НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map

Виконав: студент 4 курсу, групи ІМ-13
(шифр групи)

Нестеров Дмитро Васильович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент, доктор філософії Шульга М.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) старший викладач, доктор філософії комп’ютерної інженерії Міщенко Л. Д. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент асистент кафедри БМК Матвійчук О.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Нестерова Дмитра Васильовича

1. Тема проєкту *Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map*
Керівник проєкту ____ асистент, доктор філософії Шульга М.В._____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року № 1705-с
2. Термін здачі студентом закінченого проєкту _____*1 червня 2025 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні відомості.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1: Аналіз предметної області.
Розділ 2: Вибір технологічного стеку.
Розділ 3: Розробка функціоналу.
Розділ 4: Тестування додатку.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): алгоритм дії веб-застосунку (принципова схема), діаграма класів дорожньої карти (функціональна схема), архітектура додатку (структурна схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	Міщенко Л.Д.		

7. Дата видачі завдання___15 лютого_2025_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.02.2025-15.02.2025*	
2.	*Вивчення та аналіз завдання*	*15.02.2025-15.04.2025*	
3.	*Розробка архітектури та загальної структури системи*	*01.03.2025-10.03.2025*	
4.	*Розробка структур окремих підсистем*	*11.03.2025-31.03.2025*	
5.	*Програмна реалізація системи*	*01.04.2025-30.04.2025*	
6.	*Оформлення пояснювальної записки*	*01.05.2025-31.05.2025*	
7.	*Тестування, доопрацювання*	*01.06.2025-10.05.2025*	
8.	*Передзахист*	*11.06.2025*	
9.	*Захист*	*17.06.2025*	

Студент-дипломник _____Дмитро НЕСТЕРОВ___
(підпис)

Керівник проєкту _____Максим ШУЛЬГА___
(підпис)

Анотація

У бакалаврському проєкті було розроблено сучасний веб-додаток для підтримки командної співпраці та управління проєктами. Система поєднує в собі списки завдань, дошки Kanban та інтерактивну дорожню карту для візуалізації графіків і етапів проєкту. Користувачі можуть створювати завдання, призначати відповідальних учасників, відстежувати прогрес і отримувати сповіщення в реальному часі. Інтерфейс реалізовано за допомогою TypeScript і React, які об’єднані через Vite для швидкої розробки. Стилізація базується на SCSS модулях із підтримкою світлих/темних тем і анімації. Керування даними здійснюється через зовнішній REST API, тоді як глобальний стан обробляється за допомогою Redux Toolkit із Redux Thunk і Reselect. Компонентна архітектура забезпечує масштабованість і зручність обслуговування.

Annotation

In this Bachelor's Degree project, a modern web application was developed to support team collaboration and project management. The system combines task lists, Kanban boards, and an interactive roadmap for visualizing project timelines and milestones. Users can create tasks, assign responsible members, track progress, and receive real-time notifications. The frontend is implemented with TypeScript and React, bundled via Vite for fast development. Styling is based on modular SCSS with support for light/dark themes and animations. Data is managed through an external REST API, while global state is handled using Redux Toolkit with Redux Thunk and Reselect. A component-based architecture ensures scalability and maintainability.

Довідки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Інтерактивна платформа для	3		
			організування проєктів на базі			
			методології Kanban з використанням			
			метрик та таблиць Road map			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Інтерактивна платформа для	93		
			організування проєктів на базі			
			методології Kanban з використанням			
			метрик та таблиць Road map			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Інтерактивна платформа для	1		
			організування проєктів на базі			
			методології Kanban з використанням			
			метрик та таблиць Road map			
			Алгоритм дії веб-застосунку			
			(принципова схема)			
	А4	***ІАЛЦ.467100.005 Д2***	Інтерактивна платформа для	1		
			організування проєктів на базі			
			методології Kanban з використанням			
			метрик та таблиць Road map			
			Діаграма класів дорожньої карти			
			(функціональна схема)			

Зм	*Арк.*	*№ докум.*	*Підпис*	*Дата*

ІАЛЦ.467200.001 ОА

Розроб.	*Нестеров Д.В.*		
Перевір.	*Шульга М.В.*		
Реценз.	*Матвійчук О.В.*		
Н. контр.	*Міщенко Л.Д.*		
Затверд.	*Новотарський М.А.*		

Інтерактивна платформа для організування проєктів на базі методології Kanban з використаннямметрик та таблиць Road map
Опис Альбому

Літ.	***Арк.***	***Аркушів***
	1	2

КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-13

	А4	*ІАЛЦ.467200.006 Д3*		Інтерактивна платформа для	1		
				організування проєктів на базі			
				методології Kanban з використанням			
				метрик та таблиць Road map			
				Архітектура додатку			
				(структурна схема)			
	А4	*ІАЛЦ.467200.007 Д4*		Інтерактивна платформа для	37		
				організування проєктів на базі			
				методології Kanban з використанням			
				метрик та таблиць Road map			
				Текст програмного коду			

Зм	*Арк.*	*№ докум.*	*Підпис*	*Дата*	**ІАЛЦ.467200.001 ОА**	**Арк.**
						2

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інтерактивна платформа для організування проєктів на базі методології Kanban з використаннямметрик та таблиць Road map*»

Київ – 2025

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
Зм	***Арк.***	***№ докум.***	***Підпис***	***Дата***			
Розроб.		*Нестеров Д.В.*			*Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map* Технічне завдання	***Літ.***	***Арк.*** 1 / ***Аркушів*** 3
Перевір.		*Шульга М.В.*					
Реценз.		*Матвійчук О. В.*				***КПІ ім. Ігоря Сікорського, ФІОТ ІМ-13***	
Н. контр.		*Міщенко Л.Д.*					
Затверд.		*Новотарський М.А.*					

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Це технічне завдання поширюється на розробку вебзастосунку для організації командної роботи над завданнями та проєктами. Область застосування, це використання у різних за розміром командах для ефективного керування завданнями, планування проєктів та спільної роботи через списки справ, канбан-дошки та дорожні карти.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Причиною розробки є завдання бакалаврського проєкту за освітньо-професійною програмою "Інженерія програмного забезпечення комп'ютерних систем" спеціальності 121 " Інженерія програмного забезпечення", затверджене кафедрою Обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

3. ЦІЛЬ ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є створення сучасного вебзастосунку для управління завданнями та проєктами, який об'єднує функціонал списків справ, канбан-дошок, дорожніх карт, делегування задач, отримання сповіщень та відстеження прогресу виконання завдань.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є:

- Науково-технічна література з розробки застосунків та організації командної роботи.
- Статті та публікації у періодичних виданнях та на наукових конференціях з питань управління проєктами і розробки інтерфейсів користувача.
- Документація сучасних технологій фронтенд-розробки (React, Redux, Vite тощо).
- Аналітичні огляди існуючих систем управління завданнями, публікації в Інтернеті.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до продукту, що розробляється

- Підтримка основного функціоналу управління завданнями: створення, редагування, категоризація завдань і підзадач, делегування відповідальних осіб, побудова канбан-дошок і дорожніх карт.
- Інтуїтивно зрозумілий адаптивний інтерфейс: підтримка світлої та темної тем, плавні анімації переходів, коректна робота на різних розмірах екранів.

- Система сповіщень та мультимовна підтримка: отримання повідомлень про зміни у завданнях і проєктах, можливість перемикання мов інтерфейсу без перезавантаження сторінки.

5.2 Вимоги до програмного забезпечення

- Підтримка сучасних браузерів: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari.
- Робота через зовнішнє API: безпечна передача даних, обробка запитів у реальному часі, відповідність стандартам REST.
- Фронтенд-стек розробки: використання мови програмування TypeScript, бібліотеки React, системи збірки Vite, стилізації SCSS-модулів та управління станом через Redux Toolkit.

5.3 Вимоги до апаратної частини

- Клієнтські пристрої: персональний комп’ютер, ноутбук, планшет або смартфон.
- Оперативна пам’ять не менше 1 ГБ.
- Підключення до Інтернету: стабільне з’єднання для повноцінної роботи застосунку.

6. ЕТАПИ РОЗРОБКИ

	Дата
6.1 Вивчення джерел, літератури, аналіз існуючих рішень	20.02. 2025
6.2 Складання і узгодження технічного завдання	01.03. 2025
6.3 Проєктування архітектури вебзастосунку та структури модуля	10.03. 2025
6.4 Реалізація функціональних модулів (канбан, дорожні карти, тощо.)	10.04. 2025
6.5 Тестування функціональності, адаптивності, багатомовності	01.05. 2025
6.6 Оптимізація і виправлення помилок	20.05. 2025
6.7 Оформлення документації дипломного проєкту	10.06. 2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інтерактивна платформа для організування проєктів на базі методології Kanban з використаннямметрик та таблиць Road map»*

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SPA – single page application (односторінковий застосунок)

JS – JavaScript (джаваскрипт)

ПК – персональний комп’ютер

API – application programming interface (інтерфейс прикладного програмування)

IDE – integrated development environment (інтегроване середовище розробки)

TS – TypeScript (тайпскрипт)

ESM – ECMAScript modules (модулі ECMAScript)

SCSS – Sassy CSS (розширення CSS з підтримкою змінних, вкладеності та міксинів)

JSX – JavaScript XML (синтаксис, що дозволяє писати HTML-подібний код у JS)

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

DOM – Document Object Model (об’єктна модель документа)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

SFC – single file component (компонент одного файла)

CLI – command line interface (інтерфейс командного рядка)

MVVM – model-view-viewmodel (архітектурний шаблон модель-представлення-модель представлення)

HTTP – hypertext transfer protocol (протокол передавання гіпертексту)

AOT – ahead-of-time compilation (компіляція наперед)

RTK – Redux Toolkit (офіційний набір інструментів для Redux)

MVC – model-view-controller (модель-представлення-контролер)

SASS – syntactically awesome stylesheets (покращене розширення CSS)

ВСТУП

Веб-технології відіграють вирішальну роль у забезпеченні ефективного спілкування, співпраці та координації завдань між розподіленими командами. Через пандемію радикально зріс попит на системи керування завданнями, вони стали частиною щоденної цифрової інфраструктури. Ці системи використовуються не лише в корпоративному середовищі, але й звичайними користувачами для планування задач, роботи з проєктами разом з іншими користувачами, наприклад при роботі над студентським проєктом.

Швидка еволюція технологій, наприклад бібліотек керування станом, дозволила розробникам легко створювати адаптивні, масштабовані та зручні для користувачів програми. У цьому контексті розробка сучасної веб-системи керування задачами, що підтримує спільні робочі процеси, дошки канбан та дорожні карти, є своєчасною та практичною.

Метою цього дипломного проєкту є розробка та реалізація комплексного веб-додатку для керування завданнями, який поєднує в собі управління задачами, роботу з канбан проєктами та дорожніми картами. Система підтримує багатокористувацьку співпрацю, а також дає стимул користувачам активно взаємодіяти, наприклад за допомогою ефектів профілю та аватару. Цей проєкт відповідає потребам реальних користувачів у організації роботи, делегуванні задач іншим користувачам та плануванню довгострокових стратегій при роботі з проєктами.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Історія та розвиток систем управління завданнями

Концепція управління завданнями була суттєвим елементом людської організації з найдавніших цивілізацій. Стародавні суспільства, від єгиптян до римлян, вимагали складних методів управління робочою силою для складних проєктів, таких як будівництво або військові кампанії. Перше управління завданнями покладалося на вербальне спілкування і ручні записи.

З промисловою революцією управління завданнями перетворилося на більш формалізовані практики. Запровадження виробничих ліній і великих підприємств призвело до створення ранніх методологій управління проєктами, таких як діаграми Ганта, винайдені Генрі Гантом у 1910-х роках. Ці діаграми дозволяли менеджерам візуалізувати завдання з плином часу, що є фундаментальною концепцією, яка продовжує лежати в основі сучасного програмного забезпечення для керування завданнями.

Цифрова революція різко прискорила розвиток систем управління завданнями. Наприкінці 20-го століття організації почали використовувати комп’ютерні інструменти управління проєктами для більш ефективної організації робочих процесів. Перше програмне забезпечення, таке як Microsoft Project, представлене в 1984 році, перенесло принципи управління задачами у цифровий формат, дозволяючи користувачам працювати з задачами на персональних комп’ютерах.

У 2000-х роках поява хмарних обчислень привела до появи нового покоління платформ керування завданнями. Ці системи наголошували на співпраці в режимі реального часу та доступності на різних пристроях. Сучасні рішення зосереджені не лише на відстеженні окремих завдань, але й

на сприянні командній співпраці, гнучкості та прозорості.

1.2 Сучасні вимоги до вебзастосунків для командної роботи

У сучасному середовищі з плином часу очікування щодо веб-програм для командної співпраці значно змінилися. Більше недостатньо основних списків завдань або статичних планів проєкту. Сучасні команди – від стартапів до великих підприємств – потребують динамічних, інтуїтивно зрозумілих і дуже гнучких платформ, які не лише підтримують керування завданнями, але й сприяють співпраці, прозорості та продуктивності в різних ролях і часових поясах. Однією з основних вимог є співпраця в реальному часі. Члени команди очікують можливості миттєво переглядати зміни, призначати завдання, оновлювати статуси та коментувати без затримок. Програми, пропонуючи синхронізацію у реальному часі, встановлюють стандарт, дозволяючи командам працювати разом, ніби вони знаходяться в одній кімнаті, навіть коли користувачі розподілені по континентах.

Варіативність завдань і проєктів також стала основною вимогою. Інструменти повинні дозволяти користувачам легко перемикатись між різними режимами – такими як списки, дошки, часові шкали та календарі – залежно від їхніх уподобань і потреб проєкту.

Ще одним важливим очікуванням є інтеграція із зовнішніми інструментами. Сучасні робочі екосистеми включають різноманітні сервіси, такі як Slack, Google Drive або GitHub та багато інших. Ефективні платформи командної співпраці повинні бездоганно інтегруватися з цими інструментами, щоб уникнути перемикання контексту та підвищити ефективність.

Ролі користувачів і керування дозволами однаково важливі, особливо у великих командах, де безпека даних і делегування відповідальності мають велике значення. Гнучкі налаштування прав у проєкті дозволяють зручно і

надійно контролювати, хто може переглядати, створювати чи видаляти задачі, редагувати або керувати певними завданнями та проєктами, забезпечуючи захист конфіденційної інформації та сприяючи чіткій ієрархії.

Дуже важливою функцією серед сучасних користувачів є дорожня карта та відстеження віх. Командам потрібно не лише керувати щоденними завданнями, але й узгоджувати їх із стратегічними цілями протягом місяців або навіть років.

Мобільність і швидкість реагування пристрою сьогодні не підлягають обговоренню. Користувачам потрібен повний функціонал не лише у комп'ютерних браузерах, але і в планшетах і смартфонах. Хмарна архітектура та принципи адаптивного дизайну гарантують, що співпраця можлива будь-коли та будь-де.

Ще одна важлива вимога – простота використання у поєднанні з налаштуванням. У той час як користувачі очікують, що програми будуть інтуїтивно зрозумілими та простими в установці, вони також шукають потужні параметри налаштування, які можуть пристосувати інструмент до їх робочого процесу. Це демонструє, наскільки важливий баланс між простотою та гнучкістю.

Нарешті, візуальна персоналізація та функції мотивації набули значення. Користувачі все більше цінують можливість персоналізувати свої профілі, аватари або навіть зовнішній вигляд проєкту, створюючи більш привабливе та мотивуюче робоче середовище.

1.3 Огляд існуючих рішень

1.3.1 Asana

Asana – одна з найвідоміших платформ у галузі керування завданнями, яку часто хвалять за її простоту, гнучкість і безперебійну роботу. Це

дозволяє користувачам створювати завдання, призначати членів команди як виконавців, встановлювати кінцеві терміни виконання та відстежувати прогрес виконання за допомогою списків, дошок, календарів і часових шкал. Простий інтерфейс і дизайн, орієнтований на користувача, зробили Asana улюбленим додатком серед малих і середніх команд. Однією з головних сильних сторін Asana є її інтуїтивно зрозумілий інтерфейс користувача, який дозволяє користувачам швидко орієнтуватися та керувати проєктами без тривалого навчання. Крім того, широка інтеграційна екосистема Asana забезпечує сумісність із такими інструментами, як Slack, Google Workspace і Zoom, що дозволяє командам ефективно синхронізувати свої робочі процеси [1].

Однак, незважаючи на свої переваги, Asana має помітні обмеження, наприклад не має вбудованого модуля дорожньої карти, який пов’язує окремі завдання з довгостроковими стратегічними цілями. Крім того, Asana не надає можливостей внутрішнього спілкування, є лише коментарі до завдань. Для чатів або дзвінків у реальному часі користувачі повинні покладатися на сторонні платформи, такі як Slack. Можливості персоналізації в Asana мінімальні. Профілі користувачів є базовими, без функцій глибокого налаштування.

1.3.2 Jira

Jira, розроблена Atlassian, стала основним продуктом управління проєктами розробки програмного забезпечення, особливо для Agile-команд [2]. Вона пропонує такі розширені функції, як налаштування робочих процесів, відстеження проблем, планування спринтів, керування невиконаними задачами та детальні звіти. Модульність Jira дозволяє адаптувати її до різних галузей, хоча вона залишається найпопулярнішим рішенням серед команд інженерів та менеджерів. Головною перевагою Jira є

гнучкість. Команди можуть створювати складні робочі процеси, правила автоматизації та схеми дозволів. Вона підтримує Scrum, Kanban і гібридні методології Agile, що робить її важливим інструментом для технічного управління проєктами.

