спринтами:

1) створення, модифікація та видалення карток завдань (issues);

2) планування ітерацій (спринтів) із зазначенням термінів виконання та статусів;

3) деталізація завдань: додавання описів, пріоритетів та призначення відповідальних виконавців;

4) система коментування завдань для обговорення робочих процесів у реальному часі;

г) комунікація та оновлення в реальному часі:

1) миттєва синхронізація змін в інтерфейсі через WebSockets (Socket.io) без оновлення сторінки;

2) система внутрішніх сповіщень про важливі події (призначення завдання, зміна статусу);

3) інтеграція з поштовим сервісом для надсилання критичних оновлень на email користувача;

4) модуль підтримки для зв'язку з адміністрацією системи;

д) візуалізація та структурування даних:

1) перегляд завдань у формі інтерактивних таблиць та гнучких сіток (grids);

2) фільтрація та сортування завдань за різними критеріями (статус, виконавець, пріоритет);

3) можливість перемикання між активними та завершеними спринтами.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні атрибути системи, такі як надійність, продуктивність та безпека:

а) безпека та захист інформації:

1) надійне шифрування паролів перед збереженням у базу даних за допомогою bcrypt;

2) валідація всіх вхідних даних на серверному рівні (DTO) для запобігання SQL-ін'єкціям та XSS-атакам;

3) ізоляція даних: користувачі можуть бачити лише ті проєкти, до яких вони офіційно долучені;

б) вимоги до продуктивності:

1) час відгуку API на стандартні запити не повинен перевищувати 300 мс;

2) оптимізація фронтенду через серверний рендеринг (Next.js SSR) для швидкого початкового завантаження сторінок;

3) мінімізація навантаження на мережу завдяки кешуванню запитів на стороні клієнта (React Query);

в) користувацький інтерфейс та зручність (UI/UX):

1) адаптивна верстка, що забезпечує коректне відображення на мобільних пристроях та десктопах;

2) наявність темної та світлої тем оформлення з можливістю автоматичного перемикавання;

3) використання сучасних UI-компонентів (Shadcn/UI, Tailwind CSS) для створення естетичного та зрозумілого інтерфейсу;

г) надійність та сумісність:

1) забезпечення цілісності даних при міграціях бази даних через Prisma ORM;

2) контейнеризація за допомогою Docker для швидкого розгортання та стабільної роботи в різних середовищах. резервне копіювання даних;

3) підтримка актуальних версій популярних браузерів (Chrome, Safari, Firefox, Edge);

д) локалізація та підтримка:

1) інтерфейс має бути повністю адаптований для англійської мови;

2) наявність вбудованої форми зворотного зв'язку для повідомлень про помилки.

1.4 Постановка завдання роботи

Метою даної роботи є розроблення кросплатформеної вебсистеми «Planner» для ефективного управління командними проєктами та завданнями за методологією Agile.

Основні завдання розробки:

- проєктування архітектури. Вивчити створення модульної структури застосунку на базі NestJS (backend) та Next.js (frontend) з використанням PostgreSQL як основного сховища даних;

- реалізація серверної частини. Розробка RESTful API для керування користувачами, проєктами, спринтами та завданнями; налаштування взаємодії з БД через Prisma;

- впровадження Real-time функціоналу. інтеграція Socket.io для забезпечення миттєвого обміну даними між учасниками команди;

- розробка клієнтського інтерфейсу. Створення динамічного фронтенду з високим рівнем інтерактивності, системою модальних вікон та зручною навігацією;

- забезпечення безпеки. Впровадження механізмів авторизації на основі JWT та захисту приватних даних;

- тестування та валідація. Перевірка коректності роботи всіх модулів системи, оптимізація швидкодії та розгортання тестового середовища через Docker Compose.

Цілі розробки:

- створити надійний інструмент для координації роботи IT-команд;
- забезпечити високу швидкість обміну інформацією через WebSocket;
- гарантувати конфіденційність та розмежування прав доступу учасників проєкту;
- надати користувачам зручний інтерфейс для планування робочих ітерацій.

1.5 Висновки за розділом 1

Дослідження сучасних методологій командної розробки підтверджує, що ефективна організація робочих процесів залишається критичним викликом через надмірну складність або функціональні прогалини наявних інструментів. Популярні платформи, такі як Jira, Trello чи Linear, пропонують широкі можливості для відстеження завдань, проте вони часто перевантажені зайвими елементами або не забезпечують достатньої оперативності у взаємодії учасників. Головними перешкодами для продуктивної співпраці є затримки в оновленні даних, складність налаштування рівнів доступу та громіздкість інтерфейсів. Вебсистема «Planner» покликана усунути ці недоліки завдяки використанню технології WebSockets для миттєвої синхронізації, впровадженню гнучкої системи ролей, забезпеченню надійного захисту сесій через JWT та створенню максимально інтуїтивного середовища для управління спринтами.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Структура системи

2.1.1 Основні компоненти системи

Архітектурна побудова системи «Planner» базується на принципі розділення відповідальності між клієнтською та серверною частинами, що забезпечує масштабованість та зручність підтримки коду. Проєкт реалізовано за багатошаровою моделлю, де кожен рівень виконує чітко визначені функції [8].

Клієнтський рівень (Frontend – Next.js):

- екрани (Screens) Інтерфейсні компоненти (Components & UI): набір візуальних елементів (на основі Shadcn/UI та Tailwind CSS), що формують зовнішній вигляд системи та взаємодіють з користувачем;
- рівень маршрутизації (App Router): структура сторінок застосунку, яка відповідає за навігацію та відображення контенту залежно від URL;
- управління станом та даними (Hooks & Context): використання React Context API для автентифікації та React Query для синхронізації даних із сервером і кешування.

Серверний рівень (Backend – NestJS):

- контролери (Controllers): відповідають за обробку вхідних HTTP-запитів, валідацію вхідних даних через DTO та повернення відповідей клієнту;
- провайдери бізнес-логіки (Services): містять основні алгоритми роботи системи, взаємодіють з базою даних та іншими модулями;
- Модульна структура (Modules): логічне групування функціонала (наприклад, AuthModule, IssuesModule, SprintsModule) для забезпечення високої зв'язності та низької залежності компонентів.

Рівень роботи з даними (Persistence Layer):

- ORM-шар (Prisma): забезпечує типізовану взаємодію з базою даних, мапінг моделей на таблиці та виконання запитів;

- база даних (PostgreSQL): реляційне сховище для надійного зберігання інформації про користувачів, проєкти та завдання.

Спеціалізовані сервіси (Core Services):

- сервіс автентифікації (AuthService): реалізує механізми перевірки прав доступу та генерації токенів JWT.
- сервіс бази даних (DbService);
- сервіс реального часу (Socket.io Gateway): забезпечує миттєву передачу повідомлень та оновлення статусів завдань без перезавантаження сторінок;
- сервіс взаємодії з БД (PrismaService): централізований вузол для керування підключеннями до бази даних;
- поштовий сервіс (MailService): відповідає за надсилання сервісних повідомлень на електронні адреси користувачів.

Зовнішні інтеграції та інфраструктура:

- вебсокет-з'єднання: канал для двостороннього зв'язку між клієнтом і сервером;
- SMTP-протокол: взаємодія з поштовими серверами для розсилки сповіщень;
- Docker-середовище: контейнеризація для уніфікації процесів розгортання та роботи всіх компонентів системи.

2.1.2 Взаємодія компонентів системи

Внутрішні комунікації в межах платформи «Planner» базуються на чіткому розподілі потоків даних між клієнтом, сервером та базою даних.

Клієнтський інтерфейс – Серверний API:

- користувач ініціює дії через UI-компоненти (форми, кнопки, модальні вікна);
- запити передаються до NestJS-контролерів через HTTP-протокол (REST) або постійне з'єднання (WebSockets);

- отримані відповіді оновлюють стан фронтенду за допомогою React Query або хуків.

Бізнес-логіка (Services) – База даних (Prisma):

- сервіси обробляють логіку запиту та звертаються до Prisma ORM для маніпуляції даними;
- Prisma забезпечує безпечне виконання запитів до PostgreSQL та повертає структуровані об'єкти;
- дані трансформуються сервісами у формат, придатний для відправлення клієнту.

Реальний час та сповіщення:

- при зміні даних на сервері (наприклад, оновлення статусу завдання) спрацьовує WebSocket-шлюз (Gateway);
- подія миттєво транслюється всім підключеним учасникам проєкту;
- одночасно може ініціюватися MailService для відправлення критичних повідомлень на зовнішні поштові сервери.

2.2 Функціонування системи

2.2.1 Основні процеси системи

Робота застосунку складається з декількох циклічних процесів, що забезпечують повний життєвий цикл управління проєктом.

Процедура доступу та ідентифікації::

- користувач вводить облікові дані, які перевіряються сервісом автентифікації;
- система генерує захищений JWT-токен, що зберігається на стороні клієнта для підписання наступних запитів;
- доступ до приватних сторінок та API-ендпоінтів регулюється спеціальними захисними механізмами (Guards) на основі ролей.

Організація командного середовища:

- створюється новий проєкт, де ініціатор автоматично отримує права адміністратора;
- формується список учасників через механізм запрошень за електронною поштою;
- система розподіляє права доступу, дозволяючи кожному учаснику бачити лише закріплені за ним завдання та проєкти.

Управління життєвим циклом завдань:

- учасник створює завдання (issue), заповнюючи необхідні поля (пріоритет, опис, виконавець);
- дані валідуються на сервері та зберігаються в БД, після чого статус завдання може змінюватися в межах спринту;
- будь-які коригування (коментарі, зміна статусу) супроводжуються автоматичним оновленням інтерфейсу в усіх членів команди завдяки вебсокетам.

Планування та робота зі спринтами:

- адміністратор проєкту формує часові ітерації (спринти), додаючи до них завдання з беклогу;
- система відстежує активність спринту та дозволяє перемикатися між різними етапами розробки;
- після завершення терміну спринту завдання можуть бути закриті або перенесені до наступної ітерації.

Комунікація та зворотний зв'язок:

- у межах кожної картки завдання реалізовано потік обговорення (коментарі);
- у разі виникнення технічних проблем користувач може скористатися модулем підтримки для надсилання звіту адміністрації;
- система сповіщень (як внутрішніх, так і через email) гарантує, що жодна важлива зміна не залишиться поза увагою команди.

2.2.3 Логіка роботи системи

Функціонування платформи «Planner» базується на архітектурних засадах, що гарантують стабільність та зручність користування:

- принцип миттєвої синхронізації (Real-time sync): за рахунок інтеграції WebSockets будь-які зміни в проєкті чи статусі завдань одразу відображаються у всіх підключених користувачів. Це усуває потребу в ручному оновленні сторінок та забезпечує безперервність командної роботи;
- принцип безпеки та конфіденційності: доступ до інформації суворо обмежений механізмами автентифікації на базі JWT. Паролі користувачів піддаються незворотному хешуванню перед збереженням у базу даних, а передача всієї інформації відбувається через захищені протоколи;
- принцип рольового розмежування (RBAC): архітектура системи передбачає чітку ієрархію прав. Кожна дія користувача (створення спринту, видалення завдання чи зміна складу команди) перевіряється на відповідність його ролі в конкретному проєкті;
- принцип ефективного керування станом (State management): використання сучасних інструментів кешування на стороні клієнта дозволяє мінімізувати кількість запитів до сервера та забезпечує «плавність» роботи інтерфейсу навіть при нестабільному з'єднанні.

2.3 Вибір мови програмування та інструментарію для розробки програми

2.3.1 Мова програмування

Для розробки системи Planner обрано TypeScript як єдину мову для фронтенду (Next.js) та бекенду (NestJS) [9].

Таке рішення обґрунтоване наступними перевагами:

- надійність: сувора типізація мінімізує помилки та полегшує масштабування;

- універсальність: спільні типи даних (DTO/interfaces) для клієнта та сервера прискорюють розробку;
- продуктивність: Next.js забезпечує швидке завантаження сторінок через SSR, а NestJS – стабільну модульну архітектуру;
- екосистема: використання Prisma ORM та Tailwind CSS дозволяє фокусуватися на бізнес-логіці;
- асинхронність: ефективна обробка паралельних запитів без блокування системи.

2.3.1.1 Опис переваг та обмежень вибраної мови

Переваги використання мови TypeScript у системі «Planner»:

- уніфікація розробки: використання однієї мови для клієнта та сервера дозволяє ділити спільні інтерфейси/DTO, що прискорює розробку та усуває помилки в API;
- надійність: статична типізація виявляє помилки до запуску, що важливо для складних структур спринтів;
- продуктивність: асинхронність Node.js забезпечує швидкий відгук інтерфейсу (до 0,5 с);
- екосистема: доступ до потужних інструментів (Prisma, Socket.io, React/Next.js) для створення інтерактивних дошок.

Обмеження та виклики:

- поріг входження: необхідне знання складних патернів (DI, декоратори);
- об'єм коду: ризик великого розміру клієнтських скриптів, що потребує оптимізації завантаження;
- час компіляції: додаткові витрати ресурсів на збірку проєкту.

Порівняльна характеристика мов програмування для вебплатформ наведена в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз інструментаріїв розробки

Критерій	TypeScript (Next.js/NestJS)	Python (Django/FastAPI)	Java (Spring Boot)
Типізація	Строга (статична)	Динамічна / Опціональна	Строга (статична)
Швидкість розробки	Дуже висока (Full-stack)	Висока	Середня
Робота в реальному часі	Відмінна (WebSockets)	Добра	Добра
Синтаксис	Сучасний, лаконічний	Дуже простий	Складний (багатослівний)
Екосистема	Найбільша у світі (npm)	Велика	Величезна (корпоративна)
Масштабованість	Висока (модульна)	Середня	Дуже висока
Відгук інтерфейсу	< 0,5 с	0,5 – 1,0 с	0,5 – 1,0 с

TypeScript було обрано як найбільш оптимальне рішення для системи «Planner» завдяки можливості створення цілісної архітектури, високій швидкості ітерацій розробки та найкращій підтримці функцій взаємодії користувачів у реальному часі.

2.3.2 Середовище розробки

Visual Studio Code (VS Code) обрано основним середовищем розробки системи завдяки його швидкодії та гнучкості.

Ключові чинники вибору:

- нативна підтримка TypeScript: найкраща інтеграція, миттєва перевірка типів та інтелектуальні підказки;
- екосистема розширень: плагіни для роботи з Prisma, Tailwind CSS та автоформатування коду;
- інтегрований термінал: зручне керування сервером NestJS та клієнтом Next.js в одному вікні;
- продуктивність: низьке споживання ресурсів та швидкий відгук інтерфейсу;

- контроль версій: будовані інструменти Git для керування гілками та змінами.

2.3.2.1 Обґрунтування вибору та опис його можливостей

VS Code – збалансоване Full-stack рішення, що поєднує легкість та потужність.

Ключові інструменти:

- IntelliSense: розумне автодоповнення з урахуванням типів та модулів;
- Debugging: вбудований дебаггер для аналізу Node.js у реальному часі;
- екосистема (ESLint/Prettier): контроль єдиного стилю коду системи;
- робота з БД: плагіни Prisma для візуалізації схем та генерації клієнта.
- кастомізація: персоналізація інтерфейсу та гарячих клавіш для комфортної роботи.

Порівняння популярних середовищ для веброзробки представлено в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика інструментів розробки

Критерій	Visual Studio Code	WebStorm (JetBrains)	Sublime Text
1	2	3	4
Підтримка TypeScript	Повна (вбудована)	Повна (IDE-рівень)	Через плагіни
Споживання ресурсів	Низьке	Високе	Мінімальне
Інтегрований термінал	Присутній	Присутній	Через плагіни

Продовження таблиці 2.2

1	2	3	4
Вартість	Безкоштовно	Платно	Умовно- безкоштовно
Екосистема плагінів	Найбільша у вебi	Вбудовані функції	Обмежена
Зручність Git	Висока	Дуже висока	Базова

Visual Studio Code є оптимальним вибором для реалізації проєкту «Planner», оскільки він пропонує найкращу підтримку обраного технологічного стеку при мінімальних вимогах до апаратного забезпечення та високій швидкості роботи.

2.3.3 База даних

Для стабільного функціонування платформи «Planner», що передбачає складні зв'язки між проєктами, завданнями та користувачами, було обрано реляційну модель збереження даних. Основним технологічним стеком для роботи з інформацією стали PostgreSQL як серверна база даних та Prisma як інструмент об'єктно-реляційного відображення (ORM) [10]-[11].

2.3.3.1 Реляційна база даних PostgreSQL

PostgreSQL є потужною системою керування базами даних з відкритим кодом, яка забезпечує високу надійність та цілісність інформації. У проєкті «Planner» вона виступає централізованим сховищем усіх даних.

Ключові переваги:

- надійність та ACID-відповідність: гарантує коректність виконання транзакцій, що є критичним при зміні статусів завдань або фінансових параметрів проєкту;

- підтримка складних зв'язків: архітектура PostgreSQL ідеально підходить для реалізації деревоподібних структур (Проект – Спринт – Завдання – Коментарі);

- висока продуктивність: здатність ефективно обробляти великі обсяги даних та складні аналітичні запити без втрати швидкодії.

Обмеження: потребує налаштованої серверної інфраструктури та постійного мережевого з'єднання для доступу клієнта до даних.

2.3.3.2 Об'єктно-реляційне відображення (Prisma ORM)

Prisma використовується як сучасний шар абстракції між кодом застосунку (NestJS) та базою даних. Вона забезпечує розробника типізованим клієнтом для виконання запитів.

Переваги використання Prisma:

- типобезпека (Type-safety): автоматична генерація типів на основі схеми БД виключає помилки невідповідності даних у кодї;

- декларативні міграції: спрощує процес оновлення структури бази даних, дозволяючи відстежувати зміни в системі контролю версій;

- зручність розробки: високорівневий API дозволяє замість складних SQL-запитів використовувати зрозумілі методи об'єктів.

Для порівняльного аналізу було розглянуто альтернативні варіанти, такі як NoSQL рішення (MongoDB) та локальні бази (SQLite). Хоча MongoDB пропонує гнучкість схеми, вона поступається у швидкості обробки реляційних зв'язків. SQLite, своєю чергою, не підходить для багатокористувацької вебсистеми через обмеження у масштабуванні. Таким чином, комбінація PostgreSQL та Prisma була визначена як найбільш збалансована для реалізації вимог проекту.

Порівняння різних систем керування даними наведено в таблиці 2.3.

Таблиця 2.3 – Порівняльний аналіз систем керування базами даних

Критерій	PostgreSQL (з Prisma)	MongoDB	SQLite
Тип моделі	Реляційна (SQL)	Документо-орієнтована (NoSQL)	Реляційна (локальна)
Цілісність даних	Дуже висока (ACID)	Базова	Висока (локальна)
Складні запити	Відмінно підтримує	Обмежено	Базовий рівень
Масштабованість	Висока (вертикальна/горизонтальна)	Дуже висока	Низька
Типізація	Суворі (через Prisma)	Відсутня (динамічна)	Базова
Синхронізація	Через API сервіси	Через хмарні атласи	Ручна / Відсутня
Складність зв'язків	Будь-яка складність	Ускладнена	Обмежена

Використання PostgreSQL у парі з Prisma ORM забезпечує платформі «Planner» необхідний рівень надійності, високу швидкість розробки завдяки автоматизації запитів та гарантує цілісність даних у складному командному середовищі.

2.4 Вимоги до технічних і програмних засобів

2.4.1 Необхідні технічні засоби для розробки та експлуатації

Для успішної побудови та експлуатації платформи «Planner» необхідний наступний комплекс технічних засобів:

- робоча станція розробника: комп'ютер з достатньою потужністю для одночасного запуску серверної та клієнтської частин застосунку, а також бази даних;
- серверна інфраструктура (або Docker-контейнер): середовище для розгортання PostgreSQL та виконання серверного коду на базі Node.js;

- клієнтські пристрої: будь-які девайси з сучасними веббраузерами (десктопи, ноутбуки, планшети) для перевірки адаптивності та функціональності інтерфейсу;
- мережеве з'єднання: стабільний канал зв'язку для роботи WebSocket-протоколу та взаємодії з поштовими сервісами.

2.4.2 Вимоги до апаратного та програмного забезпечення

Апаратні параметри (мінімальні):

- процесор: багатоядерний Intel Core i5 (8-го покоління або новіше) або аналогічний Apple M-серії для швидкої транспіляції TypeScript-коду;
- оперативна пам'ять: не менше 8 Гб (рекомендовано 16 Гб) для забезпечення стабільної роботи Docker-контейнерів та декількох запущених екземплярів Node.js;
- дисковий простір: від 5 Гб вільного місця для встановлення Node.js, залежностей проєкту (node_modules) та зберігання логів бази даних.

Програмне забезпечення:

- операційна система: macOS (12.0+), Windows (10/11) з підтримкою WSL2 або дистрибутиви Linux (Debian/Ubuntu);
- середовище розробки: Visual Studio Code з актуальними розширеннями для TypeScript та Prisma;
- середовище виконання: Node.js версії 18.x або новішої та менеджер пакетів npm/yarn;
- система керування базами даних: PostgreSQL версії 14+ або вище;
- додаткові інструменти: Docker та Docker Compose для уніфікації середовища розгортання.

2.5 Висновки за розділом 2

У межах другого розділу було детально спроектовано архітектуру вебсистеми «Planner» та визначено механізми взаємодії її ключових модулів.

Обрана модель Full-stack розробки на базі мови TypeScript дозволила створити цілісну екосистему, де Next.js відповідає за динамічний інтерфейс, а NestJS – за стабільну серверну логіку. Використання Visual Studio Code як основного інструменту розробки обґрунтовано його легкістю та нативною підтримкою обраного стеку, що значно прискорює цикл написання та налагодження коду. Впровадження реляційної бази даних PostgreSQL у поєднанні з Prisma ORM забезпечило надійність збереження інформації та високу швидкість виконання складних запитів. Реалізація функцій реального часу через WebSockets стала ключовим елементом для забезпечення інтерактивності командної роботи. Запропонована архітектура повністю відповідає вимогам щодо продуктивності, безпеки та масштабованості системи.

З РОЗРОБКА ПРОГРАМИ

3.1 Архітектура системи

Архітектура застосунку «Planner» побудована на базі клієнт-серверної моделі з модульною структурою на бекенді та компонентним підходом на фронтенді. Такий підхід забезпечує чітке розділення бізнес-логіки, представлення даних та взаємодії з базою даних, що сприяє високій продуктивності, масштабованості та зручності підтримки системи.

Основні компоненти архітектури.

Data Layer (Рівень даних / Модель) – відповідає за структуру даних та взаємодію з базою даних (PostgreSQL):

- класи моделей даних, описані через Prisma (User, Project, Issue, Sprint, Comment, Notification);
- рівень доступу до даних (PrismaService), що забезпечує виконання запитів та дотримання цілісності даних.

Presentation Layer (Рівень представлення / View) – відповідає за візуалізацію інтерфейсу та взаємодію з користувачем (Next.js):

- сторінки та макети (Dashboard, ProjectsPage, IssuesPage, Settings тощо);
- інтерактивні компоненти (AppSidebar, IssueDetailModal, CreateProjectModal, KanbanBoard);
- UI-бібліотека на базі Shadcn UI для забезпечення єдиного стилю.

Logic Layer (Рівень логіки / Controller & Service) – з'єднує інтерфейс із даними та обробляє запити:

- контролери бекенду, що обробляють HTTP-запити (AuthController, IssuesController, ProjectsController);
- клієнтські провайдери та хуки (AuthProvider, QueryProvider, SocketProvider) для управління станом на стороні фронтенду.