Тим не менш, саме складність Jira є одним з найбільших недоліків платформи. Налаштування проєкту часто вимагає глибокого технічного розуміння та адміністративних зусиль. Для нетехнічних користувачів або невеликих команд переважна кількість опцій Jira може перешкодою. Ще один важливий аспект – візуальне та естетичне враження. Хоча інтерфейс Jira функціональний, він часто здається важким і застарілим. Крім того, перемикання тем може спричинити такі збої, як неправильне відтворення кольорів, непослідовна видимість компонент або пошкоджені структури макета. Jira також не зосереджена на персоналізації. Профілі користувачів у прості, без візуальних налаштувань або мотиваційних функцій.

1.3.3 Aha! Roadmaps

Aha! Roadmaps – це платформа, призначена головним чином для менеджерів із продуктів, яким потрібно визначати стратегічні цілі, планувати випуски продуктів і узгоджувати зусилля розробників з бізнес-цілями. Вона спеціалізується на створенні візуальних дорожніх карт, організації функцій та ініціатив і їх пов’язуванні з ширшими стратегіями компанії. Головною перевагою Aha! Roadmaps є зосередженість на стратегічному плануванні. Для компаній, які віддають перевагу стратегічному плануванню над повсякденними завданнями, Aha! Roadmaps є ідеальним вибором [3].

Однак, Aha! Roadmaps не є комплексною системою керування завданнями. Незважаючи на те, що вона чудова у стратегічній візуалізації, користувачам часто потрібні додаткові інструменти від інших платформ, такі як Asana, Jira або щось схоже на Trello, для детального відстеження

повсякденних завдань і командної співпраці. Це призводить до фрагментації робочих процесів і додаткової операційної складності. Крім того, у Aha! Roadmaps відсутні функції спілкування в реальному часі. Співпраця обмежена коментарями в межах завдань або документів, і немає підтримки для вбудованих чатів або голосових/відеодзвінків. З точки зору персоналізації, Aha! Roadmaps є досить нудним інтерфейсом. Профілі користувачів мінімальні та не пропонують параметрів візуального налаштування. Крім того, користувальницький інтерфейс Aha! Roadmaps, хоч і чистий, не має багатої візуальної динаміки та інтерактивної анімації.

1.3.4 Zenkit

Zenkit позіціонує себе як гнучку багатопрограмну платформу, що дозволяє користувачам керувати інформацією в різних форматах, включаючи списки, дошки Канбан, таблиці, календарі та інтелектуальні карти [4]. ЇЇ адаптивність робить платформу привабливою для користувачів, яким потрібні різноманітні вже налаштовані структури без жорстких робочих процесів. Основна перевага Zenkit – це її універсальність. Команди можуть адаптувати Zenkit до різноманітних робочих процесів, перемикаючись між різними представленнями за потреби без суттєвої зміни структури. Вона також підтримує роботу в автономному режимі, що є цінною функцією для мобільних команд.

Тим не менш, Zenkit має кілька обмежень. Незважаючи на свою гнучкість, їй не вистачає більш глибоких функцій виконання завдань, таких як вбудовані розширені звіти. Також тут відсутні засоби зв’язку та надто складна автоматизація, доступна у великих системах, таких як Jira. Більше того, хоча Zenkit надає кілька переглядів, вона не пропонує персоналізованого досвіду користувача. Профілі є статичними, і в них немає вбудованих візуальних ефектів або стимулів для залучення користувачів. З

точки зору дизайну, візуальний стиль Zenkit чистий, але статичний. Переходів небагато, і взаємодія виглядає простою порівняно з більш сучасними програмами. Нарешті, Zenkit не має інтегрованих комунікаційних можливостей. Як і багато інших конкурентів, вона змушує користувачів покладатися на зовнішні інструменти для обміну повідомленнями та обговорень.

ВИСНОВОК ДО РОЗДІЛУ 1

Існуючі платформи керування завданнями та співпраці багаті спеціалізованими рішеннями, кожне з яких перевершує в одних сферах, але поступається в інших:

- Asana простота, але їй бракує стратегічного планування та спілкування.
- Jira забезпечує глибокий контроль, але перевантажує користувачів.
- Aha! Roadmaps потребує зовнішніх інструментів керування завданнями.
- Zenkit пропонує гнучкість, але не має функцій взаємодії користувачів.

MyTaskManager було розроблено для усунення цих суттєвих недоліків шляхом логічного поєднання:

- Комплексного управління завданнями та проєктами.
- Стратегічного довгострокового планування (дорожні карти).
- Вбудованих засобів спілкування (чати, дзвінки, зустрічі).
- Широкої персоналізації профілю та анімацій.
- Стабільного, динамічного і сучасного інтерфейсу користувача.

Цей цілісний підхід позиціонує MyTaskManager не лише як конкурента, але й як рішення нового покоління для керування завданнями, людьми та цілями в інтегрованому, мотивуючому та візуально привабливому середовищі.

РОЗДІЛ 2
ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Вибір мови програмування та інструментів розробки

Під час планування розробки сучасної, адаптивної та багатофункціональної веб-програми одним із найважливіших рішень є вибір відповідної мови програмування та інструментів розробки.

2.1.1 JavaScript

JavaScript залишається найбільш поширеною мовою для веб-розробки, що розвивається. За багатьма опитуваннями розробників (такими як опитування серед розробників на Github або на Stack Overflow), JavaScript уже довгий час незмінно вважається найпоширенішою мовою програмування в усьому світі. Це фактичний стандарт для розробки на основі браузера, а з появою Node.js JS також став потужним варіантом для розробки на стороні сервера. Поки що не видно тенденції на те, що з'явиться мова програмування, що замінить JavaScript у розробці клієнтського інтерфейсу. Існують навіть інструменти як Electron або React native для розробки програм на ПК або смартфони, як на IOS, так і на Android [5].

Однією з основних причин вибору JavaScript є її універсальність: одну мову можна використовувати в усьому стеку – від браузера (frontend) до сервера (backend) і навіть для інструментів створення, тестових інфраструктур та автоматизації розгортання та інших задач інфраструктури. Ця уніфікація спрощує технологічний стек, зменшує перемикання контексту та прискорює швидкість розробки. Також це дозволяє легше перекваліфікуватися з frontend у backend та навпаки.

2.1.2 Перехід до TypeScript

Хоча JavaScript є дуже гнучкою мовою, ця гнучкість може призвести до помилок під час виконання та проблем з підтримкою коду у великих програмах. З цієї причини було вирішено використовувати TypeScript – статично типізований наднабір JavaScript, розроблений Microsoft. TypeScript реалізує додаткову підтримку введення типів, інтерфейсів, переліків і розширених інструментів, що робить масштабну розробку інтерфейсу безпечнішою та передбачуванішою [6].

Використання TypeScript дозволило:

- Виявляти під час компіляції потенційні помилки.
- Дотримуватися суворіших контрактів між компонентами та API.
- Підвищити продуктивність розробника завдяки інтеграції IDE.
- Писати зручніший, масштабований код.

TypeScript легко інтегрується з сучасними фреймворками, такими як React (який також використовується у цьому проєкті), і його впровадження швидко зростає в усій галузі. TS дає багато переваг мов зі статичною типізацією, таких як Java або C#, не жертвуючи гнучкістю та виразністю JavaScript.

2.1.3 Багатопарадигмена гнучкість

Іншим важливим фактором у виборі JavaScript/TypeScript є їх мультипарадигмальний підхід. Мова підтримує функціональний, імперативний/декларативний та об’єктно-орієнтований стилі. Це дозволяє вибрати найбільш відповідну парадигму для певної частини програми. Наприклад:

- У логіці управління станом було використано чисті функції та принципи незмінності (як це видно в редьюсерах Redux Toolkit).
- У компонентній архітектурі було використано композицію.

- Для обробки подій і логіки життєвого циклу було використано замикання та першокласні функції (функції старшого порядку).
- Для API створено класи з методами, щоб ними було зручніше користуватися.

2.1.4 Інструменти розробки

Для вищої продуктивності і зменшення кількості помилок під час розробки, було обрано такі інструменти в поєднанні з JavaScript/TypeScript:

- Vite – швидкий і легкий інструмент для збірки, який використовує власні модулі ESM і значно покращує час запуску порівняно з традиційними пакетами, такими як Webpack або Gulp [7].
- ESLint у зв'язці з Prettier – для узгодження форматування коду.
- Модулі SCSS – для стилів з підтримкою змінних і міксинів.
- React DevTools – для налагодження та аналізу дерев компонентів.
- Redux DevTools – для аналізу оновлень стану.
- Jest разом з React Testing Library – для юніт тестів додатку.

2.1.5 Підсумок вибору мови та інструментів розробки

Підсумовуючи, вибір JavaScript (з TypeScript) для цього проєкту був природнім і добре обґрунтованим рішенням, підтвердженим часом зрілості такої екосистеми, наявності необхідних інструментів і парадигменої гнучкості. Використання уніфікованої мови в усьому стеку в поєднанні з сучасними інструментами, такими як Vite, і вдосконаленою безпекою типів від TypeScript, забезпечили оптимальне середовище розробки, яке дозволило швидку розробку та довгострокову масштабованість.

2.2. Огляд сучасних фронтенд-фреймворків

Веб розробка завжди стрімко розвивалася, а за останні роки особливо виділяється появою надійних фреймворків і бібліотек JavaScript, які спрямовані на спрощення розробки та оптимізацію динамічних інтерфейсів користувача.

Серед них є три найпопулярніші, кожен зі своєю філософією, особливими підходами та архітектурним стилем:

- Vue – найпростіша серед перерахованих бібліотека, дозволяє писати невеликі додатки.
- Angular – важкий фреймворк, заточений під величезні додатки.
- React – бібліотека – золота середина.

У цьому розділі досліджено ці три технології, виділено їхні характеристики, охарактеризовано екосистему та практичні переваги, а нижче буде наведено порівняльний аналіз. Це аргументовано обґрунтує вибір саме React.js для реалізації додатку.

2.2.1 Огляд React.js

React.js – JS бібліотека для створення користувальницьких інтерфейсів, що була розроблена Facebook у 2013 році [8]. Основною мотивацією при написанні були проблеми при розробці своєї соцмережі, виправлення яких займало дуже багато часу і позбавлення однієї створювало багато нових. Основна ідея полягає в компонентній архітектурі, де кожен елемент інтерфейсу можна розбити на ізольовані компоненти з внутрішнім станом і логікою відтворення. Такий код легше підтримувати та виправляти у ньому помилки. React – це не повноцінний фреймворк, а бібліотека для створення інтерфейсу користувача, яка дає розробникам свободу вибору додаткових бібліотек для маршрутизації, управління станом і стилізації візуалу [9].

2.2.1.1 Ключові особливості React

- Декларативний синтаксис – розробники описують, як має виглядати інтерфейс користувача, а React обробляє відтворення.
- Однонапрямлений потік даних – при функціональному підході можна лише передати дані від батьківської компоненти до дочірньої.
- Синтаксис JSX – поєднує HTML із JavaScript.
- Легке розділення коду, lazy-loading [10].
- Virtual DOM – Не має сенсу маніпулювати всім DOM-деревом, це найважча операція у браузері. Достатньо робити його легке дерево-представлення та працювати з ним, використовується евристичний алгоритм зі складністю O(n) [11].
- Потужна екосистема – велика колекція сторонніх бібліотек.
- Підтримка спільноти – це одна з найбільш поширених бібліотек у світі, має чудову офіційну документацію та ресурси спільноти.

2.2.1.2 Аргументація вибору React

React забезпечує швидке створення прототипів, відмінний досвід розробника та масштабовану архітектуру. У поєднанні з Redux Toolkit для управління станом і Vite для продуктивності React забезпечує швидку розробку надійних програм, які зручно підтримувати та покращувати.

2.2.2 Огляд Vue.js

Vue.js – це легка і очевидна бібліотека JavaScript. Створена Еваном Ю у 2014 році. Він розроблений таким чином, щоб бути доступним, зрозумілим і

гнучким. Vue поєднує в собі простоту традиційного HTML/CSS/JS із потужними інструментами для розробки на основі компонентів [12].

Основні характеристики Vue:

- Просте навчання.
- Компоненти одного файла (SFC), містять шаблон, логіку та стилі.
- Реактивне зв’язування даних. Тут двонапрямлений потік даних.
- Vue CLI – вбудований з набором команд та чудова документація.
- Хоча Vue є чудовою структурою, її не було обрано з таких причин:
- Трохи менш розвинена екосистема порівняно з React.
- Менше поширення у великих корпоративних системах..
- Гірша TypeScript підтримка у своїх ранніх версіях.

2.2.3 Огляд Angular

Angular – це важкий для вивчення фреймворк, розроблений Google. Він використовує TypeScript з коробки та дотримується суворо впевненого архітектурного шаблону (MVVM). Angular особливо добре підходить для додатків корпоративного рівня зі складним керуванням станом і суворими архітектурними потребами, наприклад банківські додатки [13].

Основні характеристики Angular:

- Вбудована маршрутизація, обробка форм, клієнт для HTTP запитів.
- Dependency injection система за замовчуванням.
- Попередня компіляція (AOT) для швидшого рендерингу.
- Angular CLI для роботи, є об’ємна та детальна документація.

Недоліки Angular:

- Крутіша крива навчання. Це сповільнило б розробку проєкту.

- Синтаксис шаблону більш складний і багатослівний.
- Вимагає суворого дотримання структури та парадигм Angular.
- Порівняльну характеристику наведено у таблиці 2.1

Таблиця 2.1 – Порівняльна характеристика інтерфейсних бібліотек

Характеристика / Аспект	**React**	**Vue**	**Angular**
Тип	Бібліотека	Фреймворк	Фреймворк
Мова	JS / TS	JS / TS	TS
Складність	Середня	Низька	Висока
Поріг входу	Помірний	Легкий	Крутий
Гнучкість	Дуже висока	Висока	Низька
Спільнота	Величезна	Велика	Велика
Структура компонентів	Функціональні + хуки	SFC (Шаблон + Скрипт + Стилі)	Класи + Декоратори
Продуктивність	Висока (віртуальний DOM)	Висока (віртуальний DOM)	Хороша (AOT-компіляція)
Менеджмент стану	Redux, Zustand, MobX тощо	Vuex, Pinia	RxJS, сервіси
Маршрутизація	react-router	vue-router	@angular/router
Відповідність цьому проєкту	Найкращий вибір	Додатковий варіант	Занадто важкий

2.2.4 Підсумок вибору стеку

React було обрано як найбільш оптимальний варіант для цього проєкту на основі таких критеріїв:

- Забезпечує найкращий баланс продуктивності та гнучкості.
- Інтеграція з модулями TypeScript, Redux Toolkit, Vite та SCSS є бездоганною та перевіреною часом.
- Декларативний підхід React ідеально підходить для модульних інтерфейсів користувача.
- Потужна підтримка спільноти забезпечує підтримку, розробку і масштабованість у довгостроковій перспективі.

Підсумовуючи, використання React забезпечило ідеальну основу для створення швидкої, масштабованої та інтуїтивно зрозумілої програми для керування проєктами, призначеної як для індивідуального використання, так і для командної співпраці.

2.3 Порівняння бібліотек управління станом

Управління станом є одним із найважливіших архітектурних питань у сучасній розробці інтерфейсу, особливо під час створення великих динамічних програм із складною взаємодією користувачів, спільними компонентами та багатьма асинхронними операціями. У цьому розділі буде розглянуто кілька найбільш популярних рішень для управління станом, порівняно їхні архітектурні моделі та пояснено, чому для цього проєкту було обрано саме Redux Toolkit.

2.3.1 Огляд Redux Toolkit

Redux Toolkit (RTK) – це офіційно рекомендований розробниками Redux і модернізований підхід для роботи з Redux – контейнером передбачуваного стану для програм JavaScript [14]. За своєю суттю Redux побудований на принципах архітектури Flux, запропонованої Facebook [15]. Flux – це односпрямований шаблон потоку даних, де стан програми керується в одному сховищі, а зміни запускаються виключно через дії, оброблені чистими функціями, відомими як редьюсери (або ще називають редукторами) [16]. Це контрастує з традиційними шаблонами MVC (Model-View-Controller) і MVVM (Model-View-ViewModel), де двонаправлене зв’язування даних і децентралізована логіка можуть призвести до непередбачуваних станів і збільшення складності налагодження [17].

Враховуючи складність проєкту – з її системою автентифікації, декількома взаємопов’язаними модулями (Kanban, Roadmaps, Profile тощо) і асинхронними операціями – було необхідне масштабоване та передбачуване рішення для керування станом.

Redux Toolkit було обрано, оскільки:

- Забезпечує чітку структуру без шкоди для гнучкості.
- Підтримує проміжне програмне забезпечення та керування побічними ефектами (наприклад, logging, обробка помилок, async thunks).
- Централізований стан робить міжкомпонентний зв’язок безперебійним.
- Redux DevTools пропонує потужні можливості налагодження.

2.3.2 Огляд Context та useReducer

React Context проста, але потужна альтернатива стороннім бібліотекам для управління станом. Контекст дозволяє передавати дані через дерево компонентів без необхідності вручну кидати пропси на кожному рівні – це вирішує проблему «props drilling» і забезпечує доступ до глобального стану з будь-якої точки програми. Хук useReducer, у свій час, надає чіткий і передбачуваний спосіб оновлення стану, особливо якщо логіка оновлення є складною або включає кілька кроків [18]. У поєднанні ці інструменти можуть побудувати централізовану систему керування станом на базі стандартного функціоналу React без потреби в зовнішніх бібліотеках, таких як Redux. Це особливо зручно для невеликих застосунків, де в пріоритеті простота та контроль.

2.3.2.1 Мотивація створення Context API

Context у React був представлений для вирішення prop drilling. Прокидання параметрів вглибину відбувається, коли глибоко вкладеним компонентам потрібні дані від компонент, що на кілька порядків старше, але тоді дані повинні передаватися через кожен проміжний рівень, навіть якщо ці рівні безпосередньо не використовують їх. Це не тільки призводить до роздутого та менш читабельного коду, але й тісно пов'язує компоненти, які в іншому випадку повинні залишатися незалежними. Context API вирішує цю проблему. Для цього надаються засіб доступу до спільних даних для будь-якого компонента, незалежно від їх глибини в дереві, без передачі атрибутів вручну. Контекст діє як глобальна система ін'єкції залежностей, де стан або конфігурацію можна споживати через хук useContext [19].

2.3.2.2 Принцип роботи useContext + useReducer

Хук useReducer – це реалізація шаблону редуктора React, схожа за концепцією на редуктори Redux. У поєднанні з Context API він дає змогу створювати централізовану логіку стану [18]:

- Функція редуктора обробляє переходи між станами.
- Провайдер контексту обгортає компоненти і надає стан і метод dispatch.
- Будь-який компонент всередині <Provider> може отримати доступ до дій або стану.
- Ця модель ідеально підходить для обробки простих сценаріїв глобального стану, таких як автентифікація користувача, перемикання тем, мовні параметри або стан відкриття модалок.

2.3.2.3 Ключові переваги

- На відміну від Redux, Context + useReducer не потребує зовнішніх пакетів, конфігурації проміжного програмного забезпечення або спеціальних інструментів.
- І стан, і пов’язана з ним логіка оновлення знаходяться разом в одному файлі/модулі.
- Оскільки все працює в хуках React, немає концепції проміжного програмного забезпечення, як у Redux.
- Хоча Context + useReducer не мають автоматичної типізації, TypeScript і може бути доданий явно з мінімальними зусиллями.

2.3.2.4 Обмеження

Незважаючи на свою простоту і елегантність, цей підхід починає руйнуватися у більших програмах, таких як цей проєкт. Обмеження стають особливо очевидними в таких областях:

- Одним із найбільших недоліків Context є те, що будь-яка зміна в наданому значенні спричиняє повторний ререндер всіх споживаючих компонент, навіть якщо вони використовують лише частину стану [19].
- У Redux Toolkit та усіх інших популярних стейт менеджерів є власний DevTools, що пропонує відстеження дій і налагодження стану. Але Context + useReducer не має цього самоаналізу.
- Керування декількома контекстами та редукторами для різних сегментів стану (наприклад, профіль, дорожні карти) швидко стає громіздким. На відміну від Redux, який природно підтримує фрагментацію та проміжні бібліотеки, Context не пропонує

вбудованої стратегії для поділу коду, асинхронної логіки чи структурованого глобального стану.

- Відсутність вбудованої підтримки асинхронних дій змушує розробників обробляти асинхронну логіку на рівні компонент або за допомогою спеціальних абстракцій. Це призводить до повторюваних шаблонів, розкиданих проблем і зниження ремонтопридатності.

2.3.2.5 Причини не обирати Context

У цьому проєкті архітектура програми вимагала спільного стану для багатьох модулів, надійної асинхронної логіки, проміжного програмного забезпечення та передбачуваних переходів між станами наступних функцій:

- Спільне редагування завдань і дошки
- Системи оповіщення
- Керування профілем і налаштуваннями
- Візуалізація інтерфейсу користувача на основі ролей

Redux Toolkit надає все це з коробки разом із потужною екосистемою інструментів і архітектурою для довгострокової розробки. Це заохочує розділяти код, сприяє узгодженості в оновленнях стану, і його легше тестувати та налагоджувати в командах. Хоча Context + useReducer добре працювало би в меншому автономному проєкті (наприклад, перемикачі тем або локалізація текстів), це вимагало би значних архітектурних витрат, щоб відповідати можливостям Redux у цьому контексті.

2.3.3 Огляд MobX

MobX – це бібліотека керування станом для програм JavaScript, яка базується на парадигмі реактивного програмування. На відміну від Redux або

useReducer, які покладаються на явні переходи між станами та незмінність, MobX дозволяє розробникам безпосередньо змінювати стан і автоматично відстежує, які компоненти залежать від яких даних. Це робить її дуже інтуїтивною у використанні та значно зменшує шаблонний код [20].

В основі MobX лежить принцип прозорої реактивності. Стан стає доступним для спостереження, і будь-який компонент або обчислення, які його використовують, стають автоматично реактивними. Коли спостережувані дані змінюються, усі залежні реакції (компоненти, обчислення тощо) автоматично й ефективно оновлюються [21].

MobX досягає цього за допомогою трьох основних концепцій:

- Спостережуваний стан: будь-який об’єкт, масив або примітивне значення можна зробити спостережуваним.
- Обчислені значення: похідні дані, які автоматично оновлюються, коли спостережувані параметри залежать від змін.
- Реакції: функції, які виконують побічні ефекти (наприклад, рендеринг інтерфейсу користувача) при зміні спостережуваних.

2.3.3.1 Переваги MobX

- MobX усуває необхідність визначення типів дій, редукторів або диспетчерів. Необхідно написати стан як звичайні класи або об’єкти JavaScript, що спрощує навчання для початківців і забезпечує швидшу розробку проєктів малого та середнього розміру.
- MobX відстежує залежності даних під час виконання, тож не потрібно вручну підключати компоненти до певних частин стану. Це означає, що компоненти будуть повторно ререндиритись лише

тоді, коли конкретні дані, від яких вони залежать, зміняться, покращуючи продуктивність у багатьох випадках.

- MobX оптимізує візуалізацію, використовуючи точну спостережуваність, тому дозволяє уникнути непотрібних повторних рендерингів, поширених у Context API, або простого використання Redux. Це робить MobX вибором, придатним для додатків з інтерфейсом користувача, таких як інформаційні панелі в режимі реального часу.
- MobX легко інтегрується з класовими моделями стану. Для розробників, які віддають перевагу об'єктно-орієнтованому дизайну (інкапсуляція, успадкування тощо), це велика перевага порівняно з Redux, який заохочує функціональний, незмінний дизайн.
- Сучасні версії MobX мають чудову підтримку TypeScript із такими декораторами, як @observable, @computed і @action, які сприяють безпеці та чіткості типів у складних кодових базах.

2.3.3.2 Причини не використовувати MobX

Незважаючи на те, що MobX є чудовим вибором для невеликих програм або проєктів із частими оновленнями інтерфейсу користувача, його не було обрано для цього проєкту з таких причин:

- Необхідність передбачуваних і тестованих оновлень стану в модулях.
- Вимога щодо стандартизованої структури всієї програми.
- Інтеграція з DevTools, яка пропонує можливість налагодження,.
- Redux Toolkit запропонував кращий досвід розробника завдяки своїм лаконічним API, зберігаючи чіткий розподіл завдань і цілісність архітектури.

2.3.4 Порівняння станових бібліотек

У контексті сучасної розробки інтерфейсу вибір правильної бібліотеки керування станом має вирішальне значення для зручності обслуговування, масштабованості та продуктивності. У цьому розділі порівнюються бібліотеки керування станом, які обговорювалися раніше – Redux Toolkit, Context API + useReducer, MobX, у таблиці 2.2 зображено їх порівняльну харакретистику.

Таблиця 2.2 – Порівняльна характеристика станових бібліотек

Функція / Критерій	**Redux Toolkit**	**Context API + useReducer**	**MobX**
Парадигма	Flux (іммутабельність, дії)	Reducer-орієнтована (вбудована в React)	Реактивна (ООР, шаблон спостерігача)
Шаблонний код (boilerplate)	Помірний	Низький	Дуже низький
Асинхронна логіка	Вбудована (createAsyncThunk)	Ручна (через useEffect)	Через observables
Підтримка TypeScript	Відмінна	Середня	Середня
DevTools та налагодження	Розвинуті, зрілі інструменти	Відсутні (потрібен ручний логінг)	Базові
Гранулярна реактивність	Слабка (глобальні перерендери)	Слабка	Відмінна
Екосистема	Обширна	Рідна для React	Помірна
Крива навчання	Середня	Низька	Від низької до середньої
Продуктивність	Висока (якщо добре структурована)	Середня	Висока (спостережувані дерева)

2.3.5 Підсумок вибору бібліотеки управління станом

Кожна бібліотека має унікальні сильні сторони та випадки використання:

- Redux Toolkit найкраще підходить для великих, структурованих програм, які вимагають суворого контролю та послідовності.
- Context API ідеально підходить проєктів без зовнішніх залежностей.
- MobX блищить у реактивних системах із інтенсивним використанням інтерфейсу користувача, де критично важливою є продуктивність і читабельність, але є брак структури та стандартизованості коду.

Враховуючи складність проєкту та потребу в асинхронній логіці, Redux Toolkit було обрано як найбільш відповідне рішення.

2.4 Огляд технологій стилізації інтерфейсу

У розробці інтерфейсу технології стилю відіграють ключову роль у візуальному дизайні, у зручності обслуговування та масштабованості, адже візуал – перше, що користувач бачить. З роками розвинулося кілька підходів до стилізації додатків. У цьому розділі проаналізовано три широко поширені методи: SCSS Modules, Tailwind CSS і BEM [22].

2.4.1 SCSS-модулі

Модулі SCSS поєднують потужність SASS із модулями CSS, щоб створити модульну систему стилів із обмеженим діапазоном [23]. На відміну від глобальних таблиць стилів, стилі, визначені в модулях SCSS, за замовчуванням мають локальну область видимості, що запобігає конфліктам імен і ненавмисним змінам [24].

2.4.1.1 Ключові переваги

- Назва класу є унікальною для компонента, це зменшує побічні ефекти.
- Розробники можуть визначати повторно використовувані маркери, функції та теми за допомогою функцій SCSS.
- Модулі SCSS пропонують детальний контроль над компонуванням і дизайном без обмежень від першокласних утиліт.
- Легше впроваджувати та підтримувати анімовані зміни та переходи.

2.4.1.2 Використання в цьому проєкті

У цій програмі широко використовувалися модулі SCSS. Файл theme.scss зберігає всі маркери кольорів, які імпортуються глобально. Це дозволило легко перемикатися між темами, структуровано керувати анімаційними інтервалами, точками зупинки та повністю керованою системою дизайну, адаптованою до потреб проєкту. У міру того, як додаток ускладнювався, модулі SCSS дозволяли підтримувати чіткий розподіл задач між макетом, структурою та логікою компонентів.

2.4.1.3 Обмеження

- Без динамічних стилів: на відміну від CSS-in-JS, стилі не можна змінювати під час виконання за допомогою JavaScript. Треба створювати два класи та перемикати їх у коді за певною умовою.
- Потрібне налаштування: модулі SCSS потребують належної конфігурації в пакетах (наприклад, Vite, Webpack).

2.4.2 Tailwind CSS

Tailwind CSS – це utility-first CSS-фреймворк, що сприяє використанню попередньо визначених класів безпосередньо в розмітці. Він спрямований на те, щоб усунути потребу писати власний CSS, створюючи компоненти з атомарними класами та перевикористання їх [25].

2.4.2.1 Сильні сторони

- Швидке створення прототипів: розробники можуть швидко створювати інтерфейси, не залишаючи файл компонента.
- Послідовна система дизайну: Tailwind забезпечує суворий підхід до дизайну з інтервалами, типографікою та кольорами.
- Адаптивні утиліти: вбудована підтримка медіа-запитів і точок зупину.

2.4.2.2 Причини не обрати Tailwind

- Хоча Tailwind забезпечує швидкість і послідовність, тісний зв’язок між структурою та стилем у JSX/HTML може зменшити читабельність.
- Проєкт потребує дуже налаштованих стилів, керування анімацією та кількох тем – усе це більш природно обробляється за допомогою SCSS.
- Класовий ад (надто багато вбудованих простих класів і надто важко створювати свої більш складні класи) [26].

Tailwind ідеально підходить для систем дизайну зі статичними темами або коли дизайнерські обмеження суворі. Однак у цьому випадку гнучкість і контроль SCSS були більш вигідними.

2.4.3 BEM

BEM – це конвенція про іменування та методологія, яка заохочує передбачувані семантичні імена класів для сприяння багаторазовому використанню та послідовності [27].

Переваги:

- BEM можна використовувати з будь-яким рішенням стилізації.
- Допомагає легко розробляти багаторазові компоненти інтерфейсу.

Недоліки:

- Немає фактичного визначення, це лише методологія [28].
- Багатослівність: імена класів можуть стати довгими та повторюваними.

2.4.4 Вибір у цьому проєкті

Принципи BEM були частково дотримані в іменуванні SCSS для сприяння читабельності, але повна методологія не була дотримана через природну інкапсуляцію, яку забезпечують модулі CSS. BEM і Tailwind не підходять для проєктів зі складними анімаціями, тому їх не було обрано. Враховуючи потреби проєкту в налаштуванні теми, інкапсуляції і точному контролі, модулі SCSS були обрані як баланс масштабованості, зручності та продуктивності.

2.4.5 Підсумок вибору технології стилізації

Обраний стек технологій відображає баланс між гнучкістю, стабільністю та особистим досвідом. Використання React з використанням стейт менеджеру Redux Toolkit, та стилізація через SCSS модулі формують міцну основу для сучасної розробки SPA [29], що дозволяє впроваджувати розширені функції, як-от створення кольорових тем, маршрутизацію та моніторинг продуктивності – і все це в архітектурі, яка легко підтримується та масштабується. Була інтегровано бібліотеку i18next для підтримки багатомовних інтерфейсів, що забезпечує динамічний переклад і локалізацію з мінімальними витратами [30]. У різних мовах текст різного розміру, тому за допомогою стилів це було виправлено.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було проведено комплексний аналіз сучасних фронтенд-технологій і бібліотек управління станом, щоб обґрунтувати вибір найбільш відповідних інструментів для проєкту. Базуючись на продуктивності, архітектурі, досвіді розробника і зрілості екосистеми, React було обрано як основну структуру через її гнучкість, можливість багаторазового використання та сильну підтримку спільноти.

Для керування станом Redux Toolkit було обрано замість таких альтернатив, як MobX, оскільки ця бібліотека забезпечує структурований, масштабований і передбачуваний спосіб керування станом програми. Вона добре узгоджується з архітектурою Flux, забезпечує дотримання найкращих практик і бездоганно інтегрується з TypeScript. Стилі було реалізовано за допомогою модулів SCSS, що дозволяє створювати модульний, тематичний та підтримуваний дизайн.

Загалом, вибраний стек пропонує міцну основу для створення продуктивної, зручної та зручної для користувача програми керування завданнями. Рішення ґрунтувалися як на технічній аргументації, так і на особистому досвіді розробки повного стека, що забезпечувало довгострокову життєздатність і розширюваність проєкту.

РОЗДІЛ 3
РОЗРОБКА ФУНКЦІОНАЛУ

3.1 Загальна структура проєкту

Проєкт реалізовано як React SPA додаток з domain-based архітектурою. Кожна папка – окремий модуль зі своїми задачами.

3.1.1 Опис файлової структури

Модулі розділені наступним чином:

- /pages – Компоненти верхнього рівня, що рендерять сторінки та містять в собі інші компоненти (Home, Roadmap, Profile, Board, Auth),
- /components – універсальні або ізольовані компоненти, що перевикористовуються (Header, SearchUser, Button, Footer, Preloader).
- /helpers – допоміжні чисті функції для роботи з часом, рядками, форматування даних, тощо.
- /redux – глобальний стан додатку, розділені на шари, окремо Auth, Home, Roadmap, тощо.
- /api – запити на сервер, об’єднані у класи за зонами відповідальності (окремо запити на роботу з KanBan, Roadmap, Tasks, Categories , Users, Auth, тощо.
- /hoc – компоненти вищого порядку, наприклад withAuthRedirect, обгортають сторінки типу Roadmap, та перевіряють чи автризований користувач, якщо ні, перенапраправляє на сторінку авторизації.

- /styles – Глобальні стилі з підтримкою тем, Reset.css, стилі шрифтів.
- /layouts – Шаблони для сторінок, окремий шаблон з Header та Footer, окремий для сторінок авторизації, там тільки шапка сайту, окремий шаблон для сторінки з відповідями на питання користувачів.
- /languages – файли з перекладом тексту додатку на різні мови.
- /types – глобальні типи, що перевикористовуються у додатку, Status, User, тощо.
- /mocs – об’єкти для імітації початкового стану додатку для тестування.

3.1.2 Організація роутінгу за допомогою React Router

Для маршрутизації у MyTaskManager було використано бібліотеку React Router 6 версії. Усі сторінки згруповано у три основні шаблони:

- AuthLayout – для сторінок реєстрації та авторизації.
- PageLayout – для сторінки профілю.
- FAQLayout – для сторінки відповідей на питання користувачів.
- HomeLayout – для інших сторінок (дорожні карти, канбан, тощо).

Усі існуючі сторінки:

- / – головна сторінка (ще її називають домашньою).
- /faq – сторінка з частими питаннями
- /auth/login – сторінка авторизації.
- /auth/register – сторінка реєстрації.
- /canban – список усіх канбан дошок.
- /canban/:id – окрема канбан-дошка.
- /roadmap – список усіх дорожніх карт.

- /roadmap/:id – конкретна дорожня карта.
- /profile – профіль поточного користувача.
- /profile/:id – профіль іншого користувача.
- * – будь-який інший маршрут веде на головну сторінку.

На рисунку 3.1 зображено реалізацію маршрутизації у додатку, використовуються теги Routes та Router, також тут використовується ToastifyContainer для зручної візуалізації повідомлень для користувачів, наприклад про помилки у додатку.