Services (Сервіси) – реалізують специфічну функціональність для різних модулів застосунку:

- AuthService – автентифікація користувачів (JWT) та розмежування прав доступу (RBAC);
 - PrismaService – централізована робота з базою даних;
 - IssuesService – керування життєвим циклом завдань, їх статусами та пріоритетами;
 - ProjectsService – адміністрування проєктів та управління правами учасників;
 - SprintsService – планування ітерацій та контроль виконання спринтів;
 - WebsocketGateway – забезпечення синхронізації даних між користувачами у реальному часі;
 - MailService – автоматичне надсилання поштових сповіщень про важливі події;
 - NotificationsService – система внутрішніх сповіщень користувача.
- Схема взаємодії компонентів системи зображена на рисунку 3.1.

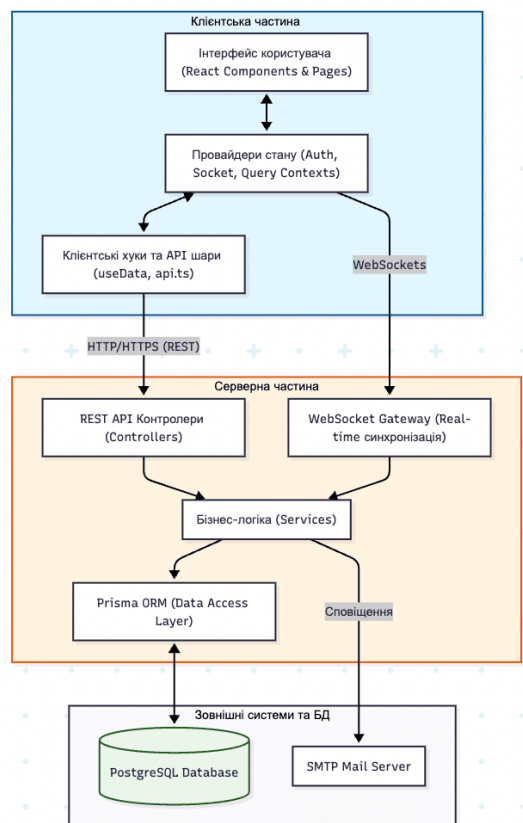


Рисунок 3.1 – Схема взаємодії компонентів системи

3.1.1 Структурна схема системи

Розробка проєкту «Planner» здійснювалася за принципом модульної декомпозиції. Програмний код розподілено на ізольовані логічні блоки відповідно до їх функціонального призначення. Такий підхід дозволяє забезпечити низьку залежність між компонентами системи (low coupling) та високу спеціалізацію кожного модуля (high cohesion), що значно спрощує процес тестування, підтримки та масштабування застосунку.

Структурну схему системи, що відображає ієрархію та склад основних модулів, зображено на рисунку 3.2.

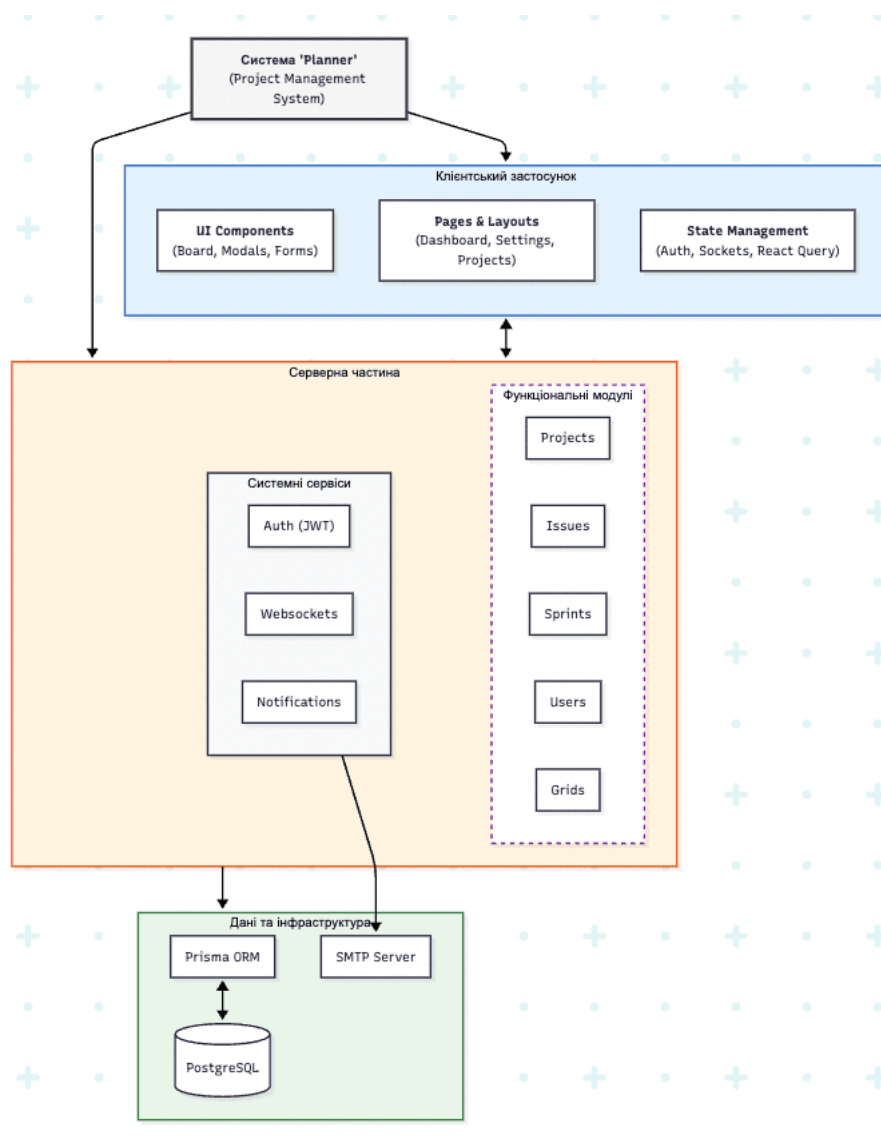


Рисунок 3.2 – Структурна схема системи «Planner»

3.2 Функціонування системи

3.2.1 Функціональна схема

Функціонування платформи «Planner» базується на взаємодії низки алгоритмів, що забезпечують ефективне управління життєвим циклом проєктів та оперативну обробку користувацьких запитів. Система підтримує стабільну роботу в багатокористувацькому режимі, гарантуючи цілісність даних та миттєву синхронізацію змін через WebSockets. Алгоритми платформи охоплюють процеси від авторизації та розмежування прав доступу до складних операцій із планування спринтів та автоматичного сповіщення учасників. Загальну функціональну схему розробленої системи представлено на рисунку 3.3.

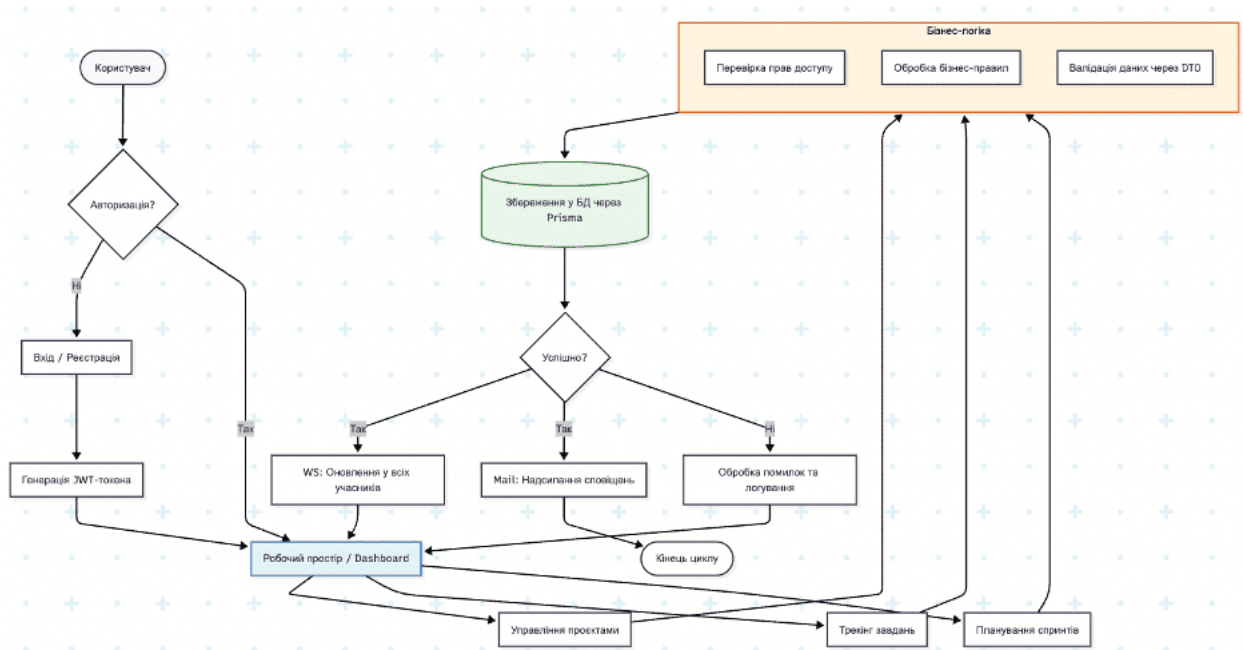


Рисунок 3.3 – Функціональна схема системи «Planner»

3.2.2 Опис процесу обміну даними у системі

Архітектура потоків даних у системі «Planner» реалізована на засадах подієво-орієнтованого підходу, що дозволяє миттєво відображати будь-які зміни в робочому просторі користувача. Обмін інформацією відбувається за

чітко визначеним циклом, який гарантує цілісність даних на сервері та їх актуальність у клієнтському інтерфейсі. Основні етапи життєвого циклу даних (рис. 3.4):

- ініціація дії користувачем. Через графічний інтерфейс (Next.js) користувач виконує операцію: створює завдання, змінює статус спринту або додає коментар;
- обробка запиту та валідація. Клієнтські API-хуки передають дані на бекенд (NestJS), де контролери та сервіси перевіряють коректність введеної інформації (через DTO) та права доступу користувача;
- персистентність у централізованому сховищі. Після успішної перевірки бізнес-логіки дані зберігаються в реляційній базі даних PostgreSQL за допомогою Prisma ORM, що забезпечує транзакційність операцій;
- широкомовна розсилка та сповіщення (Real-time Broadcast). Система ініціює подію через WebSocket-шлюз для всіх активних учасників проєкту та, за потреби, чергує відправку поштових сповіщень;
- реактивне оновлення інтерфейсу. Завдяки механізмам React Query та Socket.io, зміни автоматично підтягуються в інтерфейс без перезавантаження сторінки, забезпечуючи безперервність робочого процесу.

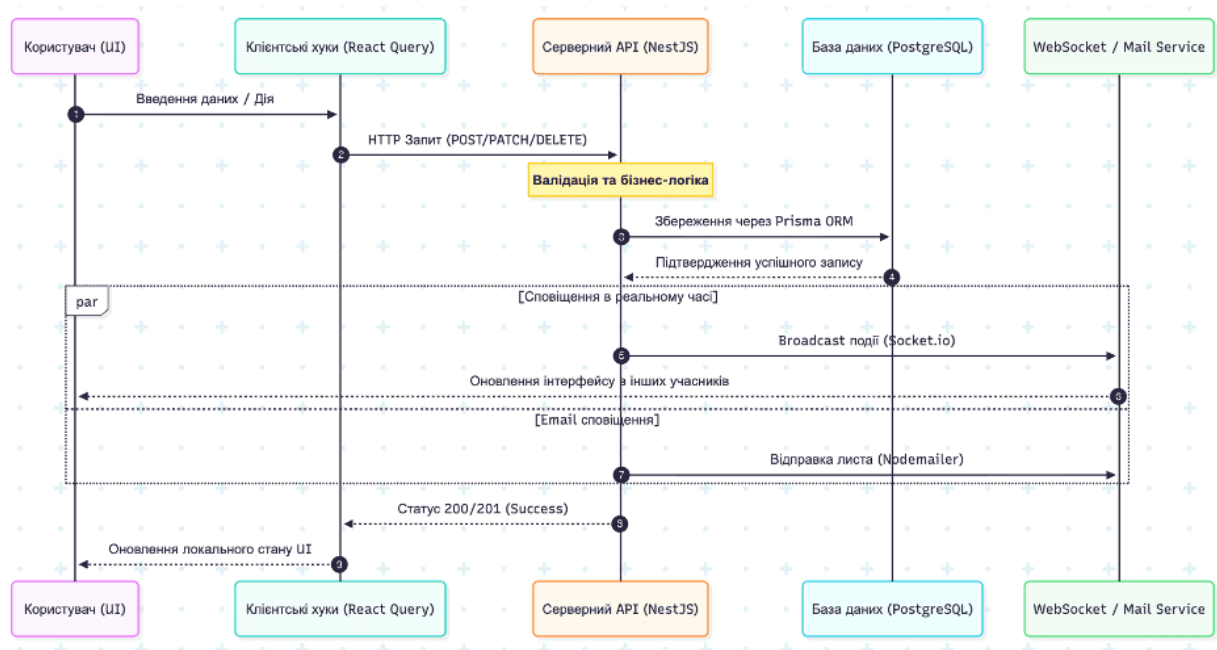


Рисунок 3.4 – Схема процесу обміну даними у системі

3.3 Проєктування бази даних

3.3.1 Структура бази даних

База даних вебзастосунку «Planner» побудована на реляційній моделі та реалізована за допомогою СУБД PostgreSQL. Для взаємодії з даними використовується сучасний інструмент Prisma ORM, що забезпечує типізацію запитів та спрощує міграції. Архітектура бази даних розроблена з урахуванням необхідності швидкого доступу до проєктів, завдань (ішувів) та забезпечення цілісності даних при спільній роботі користувачів.

Основними сутностями системи є користувачі, проєкти, учасники проєктів, спринти, завдання, коментарі та сповіщення. Кожна сутність відповідає за конкретний аспект функціоналу: від автентифікації до відстеження історії змін у завданнях.

3.3.2 Взаємозв'язки між таблицями

Взаємозв'язки в системі побудовані на основі логіки командної роботи та ієрархії управління завданнями:

- Users – Projects: зв'язок один-до-багатьох (власник) та багато-до-багатьох через таблицю ProjectMember;
- Projects – Issues: зв'язок один-до-багатьох, де кожне завдання обов'язково належить до одного проєкту;
- Issues – Sprints: зв'язок багато-до-одного, де завдання може бути призначене до спринту або знаходитися в беклозі;
- Issues – Users: два типи зв'язків: виконавець (assignee) та автор (reporter);
- Issues – Comments: зв'язок один-до-багатьох, де до кожного завдання можна додати необмежену кількість коментарів;
- Issues – IssueHistory: зв'язок один-до-багатьох для аудиту змін полів завдання.

Схему взаємозв'язків між таблицями зображено за допомогою діаграми Mermaid на рисунку 3.5.

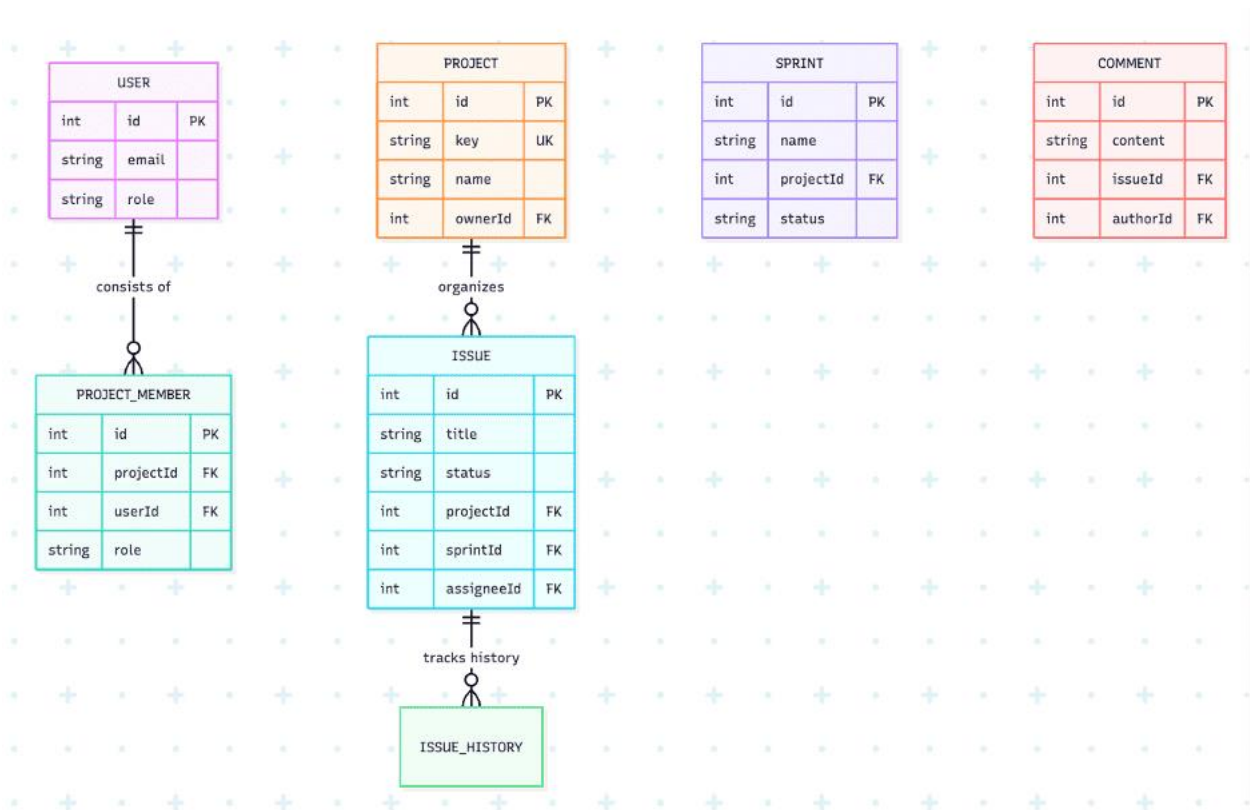


Рисунок 3.5 – Схема взаємозв'язків між таблицями бази даних «Planner»

3.4 Висновки за розділом 3

У результаті виконання третього розділу було детально описано процес проєктування та технічної реалізації багатокористувацької платформи «Planner». У ході роботи розроблено ключові алгоритми функціонування системи, зокрема механізми безпечної автентифікації, алгоритми інтерактивної зміни статусів завдань (drag-and-drop), методи миттєвої синхронізації через сокет-з'єднання та логіку ітераційного планування спринтів.

Архітектура програмного комплексу побудована на базі клієнт-серверної моделі з чітким розподілом обов'язків. Серверна частина реалізована на фреймворку NestJS із застосуванням модульного підходу, а клієнтська – на

базі React із використанням Context API та React Query для ефективного управління станом і кешування даних. Така структура забезпечує високу масштабованість системи та зручність її подальшого супроводу.

Розроблено та реалізовано сім основних модулів системи.

Модуль реєстрації та авторизації – забезпечує безпечний доступ до платформи через механізми JWT-автентифікації та хешування конфіденційних даних за допомогою Bcrypt. Ключові фрагменти програмної реалізації процесів автентифікації (валідація через DTO та генерація токенів) наведено у Додатках Б.1 та Б.2. Екран входу в систему розроблений як інтерактивна React-сторінка, що використовує клієнтські хуки для взаємодії з AuthService та автоматичного перенаправлення користувача до робочого простору після успішної перевірки даних. Реалізацію у коді наведено у Додатках Б.3 та Б.4.

Модуль управління проєктами та завданнями – ядро системи, що дозволяє структурувати робочий простір, керувати життєвим циклом завдань (Issues) та налаштовувати права доступу учасників. Ключові фрагменти програмної реалізації, що відповідають за створення та обробку нових завдань (Issues), наведено в Додатку Б.6. Детальну реалізацію логіки переміщення карток кодом показано в Додатку Б.7.

Модуль синхронізації в реальному часі – базується на протоколі WebSockets, забезпечуючи миттєве оновлення інформації у всіх користувачів без необхідності ручного перезавантаження сторінок. Ключові фрагменти коду, що реалізують синхронізацію даних між користувачами наведено в Додатках Б.8 та Б.9.

Модуль управління спринтами та ітераціями – інструментарій для короткострокового планування, що дозволяє групувати завдання за часовими межами та відстежувати прогрес виконання ітерацій. Ключові фрагменти коду для керування спринтами наведено в Додатку Б.10.

Модуль сповіщень та технічної підтримки – відповідає за інформування користувачів про важливі події в проєкті та надає канал зв'язку

з розробниками для вирішення технічних питань. Ключові фрагменти коду наведено в Додатках Б.11 та Б.12.

Модуль коментарів та командної взаємодії – забезпечує контекстне спілкування всередині карток завдань, підтримуючи спільну роботу над складними технічними рішеннями. Ключові фрагменти коду для командної взаємодії наведені в Додатку Б.13.

Модуль системних налаштувань та профілю – дозволяє кожному учаснику персоналізувати інтерфейс, керувати власними даними та налаштовувати параметри отримання нотифікацій. Ключові фрагменти коду для керування налаштуваннями наведено в Додатку Б.5.

Особливу увагу приділено проєктуванню бази даних на основі PostgreSQL. Визначено структуру таблиць, зв'язки «один-до-багатьох» та «багато-до-багатьох», а також впроваджено систему індексів для оптимізації пошукових запитів за ключами проєктів та статусами завдань. Безпека системи гарантується використанням захищених HttpOnly-cookies, валідацією даних через DTO на рівні API та розмежуванням прав доступу на основі ролей (RBAC).

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Вебзастосунок "Planner" – це сучасна платформа для управління проєктами та координації командної роботи, розроблена для оптимізації робочих процесів у динамічних середовищах. Програма дозволяє структурувати складні проєкти за допомогою гнучких сіток (grids), планувати ітерації в межах спринтів, відстежувати статус окремих завдань (issues) та забезпечувати миттєву комунікацію між учасниками команди через систему сповіщень.

Застосунок охоплює ключові аспекти проєктного менеджменту: від високорівневого планування дорожніх карт проєктів до детального обліку щоденних операцій. Функціонал включає управління життєвим циклом завдань, розподіл ролей у команді, аналіз ефективності виконання спринтів та візуалізацію прогресу через інтерактивні інтерфейси.

Цільова аудиторія охоплює ІТ-команди, стартапи, фріланс-групи та малий бізнес, де критично важливою є прозорість процесів та швидкість прийняття рішень. Окрім професійних команд, "Planner" підходить індивідуальним розробникам та студентам для організації особистих навчальних або комерційних проєктів. Завдяки інтуїтивно зрозумілому інтерфейсу, система є доступною як для досвідчених менеджерів проєктів, так і для користувачів, які тільки починають впроваджувати цифрові інструменти планування у свою діяльність.

4.1.1 Огляд планової цільової аудиторії

Для оцінки потенційного ринку та користувацької бази застосунку "Planner" було проведено аналіз сучасних трендів цифрової трансформації та ринку інструментів для спільної роботи (Collaboration Tools). Нижче наведено дві прогнозовані структури аудиторії за віковими категоріями та типами

команд (організаційною структурою). Ці дані базуються на глобальних звітах про використання SaaS-рішень для продуктивності та специфіці українського IT-сектору, який є основним споживачем подібних продуктів.

4.1.1.1 Розподіл за віком

Прогнозований розподіл базується на даних про активність професійних груп, залучених до інтелектуальної праці та управління проєктами. Згідно з дослідженнями ринку праці в IT та креативних індустріях, основне ядро користувачів інструментів планування зосереджено в сегменті молодих фахівців та менеджерів середньої ланки [12].