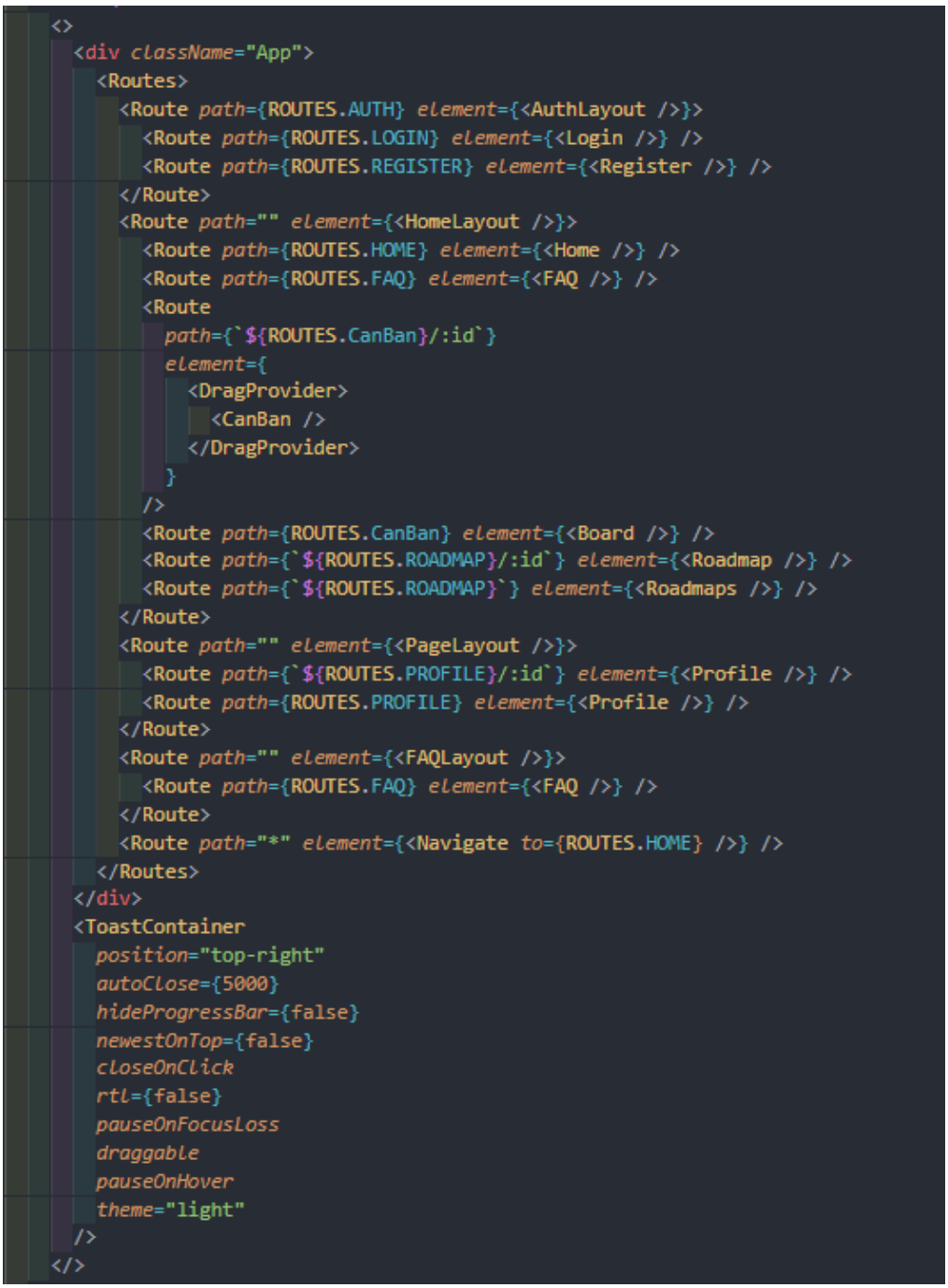


Рисунок 3.1 – Реалізація роутінгу у додатку

3.1.3 Використання Lazy loading для сторінок

React lazy ідеально підходить для оптимізації додатку, важкі компоненти, наприклад сторінку профілю, можна завантажити тільки якщо вона використовується безпосередньо зараз на екрані. Не має сенсу завантажувати цю компоненту, якщо користувач проходить авторизацію.

Приклад реалізації:

```
const FAQ = lazy(() => import('./pages/FAQ/FAQ'));
const Roadmap = lazy(() => import('./pages/Roadmap'));
```

Але на завантаження компоненти потрібен час, у цей момент потрібно показати користувачу, що іде процес завантаження. Для цього існує компонента Suspense, що обгортає завантажені через lazy loading компоненти та має атрибут fallback. Компонента очікує у якості fallback параметру компоненту, яка буде показуватись під час завантаження. У проєкті для цього було створено компоненту Preloader, а Suspense компонента використовується у файлах-шаблонах, наприклад у AuthLayout. На рисунку 3.2 зображено компоненту AuthLayout.

```tsx
import { FC,Suspense } from "react";
import { Outlet } from "react-router";

import Preloader from "../components/FallBackPreloader/FallBackPreloader";
import Footer from "../components/Footer/Footer";
import Header from "../components/Header/Header";

const AuthLayout: FC = () => {
  return (
    <>
      <Header links={[]} />
      <Suspense fallback={<Preloader />}>
        <Outlet />
      </Suspense>
      <Footer links={[]} />
    </>
  );
};

export default AuthLayout;
```

Рисунок 3.2 – AuthLayout

3.2. Реалізація базових сторінок

3.2.1 Головна сторінка (Home)

Це сторінка, на яку потрапляє користувач у першу чергу, тут знаходяться інструменти для роботи з особистими задачами, а також сюди потрапляють

задачі, які можуть бути делеговані іншими користувачами. Скріншот сторінки видно на рисунку 3.3.

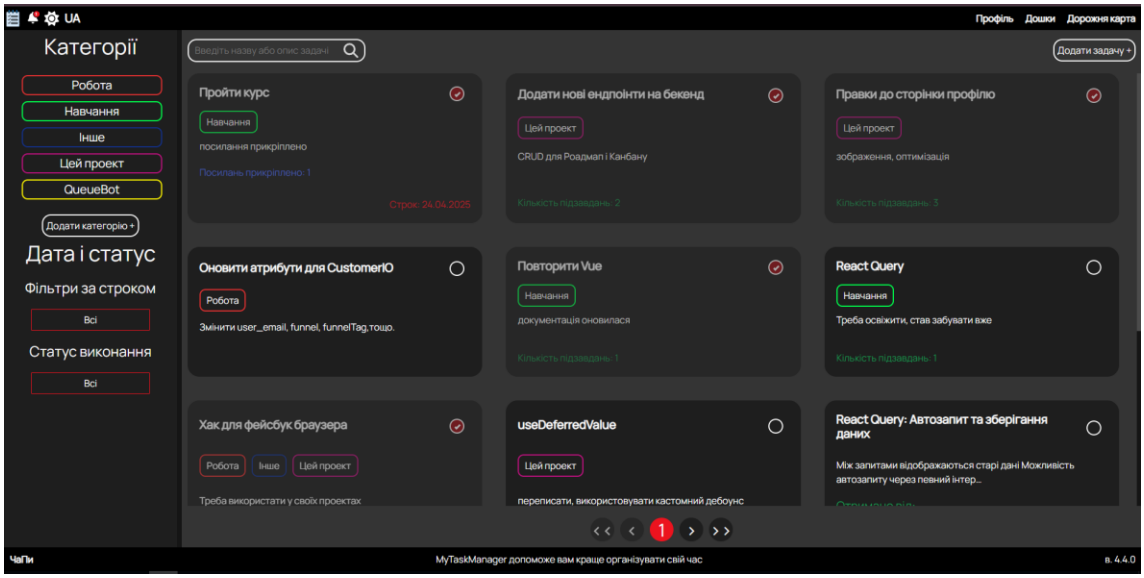


Рисунок 3.3 – Домашня сторінка додатку

Реалізовано функціонал управління особистими задачами, фільтрацію за категоріями або статусом виконання чи терміном виконання. У задач може бути до десяти прикріплених посилань, закладена архітектура під прикріплення файлів, зображень, відео та аудіо. Також у задачі може бути до десяти підзадач, які можна делегувати іншим користувачам та відстежувати статус виконання. Прийняття підзадачі, які користувачу делегували інші люди, необхідно спочатку підтвердити через систему повідомлень, а також їх можна відхилити. Статус виконання або відхилення з’являється у автора задачі. Для задач та категорій реалізовано створення, редагування та видалення, налаштовано пагінацію та фільтрацію.

Задачі відображаються у вигляді сітки з усією лаконічно виведеною інформацією, при натисканні на задачу її можна роздивитись детальніше, з повною інформацією на весь екран. Логіку рендерингу сітки задач зображено на рисунку 3.4.

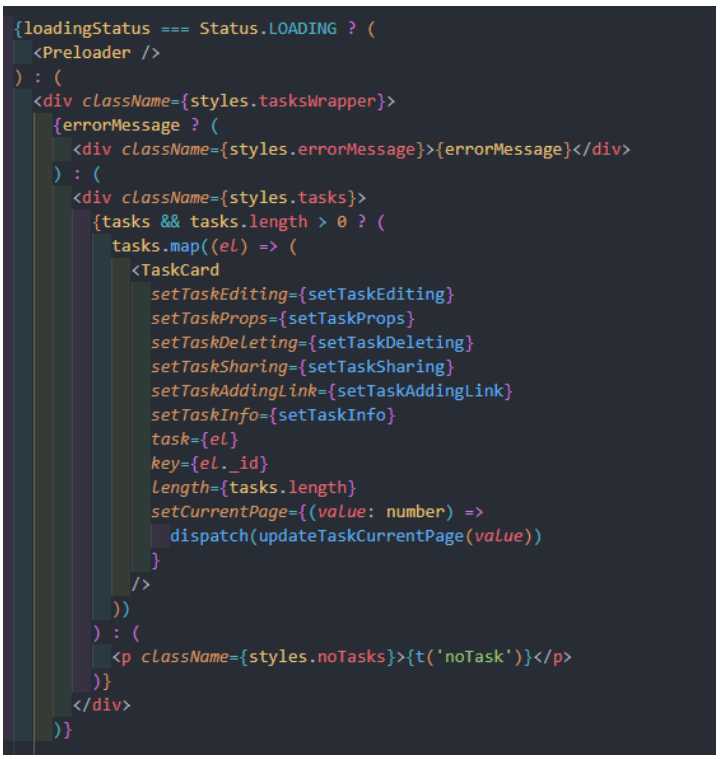


Рисунок 3.4 – Рендеринг сітки задач

Панель категорій та фільтрів знаходиться збоку, для телефонів вона займе весь екран і буде існувати кнопка, щоб відкрити це меню, або закрити. Усі елементи на сторінці розбиті на менші компоненти: Task, Pagination, Category, Categories, Tasks, тощо. Це дозволяє перевикористовувати їх, наприклад компонента Category зустрічається у боковій панелі та у сітці задач. Логіку бокової панелі та мобільного режиму зображено на рисунку 3.5.

```
const Home = () => {
  const dispatch = useAppDispatch();

  const [isMobile, setIsMobile] = useState(false);
  const [isNavBarOpened, setIsNavberOpened] = useState(true);

  useEffect(() => {
    dispatch(fetchCategories({ page: 1 }));
    return () => {
      dispatch(clearCategories());
    };
  }, []);

  useEffect(() => {
    const handleResize = () => {
      if (window.innerWidth < 680) { setIsMobile(true);
      } else { setIsMobile(false)}
    };

    handleResize();

    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  return (
    <div className={styles.row}>
      {isMobile && ( <FontAwesomeIcon icon={faBars} className={styles.menu} onClick={() => {setIsNavberOpened(true);}} />)}
      {(isNavBarOpened || !isMobile) && (<Filters isMobile={isMobile} setIsNavberOpened={setIsNavberOpened} />)}
      <Tasks isMobile={isMobile} />
    </div>
  );
};
```

Рисунок 3.5 – Логіка бокової панелі

3.2.2 Сторінка FAQ

Сторінку відповідей на часті питання було реалізовано як акордеон із запитаннями та скритими відповідями, які стають видимими після натискання на питання. Усі питання локалізовані на всі доступні у додатку мови, питання та відповіді зберігаються локально, немає взаємодії з API. На рисунку 3.6 зображено відтворення списку питань та відповідей.

```tsx
const FAQ: FC = () => {
  const { t } = useTranslation();

  const [activeIndex, setActiveIndex] = useState<number | null>(null);

  const FAQ_DATA = new Array(10).fill(1).map((_, i) => ({
    question: t(`question${i + 1}`),
    answer: t(`answer${i + 1}`),
  }));

  const toggleAccordion = (index: number) => {
    if (activeIndex === index) setActiveIndex(null);
    else setActiveIndex(index);
  };

  return (
    <div className={styles.wrapper}>
      <div className={styles.accordion}>
        <h1>{t('faqBold')}</h1>
        {FAQ_DATA.map((item, index) => (
          <div
            key={index}
            className={`${styles.item} ${
              activeIndex === index ? styles.active : ''
            }`}
          >
            <div
              className={styles.question}
              onClick={() => toggleAccordion(index)}
            >
              {item.question}
            </div>
            {
              <div className={styles.answer}>
                <div className={styles.answerBackground}>{item.answer}</div>
              </div>
            }
          </div>
        ))}
      </div>
    </div>
  );
};
```

Рисунок 3.6 – Логіка сторінки відповідей на часті питання

3.2.3 Сторінка профілю користувача (Profile)

Основна структура сторінки складається з наступних компонент:

- ProfieCard – Блок з інформацією про користувача та його зображенням.
- Chart – графік виконання задач, видимий лише для свого профілю.
- Buttons – кнопки для редагування профілю, ефектів, видалення акаунту.

З’являється цей блок лише при перегляді свого профілю, якщо ідентифікатор у посиланні співпадає з ідентифікатором користувача при реєстрації або авторизації.

Інтерфейс адаптується залежно від того, чи у своєму профілі користувач знаходиться зараз, чи ні: якщо так, з'являються інструменти для редагування інформації про профіль, зміна паролю, ефектів користувацького зображення або ефектів профілю. Інформація про ідентифікатор користувача, що авторизувався, зберігається у глобальному стані додатку. Ідентифікатор поточного профілю дістається з query параметрів браузера.

const { id } = useParams();

const ownerId = useAppSelector(selectProfile);

Приклад перевірки на те, чи власний профіль переглядає користувач, зображено на рисунку 3.7.

```
<div className={styles.column}>
  {profileStats.length > 0 && (
    <>
      {(!id || id === ownerId) && (
        <div className={styles.chartWrapper}>
          <Bar
            data={getChartData(profileStats)}
            options={chartOptions}
          />
        </div>
      )}
    </>
  )}

  {(!id || id === ownerId) && (
    <Buttons
      setIsExiting={setIsExiting}
      setIspassEditing={setIspassEditing}
      setIsAccountDeleting={setIsAccountDeleting}
      setIsEffectModalOpened={setIsEffectModalOpened}
      setIsProfileEffectModalOpened={setIsProfileEffectModalOpened}
    />
  )}
</div>
</div>

{isPassEditing && (!id || id === ownerId) && (
  <div className={styles.passwordWrapper}>
    <div className={styles.passEditing}>
      <ChangePass id={id} />
    </div>
  </div>
)}
```

Рисунок 3.7 – Перевірка ідентифікатору користувача у профілі

Для зміни ефектів використовуються модальні вікна, які зображені на рисунках 3.8 і 3.9.

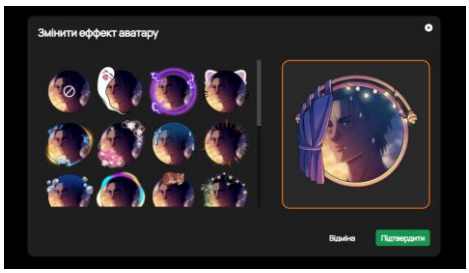


Рисунок 3.8 – Зміна ефекту аватару

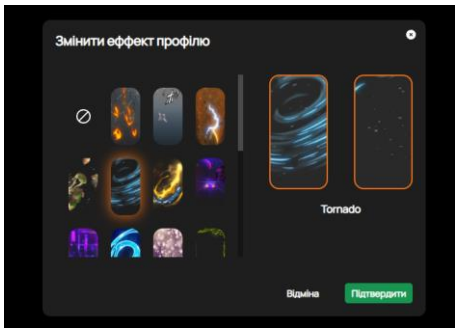


Рисунок 3.9 – Зміна ефекту сторінки профілю

Стан модальних вікон (відкрите або ні) зберігається у компоненті Profile, як наведено нижче, реалізовано за допомогою useState(). Приклад логіки модальних вікон для зміни ефектів зображено на рисунку 3.10.

```
const [isEffectModalOpened, setIsEffectModalOpened] = useState(false);
const [isProfileEffectModalOpened, setIsProfileEffectModalOpened] = useState(false);
```

```
<Modal
  active={isEffectModalOpened}
  setActive={setIsEffectModalOpened}
  ChildComponent={ChangeAvatarEffect}
  childProps={{
    toggleActive: setIsEffectModalOpened,
    isActive: isEffectModalOpened,
  }}
/>

<Modal
  active={isProfileEffectModalOpened}
  setActive={setIsProfileEffectModalOpened}
  ChildComponent={ChangeProfileEffect}
  childProps={{
    toggleActive: setIsProfileEffectModalOpened,
    isActive: isProfileEffectModalOpened,
  }}
/>
```

Рисунок 3.10 – Логіка модальних вікон для редагування профілю

Для зміни аватару користувача необхідно просто натиснути на нього та завантажити файл. На рисунку 3.11 зображено аватар користувача.

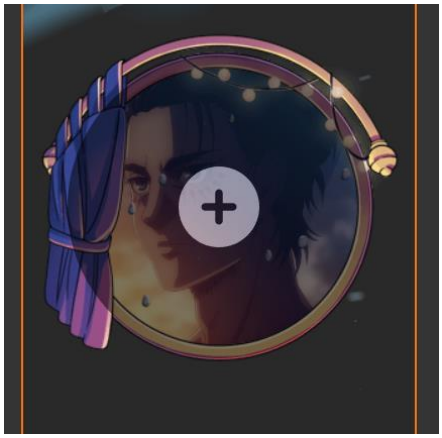


Рисунок 3.11 – Вигляд аватару користувача

На рисунку 3.12 зображено логіку зміни аватару користувача. При натисканні на аватар користувача відкриється файловий провідник, де необхідно буде обрати файл допустимого розміру та розширення, після чого буде відправлено запит на сервер на оновлення зображення.

```
const Avatar: FC<AvatarProps> = ({ isOwner }) => {
  const dispatch = useAppDispatch();
  const profile = useAppSelector(selectUserProfile);
  const inputFileRef = useRef<HTMLInputElement>(null);

  const handleChangeFile = async (event: FormEvent<HTMLInputElement>) => {
    const target = event.target as HTMLInputElement;
    const file: File = (target.files as FileList)[0];
    if (!['image/jpeg', 'image/png', 'image/jpg'].includes(file.type)) return toast.error('File type should be image, png, jpg or jpeg');
    const formData = new FormData();
    formData.append('image', file);
    try { await dispatch(changeAvatar({ image: formData })); }
    catch (e) { toast.error(`${e}`); }
  };

  return (
    <div className={styles.avatar} data-testid="avatar-container">
      {isOwner && ( <input type="file" ref={inputFileRef} onChange={handleChangeFile} data-testid="file-input-component"/>)}
      {profile?.avatarEffect?.animated ? (<img src={profile.avatarEffect.animated} className={styles.avatarEffect} alt="effect" />) : null}
      {profile?.avatar ? ( <img src={profile.avatar} alt="logo" />) : (
        <img src={'https://res.cloudinary.com/dmbythxia/image/upload/v1697126412/samples/animals/cat.jpg'} alt="default" />
      )}
      {isOwner && (
        <div className={styles.addPhoto} onClick={() => inputFileRef.current?.click()} data-testid="add-photo-component">
          <FontAwesomeIcon
            className={styles.camera}
            icon={faCirclePlus}
            data-testid="camera-icon-component"
          />
        </div>
      )}
    </div>
  );
};
```

Рисунок 3.12 – Логіка зміни аватару користувача

На сторінці профілю є гістограма виконаних задач по дням, закладена архітектура для дзвінків та текстових повідомлень, додавання користувачів у

групи. Для зміни імені необхідно просто натиснути на кнопку редагування біля нього, що з’являється при наведенні курсору. Для зміни паролю існує спеціальна секція, яка з’являється при натисканні кнопки зміни паролю. Вона вимагає вводу нового та старого паролів. Також реалізовано можливість вийти з акаунту або видалити акаунт, ці дії необхідно підтвердити, щоб користувач не міг зробити це випадково. Скріншот профілю користувача зображено на рисунку 3.13.

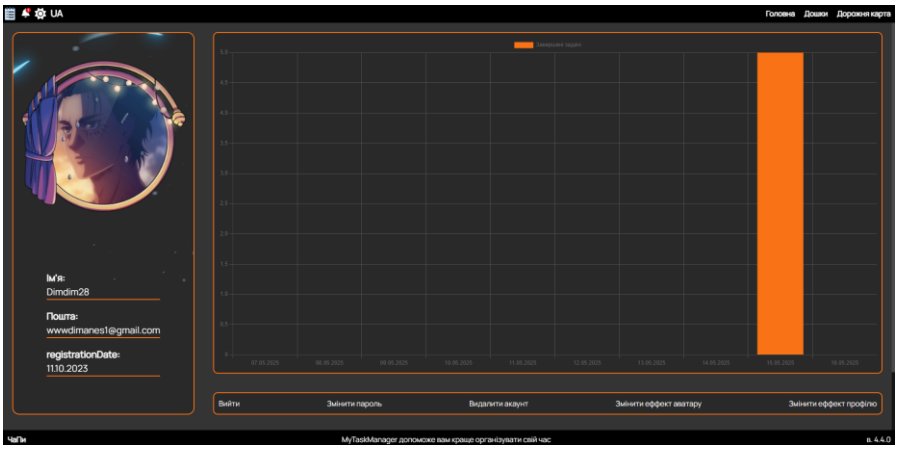


Рисунок 3.13 – Скріншот сторінки користувача

3.2.3 Усі канбан-дошки

Сторінка відповідає за усі канбан проєкти, у яких приймає участь користувач, при відкритті сторінки іде запит на сервер за допомогою fetchAllCanBanBoards. Запит повертає за раз максимум десять дошок, якщо їх стільки, що з’являється прокрутка на екрані, про прокручуванню вниз буде завантажуватися наступна сторінка. Якщо прокрутка не з’явилася через великий екран пристрою, з’явиться кнопка завантажити ще внизу. Логіку завантаження дошок зображено на рисунку 3.14.

```
useEffect(() => {
  dispatch(fetchAllCanBanBoards());
}, []);

useEffect(() => {
  const wrapper = wrapperRef.current;
  if (!wrapper) return;

  const hasScroll = wrapper.scrollHeight > wrapper.clientHeight;
  setShowLoadMoreButton(!hasScroll && currentPage < totalPages);
}, [allProjects, currentPage, totalPages]);

const handleCategoriesScroll = async (e: UIEvent<HTMLElement>) => {
  const { scrollHeight, scrollTop, clientHeight } = e.currentTarget;
  const threshold = 100;
  const isScrolled = scrollTop + clientHeight >= scrollHeight - threshold;

  if (status !== Status.LOADING && currentPage < totalPages && isScrolled) {
    dispatch(fetchAllCanBanBoards(currentPage + 1));
  }
};

if (status === Status.LOADING) return <Preloader />;

if (status === Status.ERROR) {
  return <div className={styles.error}>{errorMessage}</div>;
}
```

Рисунок 3.14 – Логіка завантаження канбан дошок

У кожній карточці-проєкті є лаконічна інформація, виділено чи це власний проєкт користувача, чи він був доданий. Є можливість видалити або редагувати (для власника) проєкт, або вийти з проєкту (якщо було додано іншим користувачем). Логіку сітки з задачами зображено на скріншоті 3.15.

```
{allProjects.length === 0 ? (
  <div className={styles.info}>{t('noCanBanBoards')}</div>
) : (
  <div className={styles.cards}>
    {allProjects.map((el) => (
      <div
        className={`${styles.card} ${currentUserProfile?._id === el.creatorId ? styles.ownerCard : null}`}
        key={el._id}
        onClick={() => navigate(`${ROUTES.CanBan}/${el._id}`)}
      >
        <div className={styles.title}>{el.title}</div>
        <div className={styles.description}>{el.description}</div>
        <div className={styles.updatedAt}>{t('updatedLabel')}: {getRelativeTime(el.updatedAt, t)}</div>
        <div className={styles.members}>{t('members')}: {el.membersCount || 1}</div>
        {currentUserProfile?._id === el.creatorId ? (
          <>
            <FontAwesomeIcon className={styles.crown} icon={faCrown} />
            <FontAwesomeIcon className={styles.deleteIcon}
              onClick={(e) => {
                e.stopPropagation();
                handleDeleteProjectClick(el._id, el.title);
              }}
              fontSize="20px"
              icon={faTrash}
            />
          </>
        ) : (
          <FontAwesomeIcon
            className={styles.deleteIcon}
            onClick={(e) => {
              e.stopPropagation();
              handleExitProjectClick(el._id, el.title);
            }}
            fontSize="20px"
            icon={faDoorOpen}
          />
        )}
      </div>
    ))}
  </div>
)}
```

Рисунок 3.15 – Логіка завантаження канбан дошок

Скріншот сторінки зображено на рисунку 3.16. Оскільки проєктів, до яких залучений користувач, не так і багато, кнопки щоб підвантажити ще проєкти немає. Візуально виділено проєкт, який користувач створив сам. При натисканні на картку проєкту користувач перенаправляється на сторінку проєкту.

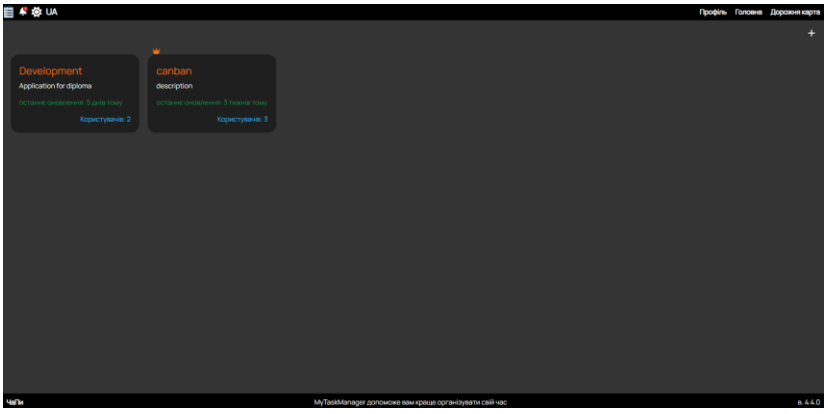


Рисунок 3.16 – Скріншот сторінки усіх канбан дошок

Логіка сторінки з усіма проєктами дорожніми картами повністю ідентична, відрізняються лише адреси запитів на сервер. На рисунку 3.17 зображено вигляд цієї сторінки.

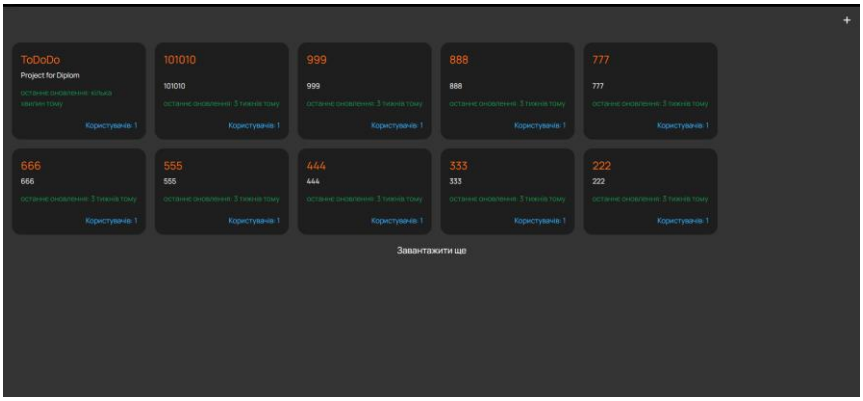


Рисунок 3.17 – Скріншот сторінки усіх дорожніх карт

3.2.4 Канбан-дошка

Сторінку реалізовано як складну drag-and-drop систему для керування задачами, всі дані при проєкту отримуються через fetchCanBanBoardById запит при відкритті сторінки. На рисунку 3.18 зображено логіку отримання

даних про проєкт та перевірку на те, чи поточний користувач є власником проєкту.

```
const { id: boardId } = useParams();

const currentUserProfile = useAppSelector(selectProfile);
const creatorId = useAppSelector(selectCanBanCreatorId);

useEffect(() => {
  if (boardId) {
    dispatch(fetchCanBanBoardById(boardId));
  }
}, [dispatch, boardId]);

if (canBanStatus === Status.LOADING) {
  return <Preloader />;
}

const isCreator = currentUserProfile?._id === creatorId;
```

Рисунок 3.18 – Логіка завантаження інформації про проєкт

Реалізовано всі функції для роботи з колонками, задачами, тегами, користувачами: створення, видалення або редагування. У випадку з користувачами їх може додавати на проєкт власник проєкту. До задачі, як виконавця, може додати будь хто з доданих до проєкту людей і будь кого. Оскільки це Drag and drop система, при перетягуванні задач між колонками необхідно автоматично прокручувати сторінку в сторону, в яку користувач тягне. На рисунку 3.19 зображено логіку автоматичної прокрутки проєкту по горизонталі при перетягуванні задачі між колонками в середині проєкту.

```
const handleBoardDragOver = (e: React.DragEvent<HTMLDivElement>) => {
  e.preventDefault();
  if (!boardRef.current) return;
  const board = boardRef.current;
  const { left, right } = board.getBoundingClientRect();
  const mouseX = e.clientX;

  if (mouseX - left < SCROLL_THRESHOLD) {
    board.scrollLeft -= SCROLL_SPEED;
  } else if (right - mouseX < SCROLL_THRESHOLD) {
    board.scrollLeft += SCROLL_SPEED;
  }
};

const handleBoardDrop = (e: React.DragEvent<HTMLDivElement>) => {
  e.preventDefault();
  setIsDragging(false);
  setDraggingTaskId(null);
};
```

Рисунок 3.19 – Логіка автопрокрутки колонок

Аналогічна логіка для прокрутки по вертикалі існує у кожній колонці. Реалізовано її у компоненті Column. На рисунку 3.20 зображено скріншот сторінки Канбан-проєкту.

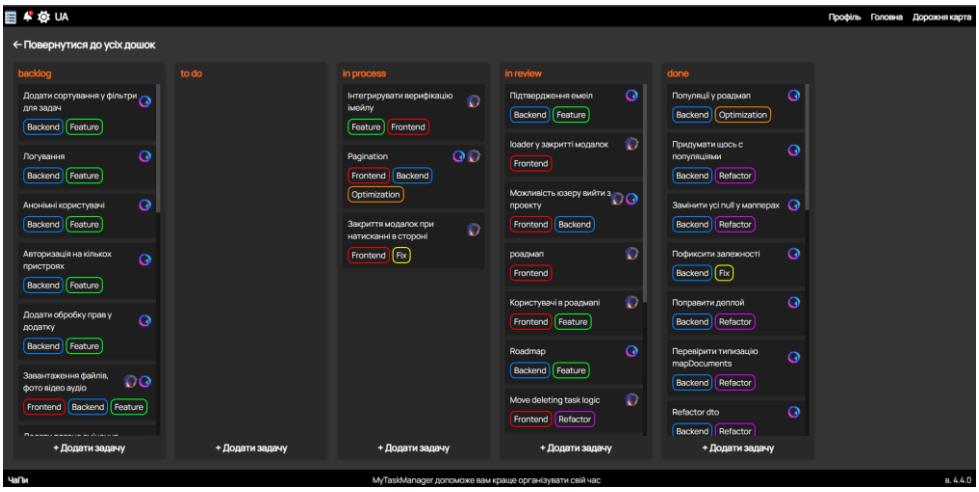


Рисунок 3.20 – Сторінка канбан проєкту

При натисканні на задачу відкриється бокова панель з повною інформацією та функціями редагування задачі. Логіка стану обробляється як і у усіх модальних вікнах за допомогою useState. Скріншот бокової панелі для редагування задачі зображено на рисунку 3.21.