Група 22–30 років (40%): Найбільш активний сегмент, що включає Junior та Middle спеціалістів, стартаперів та фрілансерів. Вони відкриті до нових інструментів, прагнуть максимально цифровізувати свої робочі процеси та цінують швидкість роботи інтерфейсу (Single Page Application).

Група 31–40 років (35%): Ключова категорія, що складається з Team Leads, Project Managers та засновників бізнесу. Їхній інтерес до застосунку зумовлений потребою в ефективному контролі за виконанням завдань підлеглими та аналізі результативності спринтів.

Група 18–21 рік (15%): Студенти технічних спеціальностей та початківці, які використовують платформу для академічних проєктів або першого досвіду командної розробки.

Група 41+ років (10%): Досвідчені керівники та консультанти. Хоча ця група менш численна, вона формує вимоги до надійності та безпеки даних у системі.

4.1.1.2 Розподіл за організаційною структурою

Прогнозована структура базується на глобальних трендах споживання SaaS-продуктів, де основний драйвер ринку — малі та середні команди, що шукають альтернативу складним Enterprise-рішенням [13].

Малі команди та стартапи (55%): Цей сегмент є домінуючим, оскільки сучасні стартапи та невеликі ІТ-команди в Україні на 13% швидше адаптують нові цифрові рішення у порівнянні з великим бізнесом. Вони надають перевагу гнучкості та швидкості розгортання [14].

Фрілансери та індивідуальні користувачі (20%): Зростання сегменту зумовлене популярністю гібридних моделей роботи та мультипроектності.

Середні компанії та Департаменти (20%): Організації, де впровадження відбувається на рівні окремих юнітів (маркетинг, дизайн, R&D).

Великі організації (5%): Сегмент «пілотних проєктів», де Planner використовується як допоміжний інструмент для швидких ітерацій окремих команд всередині корпорації [13].

4.1.2 Візуалізація планованої цільової аудиторії

На основі проведеного аналізу нижче представлено графічне відображення структури потенційних користувачів системи "Planner" за віковими показниками та форматом професійної діяльності (рис. 4.1 та 4.2).

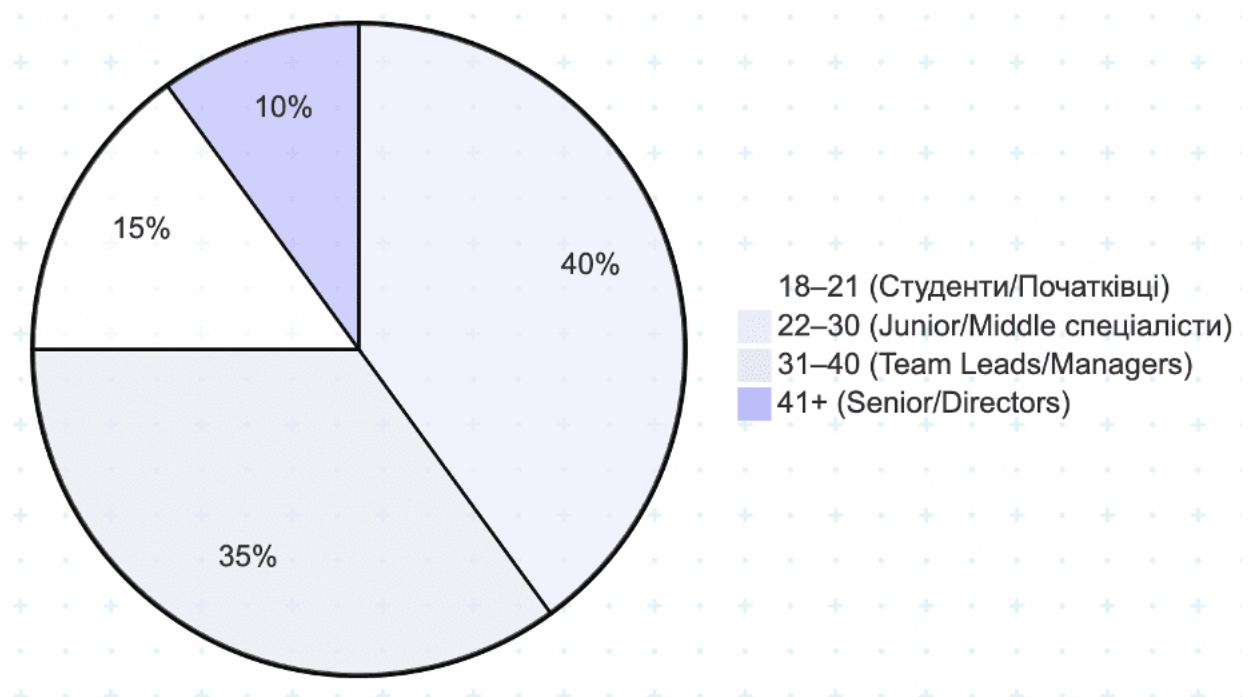


Рисунок 4.1 – Розподіл аудиторії Planner за віковими категоріями

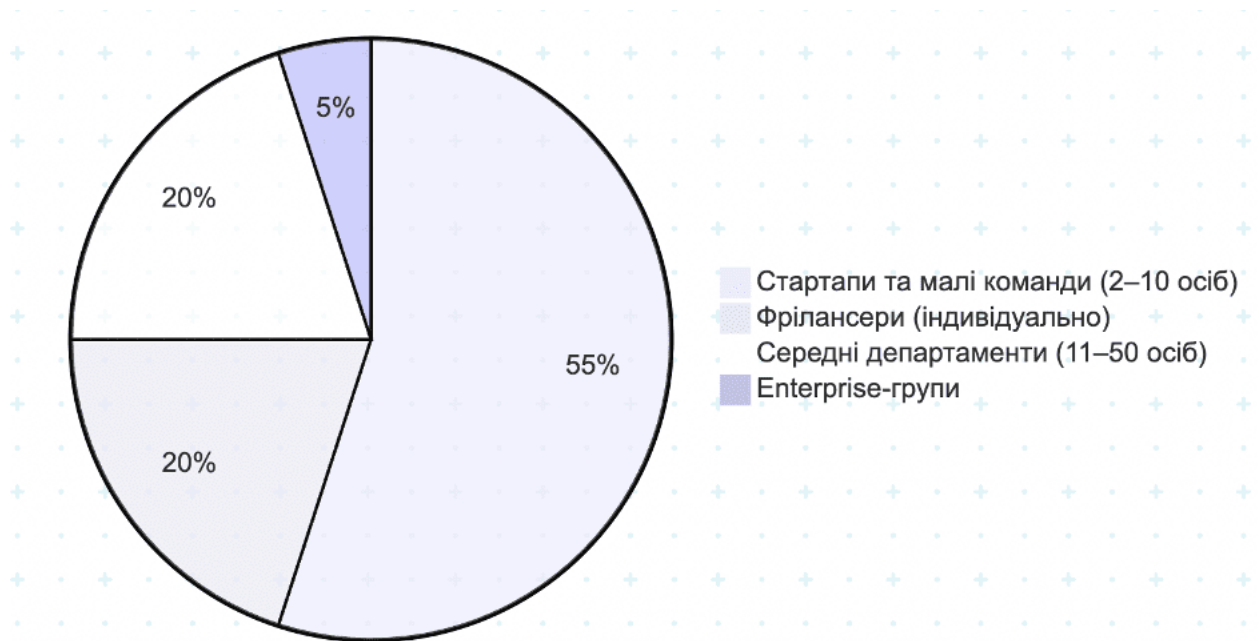


Рисунок 4.2 – Розподіл за типом організаційної структури

4.2 Інструкція по роботі з програмою

4.2.1 Встановлення програми

Оскільки "Planner" є кросплатформним вебзастосунком (SaaS), він не потребує класичного встановлення на локальний диск, проте для стабільної роботи необхідно дотримуватися технічних вимог:

а) технічні вимоги до середовища:

- 1) актуальна версія браузера (Google Chrome 90+, Firefox 88+, Safari 14+);
- 2) мінімальна роздільна здатність екрана 1280x720 пікселів;
- 3) стабільне підключення до мережі Інтернет (для роботи WebSocket-сповіщень);
- 4) увімкнена підтримка JavaScript та Cookies у налаштуваннях браузера;

б) доступ через вебінтерфейс:

- 1) перейдіть за посиланням на хостинг застосунку;
- 2) додайте сторінку в закладки браузера для оперативного доступу до робочого простору;

в) необхідні дозволи браузера:

- 1) дозвіл на надсилання сповіщень (для отримання апдейтів по завданнях);
- 2) дозвіл на використання локального сховища (LocalStorage) для збереження сесії;
- 3) дозвіл на доступ до файлової системи (для завантаження вкладень до завдань).

Схему процесу реєстрації та входу в систему зображено на рисунку 4.3.

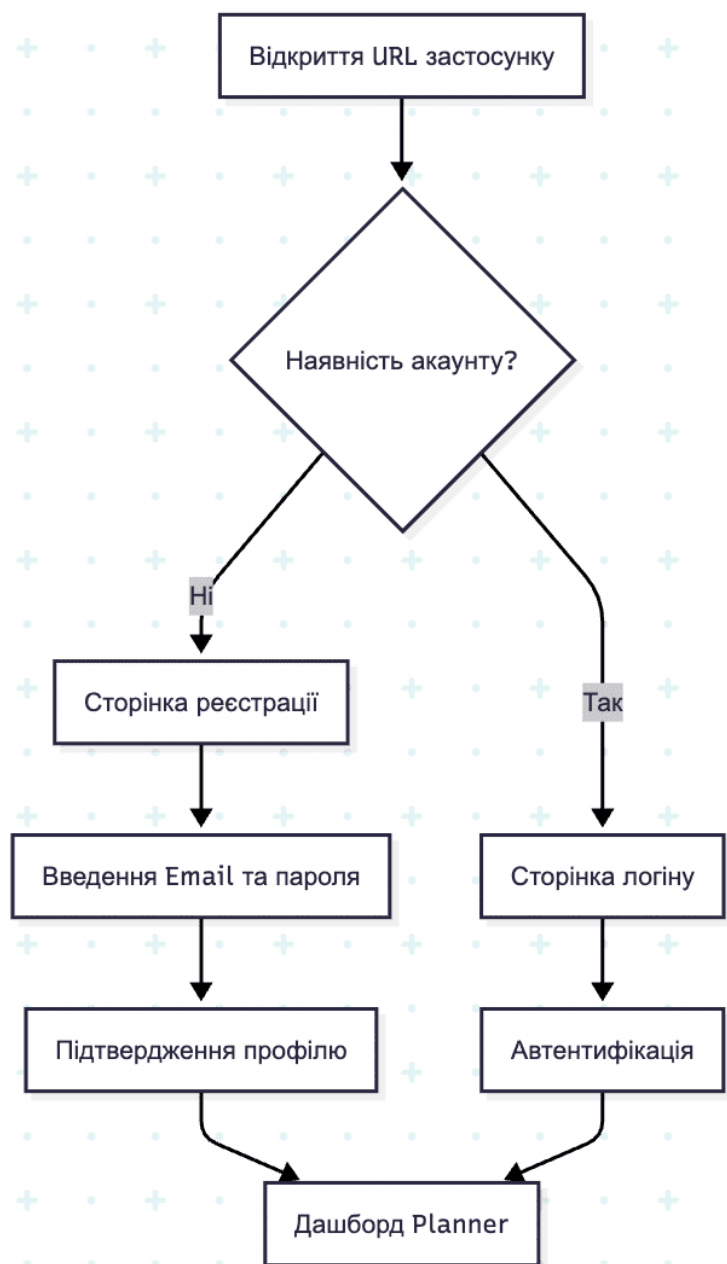


Рисунок 4.3 – Схема процесу авторизації в системі

4.2.2 Запуск та початкове налаштування програми

Після відкриття вебзастосунку користувачу необхідно виконати кроки для авторизації та конфігурації робочого простору:

а) перший запуск та реєстрація:

1) відкрийте сторінку застосунку. Якщо ви не авторизовані, система автоматично перенаправить вас на екран логіну;

2) для створення нового акаунту перейдіть до розділу реєстрації, де необхідно ввести повне ім'я (Full Name), електронну пошту та пароль (мінімум 6 символів);

3) після реєстрації ви отримаєте роль користувача (USER) за замовчуванням;

б) налаштування профілю та ідентифікація:

1) система автоматично ініціалізує ваш профіль, використовуючи введене ім'я;

2) у розділі налаштувань (/settings) ви можете змінити параметри сповіщень на пошту або оновити особисті дані;

в) створення проєкту або приєднання до команди:

1) створення проєкту: ви можете створити власний проєкт, вказавши його назву та унікальний ключ (Project Key – наприклад, "WEB" або "APP"). Як творець, ви автоматично отримуєте роль власника (Owner) та адміністратора проєкту;

2) приєднання до команди: На відміну від сімейних застосунків, у Planner приєднання відбувається через додавання вашої електронної пошти адміністратором існуючого проєкту. Після додавання ви отримаєте сповіщення і проєкт з'явиться у вашому боковому меню (Sidebar) у списку "General";

3) для кожного проєкту можна налаштувати права доступу (ADMIN, MEMBER, VIEWER), що визначає можливість редагування завдань та спринтів.

Сторінки реєстрації та авторизації, що можна побачити при першому запуску застосунку, та сторінку налаштувань наведено на рисунках 4.4, 4.5 та рисунку 4.6 відповідно.

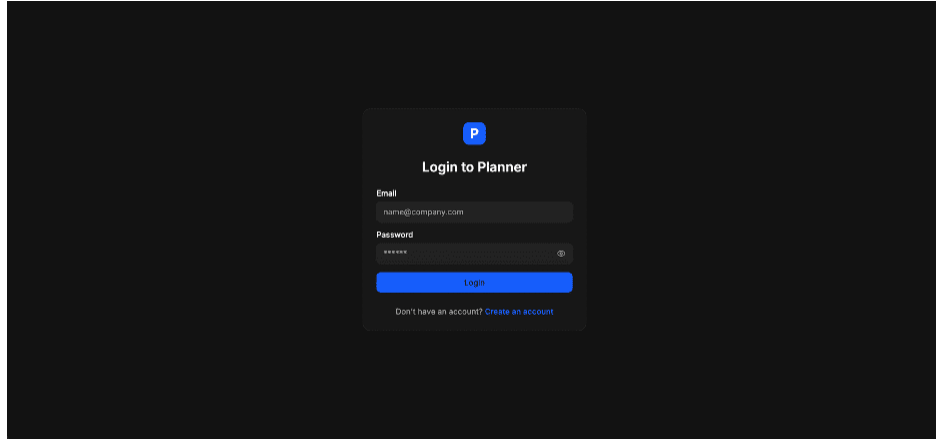


Рисунок 4.4 – Сторінка авторизації

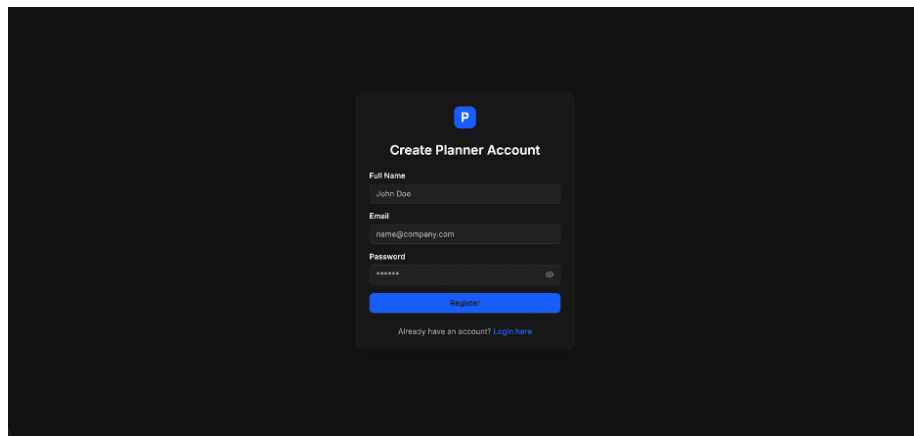


Рисунок 4.5 – Сторінка реєстрації

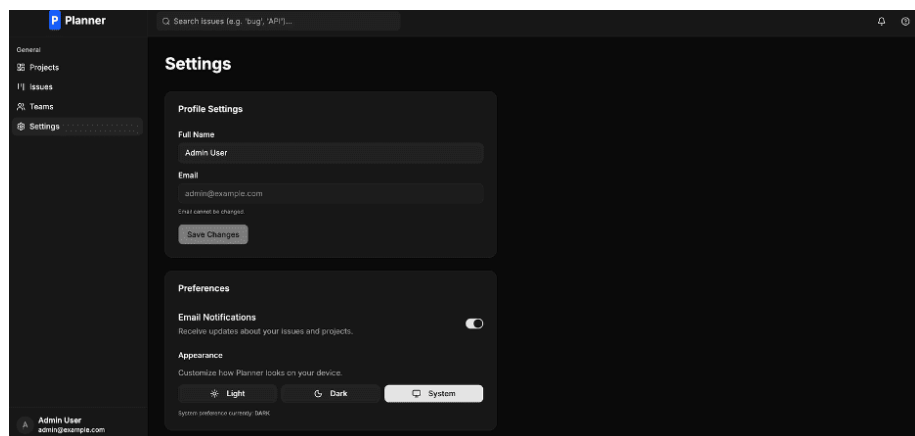


Рисунок 4.6 – Сторінка налаштувань

4.2.3 Методика роботи з програмою

Процес взаємодії з системою "Planner" базується на трьох ключових циклах (рис. 4.7):

а) управління проєктами та командами:

- 1) створення нового проєкту з унікальним ключем (Key);
- 2) налаштування гнучких сіток (Grids) для відображення даних;
- 3) керування правами доступу (ADMIN, MEMBER, VIEWER) для учасників команди;

б) робота із завданнями (Issues):

- 1) створення та редагування завдань з визначенням типу (Bug, Task, Story, Epic);
- 2) делегування завдань конкретним виконавцям (Assignee);
- 3) зміна статусів за допомогою перетягування (Drag-and-drop) на Kanban-дошці;

в) координація та моніторинг:

- 1) планування ітерацій через систему спринтів (Sprints);
- 2) коментування завдань та обговорення в режимі реального часу;
- 3) відстеження історії змін (Activity History) кожного завдання.

Для максимальної продуктивності рекомендується чітко визначати пріоритети (LOW до CRITICAL) та використовувати систему сповіщень для оперативного реагування на оновлення в проєктах. Спринтове планування дозволяє команді фокусуватися на конкретних цілях у визначені часові проміжки.

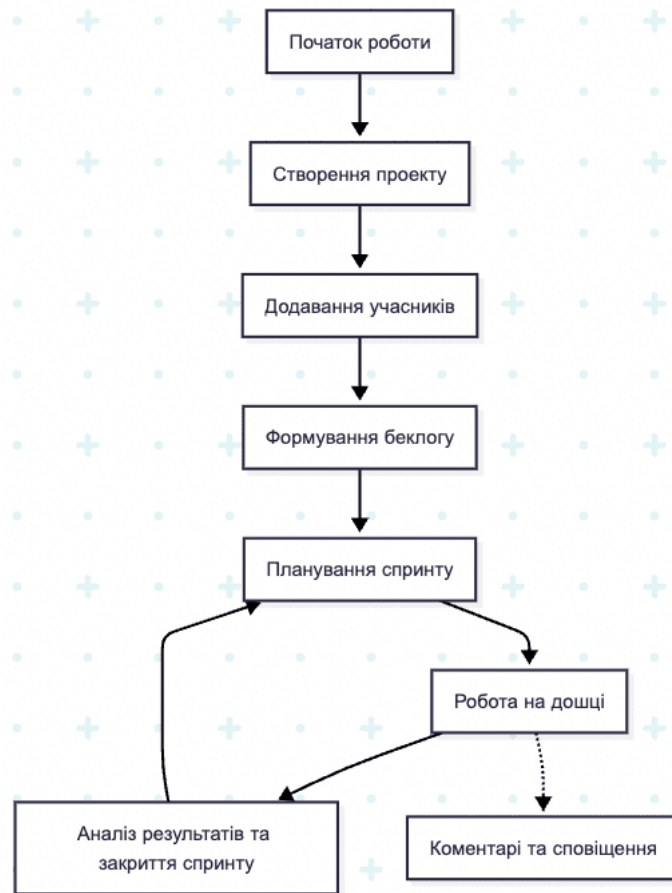


Рисунок 4.7 – Схема життєвого циклу проекту в системі

4.3 Приклад роботи користувача з програмою

Розглянемо типовий сценарій використання "Planner" на прикладі команди розробників з трьох осіб, що працюють над вебпорталом. Процес розпочинається з ініціалізації робочого простору.

Керівник проекту (Project Manager) реєструється в системі та створює проєкт під назвою «Web Development» з ключем «WEB». Після цього він переходить до розділу «Teams» та запрошує розробників через їхні Email-адреси, призначаючи їм роль MEMBER.

Далі РМ формує беклог проєкту: створює завдання (Issues), вказуючи тип, пріоритет та додаючи опис. Для початку роботи створюється «Sprint 1», до якого перетягуються пріоритетні завдання. У щоденному процесі розробники відкривають Kanban-дошку, де через зручний drag-and-drop інтерфейс переміщують завдання зі стовпця «TODO» в «IN PROGRESS».

Всі учасники отримують миттєві сповіщення (через WebSockets) про будь-які зміни: зміну статусу, нові коментарі або оновлення опису. Якщо розробнику потрібно уточнити деталі, він залишає коментар безпосередньо в картці завдання. Керівник проєкту в будь-який момент може переглянути історію змін завдання (Issue History), щоб побачити, хто і коли змінив статус чи пріоритет.

Вікна налаштування команди проєкту та створення запиту на додавання учасника та вибір йому ролі зображені на рисунках 4.8-4.9.