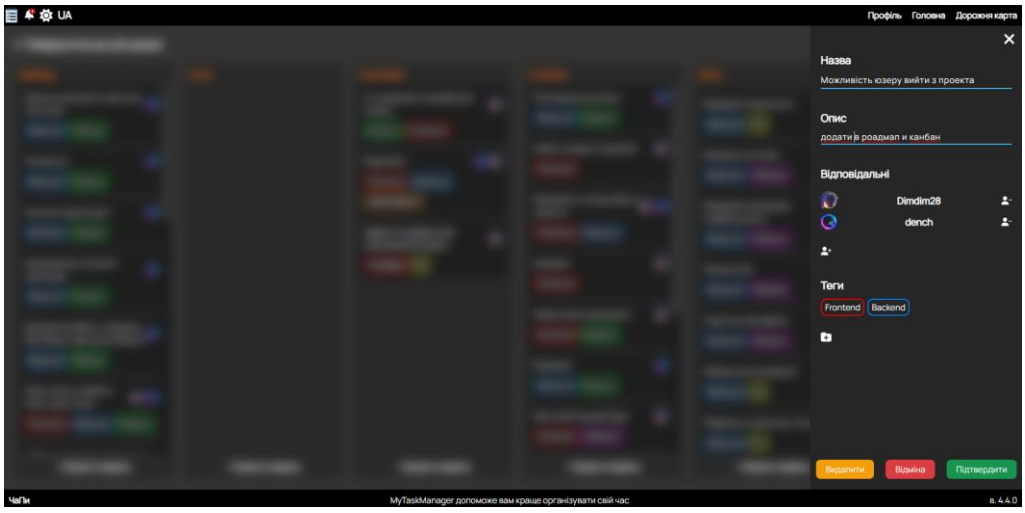


Рисунок 3.21 – Бокова панель для редагування задачі

У цій панелі можна змінити відповідальних за задачу, змінити теги, назву та опис завдання. Також реалізовано кнопку для видалення задачі та кнопку підтвердити або відмінити зміни. Ця бокова панель використовується також

при створенні нової задачі, для цього при її відкритті дістається з глобального стану інформація про обрану задачу, якщо користувач натиснув на одну з них раніше. Якщо ж інформації там немає, поля вводу та обрані теги і користувачі будуть порожніми, і при натисканні на кнопку замість запиту на оновлення задачі відбудеться запит на створення. Логіка діставання інформації зі стану зображена на рисунку 3.22.

```
const TaskInfoSideBar = () => {
  const isSideBarOpened = useAppSelector(selectIsTaskInfoSideBarOpened);
  const taskInfo = useAppSelector(selectSelectedTask);
  const members = useAppSelector(selectProjectMembers);
  const allTags = useAppSelector(selectProjectTags);
  const projectInfo = useAppSelector(selectIsProjectInfo);

  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [assigners, setAssigners] = useState<User[]>([]);
  const [tags, setTags] = useState<Tag[]>([]);
  const [isAddAssignerMenuOpened, setIsAddAssignerMenuOpened] = useState(false);
  const [isAddTagMenuOpened, setIsAddTagMenuOpened] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState('');

  const handleCLoseSideBar = () => {
    dispatch(setIsTaskInfoModalOpened(false));
    dispatch(setSelectedTask({ task: null }));
  };

  const handleEscKey = useCallback(
    (event: KeyboardEvent) => {
      if (event.key === KEY_NAME_ESC) {
        dispatch(setIsTaskInfoModalOpened(false));
        dispatch(setSelectedTask({ task: null }));
      }
    },
    [dispatch],
  );

  useEffect(() => {
    if (!taskInfo.task) {
      setTitle('');
      setDescription('');
      setAssigners([]);
      setTags([]);
    } else {
      setTitle(taskInfo.task.title);
      setDescription(taskInfo.task.description);
      setAssigners(taskInfo.task.assignees || []);
      setTags(taskInfo.task.tags);
    }
    setIsAddAssignerMenuOpened(false);
    setIsAddTagMenuOpened(false);
    setError('');
    setIsLoading(false);
  }, [taskInfo]);
```

Рисунок 3.22 – Отримання даних задачі для редагування

3.2.5 Дорожні карти (Roadmap)

Сторінка дорожньої карти дає користувачу можливість створити категорії, обравши для них колір та назву, всередині кожної категорії можна створити скільки необхідно рядків, що ідуть паралельно і у кожному з них модна створювати задачі, що не перетинаються. Також користувач може створювати квартали, що знаходяться вгорі сторінки. Видаляти квартали можна тільки з кінця, і при видаленні кварталу автоматично видаляються всі задачі, що було розпочато у цьому кварталі, а також автоматично відредаговано всі задачі, що почалися раніше але робота над ними йшла у видаленому кварталі. Їх буде автоматично скорочено, щоб вони закінчилися до початку нового кварталу. Також у верхній частині екрану присутня лінія з віхами, що дозволяє створювати подвійним натисканням миші у ній віху, також переміщати її, перейменовувати або видаляти. Задачі у рядках можна переносити у вільне місце через drag-and-drop, зажаттям лівої або правої сторінки переміщати початок та кінець задачі, а також аналогічно і перетягувати прогрес виконання від 0 до 100% у кожній задачі. Кожну задачу також можна редагувати або видаляти. Задачі створюються у кожному рядку подвійним натисканням і місце натискання буде початком задачі, якщо вистачає місця. На рисунку 3.23 зображено вигляд дорожньої карти.

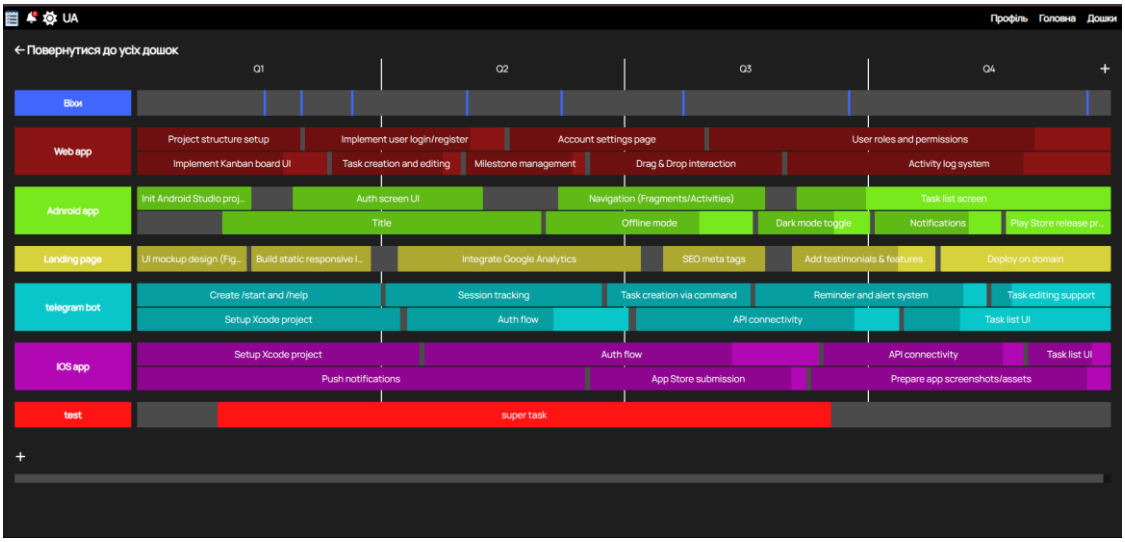


Рисунок 3.23 – Вигляд дорожньої карти

Головною вимогою до цієї сторінки є те, що залежно від розміру екрану необхідно або розтягнути існуючі квартали на всю ширину екрану, або, якщо екран надто малий для такої кількості кварталів, зробити прокрутку по горизонталі. При зміні розміру екрану необхідно перерахувати положення кварталів та задач. Логіку зміни ширини зображено на рисунку 3.24.

```
const updateWidth = () => {
  if (!roadmapRef.current) return;
  const fullWidth = roadmapRef.current.clientWidth;
  const contentWidth = fullWidth - 200 - 10;
  setRoadmapContentWidth(contentWidth);
};

useEffect(() => {
  const fetchBoardData = async (boardId: string) => {
    setIsLoading(true);
    const res = await roadmapAPI.getBoard(boardId);

    if (res.status === Status.SUCCESS) {
      dispatch(updateRoadmapData(res.data));
      setErrorMessage('');
      setIsLoading(false);
      requestAnimationFrame(updateWidth);
    } else {
      setErrorMessage(res.message);
      setIsLoading(false);
    }
  };

  if (boardId) {
    fetchBoardData(boardId);
  }
}, [boardId, dispatch]);

useEffect(() => {
  const frame = requestAnimationFrame(updateWidth);
  window.addEventListener('resize', updateWidth);

  return () => {
    cancelAnimationFrame(frame);
    window.removeEventListener('resize', updateWidth);
  };
}, []);

useEffect(() => {
  if (!isLoading && data?.quarters.length) {
    requestAnimationFrame(updateWidth);
  }
}, [isLoading, data?.quarters.length]);
```

Рисунок 3.24 – Логіка зміни ширини дорожньої карти

Для позиціонування задач та віх було зроблено наступне припущення. Кожний квартал було прийняти за 100 одиниць. У кожного рядка у стилях

було прописано position: relative; Відповідно кожній задачі було надано для абсолютного позиціонування position: absolute, top: 0, height: 100%.

Координати початку та кінця передаються як параметри у компоненту Task та додаються як inline стилі, що змінюються після дій користувача. Приклад логіки позиціонування задачі представлено на рисунку 3.25.

Рисунок 3.25 – Позиціонування задач

Таким чином задачі розташовуються абсолютно у відсотках та адаптивно змінюють свій розмір залежно від розміру екрану та кількості кварталів. Тому не має необхідності писати додаткову логіку для зміни координат задачі при зміні розміру екрану, достатньо змінити ширину на всій компоненті Roadmap, після чого задачі, віхи та рядки автоматично змінять розмір. Важливим елементом сторінки є можливість рухати початок та кінець задачі, або перетягувати її на інше місце. При натисканні мишкою на лівий або правий край задачі за допомогою обробника onMouseDown={(e) => onResizeStart(e, 'start')}викликається функція, наведена на рисунку 3.26.

```
const onResizeStart = (e: React.MouseEvent, side: 'start' | 'end') => {
  e.stopPropagation();
  e.preventDefault();
  resizingRef.current = { side, initialX: e.clientX, initialValue: localTask[side], isResizing: true };
  document.addEventListener('mousemove', onResizeMove);
  document.addEventListener('mouseup', onResizeEnd);
};
```

Рисунок 3.26 – Створення обробників для зміни координат задачі

На рисунку 3.27 зображено логіку зміни координати задачі при пересуванні миші із зажатою клавішею на краю задачі, якщо є достатньо місця.

```
const onResizeMove = (e: MouseEvent) => {
  const info = resizingRef.current;
  if (!info || !info.isResizing) return;
  e.preventDefault();
  const deltaPx = e.clientX - info.initialX;
  const deltaValue = (deltaPx / roadmapContentWidth) * (totalQuarters * 100);
  const newValue = Math.round(info.initialValue + deltaValue);
  const maxValue = totalQuarters * 100;
  if (newValue < 0 || newValue > maxValue) return;
  if (info.side === 'start') {
    const proposedStart = newValue;
    const proposedEnd = localTask.end;
    if (proposedEnd - proposedStart < 10) return;
    const isOverlapping = allTasksInRow.some((t) => {
      if (t._id === localTask._id) return false;
      return !(proposedEnd + 2 <= t.start || proposedStart >= t.end + 2);
    });
    if (isOverlapping) return;
    setLocalTask((prev) => {
      const updated = { ...prev, start: proposedStart };
      localTaskRef.current = updated;
      return updated;
    });
  }
  if (info.side === 'end') {
    const proposedStart = localTask.start;
    const proposedEnd = newValue;
    if (proposedEnd - proposedStart < 10) return;
    const isOverlapping = allTasksInRow.some((t) => {
      if (t._id === localTask._id) return false;
      return !(proposedEnd + 2 <= t.start || proposedStart >= t.end + 2);
    });
    if (isOverlapping) return;
    setLocalTask((prev) => {
      const updated = { ...prev, end: proposedEnd };
      localTaskRef.current = updated;
      return updated;
    });
  }
};
```

Рисунок 3.27 – Зміна координат задачі, якщо є місце

При відтисканні клавіши миші спрацьовує як обробник подій функція onResizeEnd, що відправить запит на сервер. Це гарантує, що буде відправлено лише один запит і не буде зайве навантаження на сервер, бо обробник події mousemove відпрацьовував би надто часто. На рисунку 3.28 зображено реалізацію цієї функції.

```
const onResizeEnd = async () => {
  if (resizingRef.current) resizingRef.current.isResizing = false;
  document.removeEventListener('mousemove', onResizeMove);
  document.removeEventListener('mouseup', onResizeEnd);
  dispatch(
    updateRoadmapTaskInCategory({
      categoryId,
      rowId,
      taskId: localTask._id,
      updates: {
        start: localTaskRef.current.start,
        end: localTaskRef.current.end,
      },
    }),
  );
  await roadmapAPI.updateTask({
    roadmapId,
    categoryId,
    rowId,
    taskId: localTask._id,
    start: localTaskRef.current.start,
    end: localTaskRef.current.end,
  });
};
```

Рисунок 3.28 – Логіка onResizeEnd

3.2.6 Сторінки авторизації та реєстрації (Login, Register)

Вони працюють по схожому принципу, з кожної сторінки можна переключитися на іншу кнопкою внизу. Також реалізовано авторизацію через Google. На рисунку 3.29 зображено сторінку авторизації.

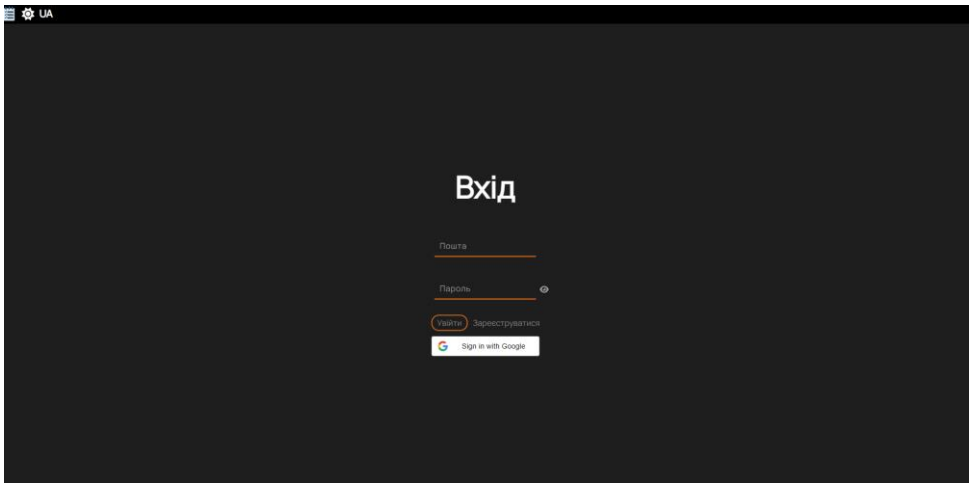


Рисунок 3.29 – Сторінка авторизації

У роботі цієї сторінки використовується бібліотека Formik, що дозволяє зручно валідувати поля, перевіряти наявність тексту у кожному полі та його, довжину, тощо. Приклад логіки зображено на рисунку 3.30.

```tsx
const Login: FC = () => {
  const dispatch = useAppDispatch();

  const [error, setError] = useState<string | null>(null);

  const { t } = useTranslation();

  const formik = useFormik({
    initialValues: {
      email: '',
      password: '',
    },
    validate,
    onSubmit: async (values, { setSubmitting }) => {
      const data: any = await dispatch(fetchUserData(values));
      if (data.error) setError(data.payload);
      if ('token' in data.payload) {
        window.localStorage.setItem('token', data.payload.token);
        socketsAPI.init(data.payload.token);
      }
      formik.resetForm();
      setSubmitting(false);
    },
  });

  return (
    <main>
      <div className={styles.wrapper}>
        <h1>{t('signInBold')}</h1>
        <form onSubmit={formik.handleSubmit}>
          <div className={styles.fieldsWrapper}>
            <FormikInput
              name="email"
              type="email"
              onChange={formik.handleChange}
              onBlur={formik.handleBlur}
              value={formik.values.email}
              title={t('email')}
            />
            {formik.errors.email && formik.touched.email ? (
              <p>{formik.errors.email}</p>
            ) : null}
          </div>
```

Рисунок 3.30 – Робота з Formik

При відправці форми іде запит на сервер, при успішному результаті якого у локальне сховище буде збережено токен користувача, що буде використовуватись при усіх наступних запитах на сервер.

Кнопку входу/реєстрації через Google було зроблено окремим компонентом, оскільки вона може бути присутня у кількох місцях. На рисунку 3.31 зображено логіку цієї компоненти.

```
import { Dispatch, FC, SetStateAction } from 'react';
import { useGoogleLogin } from '@react-oauth/google';

import socketsAPI from '../../api/socketsAPI';
import { useAppDispatch } from '../../hooks';
import { fetchAuthMe, fetchGoogleUser } from '../../redux/slices/auth/thunk';

import './styles.scss';

interface GoogleLoginProps {
  setError: Dispatch<SetStateAction<string | null>>;
}

const GoogleLogin: FC<GoogleLoginProps> = ({ setError }) => {
  const dispatch = useAppDispatch();

  const login = useGoogleLogin({
    flow: 'auth-code',
    onSuccess: async (codeResponse) => {
      const authCode = codeResponse.code;
      const data: any = await dispatch(fetchGoogleUser(authCode));

      if (data.error) {
        setError(data.payload);
      }

      if ('token' in data.payload) {
        window.localStorage.setItem('token', data.payload.token);
        socketsAPI.init(data.payload.token);
        dispatch(fetchAuthMe());
      }
    },
    onError: () => {
      setError('Google login failed');
    },
  });

  return (
    <button onClick={() => login()} className="gsi-material-button">···
    </button>
  );
};

export default GoogleLogin;
```

Рисунок 3.31 – Логіка компоненти GoogleLogin

3.3 Побудова глобального стану через Redux Toolkit

Для глобального стану додатку використовується бібліотека Redux toolkit. Глобальний стан додатку розділений на шари. Auth, home, profile, canban, roadmap. Приклад створення глобального сховища наведено на рисунку 3.32.

```
import { authReducer } from './slices/auth/auth';
import { canBanReducer } from './slices/canban/canban';
import { homeReducer } from './slices/home/home';
import { profileReducer } from './slices/profile/profile';
import { roadmapReducer } from './slices/roadmap/roadmap';

const store = configureStore({
  reducer: {
    auth: authReducer,
    home: homeReducer,
    profile: profileReducer,
    canban: canBanReducer,
    roadmap: roadmapReducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({ serializableCheck: false }),
});

export default store;
```

Рисунок 3.32 – Логіка створення глобального стану

3.3.1 Структура сховища (auth, home, profile, canban, roadmap)

Шар auth відповідає за дані авторизованого користувача, налаштування теми та мову. На рисунку 3.33. зображено початковий стан та приклад редьюсерів.

```
const initialState: AuthSliceState = {
  profile: null,
  status: Status.LOADING,
  theme: (localStorage.getItem('theme') as Theme) || Theme.DARK,
  lang: (localStorage.getItem('lang') as Language) || Language.EN,
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    logout(state) {
      state.profile = null;
      window.localStorage.removeItem('token');
      window.localStorage.removeItem('theme');
      window.localStorage.removeItem('lang');
    },
    changeTheme(state, action) {
      state.theme = action.payload;
      window.localStorage.setItem('theme', action.payload);
    },
    changeLang(state, action) {
      state.lang = action.payload;
      window.localStorage.setItem('lang', action.payload);
    },
  },
```