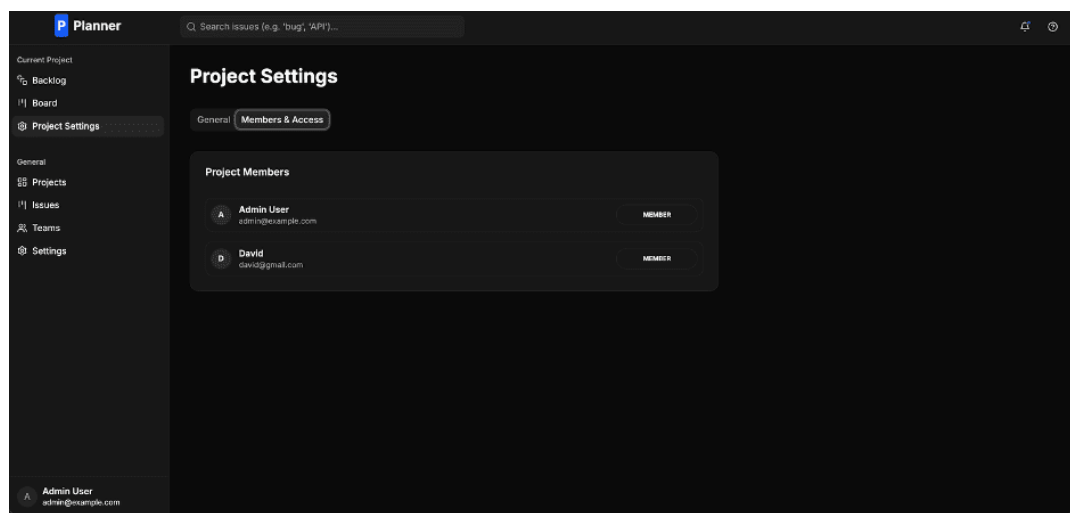


Рисунок 4.8 – Налаштування команди проєкту

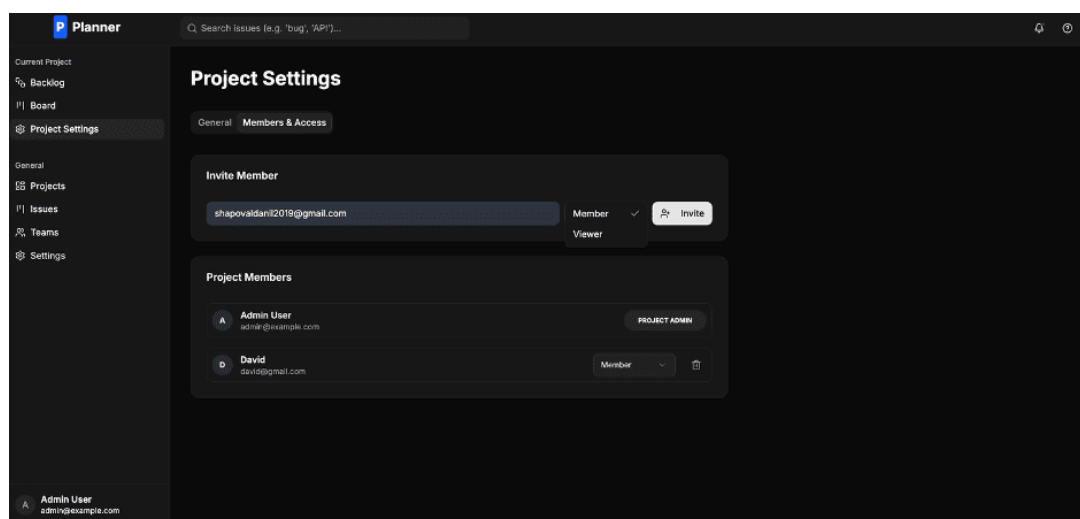


Рисунок 4.9 – Запит на додавання учасника та вибір йому ролі

Вікна створення нового завдання, приклад інтерфейсу дошки та системи сповіщень зображені на рисунках 4.10-4.12.

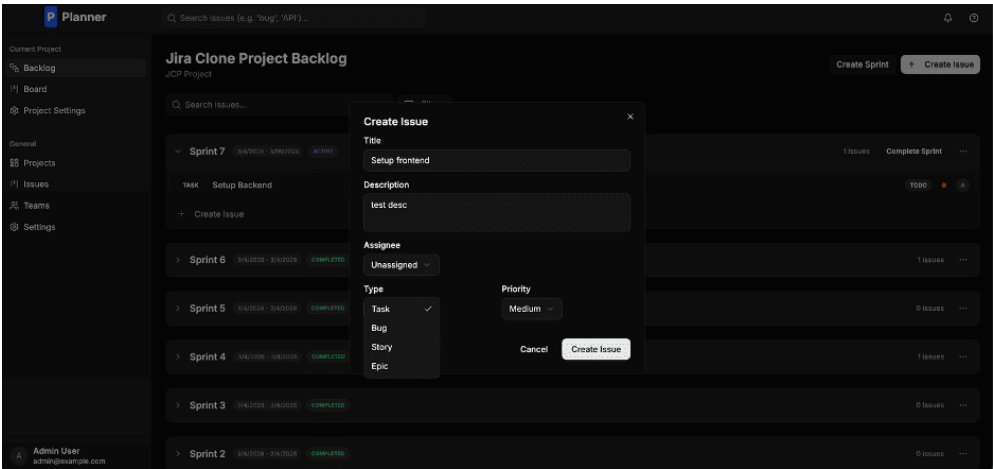


Рисунок 4.10 – Створення нового завдання

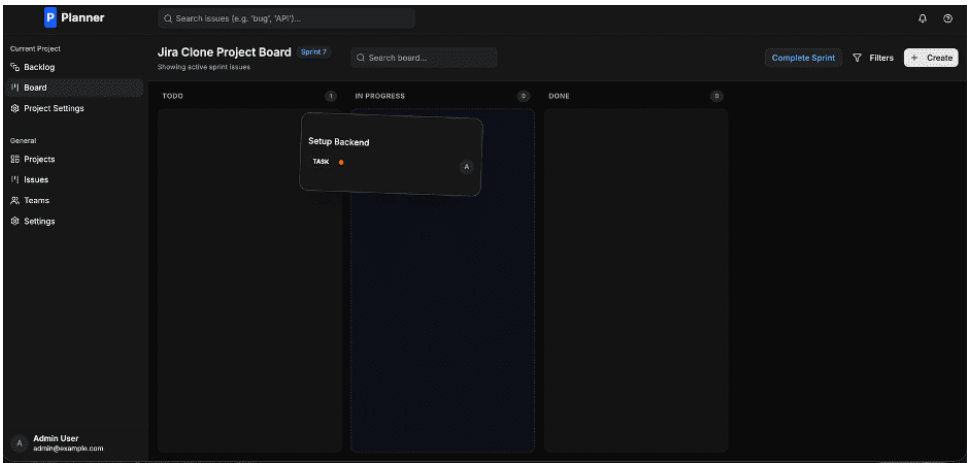


Рисунок 4.11 – Приклад інтерфейсу дошки (Board)

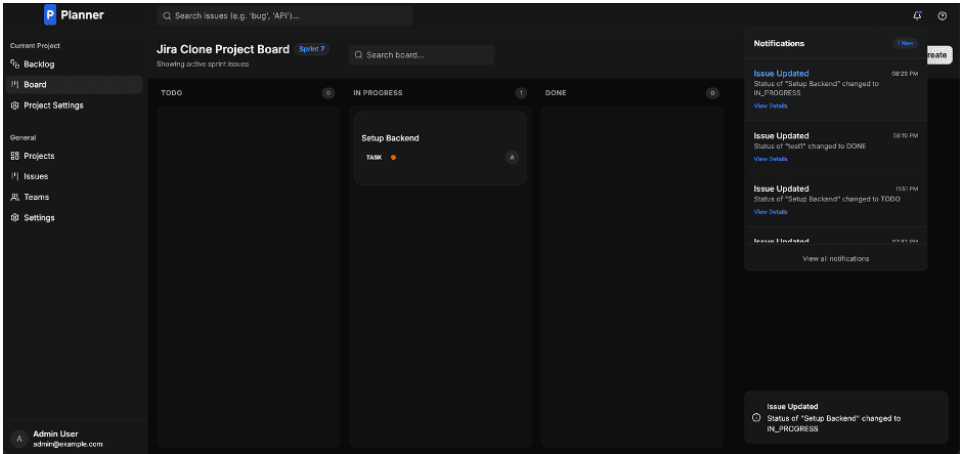


Рисунок 4.12 – Система сповіщень

Вікна створення коментаря до завдання та перегляду історії змін у завданні зображені на рисунках 4.13-4.14.

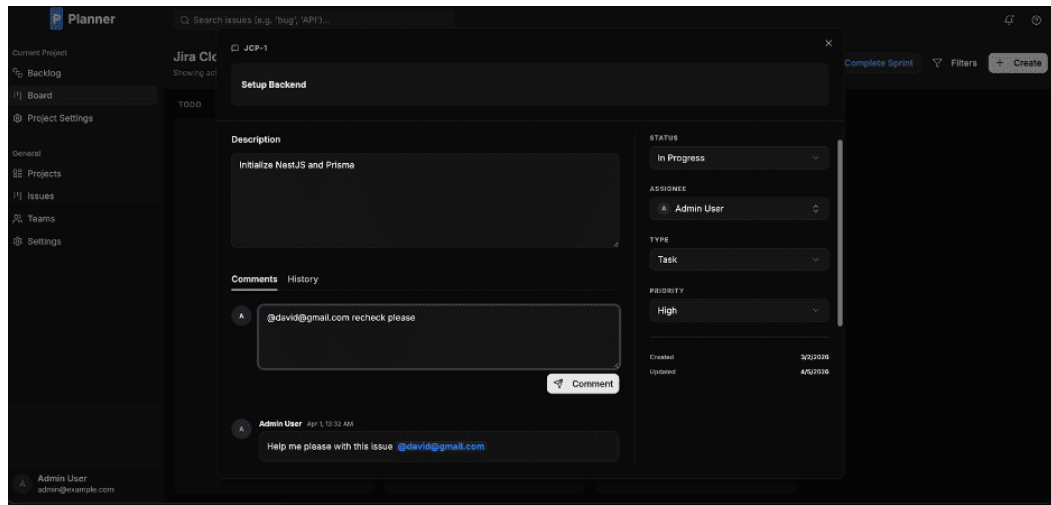


Рисунок 4.13 – Створення коментаря до завдання

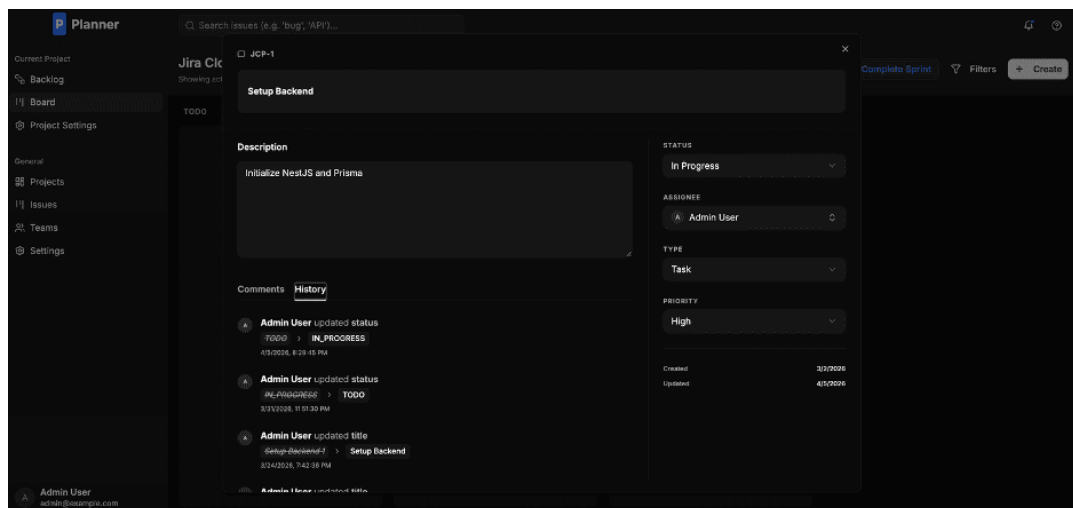


Рисунок 4.14 – Перегляд історії змін у завданні

4.4 Тестування роботи програми

4.4.1 Функціональне тестування

Для забезпечення надійності та коректності роботи всіх модулів системи "Planner" було проведено комплексне функціональне тестування, яке включало такі методи:

а) модульне тестування (Unit Testing):

- 1) перевірка окремих сервісів бекенду;
- 2) тестування бізнес-логіки розрахунку термінів спринтів та пріоритезації завдань;
- 3) валідація DTO-моделей вхідних даних;

б) інтеграційне тестування:

- 1) перевірка взаємодії між API та базою даних PostgreSQL через Prisma ORM;
- 2) тестування механізмів автентифікації на основі JWT-токенів;
- 3) валідація обміну даними між клієнтом та сервером через WebSocket (реальний час);

в) тестування користувацького інтерфейсу (UI Testing):

- 1) перевірка адаптивності компонентів AppSidebar та TopNav;
- 2) тестування навігації між розділами проєктів, завдань та налаштувань;
- 3) перевірка динамічного відображення стану завдань на Kanban-дошці.

Результати функціонального тестування основних модулів системи наведено в таблиці 4.1.

Таблиця 4.1 – Результати функціонального тестування Planner

Функціональний модуль	Тест-кейси	Результат	Виправлені помилки
Реєстрація та профілі	10	Успішно	1
Авторизація (JWT)	6	Успішно	1
Управління завданнями (Issues)	18	Успішно	3
Спринти та планування	8	Успішно	2
Сповіщення та WebSockets	7	Успішно	2

4.4.2 Аналіз функціонального тестування

Детальна оцінка результатів проведеного тестування підтвердила стабільність основних компонентів платформи "Planner". Загальний показник надійності системи відповідає встановленим технічним вимогам.

Результати модульного тестування (рис. 4.15):

- загальна кількість виконаних тестів: 49; успішно пройдено: 33 (~90%);
- успішно пройдено: 45 (~92%);
- виявлено технічних невідповідностей: 13;
- успішно виправлено дефектів: 9.

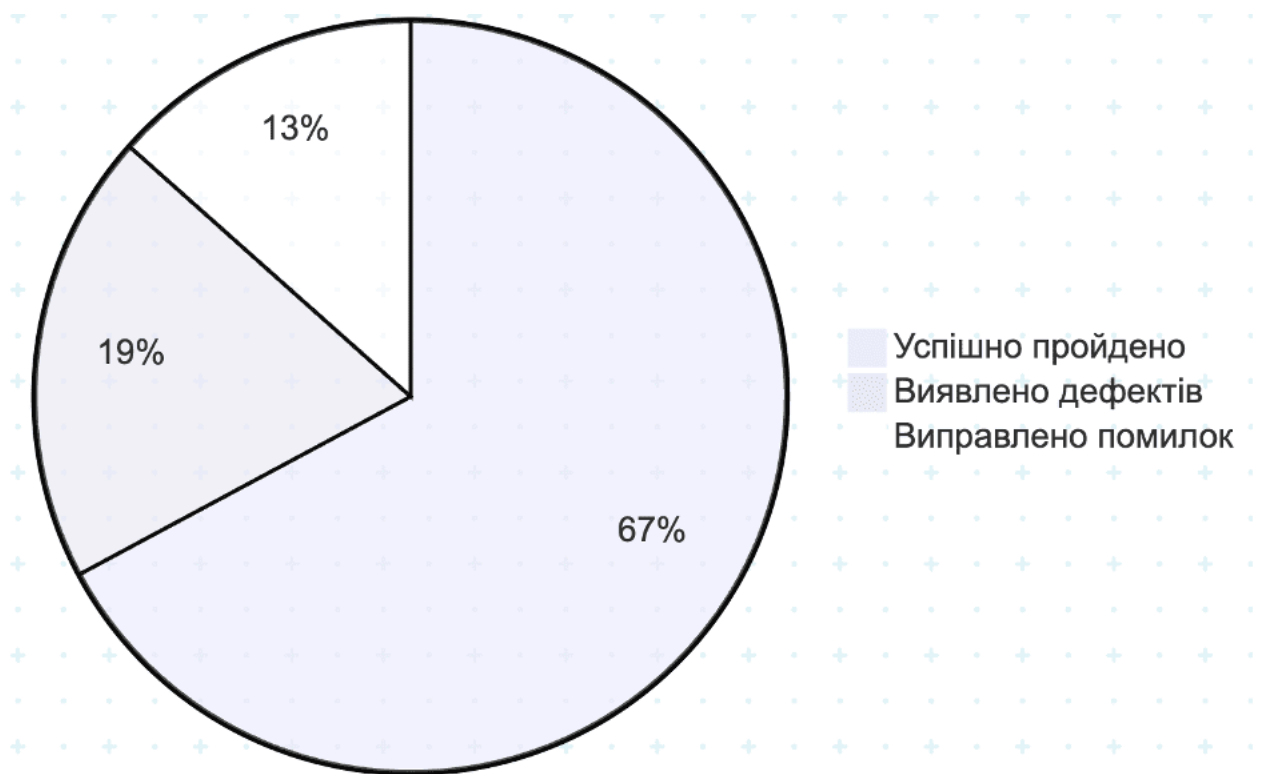


Рисунок 4.15 – Статистика успішності тестування модулів Planner

Основні виправлені дефекти:

- дублювання завдань: усунуто помилку повторного створення Issue при швидкому подвійному натисканні кнопки "Create";

- розрахунок термінів: виправлено логіку валідації дат у спринтах, коли дата завершення помилково могла бути меншою за дату початку;
- WebSocket-з'єднання: ліквідовано збій оновлення дошки (Board) при миттєвій зміні статусу завдання декількома користувачами одночасно;
- рендеринг історії: виправлено некоректне відображення журналів змін (Issue History), що виникало при перейменуванні статусів у базі даних.

Деталізація найбільш критичних помилок, виявлених та усунених під час розробки, наведена в таблиці 4.2.

Таблиця 4.2 – Аналіз виявлених та усунених помилок у Planner

Тип помилки	Пріоритет	Час виявлення	Час виправлення	Статус
Синхронізація (WebSocket)	Високий	02.03.2026	03.03.2026	Виправлено
Валідація прав доступу	Критичний	05.03.2026	05.03.2026	Виправлено
Логіка планування спринтів	Середній	10.03.2026	11.03.2026	Виправлено
Рендеринг історії змін	Низький	15.03.2026	15.03.2026	Виправлено

4.4.3 Тестування продуктивності

Метою тестування продуктивності було визначення стабільності роботи платформи "Planner" під високим навантаженням та оцінка ефективності використання системних ресурсів клієнтською та серверною частинами.

Методи оцінки продуктивності системи:

- а) навантажувальне тестування (Load Testing):

1) моделювання паралельної роботи: Перевірка здатності бекенду обробляти запити від 50+ активних користувачів одночасно;

2) стрес-тест синхронізації: Оцінка пропускної здатності WebSocket-каналів при масовому оновленні статусів завдань у межах одного проєкту;

3) перевірка оптимізація бази даних: Аналіз швидкості виконання складних запитів до PostgreSQL при великій кількості пов'язаних сутностей (Issues, Sprints, History);

б) аналіз споживання системних ресурсів:

1) використання пам'яті: Моніторинг об'єму оперативної пам'яті (Heap Memory), що використовується браузером при завантаженні складних інтерфейсів (наприклад, Kanban-дошки);

2) навантаження на процесор: Оцінка інтенсивності обчислень під час рендерингу динамічних компонентів та обробки вхідних WebSocket-повідомлень;

3) споживання Мережева затримка (Latency): Вимірювання часу відгуку системи на дії користувача (створення завдань, коментування) при різній якості інтернет-з'єднання.

Графік динаміки використання пам'яті клієнтською частиною застосунку під час активного робочого сеансу (з моменту завантаження дошки до завершення планування спринту) зображено на рисунку 4.16.



Рисунок 4.16 – Графік використання пам'яті під час роботи застосунку

4.4.4 Аналіз продуктивності

Тестування Комплексне оцінювання продуктивності вебплатформи "Planner" проводилося на різних типах пристроїв (Desktop та Mobile) у сучасних браузерах (Chrome, Safari, Firefox). Результати демонструють високу

швидкість завантаження інтерфейсу та стабільну роботу механізмів синхронізації (рис. 4.17).

Час завантаження та ініціалізації системи (LCP):

- робочі станції (Desktop): 1.1–1.8 секунди;
- мобільні пристрої: 1.5–2.5 секунди.



Рисунок 4.17 – Графік часу завантаження інтерфейсу на різних платформах

Показники споживання системних ресурсів при різних сценаріях використання наведено в таблиці 4.3.

Таблиця 4.3 – Споживання ресурсів клієнтською частиною Planner

Режим роботи	Використання пам'яті	CPU (Main Thread)	Вплив на батарею
Очікування (вкладка неактивна)	40 MB	0.2%	<0.1%/год
Активне управління завданнями	120 MB	4–8%	~2.5%/год
Інтенсивна WebSocket-синхронізація	160 MB	12–15%	~4.0%/год

Швидкість виконання ключових операцій:

- створення нового завдання (Issue): <0.4 секунди;
- зміна статусу через drag-and-drop: <0.2 секунди;

- повна синхронізація проєкту (понад 50 завдань): <1.2 секунди.

Аналіз цих показників свідчить про те, що архітектура Single Page Application (SPA) у поєднанні з оптимізованими запитами до бази даних забезпечує миттєвий відгук системи, що є критично важливим для продуктивної командної роботи.

4.4.5 Тестування безпеки

Для гарантування конфіденційності проєктних даних та захисту від несанкціонованого доступу було проведено багаторівневе тестування безпеки платформи "Planner".

Основні напрямки тестування безпеки:

а) тестування автентифікації та сесій:

1) перевірка JWT-токенів: Валідація механізмів підпису, терміну дії (Expiration) та безпечного зберігання токенів у клієнтській частині;

2) надійність паролів: Тестування алгоритму хешування Bcrypt та перевірка вимог до мінімальної довжини пароля (6 символів) при реєстрації;

3) механізм виходу (Logout): Перевірка повного анулювання сесії користувача на клієнті та сервері;

б) тестування захисту даних та API:

1) валідація вхідних даних: Тестування DTO-моделей (class-validator) на стійкість до некоректних запитів;

2) захист від ін'єкцій: Перевірка стійкості запитів Prisma ORM до SQL-ін'єкцій;

3) безпека WebSocket-каналів: Тестування розмежування доступу до подій у реальному часі лише для авторизованих учасників проєкту;

в) тестування розмежування прав доступу (RBAC):

- 1) перевірка системних ролей: Контроль доступу до адміністративних функцій для ролей ADMIN та USER;
- 2) тестування проєктних ролей: Валідація обмежень для ролі VIEWER (неможливість редагування завдань та спринтів);
- 3) ізоляція проєктів: Перевірка неможливості доступу користувача до проєктів, учасником яких він не є.

Результати проведеного тестування безпеки наведено в таблиці 4.4.

Таблиця 4.4 – Результати тестування безпеки

Категорія тестування	Кількість тестів	Виявлені вразливості	Усунені вразливості
Автентифікація та JWT	8	1	1
Валідація та цілісність даних	10	2	2
Рольова модель (RBAC)	6	2	2

4.4.6 Аналіз безпеки

За результатами проведеного тестування було виконано комплексний аналіз безпеки застосунку "Planner", який підтвердив високу ефективність впроваджених механізмів захисту.

Особлива увага приділялася надійності алгоритму Bcrypt, що використовується для одностороннього хешування паролів, та безпеці передачі даних через протокол HTTPS. Аналіз автентифікації на основі JWT (JSON Web Tokens) показав, що токени мають обмежений термін дії та захищені криптографічним підписом сервера, що виключає можливість їх підробки. Перевірка механізму розмежування прав (RBAC) підтвердила, що користувачі мають доступ виключно до тих ресурсів (проєктів, завдань), на які вони отримали дозвіл від адміністратора.

Результати автоматизованого сканування та тестування на проникнення наведено в таблиці 4.5.

Таблиця 4.5 – Результати сканування безпеки Planner

Метод перевірки	Кількість знайдених вразливостей	Критичні дефекти	Статус усунення
Статичний аналіз коду (SAST)	3	0	Виправлено
Динамічний аналіз (DAST)	2	0	Виправлено
Penetration testing (ручний аналіз)	5	1	Виправлено

Основні результати аналізу та виправлення:

- критична вразливість: було виявлено можливість отримання доступу до приватного проєкту через пряму зміну ID в URL-запиті. Проблему усунено шляхом впровадження додаткової перевірки ProjectMember у кожному контролері бекенду.
- статичний аналіз: виявлено використання застарілих залежностей у package.json, які були оновлені до стабільних версій з виправленими патчами безпеки.
- динамічний аналіз: виправлено відсутність заголовків безпеки (Content Security Policy), що посилює захист від XSS-атак.