Рисунок 3.33 – Шар auth

Шар home відповідає за домашню сторінку, категорії та персональні задачі користувача. На рисунку 3.34. зображено початковий стан та приклад редьюсерів.

```
const initialState: HomeSliceState = {
  category: {
    categories: [],
    totalPages: 0,
    currentPage: 1,
    status: Status.LOADING,
  },
  task: {
    tasks: [],
    totalPages: 0,
    currentPage: 1,
    searchPattern: '',
    isCompleted: 'false',
    date: 'all',
    activeCategories: [],
    status: Status.LOADING,
  },
};

const homeSlice = createSlice({
  name: 'home',
  initialState,
  reducers: {
    clear: () => initialState,
    clearCategories(state) {
      state.category.categories = [];
    },
```

Рисунок 3.34 – Шар home

Шар profile відповідає за сторінку профілю користувача. На рисунку 3.35. зображено початковий стан та приклад редьюсерів.

```
export type DailyStats = {
  date: string;
  counter: number;
};

export interface ProfileSliceState {
  status: Status;
  data: Profile | null;
  message?: string;
  stats: DailyStats[];
}

export type Profile = {
  _id: string;
  createdAt: string;
  email: string;
  token: string;
  updatedAt: string;
  username: string;
  avatar: string;
  isBanned?: boolean;
  profileEffect: ProfileEffect;
  avatarEffect: AvatarEffect;
};

const initialState: ProfileSliceState = {
  data: null,
  status: Status.LOADING,
  stats: [],
};

const profileSlice = createSlice({
  name: 'profile',
  initialState,
  reducers: {
    exit(state) {
      state.data = null;
    },
    clearProfileErrorMessage(state) {
      state.message = '';
    },
```

Рисунок 3.35 – Шар profile

Шар canban відповідає за сторінку канбан дошки. На рисунку 3.36. зображено початковий стан.

```
const initialColumnsState: CanBanState = {
  columns: [],
  members: [],
  selectedTask: { task: null },
  isChangeColumnNameModalOpen: false,
  isDeleteColumnModalOpen: false,
  isDeleteTaskModalOpen: false,
  processingColumnData: null,
  isTaskInfoSideBarOpened: false,
  isProjectSettingsOpened: false,
  isAddUserToProjectModalOpened: false,
  isAddTagToProjectModalOpened: false,
  selectedTag: { tag: null },
  info: null,
  tags: [],
  allProjects: [],
  creatorId: null,
  currentPage: 0,
  totalPages: 0,
};

const initialState: CanBanSliceState = {
  data: initialColumnsState,
  status: Status.SUCCESS,
};

const canBanSlice = createSlice({
  name: 'canban',
  initialState,
  reducers: {
```

Рисунок 3.36 – Шар canban

Шар roadmap відповідає за сторінку дорожньої карти. На рисунку 3.37. зображено початковий стан та приклад редьюсерів.

```
const initialState: RoadmapSliceState = {
  data: null,
  currentCategory: null,
  currentRow: null,
  currentMilestone: null,
  currentQuarter: null,
  currentTask: null,
  isDeletingCategoryOpened: false,
  isEditingCategoryOpened: false,
  isDeletingRowOpened: false,
  isDeletingQuarterModalOpened: false,
  isEditingMilestoneModalOpened: false,
  isDeletingMilestoneModalOpened: false,
  isEditingTaskModalOpened: false,
  isDeletingTaskModalOpened: false,
  clickPosition: 0,
};

const canBanSlice = createSlice({
  name: 'roadmap',
  initialState,
  reducers: {
    updateRoadmapData: (state, action: PayloadAction<RoadmapData>) => {
      state.data = action.payload;
    },
```

Рисунок 3.37 – Шар roadmap

3.3.2 Використання useAppDispatch та useAppSelector

Було створено типізовану версію хука useDispatch, яку зображено на рисунку 3.38.

```
export type AppDispatch = typeof store.dispatch;
export const useAppDispatch: () => AppDispatch = useDispatch;
```

Рисунок 3.38 – Типізація useAppDispatch

Цей хук дозволяє викликати дії, описані у редьюсері будь-якого шару. Доступ до функції dispatch відбувається наступним чином: const dispatch = useAppDispatch(); Приклад виклику зображено на рисунку 3.39. Метод setActive, що викликається при спробі закрити модального вікна, в даному випадку очищує стан категорії та рядка, а також змінює стан чи відкрите модальне вікно на false.

```
<Modal
  active={categoryDeleting}
  setActive={() => {
    dispatch(setRoadmapCurrentCategory(null));
    dispatch(setRoadmapIsDeletingCategoryOpened(false));
  }}
  ChildComponent={CategoryDeleting}
  childProps={{ roadmapId: data._id }}
/>

<Modal
  active={rowDeleting}
  setActive={() => {
    dispatch(setRoadmapCurrentRow(null));
    dispatch(setRoadmapIsDeletingRowOpened(false));
  }}
  ChildComponent={RowDeleting}
  childProps={{ roadmapId: data._id }}
/>
```

Рисунок 3.39 – Використання useAppDispatch

Функції-селектори необхідні для зручного діставання з глобального стану фрагментів даних. Приклад селекторів для шару дорожньої карти наведено на рисунку 3.40.

```
import { RootState } from '../../store';

export const selectRoadmapData = (state: RootState) => state.roadmap.data;
export const selectRoadmapCurrentCategory = (state: RootState) =>
  state.roadmap.currentCategory;
export const selectRoadmapCurrentRow = (state: RootState) =>
  state.roadmap.currentRow;
export const selectRoadmapCurrentQuarter = (state: RootState) =>
  state.roadmap.currentQuarter;
export const selectRoadmapCurrentMilestone = (state: RootState) =>
  state.roadmap.currentMilestone;
export const selectRoadmapCurrentTask = (state: RootState) =>
  state.roadmap.currentTask;
export const selectRoadmapClickPosition = (state: RootState) =>
  state.roadmap.clickPosition;
```

Рисунок 3.40 – Створення селекторів

Використовуються селектори наступним чином:

const currentQuarter = useAppSelector(selectRoadmapCurrentQuarter);

3.4 Робота із зовнішнім API

Для зручності було використано класовий підхід, усі функції для запитів на сервер були об’єднані у класи за зоною відповідальності. Окремо всі методи для роботи з дорожньою картою, окремо для роботи з задачами, окремо для канбан-дошок, тощо. Приклад такого класу зображено на рисунку 3.41.

```
import instanse from '../axios';
import { Status } from '../types/shared';

type banUserStatusResponse = {
  status: Status;
  message?: string;
};

class adminAPIClass {
  public async banUser(
    userId: string,
    isBanned: boolean,
  ): Promise<banUserStatusResponse> {
    try {
      await instanse.patch(`admin/user/${userId}/banStatus`, { isBanned });
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        message: err?.response?.data?.message || 'Error',
        status: Status.ERROR,
      };
    }
  }
}

const adminAPI = new adminAPIClass();
export default adminAPI;
```

Рисунок 3.41 – Клас для роботи з API

3.4.1 Організація доступу до API

Оскільки усі методи це публічні поля класу, можна дуже зручно експортувати отриманий в результаті екземпляр об’єкту з усіма методами викликати будь-який з них. Це було зроблено, бо експортувати кожний метод було б не зручно, а експортувати клас і створювати його екземпляр всюди було б ще довше. Можна було використати статичні поля, і тоді не мало б сенсу створювати екземпляр, а викликати методи напряму з класу, але оскільки екземпляр створюється 1 раз і експортується з файлу, різниці тут небагато. Далі ці методи можна викликати або з локальних компонент, або з асинхронної thunk функції.

3.4.2 Локальне використання API у компонентах

Цей варіант ідеальний для випадків, коли якась логіка відбувається всередині модального вікна, або наприклад змінюється задача у дорожній карті, при перенесенні її на іншу позицію має сенс кинути локально запит на сервер. На рисунку 3.42 зображено функцію для підтвердження видалення задачі.

```tsx
export const TaskDeleting: FC<TaskDeletingProps> = ({
  toggleActive,
  childProps,
}) => {
  const dispatch = useAppDispatch();
  const { t } = useTranslation();

  const currentTask = useAppSelector(selectRoadmapCurrentTask);
  const currentCategory = useAppSelector(selectRoadmapCurrentCategory);
  const currentRow = useAppSelector(selectRoadmapCurrentRow);

  const [isLoading, setIsLoading] = useState(false);
  const [message, setMessage] = useState('');

  const submit = async () => {
    if (!currentTask || !currentCategory || !currentRow) return;

    setIsLoading(true);
    const result = await roadmapAPI.deleteTask({
      roadmapId: childProps.roadmapId,
      categoryId: currentCategory._id,
      rowId: currentRow._id,
      taskId: currentTask._id,
    });

    if (result.status === Status.SUCCESS) {
      dispatch(
        deleteRoadmapTask({
          categoryId: currentCategory._id,
          rowId: currentRow._id,
          taskId: currentTask._id,
        }),
      );
      setIsLoading(false);
      toggleActive(false);
    } else {
      setMessage(result.message);
      setIsLoading(false);
    }
  };
```

Рисунок 3.42 – Локальне використання API

3.4.3 Використання AsyncThunk для глобальної роботи із даними

Якщо ж після запиту на сервер має суттєво оновитись глобальний стан додатку, має сенс використовувати createAsyncThunk для запиту на сервер, а також створити обробники статусу запиту до API, щоб залежно від цього автоматично оновлювався глобальний стан, як зображено на рисунку 3.43.

```
export const fetchAuthMe = createAsyncThunk<Profile>(
  'auth/fetchAuthMe',
  async () => {
    const { data } = await instanse.get('/user/me');
    return data;
  },
);
```

Рисунок 3.43 – Використання createAsyncThunk

На рисунку 3.44 зображено обробники, що змінюють глобальний стан та виконують додаткову логіку залежно від статусу запиту fetchAuthMe.

```
builder.addCase(fetchAuthMe.pending, (state) => {
  state.status = Status.LOADING;
  state.profile = null;
  state.message = '';
});
builder.addCase(fetchAuthMe.fulfilled, (state, action) => {
  state.profile = action.payload;
  state.status = Status.SUCCESS;
  state.message = '';
  socketsAPI.init(window.localStorage.getItem('token') || '');
});
builder.addCase(fetchAuthMe.rejected, (state, action) => {
  state.status = Status.ERROR;
  state.profile = null;
  state.message = String(action.payload);
});
```

Рисунок 3.44 – Обробники статусу fetchAuthMe

3.4.4 Причини використання різних підходів

У випадку з запитами, що змінюють глобальний стан додатку, особливо якщо це має суттєво поміняти шар (наприклад при авторизації) зручніше використати asyncThunk, якщо ж при зміні задачі у дорожній карті необхідно просто кинути запит на сервер, щоб оновити дані, що вже локально змінилися у користувача, тут достатньо локального запиту на API.

3.5 Робота зі стилями та підтримка тем оформлення

Стилізація була виконана у стилі SCSS модулів, тому файл зі стилями імпортується та використовується наступним чином:

import styles from “./Roadmap.modyle.scss”

Це гарантує, що зміна у стилях окремого модуля змінить тільки компоненту, що імпортує модуль. Такий код більш очікуваний і чистий.

3.5.1 Організація глобальних стилів та змінних теми

Глобальні стилі завантажуються як звичайний scss файл (не модуль) та імпортуються в файл App.tsx. Тут знаходяться стилі, що мають використовуватись по всьому додатку. Наприклад налаштування полос прокрутки та кольору фону сторінки. Кольорові змінні просто імпортуються у файл та використовуються через знак $, так само як і оголошуються через цей же знак: $background-dark-0: #1e1e1e.

```
@import 'partials/theme';
html,
body {
  background-color: $background-dark-0;
  color: $gray-600;
  overflow-x: hidden;
}

* {
  ::-webkit-scrollbar {
    width: 8px;
  }

  ::-webkit-scrollbar-track {
    background-color: $background-dark--10;
    transition: background-color $transition-delay;
  }
```

3.5.2 Користувацький Reset стилів

Також у файл App.tsx імпортується файл для очищення стилів, який прибирає усі відступи, border, тощо, які різні браузери по різному налаштовують за замовчуванням. Імпортується файл наступним чином: import './styles/reset.scss'; Приклад очищення стилів зображено на рисунку 3.45.

```scss
*,
*::before,
*::after {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

*:focus {
  outline: none;
  -webkit-box-shadow: none;
  -moz-box-shadow: none;
  box-shadow: none;
  resize: none;
}

a {
  text-decoration: none;
  color: inherit;
  cursor: pointer;
}

button {
  background-color: transparent;
  color: inherit;
  border-width: 0;
  padding: 0;
  cursor: pointer;
}
```

Рисунок 3.45 – Reset.scss

3.5.3 Перемикання світлої та темної теми

Для реалізації стилю світлої або темної теми document має клас, що міняється залежно від обраної теми.

```
const theme = useAppSelector(selectTheme);
document.documentElement.className = `${theme}_theme`;
```

А у кожному файлі стилів для світлої теми написано спеціальний селектор global(.light_theme), що змінює стилі компоненти як тут:

```
:global(.light_theme) .checkboxContainer {
  color: $black;

  .checkmark,
  .roundedCheckMark {
    border: 2px solid $black;
  }
}
```

3.6 Реалізація мультимовності через i18next

Мультимовність реалізована за допомогою бібліотеки i18next, реалізовано переклади на українську і англійську мови.

3.6.1 Структура перекладів

Існує два файли у папці lang, відповідно UA.ts та EN.ts. Структура перекладів зображена на рисунку 3.46. Можна як писати усі рядки на одному рівні, так і створювати вкладені структури, групуючи текст за сенсом.

```
  tags: 'Tags',
  noColumns: 'There are no columns in project right now',
  deleteTask: 'Delete',
  noFreeMembers: 'No free users',
  backToAllBoards: 'Back to all boards',
  noRoadmapBoards: 'You have no Roadmap boards',
  updatedLabel: 'Last update',
  updated: {
    justNow: 'just now',
    fewMinutesAgo: 'a few minutes ago',
    minutesAgo: '{{count}} minutes ago',
    anHourAgo: 'an hour ago',
    hoursAgo: '{{count}} hours ago',
    today: 'today',
    yesterday: 'yesterday',
    daysAgo: '{{count}} days ago',
    aWeekAgo: 'a week ago',
    weeksAgo: '{{count}} weeks ago',
  },
  loadMore: 'Load more',
};

export default EN;
```

Рисунок 3.46 – Структура перекладів

3.6.2 Підключення та використання i18next

У App.tsx підключається та ініціалізується бібліотека, для цього необхідно вказати поточну мову, а також вказати шлях до усіх існуючих файлів з перекладами. Ініціалізацію бібліотеки зображено на рисунку 3.47.

```
import TRANSLATIONS from './lang';

i18next.use(initReactI18next).init({
  lng: localStorage.getItem('lang') || Language.EN,
  debug: true,
  resources: TRANSLATIONS,
  fallbackLng: 'en',
});
```

Рисунок 3.47 – Ініціалізація i18Next

3.6.3 Використання локалізованих рядків у компонентах

Для використання локалізації у компоненті необхідно імпортувати хук useTranslation та використати його наступним чином:

import { useTranslation } from 'react-i18next';

const { t } = useTranslation();

А далі у місці, де має бути перекладений текст, викликати функцію t з переданим ключем. Приклад наведено на рисунку 3.48.

```
<div className={styles.updatedAt}>
  {t('updatedLabel')}: {getRelativeTime(el.updatedAt, t)}
</div>

<div className={styles.members}>
  {t('members')}: {el.membersCount || 1}
</div>
```

Рисунок 3.48 – Використання i18Next

3.7 Забезпечення адаптивності інтерфейсу

Залежно від пристрою користувача має змінюватися зовнішній вигляд сторінок. У кожній компоненті для цього є медіа запити, а також реалізовано хак для стабілізації висоти у мобільних браузерах, наприклад від Facebook.

3.7.1 Стабілізація висоти застосунку у мобільних браузерах

Height: 100vh дуже часто не працює, особливо у мобільних браузерах від різних компаній, які реалізують обробку висоти кожен різним методом. Для виправлення цього при відкритті застосунку або зміні висоти екрану автоматично рахується висота екрану у пікселях та зберігається у глобальниц стан як змінна стилю як наведено на рисунку 3.49.

```
function App() {
  const dispatch = useAppDispatch();

  const appHeight = () => {
    const doc = document.documentElement;
    doc.style.setProperty('--app-height', `${window.innerHeight}px`);
  };

  window.onload = function () {
    appHeight();
  };

  useEffect(() => {
    window.addEventListener('resize', appHeight);
    window.addEventListener('load', appHeight);
    appHeight();
  }, []);
```

Рисунок 3.49 – Обчислення висоти екрану

Після чого усі обгортки сторінок отримують таку висоту за допомогою стилю height: var(--app-height), та можливість вертикальної прокрутки всередині за необхідності.

3.7.2 Реалізація адаптивної верстки за допомогою SCSS-модулів

Адаптивна верстка реалізована за допомогою медіа запитів, які залежно від ширини екрану змінюють стилі. Може змінитися розмір тексту, відступи, позиціонування елементів (в колонку а не в рядок), тощо. Приклад медіа запиту зображено на рисунку 3.50.

```
@media screen and (max-width: 678px) {
  margin-top: 10px;
  h3 {
    margin-bottom: 10px;
    font-size: 20px;
  }
  h5 {
    font-size: 16px;
  }
  .deadline {
    margin: 10px 0;
  }
}
```

Рисунок 3.50 – Приклад медіа запиту

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було описано реалізацію повноцінного SPA-додатку з використанням стеку сучасних технологій. Фронтенд реалізовано на основі React 18 з підтримкою TypeScript, збірка виконується за допомогою Vite.

Для керування глобальним станом застосовано Redux Toolkit у поєднанні з Redux Thunk і Reselect. Глобальний стан розділений на шари для зручності та принципу єдиної відповідальності. Сторінки розділені через React Router v6, а дані обробляються через окремі API-класи. Залежно від ситуації використовуються або запити на API разом з локальним станом, або використовується createAsyncThunk.