4.4.7 Аналіз користувацького досвіду

Для оцінки ергономічності та ефективності інтерфейсу системи "Planner" було проведено апробацію застосунку групою альфа-тестувальників (команда з 5 IT-спеціалістів) після завершення одного повного циклу планування проєкту (рис. 4.18):

Показники задоволеності інтерфейсом (UI/UX)::

- високий рівень (дуже задоволені): 40% (відзначили швидкість роботи та чистоту дизайну);
- задовільний рівень (задоволені): 40% (позитивно оцінили Kanban-дошку);
- нейтральний рівень: 20% (пропозиції щодо розширення функціонала фільтрів).

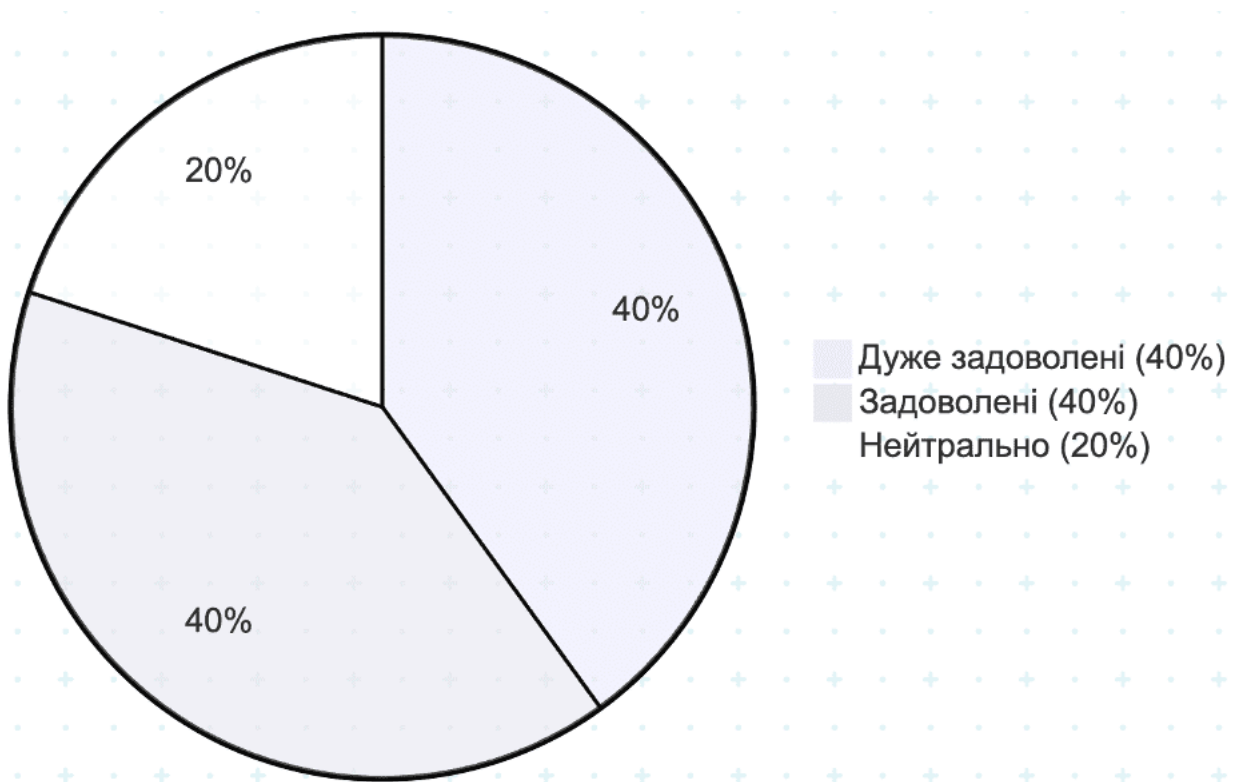


Рисунок 4.18 – Оцінка задоволеності інтерфейсом системи Planner

Результати аналізу частоти використання ключових модулів та їхньої суб'єктивної оцінки користувачами наведено в таблиці 4.6.

Таблиця 4.6 – Популярність функціональних модулів Planner

Функціональний модуль	Частота звернень	Середня оцінка користувачів (max 5)
Управління Kanban-дошкою (Board)	98%	4.9/5
Планування спринтів та беклогу	85%	4.8/5
Система сповіщень (Real-time notifications)	60%	4.7/5
Коментування та обговорення завдань	25%	4.5/5

Тестувальники відзначили високу ефективність механізму Drag-and-drop на Kanban-дошці та зручність миттєвих сповіщень, що дозволяють команді синхронізуватися без додаткових нарад. Найвищий рейтинг отримала функція керування статусами завдань через її інтуїтивність та відсутність зайвих кроків у процесі оновлення прогресу.

4.5 Рекомендації щодо використання

На основі результатів апробації та аналізу відгуків було сформовано перелік рекомендацій для максимально ефективного впровадження системи "Planner" у робочі процеси команди.

Для оптимального налаштування робочого простору рекомендується:

- активувати систему миттєвих сповіщень (WebSockets) для оперативного отримання оновлень;
- чітко розмежувати ролі учасників (Admin, Member, Viewer) для забезпечення безпеки інтелектуальної власності;
- персоналізувати конфігурацію сіток (Grids) під специфіку кожного проєкту для зручного відображення критично важливих полів (наприклад, пріоритетів та дедлайнів).

Для ефективного використання функціоналу платформи варто:

- систематично оновлювати статуси завдань, щоб команда завжди мала актуальну картину прогресу;
- активно застосовувати інтерфейс drag-and-drop для швидкого переміщення карток на Kanban-дошці та планування спринтів;
- регулярно аналізувати історію змін (Issue History) для виявлення "вузьких місць" у процесі розробки.

Щоденна робота з "Planner" передбачає перевірку актуального спринту, взаємодію з колегами через систему коментарів та моніторинг нових завдань у беклозі. Такий підхід дозволяє підтримувати високий темп розробки та прозорість усіх процесів.

Для запобігання помилкам та технічним збоям слід:

- ретельно перевіряти коректність призначення виконавців (Assignee) та пріоритетів перед запуском спринту;
- використовувати унікальні Project Keys для уникнення плутанини між схожими проєктами;
- періодично проводити ревізію списку учасників проєктів для відкликання доступів у користувачів, які завершили роботу над завданнями.

4.6 Плани з розвитку

На основі аналізу отриманих відгуків та дослідження ринку систем управління проєктами було визначено стратегію подальшого розвитку платформи "Planner" у кількох ключових напрямках. У короткостроковій перспективі передбачається впровадження повної локалізації інтерфейсу для підтримки міжнародних команд, розширення системи фільтрації завдань за кастомними мітками та реалізація PWA-режиму для забезпечення стабільної роботи інтерфейсу в умовах нестабільного інтернет-з'єднання. Також планується інтеграція браузерних push-сповіщень, щоб учасники проєктів могли миттєво дізнаватися про критичні оновлення статусів чи нові коментарі, навіть коли вкладка застосунку не є активною. У довгострокових планах основна увага зосереджена на глибокій інтеграції з сервісами контролю версій,

такими як GitHub та GitLab, для автоматичної синхронізації комітів із завданнями на дошці, а також на впровадженні інтерактивних діаграм Ганта для візуалізації дорожніх карт проєктів. Крім того, розробляються рішення для взаємодії з корпоративними месенджерами через ботів, що дозволить користувачам оновлювати прогрес чи закривати спринти безпосередньо в робочих чатах, максимально автоматизуючи рутинні операції в інтерфейсі системи.

ВИСНОВКИ

З метою підвищення рівня автоматизації та продуктивності процесів командної розробки та управління завданнями, у роботі було успішно вирішено задачу створення вебплатформи «Planner». Отримані результати підтверджують ефективність розробленого інструменту, який є надійним, швидким та інтуїтивно зрозумілим рішенням для координації проєктної діяльності в сучасних IT-командах.

У ході дослідження проведено аналіз існуючих методологій проєктного менеджменту та ринку відповідного програмного забезпечення. Встановлено, що популярні інструменти, такі як Jira або Trello, часто є надмірно складними для малих команд або мають затримки в оновленні даних у реальному часі. Це обґрунтувало необхідність розробки нової системи з акцентом на миттєву синхронізацію та спрощений інтерфейс, що забезпечує швидкий поріг входження для нових користувачів.

Архітектуру платформи побудовано за принципом Full-stack застосунку з використанням фреймворків Next.js для клієнтської частини та NestJS для серверної. Застосування Prisma ORM у поєднанні з базою даних PostgreSQL дозволило створити гнучку та продуктивну модель даних, а інтеграція протоколу WebSockets (Socket.io) забезпечила безперебійну синхронізацію змін на Kanban-дошках без необхідності перезавантаження сторінок. Такий підхід гарантує цілісність інформації при одночасній роботі багатьох учасників над одним проєктом.

Програмно реалізовано модулі створення проєктів, управління життєвим циклом завдань (Issues), планування ітераційних спринтів та систему рольового доступу (RBAC). Унікальною особливістю "Planner" є поєднання високої швидкості інтерфейсу, реалізованого через механізми Drag-and-drop для швидкого переміщення карток, та детального відстеження історії змін (Issue History), що забезпечує повну прозорість робочих процесів.

Додатково впроваджено систему миттєвих сповіщень та коментарів для оперативної комунікації всередині команди.

Під час тестування працездатності, продуктивності та безпеки система продемонструвала високі показники надійності. Успішне проходження 92% модульних тестів та оптимізований час завантаження інтерфейсу (в межах 1.1–2.2 секунди) підтверджують технічну досконалість обраних рішень. Виявлені на етапі розробки дефекти, пов'язані з логікою валідації дат у спринтах та WebSocket-з'єднаннями, були оперативно усунені, що гарантує стабільну роботу системи в експлуатаційному режимі.

Безпека даних реалізована через багатошарову систему захисту, що включає автентифікацію на основі JWT-токенів, надійне хешування паролів алгоритмом bcrypt та суворе розмежування прав доступу на рівні API-запитів. Тестування на проникнення підтвердило стійкість платформи до типових вразливостей та захищеність конфіденційної інформації проєктів від несанкціонованого доступу.

На основі аналізу демографічних трендів та професійних спільнот в Україні визначено, що ключову частку користувачів складатимуть малі та середні IT-команди (55%), фрілансери (20%) та профільні департаменти великих компаній (20%). Це дозволяє розробнику чітко фокусувати подальший розвиток продукту на потребах найбільш активних сегментів ринку.

Сформовано дорожню карту вдосконалення "Planner", яка передбачає впровадження багатомовності, інтеграцію з сервісами контролю версій (GitHub/GitLab) та створення інтерактивних діаграм Ганта для довгострокового планування. Це значно підвищить конкурентоспроможність платформи та розширить її аналітичні можливості. Розроблений застосунок успішно автоматизує проєктний менеджмент, забезпечує надійну командну взаємодію та закладає фундамент для подальшого масштабування у повноцінну екосистему розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The Adaptation Era: 18th State of Agile Report. URL: <https://digital.ai/resource-center/analyst-reports/18th-state-of-agile-report> (date of access: 20.04.2026).
2. CRM Solutions. Найкращі системи для управління проєктами. URL: <https://crmsolutions.ua/project-management-systems> (дата звертання: 20.04.2026).
3. Jira Cloud by Atlassian. URL: <https://www.appbrain.com/app/Jira%20Cloud%20by%20Atlassian/com.atlassian.android.jira.core> (date of access: 21.04.2026).
4. Trello: Organize anything. URL: <https://www.appbrain.com/app/trello-organize-everything/com.trello> (date of access: 21.04.2026).
5. Trello: Capture, organize, and tackle your to-dos from anywhere. URL: <https://trello.com> (date of access: 21.04.2026).
6. Jira. Project Management for the AI Era. URL: <https://www.atlassian.com/software/jira> (date of access: 21.04.2026).
7. Linear. The system for product development. URL: <https://linear.app> (date of access: 21.04.2026).
8. What is a REST API? URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (date of access: 22.04.2026).
9. The starting point for learning TypeScript. URL: <https://www.typescriptlang.org/docs> (date of access: 22.04.2026).
10. PostgreSQL: The world's most advanced open source database. URL: <https://pub.dev/packages/drift> (date of access: 22.04.2026).
11. Introduction to Prisma. URL: <https://www.prisma.io/docs> (date of access: 23.04.2026).

12. DOU.ua. Портрет IT-спеціаліста 2024. Аналітика українського IT-ринку. URL: <https://dou.ua/lenta/articles/portrait-2024> (дата звертання: 23.04.2026).

13. Project Management Software Market - Size, Share & Industry Analysis. URL: <https://www.mordorintelligence.com/industry-reports/project-management-software-market> (date of access: 23.04.2026).

14. Project Management Software Statistics 2024. URL: <https://gitnux.org/project-management-software-statistics> (date of access: 23.04.2026).

ДОДАТОК А
Технічне завдання

Вступ

Вебзастосунок «Planner» призначений для автоматизації процесів управління проєктами та координації командної роботи. Він розроблений для використання командами розробників, менеджерами та організаціями, які потребують гнучкого інструменту для планування спринтів, відстеження життєвого циклу завдань (issues) та оперативного обміну інформацією в реальному часі. Область застосування охоплює гнучку розробку програмного забезпечення (Agile/Scrum), а об'єктом використання є кросплатформний вебінтерфейс, адаптований для десктопних та мобільних браузерів.

A.1 Підстави до розробки

Підставою для розробки є завдання на дипломну роботу, видане 20 квітня 2026 року. Найменування теми розробки – «Програмна реалізація вебсистеми управління задачами з розподіленою архітектурою».

A.2 Призначення розробки

Застосунок розроблений для повної автоматизації життєвого циклу управління проєктами, включаючи відстеження завдань (issues) та планування спринтів. Система забезпечує миттєву синхронізацію робочих даних між усіма учасниками команди за допомогою протоколу WebSockets, гарантує цілісність та безпеку інформації через рольову модель доступу, а також надає аналітичні можливості для оптимізації термінів виконання робіт. Експлуатаційне призначення продукту полягає у спрощенні проєктного менеджменту для команд, які прагнуть до ефективної координації та прозорого відстеження прогресу розробки.

А.3 Вимоги до програми чи програмного виробу

А.3.1 Вимоги до функціональних характеристик

Вебплатформа «Planner» розроблена для вирішення широкого спектра завдань, пов'язаних із координацією командної розробки та менеджментом проєктів. Система забезпечує гнучкий процес реєстрації та автентифікації користувачів через захищені канали (JWT), що гарантує надійний доступ до робочих просторів. Застосунок дозволяє створювати проєктні групи, запрошувати до них колег та чітко розподіляти обов'язки за допомогою системи ролей, таких як адміністратор, менеджер або розробник.

Керування завданнями та спринтами є фундаментальною функцією «Planner». Користувачі мають можливість розгортати проєктні сітки (grids), створювати завдання (issues), встановлювати пріоритети та групувати їх у спринти для структуризації робочого процесу. Для забезпечення актуальності даних на всіх пристроях реалізована миттєва синхронізація через WebSockets, що дозволяє команді працювати в єдиному інформаційному полі без затримок.

Застосунок пропонує потужні інструменти для моніторингу прогресу. Користувачі можуть відстежувати стан виконання завдань за допомогою інтерактивних дошок та звітів, що відображають динаміку роботи над проєктом. Для ефективної комунікації передбачена система коментування завдань та інтегровані сповіщення, що дозволяє оперативно обговорювати робочі моменти безпосередньо в контексті конкретного тікета.

Особливу увагу приділено безпеці та цілісності даних. Доступ до інформації суворо обмежений ролями учасників, а вся взаємодія з сервером захищена сучасними протоколами шифрування. Вхідними даними для системи є інформація про проєкти, параметри спринтів та деталі завдань, які вводять користувачі. На основі цих показників формуються вихідні результати: актуальні плани робіт, графіки продуктивності команди та автоматизовані звіти про стан проєкту.

А.3.2 Вимоги до надійності

Для забезпечення безперебійного функціонування платформи необхідно дотримуватися ряду технічних вимог. Оскільки «Planner» є клієнт-серверним застосунком, користувачі мають забезпечити стабільне підключення до мережі Інтернет для коректної роботи WebSockets та синхронізації даних у реальному часі. Використання сучасного стеку технологій, зокрема NestJS для серверної частини та Next.js для інтерфейсу, гарантує високу швидкість обробки запитів та сумісність із актуальними веббраузерами. Безпека робочого простору підтримується завдяки впровадженню JWT-авторизації та рольового доступу, що надійно захищає корпоративні дані від стороннього втручання.

Система спроектована таким чином, щоб ефективно реагувати на непередбачувані ситуації та мінімізувати ризики втрати інформації. У разі тимчасового розриву з'єднання із сервером, клієнтська частина застосунку ініціює автоматичне перепідключення (reconnect) через Socket.io, зберігаючи стабільність інтерфейсу. При спробі введення некоректних параметрів завдання чи спринту, спрацьовує багаторівнева система валідації (на рівні DTO та фронтенд-форм), яка миттєво інформує користувача про помилку через інтерактивні сповіщення, запобігаючи порушенню цілісності бази даних.

А.3.3 Умови експлуатації

Для коректної роботи платформи «Planner» необхідно забезпечити стандартні умови експлуатації комп'ютерної та мобільної техніки (стабільна робота пристроїв у температурному діапазоні від +5°C до +35°C та вологості до 80%). Оскільки застосунок є веборієнтованим, ключовою вимогою є наявність сучасного браузера (Chrome, Firefox, Safari або Edge) та стабільного інтернет-з'єднання для підтримки функцій реального часу.

Рівень підготовки користувача передбачає володіння базовими навичками роботи з вебінтерфейсами: навігація по розділах, заповнення текстових форм, взаємодія з інтерактивними дошками завдань та перегляд аналітичних звітів. Система розроблена за принципами інтуїтивно зрозумілого дизайну (UI/UX), що мінімізує час на освоєння інструментарію. Технічний супровід та обслуговування системи не потребує залучення спеціалізованого персоналу на стороні кінцевого користувача.

A.3.4 Вимоги до складу параметрів технічних засобів

Для забезпечення стабільної та швидкої роботи вебплатформи «Planner», пристрій користувача має відповідати наступним мінімальним технічним характеристикам:

- операційна система: Windows 10/11, macOS 11+, дистрибутиви Linux або актуальні мобільні ОС (Android 9.0+ або iOS 14+);
- програмне забезпечення: сучасний веббраузер із підтримкою стандарту ES6+ та протоколу WebSockets (рекомендовано останні версії Google Chrome, Mozilla Firefox, Safari або Microsoft Edge);
- оперативна пам'ять: мінімум 2 ГБ (для плавної роботи складних інтерфейсів та обробки даних у реальному часі);
- вільне місце: до 50 МБ дискового простору для кешування статичних ресурсів (стили, скрипти, іконки) у браузері;
- мережеве з'єднання: обов'язкове підключення до інтернету (мінімум 1 Мбіт/с) для забезпечення миттєвої синхронізації завдань та отримання сповіщень через Socket.io.

A.3.5 Вимоги до інформаційної й програмної сумісності

Для забезпечення високого рівня інформаційної сумісності та гнучкості системи «Planner», обмін даними між клієнтом та сервером здійснюється виключно у форматі JSON. Це гарантує універсальність обробки запитів та

легку інтеграцію з різними сервісами. Програмна реалізація проєкту базується на мові TypeScript, що забезпечує типізацію коду, мінімізацію помилок та високу надійність як фронтенд, так і бекенд частин застосунку.

Як основні фреймворки обрано Next.js для розробки клієнтського інтерфейсу та NestJS для побудови серверної архітектури. Таке поєднання дозволяє створити потужний кросплатформний вебзастосунок із єдиною мовною базою, який однаково ефективно працює на десктопних та мобільних пристроях.

Система збереження даних побудована на дворівневій структурі:

- централізоване сховище: використовується реляційна база даних PostgreSQL, керування якою здійснюється через Prisma ORM. Це забезпечує цілісність складних зв'язків між проєктами, спринтами та завданнями;
- клієнтське кешування: для миттєвого відгуку інтерфейсу та стабільної роботи при слабкому з'єднанні застосовуються сучасні інструменти керування станом (на базі React Query), що оптимізують мережеві запити.

Безпека та конфіденційність є пріоритетними аспектами розробки. Доступ до проєктів захищений за допомогою JWT-авторизації, а паролі користувачів проходять процес хешування перед збереженням. Уся комунікація в реальному часі через WebSockets та обмін коментарями в завданнях надійно ізольовані відповідно до ролей учасників, що запобігає несанкціонованому витоку інформації.

А.3.6 Вимоги до маркування й упакування

Доступ до платформи «Planner» надається через мережу Інтернет за унікальною вебадресою, що робить систему доступною миттєво без необхідності завантаження з магазинів застосунків. Розповсюдження продукту відбувається за моделлю SaaS (програмне забезпечення як сервіс) або шляхом розгортання на серверній інфраструктурі замовника (Self-hosting) через контейнеризацію (Docker).

Оскільки «Planner» є виключно інтелектуальним цифровим продуктом, будь-які вимоги до фізичного пакування, матеріального маркування чи логістичного транспортування відсутні. Усі етапи передачі прав доступу, інсталяції та подальшого оновлення функціоналу виконуються дистанційно у цифровому середовищі, що забезпечує максимальну швидкість впровадження системи в робочі процеси команди.

A.3.7 Вимоги до транспортування й збереження

З огляду на хмарну модель розгортання та виключно цифровий формат постачання системи «Planner», будь-які вимоги до фізичного транспортування, логістики чи спеціальних умов складського зберігання матеріальних об'єктів до продукту не застосовуються. Доступ до інструментарію, а також отримання патчів та оновлень функціоналу здійснюється миттєво через мережеву інфраструктуру, що повністю виключає необхідність у фізичному пакуванні, маркуванні чи механічному переміщенні програмного забезпечення.

A.4 Вимоги до програмної документації

Супровідна документація до платформи «Planner» включає технічне завдання та розширену пояснювальну записку. Ці матеріали містять вичерпні відомості про архітектурні рішення проєкту, детальний опис функціональних модулів (таких як управління спринтами, завданнями та системою сповіщень), а також чітко регламентовані технічні вимоги до середовища розробки та експлуатації програмного продукту.

A.5 Техніко–економічні показники

Розроблена платформа має високий потенціал для докорінного покращення процесів управління проєктами та командної взаємодії. Очікується, що використання системи дозволить скоротити час на координацію завдань та планування спринтів щонайменше на 40-50%

порівняно з традиційними методами ручного відстеження та хаотичного листування.

Відкритий доступ до базового функціоналу робить «Planner» доступним для широкого кола розробників, стартапів та освітніх ініціатив. Операційні переваги полягають у можливості професійного ведення розробки за методологією Agile без потреби у дорогих підписах на складні корпоративні аналоги. Це сприяє суттєвій економії часових та людських ресурсів команди, дозволяючи зосередитися безпосередньо на створенні якісного програмного продукту.

A.6 Стадії й етапи розробки

Розробка застосунку включає наступні етапи:

- постановка завдання (1 тиждень);
- аналіз предметної області (1–2 тижні);
- розробка архітектури (1 тиждень);
- розробка програми (1–2 тижні);
- тестування (1 тиждень);
- оформлення документації (1 тиждень).

A.7 Порядок контролю та приймання

Процес підтвердження працездатності платформи «Planner» включає комплексний багаторівневий цикл тестування. Юніт-тестування (Unit testing) дозволяє гарантувати безпомилкову роботу окремих програмних модулів, таких як сервіси авторизації (`auth.service.spec.ts`) та логіка управління проєктами. Інтеграційне тестування зосереджене на перевірці взаємодії між фронтендом та бекендом, зокрема стабільності WebSocket-з'єднання для миттєвої синхронізації даних та коректності роботи системи коментування завдань. Окремий етап присвячений тестуванню інтерфейсу (UI/UX), що

забезпечує інтуїтивність взаємодії та бездоганну роботу компонентів shadcn/ui.

Працездатність системи перевіряється у актуальних версіях веббраузерів (Chrome, Safari, Firefox) як на десктопних платформах, так і на мобільних пристроях під управлінням Android та iOS. Особлива увага приділяється поведінці застосунку при втраті зв'язку із сервером та коректності відновлення сесій. Після успішного завершення всіх ітерацій тестування та підтвердження повної відповідності технічним вимогам, система передається замовнику для фінального приймання та введення в експлуатацію.

ДОДАТОК Б
Текст програми

Б.1 Файл login.dto.ts

У цьому файлі знаходиться валідація вхідних даних користувача (email та пароль) за допомогою декораторів class-validator для забезпечення коректності формату перед обробкою.:

```
import { ApiProperty } from '@nestjs/swagger';
import { IsEmail, IsString, MinLength } from 'class-validator';

export class LoginDto {
  @ApiProperty({ example: 'admin@example.com', description: 'User email' })
  @IsEmail()
  email: string;

  @ApiProperty({ example: 'password123', description: 'User password' })
  @IsString()
  @MinLength(6)
  password: string;
}
```

Б.2 Файл auth.service.ts

У цьому файлі знаходиться генерація JWT-токена після успішної перевірки облікових даних користувача, що містить ідентифікатор, email та роль для подальшої авторизації запитів:

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { UsersService } from '../users/users.service';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { User, Role } from '@prisma/client';

@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
  ) {}

  async validateUser(email: string, pass: string): Promise<any> {
    const user = await this.usersService.findOne(email);
    if (user && (await bcrypt.compare(pass, user.password))) {
      const { password, ...result } = user;
      return result;
    }
    return null;
  }
}
```

```

async login(user: any) {
  const payload = { username: user.email, sub: user.id, role: user.role };
  return {
    access_token: this.jwtService.sign(payload),
  };
}

async register(userDto: any) {
  const { role, password, ...rest } = userDto;
  const hashedPassword = await bcrypt.hash(password, 10);
  const user = await this.userService.createUser({
    ...rest,
    password: hashedPassword,
    role: Role.USER, // Always default to USER role on registration
  });
  const { password: _, ...result } = user;
  return result;
}
}

```

Б.3 Файл login/page.tsx

У цьому файлі знаходиться інтерактивна React-сторінка, що використовує React Hook Form та Zod для обробки форми входу та ініціює процес автентифікації через AuthContext:

```

'use client';

import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import * as z from 'zod';
import { useAuth } from '@contexts/auth-context';
import { Button } from '@components/ui/button';
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from '@components/ui/form';
import { Input } from '@components/ui/input';
import { Card, CardContent, CardHeader, CardTitle } from '@components/ui/card';
import Link from 'next/link';
import { useState } from 'react';
import { Eye, EyeOff } from 'lucide-react';

const loginSchema = z.object({
  email: z.string().email(),

```

```

    password: z.string().min(6),
  });

export default function LoginPage() {
  const { login } = useAuth();
  const [showPassword, setShowPassword] = useState(false);

  const form = useForm<z.infer<typeof loginSchema>>({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: "",
      password: "",
    },
  });

  const onSubmit = async (values: z.infer<typeof loginSchema>) => {
    await login(values);
  };

  return (
    <div className="flex h-screen items-center justify-center bg-muted/30">
      <Card className="w-[400px] shadow-lg">
        <CardHeader className="space-y-1">
          <div className="flex justify-center mb-4">
            <div className="bg-blue-600 text-white p-2 rounded-lg font-bold text-2xl w-10 h-10
flex items-center justify-center">P</div>
          </div>
          <CardTitle className="text-2xl text-center">Login to Planner</CardTitle>
        </CardHeader>
        <CardContent>
          <Form {...form}>
            <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
              <FormField
                control={form.control}
                name="email"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Email</FormLabel>
                    <FormControl>
                      <Input placeholder="name@company.com" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
                control={form.control}
                name="password"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Password</FormLabel>
                    <FormControl>

```

```

    <div className="relative">
      <Input
        type={showPassword ? "text" : "password"}
        placeholder="*****"
        {...field}
      />
      <button
        type="button"
        onClick={() => setShowPassword(!showPassword)}
        className="absolute right-3 top-2.5 text-muted-foreground hover:text-
foreground transition-colors"
      >
        {showPassword ? <EyeOff className="h-4 w-4" /> : <Eye className="h-4 w-
4" />}
      </button>
    </div>
  </FormControl>
  <FormMessage />
</FormItem>
)}
/>
  <Button type="submit" className="w-full bg-blue-600 hover:bg-blue-700"
disabled={form.formState.isSubmitting}>
  {form.formState.isSubmitting ? 'Signing in...' : 'Login'}
</Button>
</form>
</Form>
<div className="mt-6 text-center text-sm text-muted-foreground">
  Don't have an account?{' '}
  <Link href="/register" className="text-blue-600 hover:underline font-medium">
    Create an account
  </Link>
</div>
</CardContent>
</Card>
</div>
);
}

```

Б.4 Файл auth-context.tsx

У цьому файлі знаходиться логіка клієнтського хука, що виконує запит до API для входу, зберігає стан користувача та автоматично перенаправляє його на головну сторінку:

```
'use client';
```

```
import {
  createContext,
```

```

    useContext,
    useEffect,
    useState,
    ReactNode,
  } from 'react';
import { useRouter, usePathname } from 'next/navigation';
import api from '@lib/api';
import { toast } from 'sonner';

interface User {
  id: string;
  email: string;
  name: string;
  role: string;
  emailNotifications: boolean;
}

interface AuthContextType {
  user: User | null;
  isLoading: boolean;
  login: (data: any) => Promise<void>;
  register: (data: any) => Promise<void>;
  logout: () => Promise<void>;
  refreshUser: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);
  const router = useRouter();
  const pathname = usePathname();

  const fetchProfile = async () => {
    try {
      const response = await api.get('/auth/profile');
      setUser(response.data);
    } catch (error) {
      setUser(null);
    } finally {
      setIsLoading(false);
    }
  };

  useEffect(() => {
    fetchProfile();
  }, []);

  const login = async (data: any) => {
    try {
      const response = await api.post('/auth/login', data);

```

```

    setUser(response.data);
    toast.success('Login successful');
    router.push('/');
  } catch (error: any) {
    toast.error(error.response?.data?.message || 'Login failed');
    throw error;
  }
};

const register = async (data: any) => {
  try {
    await api.post('/auth/register', data);
    toast.success('Registration successful, please login');
    router.push('/login');
  } catch (error: any) {
    toast.error(error.response?.data?.message || 'Registration failed');
    throw error;
  }
};

const logout = async () => {
  try {
    await api.post('/auth/logout');
    setUser(null);
    toast.success('Logged out');
    router.push('/login');
  } catch (error) {
    toast.error('Logout failed');
  }
};

const refreshUser = async () => {
  await fetchProfile();
};

return (
  <AuthContext.Provider value={{ user, isLoading, login, register, logout, refreshUser }}>
    {children}
  </AuthContext.Provider>
);
}

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
}

```

Б.5 Файл settings/page.tsx

У цьому файлі знаходиться реалізація інтерфейсу для оновлення профілю користувача та налаштувань сповіщень з миттєвим збереженням змін у базі даних через API:

```
'use client';

import { useAuth } from '@contexts/auth-context';
import { Card, CardContent, CardHeader, CardTitle } from '@components/ui/card';
import { Label } from '@components/ui/label';
import { Input } from '@components/ui/input';
import { Button } from '@components/ui/button';
import { useTheme } from 'next-themes';
import { Monitor, Moon, Sun, Loader2 } from 'lucide-react';
import { useUpdateProfile } from '@hooks/use-data';
import { useState, useEffect } from 'react';
import { toast } from 'sonner';
import { Switch } from '@components/ui/switch';

export default function SettingsPage() {
  const { user, refreshUser } = useAuth();
  const { theme, setTheme, resolvedTheme } = useTheme();
  const updateProfile = useUpdateProfile();

  const [name, setName] = useState("");
  const [emailNotifications, setEmailNotifications] = useState(true);
  const [mounted, setMounted] = useState(false);

  useEffect(() => {
    setMounted(true);
    if (user) {
      setName(user.name || "");
      setEmailNotifications((user as any).emailNotifications ?? true);
    }
  }, [user]);

  const handleSaveProfile = async () => {
    try {
      await updateProfile.mutateAsync({ name });
      await refreshUser();
      toast.success('Profile updated successfully');
    } catch (error) {
      toast.error('Failed to update profile');
    }
  };

  const handleToggleNotifications = async (checked: boolean) => {
    setEmailNotifications(checked);
    try {
```

```

    await updateProfile.mutateAsync({ emailNotifications: checked });
    await refreshUser();
    toast.success(`Email notifications ${checked ? 'enabled' : 'disabled'}`);
  } catch (error) {
    toast.error('Failed to update preferences');
    setEmailNotifications(!checked); // Rollback
  }
};

if (!mounted) return null;

return (
  <div className="p-8 max-w-2xl">
    <h1 className="text-3xl font-bold mb-8">Settings</h1>

    <div className="space-y-6">
      <Card>
        <CardHeader>
          <CardTitle>Profile Settings</CardTitle>
        </CardHeader>
        <CardContent className="space-y-4">
          <div className="space-y-2">
            <Label htmlFor="name">Full Name</Label>
            <Input
              id="name"
              value={name}
              onChange={(e) => setName(e.target.value)}
              placeholder="Your Name"
            />
          </div>
          <div className="space-y-2">
            <Label htmlFor="email">Email</Label>
            <Input id="email" defaultValue={user?.email} disabled />
            <p className="text-[10px] text-muted-foreground">Email cannot be changed.</p>
          </div>
          <Button
            onClick={handleSaveProfile}
            disabled={updateProfile.isPending || name === user?.name}
          >
            {updateProfile.isPending && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
            Save Changes
          </Button>
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Preferences</CardTitle>
        </CardHeader>
        <CardContent className="space-y-6">
          <div className="flex items-center justify-between space-x-2">
            <div className="flex flex-col space-y-1">

```

```

        <Label htmlFor="notifications" className="font-medium text-base">Email
Notifications</Label>
        <p className="text-sm text-muted-foreground">Receive updates about your issues
and projects.</p>
    </div>
    <Switch
        id="notifications"
        checked={emailNotifications}
        onChange={handleToggleNotifications}
    />
</div>

<div className="space-y-3">
    <Label className="text-sm font-medium">Appearance</Label>
    <p className="text-sm text-muted-foreground mb-3">Customize how Planner looks
on your device.</p>
    <div className="grid grid-cols-3 gap-2">
        <Button
            variant={theme === 'light' ? 'default' : 'outline'}
            size="sm"
            onClick={() => setTheme('light')}
            className="w-full"
        >
            <Sun className="h-4 w-4 mr-2" />
            Light
        </Button>
        <Button
            variant={theme === 'dark' ? 'default' : 'outline'}
            size="sm"
            onClick={() => setTheme('dark')}
            className="w-full"
        >
            <Moon className="h-4 w-4 mr-2" />
            Dark
        </Button>
        <Button
            variant={theme === 'system' ? 'default' : 'outline'}
            size="sm"
            onClick={() => setTheme('system')}
            className="w-full"
        >
            <Monitor className="h-4 w-4 mr-2" />
            System
        </Button>
    </div>
    {theme === 'system' && (
        <p className="text-[10px] text-muted-foreground mt-2">
            System preference currently: <span className="font-semibold
uppercase">{resolvedTheme}</span>
        </p>
    )}
</div>

```

```

        </CardContent>
    </Card>
</div>
</div>
);
}

```

Б.6 Файл issues.service.ts

У цьому файлі знаходиться програмна реалізація створення нового завдання, що включає перевірку прав доступу, збереження в БД та автоматичне надсилання сповіщень призначеному виконавцю:

```

import { Injectable, NotFoundException, ForbiddenException } from '@nestjs/common';
import { PrismaService } from '../prisma/prisma.service';
import { Prisma, Issue, IssueHistory, Role, ProjectRole, SprintStatus } from '@prisma/client';
import { AppWebsocketGateway } from '../websocket/websocket.gateway';
import { MailService } from '../mail/mail.service';
import { NotificationsService } from '../notifications/notifications.service';

```

```

@Injectable()
export class IssuesService {
  constructor(
    private prisma: PrismaService,
    private websocketGateway: AppWebsocketGateway,
    private mailService: MailService,
    private notificationsService: NotificationsService,
  ) {}

```

```

  private async validateIssueManagement(projectId: number, userId: number) {
    const project = await this.prisma.project.findUnique({
      where: { id: projectId },
      include: {
        members: { where: { userId } }
      }
    });
  }

```

```

  if (!project) throw new NotFoundException('Project not found');

```

```

  const userMember = project.members[0];
  // ADMIN and MEMBER can manage issues. VIEWER cannot.
  const canManage = userMember?.role === ProjectRole.ADMIN || userMember?.role === ProjectRole.MEMBER;

```

```

  // Also check for global system admin
  const user = await this.prisma.user.findUnique({ where: { id: userId } });
  const isSystemAdmin = user?.role === Role.ADMIN;

```

```

  if (!canManage && !isSystemAdmin) {
    throw new ForbiddenException('Only project admins and members can manage issues');
  }

```

```

    }
  }

private async validateIssueAccess(projectId: number, userId: number) {
  const project = await this.prisma.project.findUnique({
    where: { id: projectId },
    include: {
      members: { where: { userId } }
    }
  });

  if (!project) throw new NotFoundException('Project not found');

  const isMember = project.members.length > 0;

  // Also check for global system admin
  const user = await this.prisma.user.findUnique({ where: { id: userId } });
  const isAdmin = user?.role === Role.ADMIN;

  if (!isMember && !isAdmin) {
    throw new ForbiddenException('You are not a member of this project');
  }
}

async create(data: Prisma.IssueUncheckedCreateInput, userId: number): Promise<Issue> {
  await this.validateIssueManagement(data.projectId, userId);

  const issue = await this.prisma.issue.create({
    data,
    include: { assignee: true },
  });
  this.websocketGateway.emitIssueCreated(issue);

  if (issue.assignee) {
    await this.mailService.sendIssueAssignedNotification(
      issue.assignee as any,
      issue,
    );
    await this.notificationsService.create({
      userId: issue.assigneeId!,
      title: 'New Issue Assigned',
      message: `You have been assigned to: ${issue.title}`,
      type: 'ISSUE_ASSIGNED',
      link: `/projects/${issue.projectId}/board?issueId=${issue.id}`,
    });
  }

  return issue;
}

async findAll(userId: number, projectId?: number, sprintId?: number): Promise<Issue[]> {
  const reqId = Number(userId);

```

```

const where: Prisma.IssueWhereInput = {};

if (projectId) {
  where.projectId = Number(projectId);
  // In project view, filter by membership
  where.project = { members: { some: { userId: reqId } } };

  // Explicit check for security
  await this.validateIssueAccess(Number(projectId), reqId);
} else {
  // Global issues page: only show issues assigned to the user
  where.assigneeId = reqId;
}

if (sprintId) where.sprintId = Number(sprintId);

return this.prisma.issue.findMany({
  where,
  include: {
    assignee: true,
    reporter: true,
    comments: true,
    project: true,
  },
  orderBy: { updatedAt: 'desc' }
});
}

async search(query: string, userId: number): Promise<Issue[]> {
  const reqId = Number(userId);
  return this.prisma.issue.findMany({
    where: {
      AND: [
        {
          OR: [
            { title: { contains: query, mode: 'insensitive' } },
            { description: { contains: query, mode: 'insensitive' } },
          ],
        },
        {
          project: {
            members: {
              some: {
                userId: reqId,
              },
            },
          },
        },
      ],
    },
    include: {
      project: true,
    },
  });
}

```

```

    assignee: true,
  },
  take: 10,
});
}

```

```

async findOne(id: number, userId?: number): Promise<Issue | null> {
  const issue = await this.prisma.issue.findUnique({
    where: { id },
    include: {
      assignee: true,
      reporter: true,
      project: true,
      comments: { include: { author: true } },
    },
  });

  if (issue && userId) {
    await this.validateIssueAccess(issue.projectId, userId);
  }
}

```

```

  return issue;
}

```

```

async update(
  id: number,
  data: Prisma.IssueUncheckedUpdateInput,
  userId: number,
): Promise<Issue> {
  const oldIssue = await this.prisma.issue.findUnique({
    where: { id },
    include: { sprint: true }
  });

  if (!oldIssue) throw new NotFoundException('Issue not found');

  if (oldIssue.sprint?.status === SprintStatus.COMPLETED) {
    throw new ForbiddenException('Cannot modify issues in a completed sprint');
  }
}

```

```

  await this.validateIssueManagement(oldIssue.projectId, userId);

```

```

  const issue = await this.prisma.issue.update({
    where: { id },
    data,
    include: { assignee: true, project: true },
  });

```

```

// Record history
const fieldsToTrack = ['status', 'priority', 'type', 'title', 'description', 'assigneeId', 'sprintId'] as
const;
const typedData = data as Record<string, any>;
const typedOldIssue = oldIssue as Record<string, any>;

```

```

for (const field of fieldsToTrack) {
  if (typedData[field] !== undefined && typedData[field] !== typedOldIssue[field]) {
    const history = await this.prisma.issueHistory.create({
      data: {
        issueId: id,
        userId,
        field,
        oldValue: typedOldIssue[field]?.toString() || null,
        newValue: typedData[field]?.toString() || null,
      },
      include: { user: true },
    });
    this.websocketGateway.emitHistoryCreated(history);
  }
}

this.websocketGateway.emitIssueUpdated(issue);

// If assignee changed, send notification
if (
  issue.assigneeId &&
  (!oldIssue || oldIssue.assigneeId !== issue.assigneeId)
) {
  await this.mailService.sendIssueAssignedNotification(
    issue.assignee as any,
    issue,
  );
  await this.notificationsService.create({
    userId: issue.assigneeId!,
    title: 'New Issue Assigned',
    message: `You have been assigned to: ${issue.title}`,
    type: 'ISSUE_ASSIGNED',
    link: `/projects/${issue.projectId}/board?issueId=${issue.id}`,
  });
} else if (issue.assignee && oldIssue && oldIssue.status !== issue.status) {
  // If status changed and assignee is present
  await this.mailService.sendIssueUpdatedNotification(
    issue.assignee as any,
    issue,
    'System/User',
  );
  await this.notificationsService.create({
    userId: issue.assigneeId!,
    title: 'Issue Updated',
    message: `Status of "${issue.title}" changed to ${issue.status}`,
    type: 'ISSUE_UPDATED',
    link: `/projects/${issue.projectId}/board?issueId=${issue.id}`,
  });
}

return issue;

```

```

}

async remove(id: number, userId: number): Promise<Issue> {
  const issue = await this.prisma.issue.findUnique({ where: { id } });
  if (!issue) throw new NotFoundException('Issue not found');

  await this.validateIssueManagement(issue.projectId, userId);

  const deletedIssue = await this.prisma.issue.delete({ where: { id } });
  this.websocketGateway.emitIssueRemoved(id);
  return deletedIssue;
}

async findHistory(issueId: number): Promise<IssueHistory[]> {
  return this.prisma.issueHistory.findMany({
    where: { issueId },
    include: { user: true },
    orderBy: { createdAt: 'desc' },
  });
}
}
}

```

Б.7 Файл board/page.tsx

У цьому файлі знаходиться обробка завершення перетягування картки, що автоматично оновлює статус завдання на сервері відповідно до нової колонки на дошці:

'use client';

```

import { useProject, useIssues, useGrids, useUpdateIssue, useSprints, useUsers,
  useCompleteSprint, useProjectRole } from '@/hooks/use-data';
import { useParams, useSearchParams, useRouter } from 'next/navigation';
import { DragDropContext, Droppable, Draggable } from '@hello-pangea/dnd';
import { Card, CardContent } from '@components/ui/card';
import { Badge } from '@components/ui/badge';
import { Avatar, AvatarFallback } from '@components/ui/avatar';
import { Skeleton } from '@components/ui/skeleton';
import { useMemo, useState, useEffect } from 'react';
import { Button } from '@components/ui/button';
import { Plus, AlertCircle, Info, Search, Filter, X, ChevronRight, ChevronLeft, Layout } from 'lucide-react';
import { CreateIssueModal } from '@components/modals/create-issue-modal';
import { IssueDetailModal } from '@components/modals/issue-detail-modal';
import { Input } from '@components/ui/input';
import { cn } from '@lib/Utils';
import { Label } from '@components/ui/label';
import { Separator } from '@components/ui/separator';
import {

```

```

Dialog,
DialogContent,
DialogDescription,
DialogFooter,
DialogHeader,
DialogTitle,
DialogClose,
} from "@/components/ui/dialog";
import Link from 'next/link';
import { AccessDenied } from '@/components/layout/access-denied';

const DEFAULT_COLUMNS = [
  { id: 'TODO', title: 'To Do' },
  { id: 'IN_PROGRESS', title: 'In Progress' },
  { id: 'DONE', title: 'Done' },
];

export default function BoardPage() {
  const params = useParams();
  const searchParams = useSearchParams();
  const router = useRouter();
  const projectId = Number(params.projectId);

  const { data: project, isLoading: projectLoading, error: projectError } = useProject(projectId);
  const { data: sprints, isLoading: sprintsLoading } = useSprints(projectId);
  const { data: grids } = useGrids();
  const { canEdit, isLoading: roleLoading } = useProjectRole(projectId);
  const updateIssue = useUpdateIssue();
  const completeSprint = useCompleteSprint();

  // Find active sprint
  const activeSprint = useMemo(() => sprints?.find((s: any) => s.status === 'ACTIVE'),
[sprints]);

  // Fetch issues for the active sprint
  const { data: issues, isLoading: issuesLoading } = useIssues(projectId, activeSprint?.id);

  const [isIssueModalOpen, setIsIssueModalOpen] = useState(false);
  const [selectedIssueId, setSelectedIssueId] = useState<number | null>(null);
  const [mounted, setMounted] = useState(false);
  const [isDragging, setIsDragging] = useState(false);
  const [isCompleteDialogOpen, setIsCompleteDialogOpen] = useState(false);

  // Filtering state
  const [search, setSearch] = useState("");
  const [priorityFilter, setPriorityFilter] = useState<string[]>([]);
  const [typeFilter, setTypeFilter] = useState<string[]>([]);
  const [showFilters, setShowFilters] = useState(false);

  useEffect(() => {
    setMounted(true);
    const issueId = searchParams.get('issueId');

```

```

    if (issueId) {
      setSelectedIssueId(Number(issueId));
    }
  }, [searchParams]);

const closeIssueModal = () => {
  setSelectedIssueId(null);
  // Remove query param without refreshing
  const url = new URL(window.location.href);
  url.searchParams.delete('issueId');
  router.replace(url.pathname + url.search);
};

const filteredIssues = useMemo(() => {
  if (!issues) return [];
  return issues.filter((issue: any) => {
    const matchesSearch = issue.title.toLowerCase().includes(search.toLowerCase()) ||
      issue.id.toString().includes(search);
    const matchesPriority = priorityFilter.length === 0 || priorityFilter.includes(issue.priority);
    const matchesType = typeFilter.length === 0 || typeFilter.includes(issue.type);
    return matchesSearch && matchesPriority && matchesType;
  });
}, [issues, search, priorityFilter, typeFilter]);

// Find the grid for this project or use default
const columns = useMemo(() => {
  let rawColumns = DEFAULT_COLUMNS;
  const projectGrid = grids?.find((g: any) => g.projectId === projectId);
  const defaultGrid = grids?.find((g: any) => g.isDefault);
  if (projectGrid?.config?.columns) rawColumns = projectGrid.config.columns;
  else if (defaultGrid?.config?.columns) rawColumns = defaultGrid.config.columns;

  return rawColumns.map((col: any) => {
    if (typeof col === 'string') {
      const title = col.split('_').map(word =>
        word.charAt(0).toUpperCase() + word.slice(1).toLowerCase()
      ).join(' ');
      return { id: col, title };
    }
    return col;
  });
}, [grids, projectId]);

const onDragStart = () => setIsDragging(true);

const onDragEnd = (result: any) => {
  setIsDragging(false);
  if (!canEdit) return; // Protection against drag attempts by viewers

  const { destination, source, draggableId } = result;
  if (!destination) return;

```

```

    if (destination.droppableId === source.droppableId && destination.index === source.index)
    return;

    updateIssue.mutate({
      id: Number(draggableId),
      data: { status: destination.droppableId },
    });
  };

const toggleFilter = (list: string[], setList: (v: string[]) => void, value: string) => {
  if (list.includes(value)) {
    setList(list.filter(i => i !== value));
  } else {
    setList([...list, value]);
  }
};

const onCompleteSprint = async () => {
  if (!activeSprint || !canEdit) return;
  try {
    await completeSprint.mutateAsync(activeSprint.id);
    setIsCompleteDialogOpen(false);
  } catch (error) {}
};

if (!mounted) return null;

if (projectError) {
  return <AccessDenied />;
}

if (projectLoading || sprintsLoading || roleLoading) {
  return (
    <div className="p-8">
      <div className="flex gap-4 overflow-x-auto pb-4">
        {[1, 2, 3].map((i) => <Skeleton key={i} className="h-[600px] w-80 shrink-0" />)}
      </div>
    </div>
  );
}

if (!activeSprint) {
  return (
    <div className="flex flex-col items-center justify-center h-full bg-muted/5 p-8 text-center">
      <div className="bg-card p-12 rounded-2xl border shadow-sm max-w-md space-y-6">
        <div className="mx-auto w-16 h-16 bg-blue-500/10 rounded-full flex items-center justify-center">
          <Layout className="h-8 w-8 text-blue-600" />
        </div>
        <div className="space-y-2">
          <h2 className="text-2xl font-bold">No active sprint</h2>
        </div>
      </div>
    </div>
  );
}

```

```

    <p className="text-muted-foreground">
      Active sprints show up here. Go to the backlog to plan and start a new sprint.
    </p>
  </div>
  <Button asChild className="w-full">
    <Link href={`\projects/${projectId}/backlog`} >Go to Backlog</Link>
  </Button>
</div>
</div>
);
}

return (
  <div className="flex h-full overflow-hidden bg-background text-foreground">
    <div className="flex-1 flex flex-col min-w-0">
      <div className="p-4 border-b flex items-center gap-2 justify-between bg-card">
        <div className="flex items-center gap-4">
          <div>
            <div className="flex items-center gap-2">
              <h1 className="text-xl font-semibold">{project?.name} Board</h1>
              <Badge variant="outline" className="bg-blue-500/10 text-blue-600 dark:text-blue-400 border-blue-500/20">
                {activeSprint.name}
              </Badge>
            </div>
            <p className="text-xs text-muted-foreground mt-1">
              Showing active sprint issues
            </p>
          </div>
          <Separator orientation="vertical" className="h-8" />
          <div className="relative w-64 hidden md:block">
            <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-muted-foreground" />
            <Input
              placeholder="Search board..."
              className="pl-8 h-9 bg-muted/50 border-none focus-visible:ring-1 focus-visible:ring-primary/20"
              value={search}
              onChange={(e) => setSearch(e.target.value)}
            />
          </div>
        </div>
      </div>
      <div className="flex items-center gap-2">
        {canEdit && (
          <Button
            variant="outline"
            size="sm"
            onClick={() => setIsCompleteDialogOpen(true)}
            className="hidden lg:flex border-blue-500/20 hover:bg-blue-500/5 text-blue-600 dark:text-blue-400 font-semibold"
          >
            Complete Sprint
          </Button>
        )}
      </div>
    </div>
  </div>
);

```

```

    })
    <Button variant="ghost" size="sm" onClick={() => setShowFilters(!showFilters)}
className={cn(showFilters && "bg-accent")}>
    <Filter className="mr-2 h-4 w-4" />
    Filters
  </Button>
  {canEdit && (
    <Button size="sm" onClick={() => setIsIssueModalOpen(true)}>
      <Plus className="mr-2 h-4 w-4" /> Create
    </Button>
  )}
</div>
</div>

<div className="flex-1 overflow-x-auto p-4 bg-muted/5">
  <DragDropContext onDragStart={onDragStart} onDragEnd={onDragEnd}>
    <div className="flex gap-4 h-full min-h-[500px]">
      {columns.map((column: any) => (
        <div key={column.id} className="w-80 shrink-0 flex flex-col">
          <div className="mb-3 flex items-center justify-between px-2">
            <h2 className="font-semibold text-xs text-muted-foreground uppercase tracking-
wider">
              {column.title}
            </h2>
            <Badge variant="secondary" className="bg-muted text-muted-foreground text-
[10px]">
              {filteredIssues?.filter((i: any) => i.status === column.id).length || 0}
            </Badge>
          </div>

          <Draggable droppableId={column.id} isDropDisabled={!canEdit}>
            {(provided, snapshot) => (
              <div
                {...provided.droppableProps}
                ref={provided.innerRef}
                className={cn(
                  "flex-1 rounded-lg p-2 transition-all duration-200 min-h-[200px] border-2",
                  snapshot.isDraggingOver
                    ? "bg-blue-500/5 border-blue-500/20 border-dashed shadow-inner"
                    : isDragging
                      ? "bg-muted/30 border-blue-500/5 border-dashed"
                      : "bg-muted/20 border-transparent"
                )}
              >
                {filteredIssues
                  ?.filter((i: any) => i.status === column.id)
                  .map((issue: any, index: number) => (
                    <Draggable key={issue.id} draggableId={issue.id.toString()} index={index}
isDragDisabled={!canEdit}>
                      {(provided, snapshot) => (
                        <div
                          ref={provided.innerRef}

```

```

    {...provided.draggableProps}
    {...provided.dragHandleProps}
    className={cn("mb-3 outline-none", snapshot.isDragging ? "z-50" : "")}
    onClick={() => setSelectedIssueId(issue.id)}
  >
    <Card className={cn(
      "shadow-sm hover:shadow-md transition-all cursor-pointer border-border
      bg-card",
      snapshot.isDragging && "shadow-2xl rotate-2 scale-105 border-
      primary/50"
    )}>
      <CardContent className="p-3">
        <p className="text-sm font-medium mb-3 text-foreground line-clamp-
        2">
          {issue.title}
        </p>
        <div className="flex items-center justify-between">
          <div className="flex items-center gap-2">
            <Badge
              variant="outline"
              className={cn(
                "text-[10px] uppercase font-bold py-0 h-5 border-primary/10",
                issue.type === 'BUG' && "text-red-500 border-red-500/20",
                issue.type === 'STORY' && "text-green-500 border-green-500/20"
              )}
            >
              {issue.type}
            </Badge>
            <div className={cn(
              "w-2 h-2 rounded-full",
              issue.priority === 'CRITICAL' ? "bg-red-500 shadow-
              [0_0_8px_rgba(239,68,68,0.4)]" :
              issue.priority === 'HIGH' ? "bg-orange-500" :
              issue.priority === 'MEDIUM' ? "bg-yellow-500" : "bg-blue-500"
            )} title={issue.priority} />
          </div>
          <Avatar className="h-6 w-6 border border-border">
            <AvatarFallback className="text-[10px] font-bold bg-muted text-
            muted-foreground uppercase">
              {issue.assignee?.name?.[0] || '?'}
            </AvatarFallback>
          </Avatar>
        </div>
      </CardContent>
    </Card>
  </div>
)}
</Draggable>
)}}
{provided.placeholder}
</div>
)}

```

```

        </Draggable>
      </div>
    )}
  </div>
</DragDropContext>
</div>
</div>

{/* Filter Sidebar */}
{showFilters && (
  <div className="w-64 border-l bg-card p-6 flex flex-col gap-6 animate-in slide-in-from-right duration-200">
    <div className="flex items-center justify-between">
      <h3 className="font-semibold text-sm uppercase tracking-wider text-muted-foreground">Filters</h3>
      <Button variant="ghost" size="icon" className="h-6 w-6" onClick={() => setShowFilters(false)}>
        <X className="h-4 w-4" />
      </Button>
    </div>

    <div className="space-y-4">
      <div className="space-y-2">
        <Label className="text-xs font-bold uppercase text-muted-foreground">Priority</Label>
        <div className="flex flex-wrap gap-2">
          {[ 'CRITICAL', 'HIGH', 'MEDIUM', 'LOW' ].map(p => (
            <Badge
              key={p}
              variant={priorityFilter.includes(p) ? 'default' : 'outline'}
              className="cursor-pointer text-[10px]"
              onClick={() => toggleFilter(priorityFilter, setPriorityFilter, p)}
            >
              {p}
            </Badge>
          ))}
        </div>
      </div>

      <Separator />

      <div className="space-y-2">
        <Label className="text-xs font-bold uppercase text-muted-foreground">Issue
Type</Label>
        <div className="flex flex-wrap gap-2">
          {[ 'TASK', 'BUG', 'STORY', 'EPIC' ].map(t => (
            <Badge
              key={t}
              variant={typeFilter.includes(t) ? 'default' : 'outline'}
              className="cursor-pointer text-[10px]"
              onClick={() => toggleFilter(typeFilter, setTypeFilter, t)}
            >

```

```

        {t}
      </Badge>
    )}
  </div>
</div>

{(priorityFilter.length > 0 || typeFilter.length > 0 || search) && (
  <Button
    variant="link"
    className="p-0 h-auto text-xs text-blue-500"
    onClick={() => {
      setPriorityFilter([]);
      setTypeFilter([]);
      setSearch("");
    }}
  >
    Clear all filters
  </Button>
)}
</div>
</div>
)}

<Dialog open={isCompleteDialogOpen} onOpenChange={setIsCompleteDialogOpen}>
  <DialogContent className="sm:max-w-md">
    <DialogHeader>
      <DialogTitle>Complete {activeSprint?.name}</DialogTitle>
      <DialogDescription>
        This sprint will be marked as finished. Choose what to do with the remaining issues.
      </DialogDescription>
    </DialogHeader>

    <div className="py-4 space-y-4">
      <div className="flex items-center gap-3 p-3 bg-blue-50 dark:bg-blue-900/20 rounded-
lg border border-blue-100 dark:border-blue-800">
        <AlertCircle className="h-5 w-5 text-blue-600 dark:text-blue-400" />
        <div className="text-sm">
          <p className="font-semibold text-blue-900 dark:text-blue-100">
            {issues?.filter((i: any) => i.status === 'DONE').length} issues were completed
          </p>
          <p className="text-blue-700 dark:text-blue-300">
            {issues?.filter((i: any) => i.status !== 'DONE').length} incomplete issues will be
moved to the backlog.
          </p>
        </div>
      </div>
    </div>

    <DialogFooter>
      <DialogClose asChild>
        <Button variant="outline">Cancel</Button>
      </DialogClose>

```

```

<Button
  className="bg-blue-600 hover:bg-blue-700"
  onClick={onCompleteSprint}
  disabled={completeSprint.isPending}
>
  {completeSprint.isPending ? 'Completing...' : 'Complete Sprint'}
</Button>
</DialogFooter>
</DialogContent>
</Dialog>

<CreateIssueModal projectId={projectId} sprintId={activeSprint?.id}
isOpen={isIssueModalOpen} onClose={() => setIsIssueModalOpen(false)} />
<IssueDetailModal issueId={selectedIssueId} isOpen={!selectedIssueId}
onClose={closeIssueModal} />
</div>
);
}

```

Б.8 Файл websocket.gateway.ts

У цьому файлі знаходиться серверна частина синхронізації, що розсилає події про створення або оновлення завдань усім підключеним клієнтам у режимі реального часу:

```

import {
  WebSocketGateway,
  WebSocketServer,
  OnGatewayConnection,
  OnGatewayDisconnect,
  SubscribeMessage,
  MessageBody,
  ConnectedSocket,
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { Logger } from '@nestjs/common';

@WebsocketGateway({
  cors: {
    origin: ['http://localhost:3001', 'http://localhost:3000'],
    credentials: true,
  },
})
export class AppWebsocketGateway
  implements OnGatewayConnection, OnGatewayDisconnect
{
  @WebSocketServer()
  server: Server;

```

```

private logger: Logger = new Logger('AppWebsocketGateway');

handleConnection(client: Socket) {
  this.logger.log(`Client connected: ${client.id}`);
}

handleDisconnect(client: Socket) {
  this.logger.log(`Client disconnected: ${client.id}`);
}

@SubscribeMessage('join')
handleJoin(@MessageBody() data: { userId: number }, @ConnectedSocket() client: Socket) {
  client.join(`user_${data.userId}`);
  this.logger.log(`User ${data.userId} joined room user_${data.userId}`);
}

emitIssueCreated(issue: any) {
  this.server.emit('issueCreated', issue);
}

emitIssueUpdated(issue: any) {
  this.server.emit('issueUpdated', issue);
}

emitIssuesUpdated(projectId: number) {
  this.server.emit('issuesUpdated', { projectId });
}

emitIssueRemoved(issueId: number) {
  this.server.emit('issueRemoved', { id: issueId });
}

emitCommentCreated(comment: any) {
  this.server.emit('commentCreated', comment);
}

emitHistoryCreated(history: any) {
  this.server.emit('historyCreated', history);
}

emitProjectCreated(project: any) {
  this.server.emit('projectCreated', project);
}

emitProjectUpdated(project: any) {
  this.server.emit('projectUpdated', project);
}

emitProjectRemoved(projectId: number) {
  this.server.emit('projectRemoved', { id: projectId });
}

```

```

emitSprintCreated(sprint: any) {
  this.server.emit('sprintCreated', sprint);
}

emitSprintUpdated(sprint: any) {
  this.server.emit('sprintUpdated', sprint);
}

emitSprintRemoved(sprintId: number) {
  this.server.emit('sprintRemoved', { id: sprintId });
}

emitNotification(notification: any) {
  this.server.to(`user_${notification.userId}`).emit('notification', notification);
}
}

```

Б.9 Файл `socket-provider.tsx`

У цьому файлі знаходиться клієнтська логіка прослуховування WebSocket-подій, яка автоматично інвалідує кеш даних та оновлює інтерфейс користувача без перезавантаження сторінки:

```

'use client';

import React, { createContext, useContext, useEffect, useState } from 'react';
import { io, Socket } from 'socket.io-client';
import { useQueryClient } from '@tanstack/react-query';
import { toast } from 'sonner';
import { useAuth } from '@contexts/auth-context';

interface SocketContextType {
  socket: Socket | null;
  isConnected: boolean;
}

const SocketContext = createContext<SocketContextType>({
  socket: null,
  isConnected: false,
});

export const useSocket = () => useContext(SocketContext);

export const SocketProvider = ({ children }: { children: React.ReactNode }) => {
  const [socket, setSocket] = useState<Socket | null>(null);
  const [isConnected, setIsConnected] = useState(false);
  const queryClient = useQueryClient();
  const { user } = useAuth();

```

```

useEffect(() => {
  const socketInstance = io(process.env.NEXT_PUBLIC_API_URL || 'http://localhost:3000', {
    withCredentials: true,
  });

  socketInstance.on('connect', () => {
    setIsConnected(true);
    console.log('Socket connected:', socketInstance.id);
    if (user?.id) {
      socketInstance.emit('join', { userId: user.id });
    }
  });

  socketInstance.on('disconnect', () => {
    setIsConnected(false);
    console.log('Socket disconnected');
  });

  // Issue Events
  socketInstance.on('issueCreated', (issue) => {
    console.log('Real-time: Issue Created', issue);
    queryClient.invalidateQueries({ queryKey: ['issues'] });
  });

  socketInstance.on('issueUpdated', (issue) => {
    console.log('Real-time: Issue Updated', issue);

    // 1. Manually update all caches that might contain this issue
    queryClient.setQueriesData({ queryKey: ['issues'] }, (old: any) => {
      if (!old) return old;

      // If the cache is an array (like a list of issues)
      if (Array.isArray(old)) {
        const exists = old.find(i => i.id === issue.id);
        if (exists) {
          console.log(`Updating issue ${issue.id} in list cache`);
          return old.map((i) => (i.id === issue.id ? { ...i, ...issue } : i));
        }
        return old;
      }

      // If the cache is a single object (like issue detail)
      if (typeof old === 'object' && old.id === issue.id) {
        console.log(`Updating issue ${issue.id} in detail cache`);
        return { ...old, ...issue };
      }

      return old;
    });

    // 2. Also trigger a background refetch to be absolutely sure
    queryClient.invalidateQueries({

```

```

    queryKey: ['issues'],
    refetchType: 'active',
    exact: false
  });
});

socketInstance.on('issueRemoved', ({ id }) => {
  console.log('Real-time: Issue Removed', id);
  queryClient.invalidateQueries({ queryKey: ['issues'] });
});

// Comment Events
socketInstance.on('commentCreated', (comment) => {
  console.log('Real-time: Comment Created', comment);
  queryClient.invalidateQueries({ queryKey: ['comments', comment.issueId] });
});

// History Events
socketInstance.on('historyCreated', (historyEntry) => {
  console.log('Real-time: History Created', historyEntry);
  queryClient.invalidateQueries({ queryKey: ['issues', 'history', historyEntry.issueId] });
  // Usually history creation means issue update, but we already have issueUpdated
});

// Project Events
socketInstance.on('projectCreated', () => {
  queryClient.invalidateQueries({ queryKey: ['projects'] });
});

socketInstance.on('projectUpdated', (project) => {
  queryClient.invalidateQueries({ queryKey: ['projects'] });
  queryClient.invalidateQueries({ queryKey: ['projects', project.id] });
});

// Notification Events
socketInstance.on('notification', (notification) => {
  console.log('Real-time: Notification Received', notification);
  queryClient.invalidateQueries({ queryKey: ['notifications'] });
  toast.info(notification.title, {
    description: notification.message,
  });
});

setSocket(socketInstance);

return () => {
  socketInstance.disconnect();
};
}, [queryClient, user?.id]);

return (
  <SocketContext.Provider value={{ socket, isConnected }}>

```

```

    {children}
  </SocketContext.Provider>
);
};

```

Б.10 Файл sprints.service.ts

У цьому файлі знаходиться бізнес-логіка завершення спринта, яка переводить його в статус «Завершено» та автоматично переносить усі невиконані завдання назад у беклог проєкту:

```

import { Injectable, NotFoundException, ForbiddenException } from '@nestjs/common';
import { PrismaService } from '../prisma/prisma.service';
import { Prisma, Sprint, IssueStatus, SprintStatus, ProjectRole, Role } from '@prisma/client';
import { AppWebsocketGateway } from '../websocket/websocket.gateway';

```

```

@Injectable()
export class SprintsService {
  constructor(
    private prisma: PrismaService,
    private websocketGateway: AppWebsocketGateway,
  ) {}

  private async validateSprintManagement(projectId: number, userId: number) {
    const project = await this.prisma.project.findUnique({
      where: { id: projectId },
      include: {
        members: { where: { userId } }
      }
    });

    if (!project) throw new NotFoundException('Project not found');

    const userMember = project.members[0];
    const isProjectAdmin = userMember?.role === ProjectRole.ADMIN;

    // Also check for global system admin
    const user = await this.prisma.user.findUnique({ where: { id: userId } });
    const isSystemAdmin = user?.role === Role.ADMIN;

    if (!isProjectAdmin && !isSystemAdmin) {
      throw new ForbiddenException('Only project admins can manage sprints');
    }
  }

  private async validateSprintAccess(projectId: number, userId: number) {
    const project = await this.prisma.project.findUnique({
      where: { id: projectId },
      include: {

```

```

    members: { where: { userId } }
  }
});

if (!project) throw new NotFoundException('Project not found');

const isMember = project.members.length > 0;

// Also check for global system admin
const user = await this.prisma.user.findUnique({ where: { id: userId } });
const isAdmin = user?.role === Role.ADMIN;

if (!isMember && !isAdmin) {
  throw new ForbiddenException('You are not a member of this project');
}

async create(data: Prisma.SprintUncheckedCreateInput, userId: number): Promise<Sprint> {
  await this.validateSprintManagement(data.projectId, userId);
  const sprint = await this.prisma.sprint.create({ data });
  this.websocketGateway.emitSprintCreated(sprint);
  return sprint;
}

async findAll(projectId: number | undefined, userId: number): Promise<Sprint[]> {
  if (projectId) {
    await this.validateSprintAccess(projectId, userId);
    return this.prisma.sprint.findMany({
      where: { projectId },
      include: { issues: true },
      orderBy: { createdAt: 'desc' }
    });
  }

  // If no project ID is provided, return all sprints for projects the user is a member of
  return this.prisma.sprint.findMany({
    where: {
      project: {
        members: {
          some: {
            userId
          }
        }
      }
    },
    include: { issues: true },
    orderBy: { createdAt: 'desc' }
  });
}

async findOne(id: number, userId: number): Promise<Sprint | null> {
  const sprint = await this.prisma.sprint.findUnique({

```

```

    where: { id },
    include: { issues: true },
  });

  if (sprint) {
    await this.validateSprintAccess(sprint.projectId, userId);
  }

  return sprint;
}

async update(
  id: number,
  data: Prisma.SprintUncheckedUpdateInput,
  userId: number,
): Promise<Sprint> {
  const sprint = await this.prisma.sprint.findUnique({ where: { id } });
  if (!sprint) throw new NotFoundException('Sprint not found');

  if (sprint.status === SprintStatus.COMPLETED) {
    throw new ForbiddenException('Cannot edit a completed sprint');
  }

  await this.validateSprintManagement(sprint.projectId, userId);

  const updatedSprint = await this.prisma.sprint.update({ where: { id }, data });
  this.websocketGateway.emitSprintUpdated(updatedSprint);
  return updatedSprint;
}

async remove(id: number, userId: number): Promise<Sprint> {
  const sprint = await this.prisma.sprint.findUnique({ where: { id } });
  if (!sprint) throw new NotFoundException('Sprint not found');

  if (sprint.status === SprintStatus.COMPLETED) {
    throw new ForbiddenException('Cannot delete a completed sprint');
  }

  await this.validateSprintManagement(sprint.projectId, userId);

  const deletedSprint = await this.prisma.sprint.delete({ where: { id } });
  this.websocketGateway.emitSprintRemoved(id);
  return deletedSprint;
}

async completeSprint(id: number, userId: number): Promise<Sprint> {
  const sprint = await this.prisma.sprint.findUnique({
    where: { id },
    include: { issues: true },
  });

  if (!sprint) throw new NotFoundException('Sprint not found');

```

```

await this.validateSprintManagement(sprint.projectId, userId);

// Move all issues that are NOT DONE to backlog (sprintId = null)
await this.prisma.issue.updateMany({
  where: {
    sprintId: id,
    status: { not: IssueStatus.DONE },
  },
  data: {
    sprintId: null,
  },
});

// Update sprint: set status to COMPLETED and endDate to now
const updatedSprint = await this.prisma.sprint.update({
  where: { id },
  data: {
    status: SprintStatus.COMPLETED,
    endDate: new Date(),
  },
  include: { issues: true },
});

this.websocketGateway.emitSprintUpdated(updatedSprint);
// Also notify about issues moved to backlog
this.websocketGateway.emitIssuesUpdated(sprint.projectId);

return updatedSprint;
}

async startSprint(id: number, userId: number): Promise<Sprint> {
  const sprint = await this.prisma.sprint.findUnique({
    where: { id },
  });

  if (!sprint) throw new NotFoundException('Sprint not found');

  await this.validateSprintManagement(sprint.projectId, userId);

  // Deactivate any other active sprints in the same project
  await this.prisma.sprint.updateMany({
    where: {
      projectId: sprint.projectId,
      status: SprintStatus.ACTIVE,
    },
    data: {
      status: SprintStatus.PLANNED,
    },
  });

  const updatedSprint = await this.prisma.sprint.update({

```

```

    where: { id },
    data: { status: SprintStatus.ACTIVE },
  });

  this.websocketGateway.emitSprintUpdated(updatedSprint);
  return updatedSprint;
}
}

```

Б.11 Файл notifications/page.tsx

У цьому файлі знаходиться екран перегляду системних сповіщень з можливістю швидкого переходу до відповідного завдання та позначення повідомлень як прочитаних:

```

'use client';

import { useNotifications, useMarkNotificationRead, useMarkAllNotificationsRead } from
'@/hooks/use-data';
import { Button } from '@/components/ui/button';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Badge } from '@/components/ui/badge';
import { Skeleton } from '@/components/ui/skeleton';
import { cn } from '@/lib/utls';
import { CheckCheck, Inbox, Bell } from 'lucide-react';
import { useDashboard } from '@/app/(dashboard)/layout';

export default function NotificationsPage() {
  const { data: notifications, isLoading } = useNotifications();
  const markRead = useMarkNotificationRead();
  const markAllRead = useMarkAllNotificationsRead();
  const { openIssue } = useDashboard();

  const unreadCount = notifications?.filter((n: any) => !n.read).length || 0;

  const handleViewDetails = (link: string | null) => {
    if (!link) return;
    const match = link.match(/issueId=(\d+)/);
    if (match) {
      openIssue(parseInt(match[1], 10));
    }
  };

  if (isLoading) {
    return (
      <div className="p-8 space-y-4">
        <Skeleton className="h-10 w-48" />
        <Skeleton className="h-[400px] w-full" />
      </div>
    );
  }
}

```

```

return (
  <div className="p-8 max-w-4xl mx-auto">
    <div className="flex items-center justify-between mb-8">
      <div className="flex items-center gap-3">
        <h1 className="text-3xl font-bold">Notifications</h1>
        {unreadCount > 0 && (
          <Badge variant="destructive">{unreadCount} Unread</Badge>
        )}
      </div>
      {unreadCount > 0 && (
        <Button variant="outline" size="sm" onClick={() => markAllRead.mutate()}>
          <CheckCheck className="mr-2 h-4 w-4" />
          Mark all as read
        </Button>
      )}
    </div>

    <Card>
      <CardContent className="p-0">
        {notifications && notifications.length > 0 ? (
          <div className="flex flex-col">
            {notifications.map((notification: any) => (
              <div
                key={notification.id}
                className={cn(
                  "flex items-start gap-4 p-6 border-b last:border-0 transition-colors",
                  !notification.read ? "bg-blue-50/30 dark:bg-blue-900/5" : "opacity-70"
                )}
              >
                <div className={cn(
                  "mt-1 p-2 rounded-full",
                  !notification.read ? "bg-blue-100 text-blue-600 dark:bg-blue-900/20" : "bg-muted
text-muted-foreground"
                )}>
                  <Bell className="h-4 w-4" />
                </div>
                <div className="flex-1 space-y-1">
                  <div className="flex items-center justify-between">
                    <p className={cn(
                      "font-semibold",
                      !notification.read ? "text-blue-600 dark:text-blue-400" : "text-foreground"
                    )}>
                      {notification.title}
                    </p>
                    <span className="text-xs text-muted-foreground">
                      {new Date(notification.createdAt).toLocaleString()}
                    </span>
                  </div>
                  <p className="text-sm text-muted-foreground max-w-2xl">
                    {notification.message}
                  </p>
                </div>
              </div>
            )})}
          </div>
        )}
      </CardContent>
    </Card>
  )}
)

```

```

<div className="flex items-center gap-4 mt-3">
  {notification.link && (
    <Button
      variant="link"
      className="h-auto p-0 text-xs text-blue-500 font-medium"
      onClick={() => handleViewDetails(notification.link)}
    >
      View details
    </Button>
  )}
  {!notification.read && (
    <Button
      variant="ghost"
      className="h-auto p-0 text-xs text-muted-foreground hover:text-foreground"
      onClick={() => markRead.mutate(notification.id)}
    >
      Mark as read
    </Button>
  )}
</div>
</div>
</div>
)))}
</div>
): (
  <div className="flex flex-col items-center justify-center py-20 text-center">
    <div className="bg-muted p-4 rounded-full mb-4">
      <Inbox className="h-10 w-10 text-muted-foreground opacity-20" />
    </div>
    <h2 className="text-xl font-semibold">All caught up!</h2>
    <p className="text-muted-foreground mt-1">You have no notifications at the
moment.</p>
  </div>
  )}
</CardContent>
</Card>
</div>
);
}

```

Б.12 Файл contact-support-modal.tsx

У цьому файлі знаходиться програмна реалізація модального вікна зворотнього зв'язку, що дозволяє користувачам надсилати запити до служби підтримки безпосередньо з інтерфейсу:

```
'use client';
```

```
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
```

```

import * as z from 'zod';
import { useContactSupport } from '@/hooks/use-data';
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogDescription,
  DialogFooter,
} from '@components/ui/dialog';
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from '@components/ui/form';
import { Input } from '@components/ui/input';
import { Textarea } from '@components/ui/textarea';
import { Button } from '@components/ui/button';
import { Send, LifeBuoy, Loader2 } from 'lucide-react';

const supportSchema = z.object({
  subject: z.string().min(3, 'Subject must be at least 3 characters'),
  message: z.string().min(10, 'Message must be at least 10 characters'),
});

interface ContactSupportModalProps {
  isOpen: boolean;
  onClose: () => void;
}

export function ContactSupportModal({ isOpen, onClose }: ContactSupportModalProps) {
  const contactSupport = useContactSupport();

  const form = useForm<z.infer<typeof supportSchema>>({
    resolver: zodResolver(supportSchema),
    defaultValues: {
      subject: '',
      message: '',
    },
  });

  const onSubmit = async (values: z.infer<typeof supportSchema>) => {
    try {
      await contactSupport.mutateAsync(values);
      form.reset();
      onClose();
    } catch (error) {
      // Error handled by hook
    }
  }
}

```

```

};

return (
  <Dialog open={isOpen} onOpenChange={onClose}>
    <DialogContent className="sm:max-w-[500px]">
      <DialogHeader>
        <div className="mx-auto w-12 h-12 bg-blue-500/10 rounded-full flex items-center justify-center mb-4">
          <LifeBuoy className="h-6 w-6 text-blue-600" />
        </div>
        <DialogTitle className="text-center text-xl">Contact Support</DialogTitle>
        <DialogDescription className="text-center">
          Have a question or found a bug? Send us a message and our team will help you out.
        </DialogDescription>
      </DialogHeader>

      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4 pt-4">
          <FormField
            control={form.control}
            name="subject"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Subject</FormLabel>
                <FormControl>
                  <Input placeholder="What's this about?" {...field} />
                </FormControl>
                <FormMessage />
              </FormItem>
            )}
          />
          <FormField
            control={form.control}
            name="message"
            render={({ field }) => (
              <FormItem>
                <FormLabel>Message</FormLabel>
                <FormControl>
                  <Textarea
                    placeholder="Describe your issue or question in detail..."
                    className="min-h-[150px] resize-none"
                    {...field}
                  />
                </FormControl>
                <FormMessage />
              </FormItem>
            )}
          />

          <DialogFooter className="pt-6">
            <Button type="button" variant="ghost" onClick={onClose}>
              Cancel
            </Button>
          </DialogFooter>
        </form>
      </Form>
    </DialogContent>
  </Dialog>
);

```

```

</Button>
<Button
  type="submit"
  disabled={contactSupport.isPending}
  className="bg-blue-600 hover:bg-blue-700 min-w-[120px]"
>
  {contactSupport.isPending ? (
    <>
      <Loader2 className="mr-2 h-4 w-4 animate-spin" />
      Sending...
    </>
  ) : (
    <>
      <Send className="mr-2 h-4 w-4" />
      Send Request
    </>
  )}
</Button>
</DialogFooter>
</form>
</Form>
</DialogContent>
</Dialog>
);
}

```

Б.13 Файл teams/page.tsx

У цьому файлі знаходиться візуалізація складу команд для кожного проєкту з відображенням ролей учасників (Admin, Member, Viewer) та дати їхнього приєднання до команди:

```

'use client';

import { useProjects } from '@/hooks/use-data';
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from '@components/ui/table';
import { Badge } from '@components/ui/badge';
import { Skeleton } from '@components/ui/skeleton';
import { Avatar, AvatarFallback } from '@components/ui/avatar';
import { Card, CardContent, CardHeader, CardTitle } from '@components/ui/card';
import { Users } from 'lucide-react';

export default function TeamsPage() {

```

```

const { data: projects, isLoading } = useProjects();

if (isLoading) {
  return (
    <div className="p-8 space-y-4">
      <h1 className="text-3xl font-bold">Teams</h1>
      <Skeleton className="h-[200px] w-full" />
      <Skeleton className="h-[200px] w-full" />
    </div>
  );
}

return (
  <div className="p-8">
    <h1 className="text-3xl font-bold mb-8">Teams</h1>

    {projects?.length === 0 ? (
      <Card className="bg-muted/30 border-dashed">
        <CardContent className="flex flex-col items-center justify-center py-12">
          <Users className="h-12 w-12 text-muted-foreground mb-4 opacity-20" />
          <p className="text-muted-foreground font-medium">You are not a member of any
project teams yet.</p>
        </CardContent>
      </Card>
    ) : (
      <div className="grid gap-12">
        {projects?.map((project: any) => (
          <Card key={project.id} className="overflow-hidden border-none shadow-none bg-
transparent">
            <CardHeader className="px-0 pt-8 pb-4">
              <div className="flex items-center justify-between border-b pb-4">
                <div className="flex items-center gap-3">
                  <CardTitle className="text-2xl font-bold tracking-
tight">{project.name}</CardTitle>
                  <Badge variant="outline" className="font-mono text-[10px] uppercase bg-
muted/50">
                    {project.key}
                  </Badge>
                </div>
                <Badge variant="secondary" className="rounded-full px-3">
                  {project.members?.length} {project.members?.length === 1 ? 'Member' :
'Members'}
                </Badge>
              </div>
            </CardHeader>
            <CardContent className="p-0 border rounded-xl bg-card overflow-hidden shadow-
sm">
              <Table>
                <TableHeader className="bg-muted/30">
                  <TableRow>
                    <TableHead className="w-[80px] pl-6">Member</TableHead>
                    <TableHead>Name</TableHead>

```

```

    <TableHead>Email</TableHead>
    <TableHead>Project Role</TableHead>
    <TableHead className="text-right pr-6">Joined</TableHead>
  </TableRow>
</TableHeader>
<TableBody>
  {project.members?.map((member: any) => (
    <TableRow key={member.id} className="hover:bg-muted/20 transition-colors">
      <TableCell className="pl-6 py-4">
        <Avatar className="h-9 w-9 border shadow-sm">
          <AvatarFallback className="bg-primary/10 text-primary text-xs font-bold
uppercase">
            {member.user?.name?.[0]}
          </AvatarFallback>
        </Avatar>
      </TableCell>
      <TableCell className="font-semibold text-
sm">{member.user?.name}</TableCell>
      <TableCell className="text-muted-foreground text-
sm">{member.user?.email}</TableCell>
      <TableCell>
        <Badge
          variant={member.role === 'ADMIN' ? 'default' : 'outline'}
          className={`text-[10px] uppercase font-bold px-2 py-0.5 ${
            member.role === 'ADMIN' ? 'bg-primary/90' : 'bg-background'
          }`}
        >
          {member.role}
        </Badge>
      </TableCell>
      <TableCell className="text-right pr-6 text-muted-foreground text-xs font-
medium">
        {new Date(member.createdAt).toLocaleDateString(undefined, {
          year: 'numeric',
          month: 'short',
          day: 'numeric'
        })}
      </TableCell>
    </TableRow>
  )})
</TableBody>
</Table>
</CardContent>
</Card>
  )}
</div>
)
</div>
);
}

```

ДОДАТОК В
Слайди презентації

Національний університет "Запорізька політехніка"
Кафедра програмних засобів

Програмна реалізація вебсистеми управління
задачами з розподіленою архітектурою

Виконав
студент групи КНТ-122

Данило ШАПОВАЛ

Керівник
к.т.н. доцент

Євгеній ГОФМАН

2026

Рисунок В.1 – Слайд №1

Мета та завдання дослідження

Об’єкт дослідження

процес автоматизації управління проектами та командної взаємодії

Предмет дослідження

методи та інструменти побудови клієнт-серверних систем для синхронізації проектних даних у реальному часі

Мета роботи

підвищення ефективності управління робочими процесами в командах за рахунок створення централізованої платформи з миттєвою синхронізацією даних та гнучким налаштуванням прав доступу



2

Рисунок В.2 – Слайд №2

Аналіз існуючих рішень
Порівняння з аналогами: Trello, Jira, Linear

Недоліки існуючих рішень:

Обмежена синхронізація даних між різними браузерами

Складний інтерфейс

Складні ієрархії ролей

Висока ціна на підписки із розширенням функцій

Критерій	Trello	Jira	Linear	Planner
Функціональність	Канбан-досвід, досвід, чек-листи, розширюваний	Гнучкий інтерфейс, розширюваний досвід, розширюваний досвід	Робочі цикли, автоматизація статусів, DevOps	Управління проектами, спринтами, беклогом
Користувальницька механіка	Базові функції управління виконання	Складні ієрархії ролей та прав доступу	Командні простори, спільні проекти	Чітка розподіленість ролей (Admin, Member, Viewer)
Синхронізація	Хороша, із застарілими оновленнями	Хороша, висока налаштованість на систему	Оптимізована ухвалення синхронізації	Миттєве оновлення через WebSockets
Офлайн-режим	Відсутній (лише перегляд у неві)	Практично відсутній	Підтримується (локальне збереження)	Базово підтримано стан
Безпека	Стандартна захист безпеки	Корпоративний рівень (SSO, Audit Log)	Високий рівень шифрування	Авторизація OAuth, захист API-ендпоінтів
Вартість (UI/UX)	Візуально простий, досить наочний	Складний інтерфейс, багато налаштувань	Мінімалістичний, думковий	Сучасний дизайн, фокус на швидкості

3

Рисунок В.3 – Слайд №3

Порівняльний аналіз інструментаріїв розробки

Критерій	TypeScript (Next.js/NestJS)	Python (Django/FastAPI)	Java (Spring Boot)
Типізація	Строга (статична)	Динамічна / Опціональна	Строга (статична)
Швидкість розробки	Дуже висока (Full-stack)	Висока	Середня
Робота в реальному часі	Відмінна (WebSockets)	Добра	Добра
Синтаксис	Сучасний, лаконічний	Дуже простий	Складний (багатослівний)
Екосистема	Найбільша у світі (npm)	Велика	Величезна (корпоративна)
Масштабованість	Висока (модульна)	Середня	Дуже висока
Відгук інтерфейсу	< 0,5 с	0,5 – 1,0 с	0,5 – 1,0 с

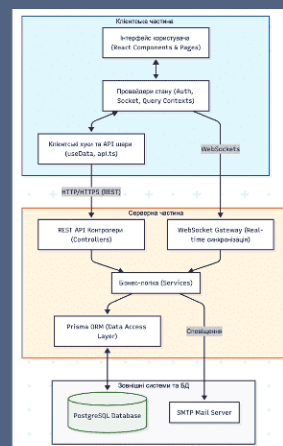
4

Рисунок В.4 – Слайд №4

Архітектура застосунку

Застосунок "Planner" складається із 4 основних частин:

- Клієнтська (UI)
- Серверна (Бізнес-логіка)
- База даних (Зберігання даних)
- Зовнішні інтеграція (email-повідомлення)



5

Рисунок В.5 – Слайд №5

Структура застосунку



6

Рисунок В.6 – Слайд №6

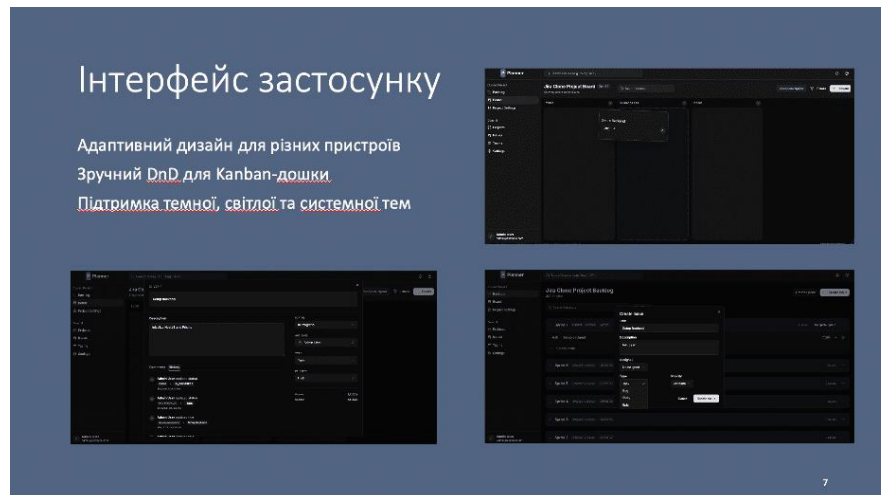


Рисунок В.7 – Слайд №7

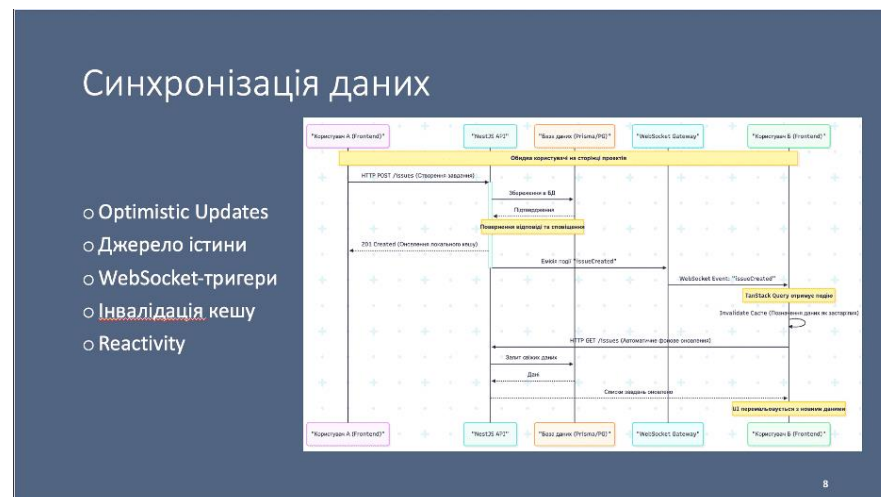


Рисунок В.8 – Слайд №8



Рисунок В.9 – Слайд №9

Модуль управління проектами та задачами

- Створення робочих просторів та управління командою
- Часові ітерації для виконання задач
- Атомарна одиниця роботи
- Візуалізація задач

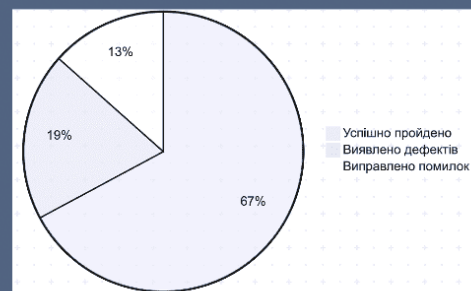


10

Рисунок В.10 – Слайд №10

Результати тестування

Функціональний модуль	Тест-кейси	Результат	Виправлені помилки
Реєстрація та профілі	10	Успішно	1
Авторизація (JWT)	6	Успішно	1
Управління завданнями (Issues)	18	Успішно	3
Спринти та планування	8	Успішно	2
Сповіщення та WebSockets	7	Успішно	2



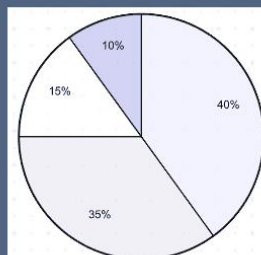
11

Рисунок В.11 – Слайд №11

Цільова аудиторія

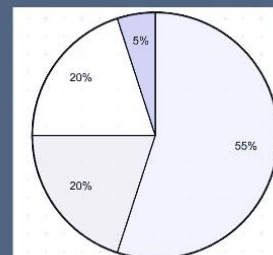
Розподілення за віком:

- 18-21 років (15%)
- 22-30 років (40%)
- 31-40 років (35%)
- 41+ років (10%)



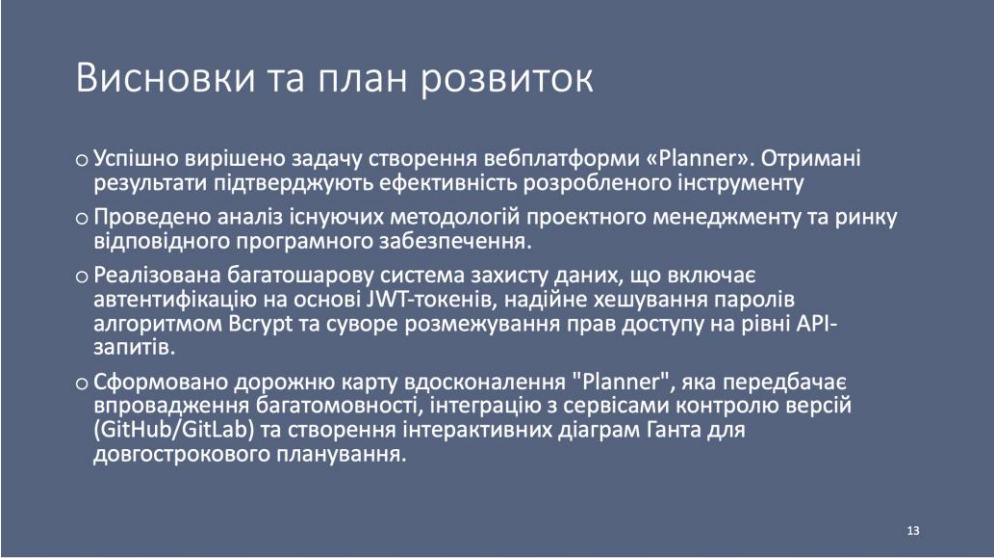
Розподілення за складом сім'ї:

- 18-21 років (15%)
- 22-30 років (40%)
- 31-40 років (35%)
- 41+ років (10%)



12

Рисунок В.12 – Слайд №12



Висновки та план розвитку

- Успішно вирішено задачу створення вебплатформи «Planner». Отримані результати підтверджують ефективність розробленого інструменту
- Проведено аналіз існуючих методологій проектного менеджменту та ринку відповідного програмного забезпечення.
- Реалізована багатoshарову систему захисту даних, що включає автентифікацію на основі JWT-токенів, надійне хешування паролів алгоритмом bcrypt та суворе розмежування прав доступу на рівні API-запитів.
- Сформовано дорожню карту вдосконалення "Planner", яка передбачає впровадження багатомовності, інтеграцію з сервісами контролю версій (GitHub/GitLab) та створення інтерактивних діаграм Ганта для довгострокового планування.

13

Рисунок В.13 – Слайд №13