Інтерфейс стилізовано за допомогою SCSS-модулів для меншої кількості випадкових змін стилів. Є підтримка світлої та темної тем та адаптація для мобільних пристроїв. Для забезпечення різних мов використано i18next, що дозволяє реалізувати шаблонізацію та перемикання мов.

Додаток має компонентну архітектуру з чітким розділенням логіки та візуальної частини. Використано хак із window.innerHeight для стабільної роботи на мобільних браузерах. Завдяки такій структурі проєкт є масштабованим, зручним у підтримці, має високу продуктивність та повністю відповідає сучасним вимогам до UI/UX та архітектури клієнтських застосунків.

РОЗДІЛ 4
ТЕСТУВАННЯ ДОДАТКУ

4.1 Тестування продуктивності та швидкості завантаження

Тестування відбувається у режимі комп'ютера та телефона за 4 критеріями:

- Продуктивність та оптимізація (час завантаження сторінки та задіяна для цього пам'ять пристрою)
- Доступність (наявність усіх необхідних атрибутів та налаштувань для роботи додаткових функцій, що корисно для людей з обмеженнями)
- Кращі практики (перевірка відповідності коду останнім стандартам)
- SEO (пошукова оптимізація, перевірка структури, мета-тегів, тощо)

На рисунку 4.1 зображено результати тестування для режиму пк. Всі характеристики мають значення 90+. Для такого об'ємного додатку продуктивність 98 зі 100 це ідеальний показник (для прикладу у Jira це значення нечасто піднімається до 90, у середньому 80-85). SEO показник максимально можливий. Було приділено багато уваги налаштуванню мета тегам та семантичній верстці. Це дозволить платформі з'являтися на перших місцях при пошуку у браузері.

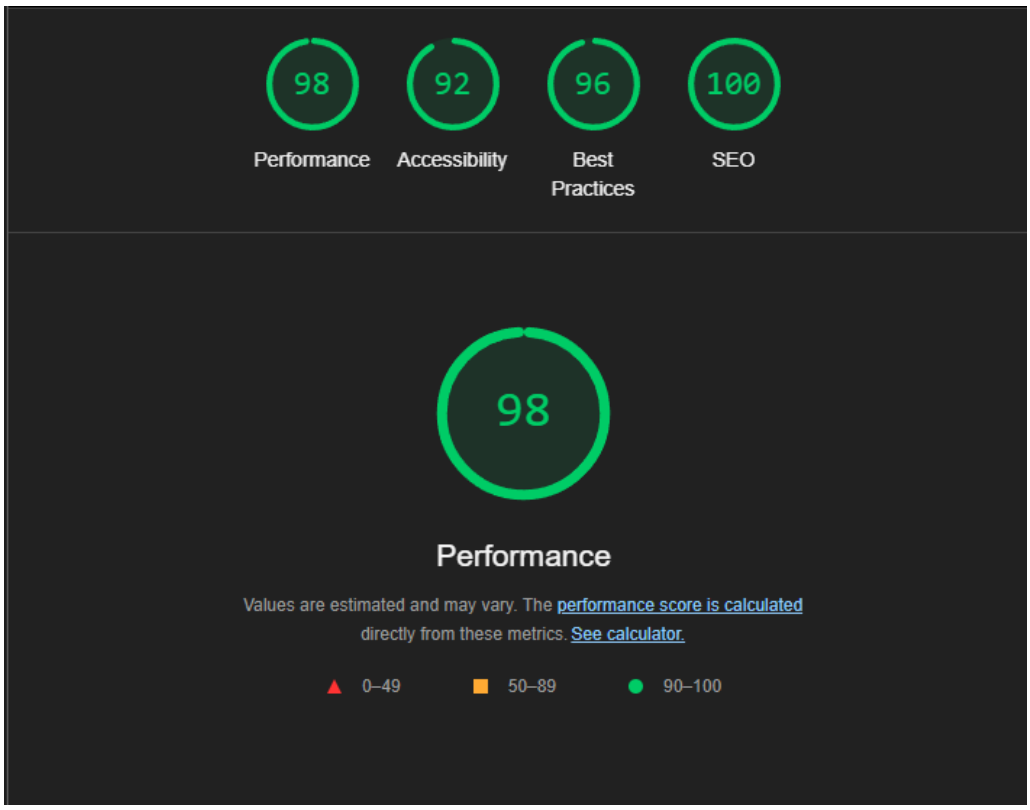


Рисунок 4.1 – Показники оптимізації для пк

На рисунку 4.2 зображено час завантаження додатку та інші показники, час блокування сторінки перед рендером та час фінального завантаження сторінки. Всі показники зеленого кольору і менше однієї секунди, для такого об’ємного додатку це ідеальні значення.

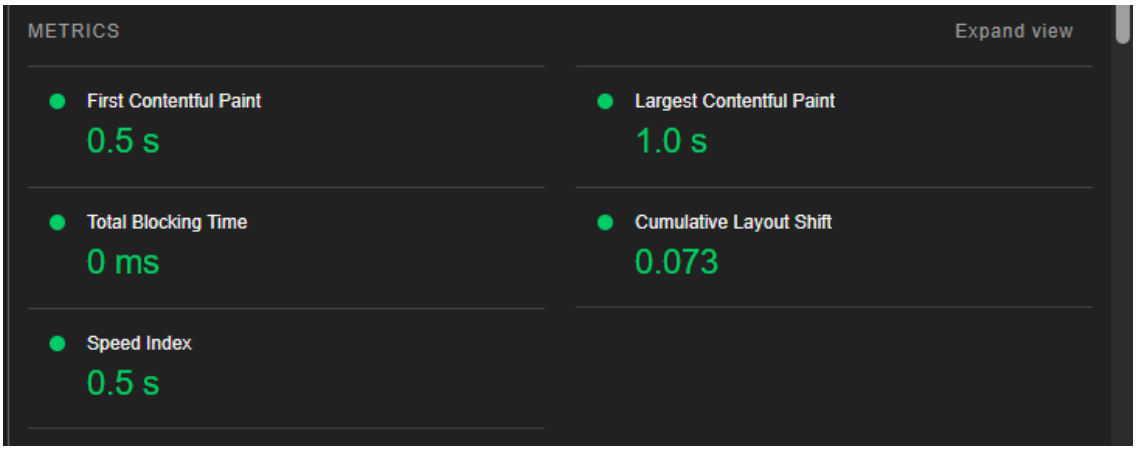


Рисунок 4.2 – Час завантаження для пк

На рисунку 4.3 зображенно показники оптимізації для телефонів. Показник продуктивності трохи зменшився, але це нормально, адже телефони менш потужні, ніж комп’ютери, така ситуація в будь-якому додатку. Головне, що він все ще вище 90 і зменшився всього на пару одиниць. Інші показники залишились такими ж, так і має бути.

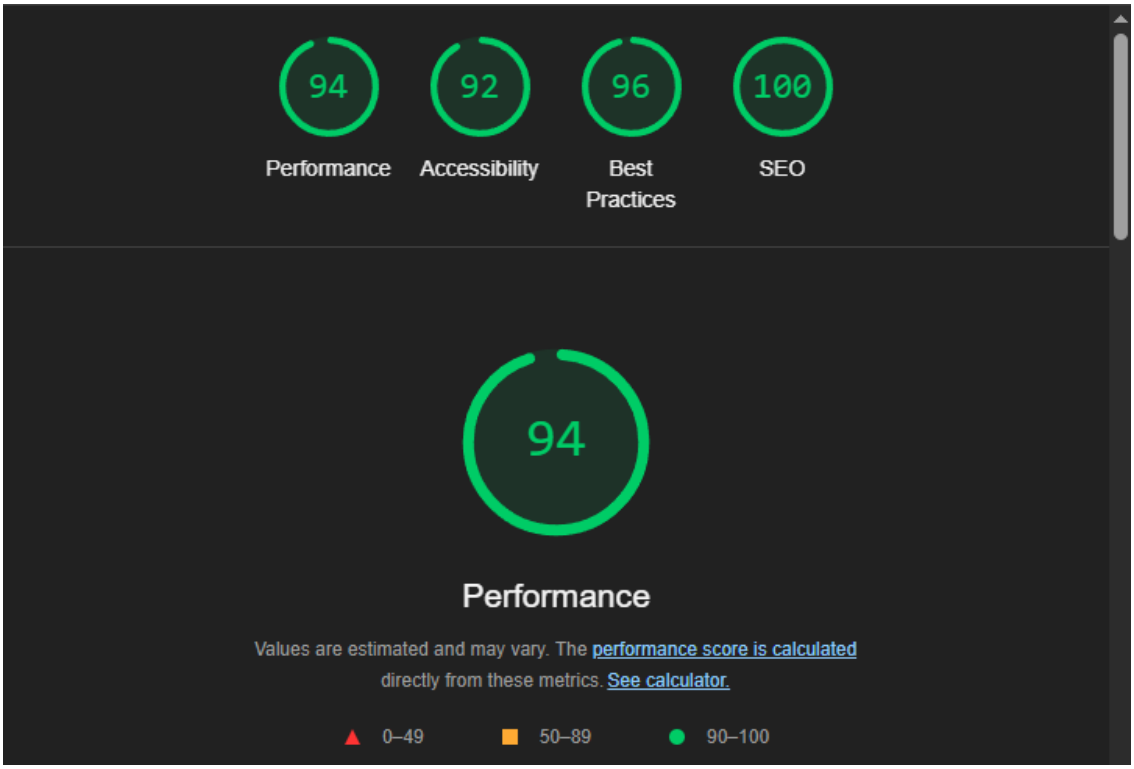


Рисунок 4.3 – Показники оптимізації для пк

На рисунку 4.4 зображено час завантаження для сторінки, час блокування та індекс швидкості на мобільному пристрої. Ці показники гірше ніж на пк, але не критично. Такими великими платформами будуть користуватися в пуршу чергу з комп’ютерів, а з телефона відкривати у крайніх випадках, але і для них найбільше значення очікування – 2.8 секунд не настільки критично. Від 4 секунд користувач починає почувати себе незручно, тому це не настільки фатально.

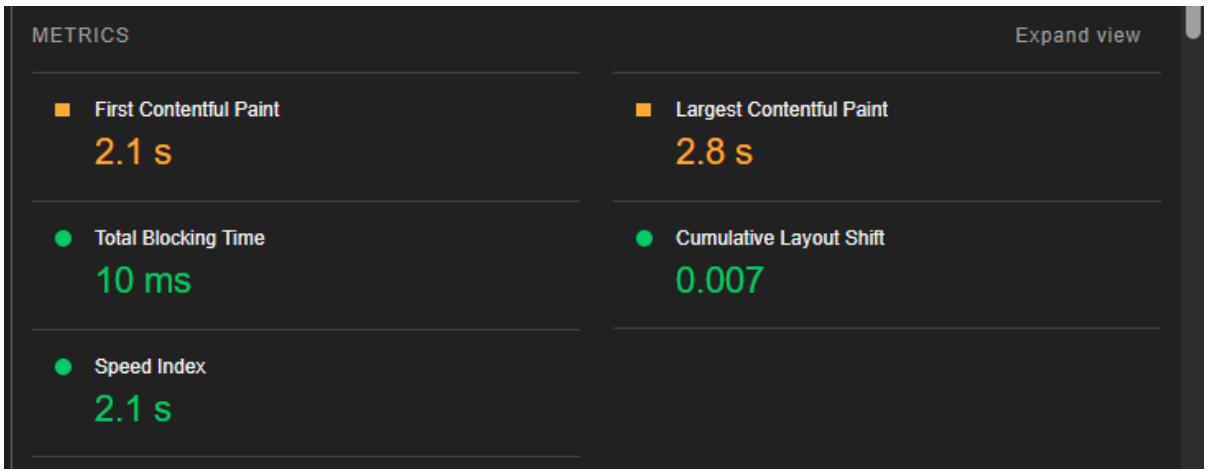


Рисунок 4.4 – Час завантаження для пк

4.2. Методика тестування вебзастосунку

Додаток покритий юніт-тестами. Були покриті тестами компоненти та глобальний стан, end-to-end тести та інтеграційні тести було замінено ручним тестуванням, адже структура сторінок та наявні компоненти часто змінювалися, отже це зайняло б багато часу. Але самі компоненти змінювалися нечасто, тому юніт-тести був сенс написати.

4.2.1. Тестування глобального стану через Redux Toolkit

Для тестування дій всередині глобального стану були створені об’єкти, які використовувалися як мок-об’єкт початкового стану. Оскільки редакс використовує підхід чистих функцій, тести пишуться дуже легко. Є початковий стан, є запущена дія, є стан-результат. Треба лише порівняти стан-результат з очікуваним станом, як показано на рисунку 4.5.

```
import {
  MOCK_OBJECT_ONE,
  MOCK_OBJECT_THREE,
  MOCK_OBJECT_TWO,
} from "../../../mocs/state";

import { clearProfileErrorMessage,exit, profileReducer } from "./profile";

describe("Testing profile slice reducers", () => {
  const exitAction = { type: exit.type };
  const clearProfileErrorMessageAction = { type: clearProfileErrorMessage.type };

  describe("exit reducer must works currectly", () => {
    it("Profile must be cleared after this reducer calling", () => {
      const firstResult = profileReducer(MOCK_OBJECT_ONE.profile, exitAction);
      const secondResult = profileReducer(MOCK_OBJECT_TWO.profile, exitAction);
      const thirdResult = profileReducer(MOCK_OBJECT_THREE.profile, exitAction);

      expect(firstResult.data).toBe(null);
      expect(secondResult.data).toBe(null);
      expect(thirdResult.data).toBe(null);
    });
  });
```

Рисунок 4.5 – Тестування дій у redux стані

Функції-селектори також легко тестуються на основі вже використаних мок-об’єктів глобального стану. Можна викликати функцію селектор і перевірити, чи було отримано очікуваний результат.

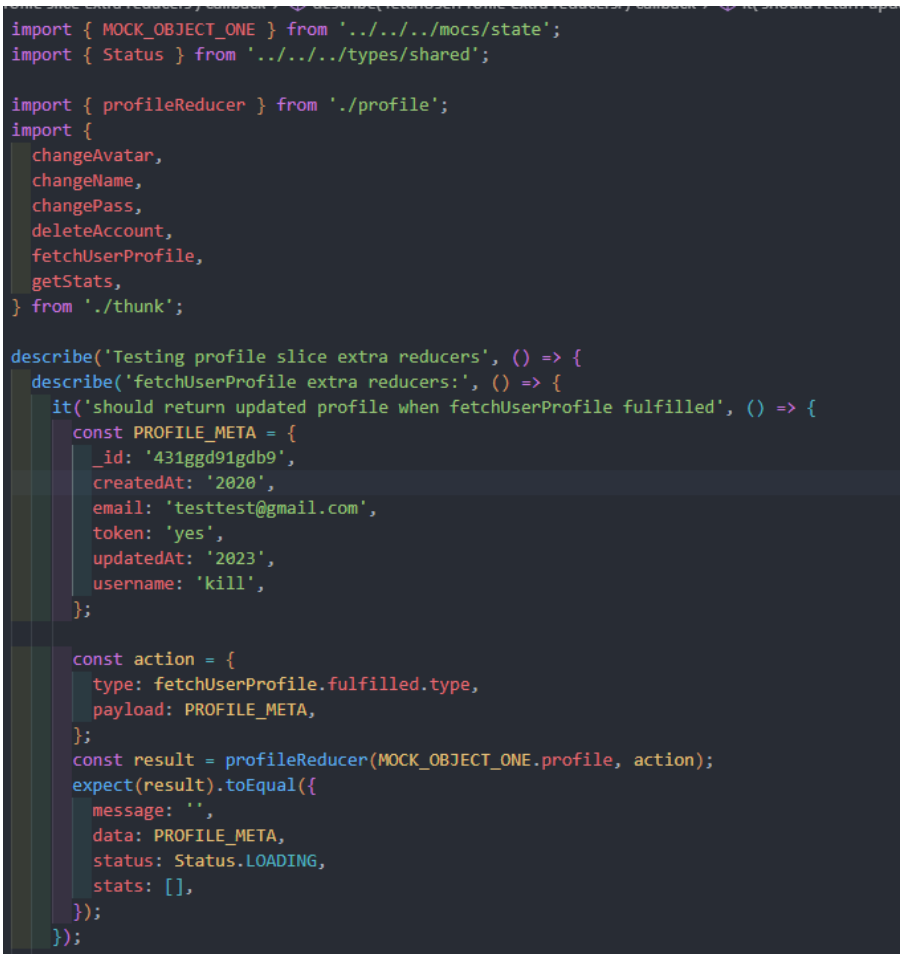
```
import {
  MOCK_OBJECT_ONE,
  MOCK_OBJECT_THREE,
  MOCK_OBJECT_TWO,
} from '../../../mocs/state';
import { Status } from '../../../types/shared';

import {
  selectProfileMessage,
  selectProfileStatus,
  selectStats,
  selectUserProfile,
} from './selectors';

describe('Testing profile slice selectors', () => {
  it('selectProfileStatus must work correctly', () => {
    expect(selectProfileStatus(MOCK_OBJECT_ONE)).toBe(Status.LOADING);
    expect(selectProfileStatus(MOCK_OBJECT_TWO)).toBe(Status.SUCCESS);
    expect(selectProfileStatus(MOCK_OBJECT_THREE)).toBe(Status.ERROR);
  });
```

Рисунок 4.6 – Тестування селекторів у redux стані

Екстраредьюсери тестуються складніше, необхідно імітувати всі стани асинхронного запиту, наприклад pending або rejected та порівняти новий стан з очікуваним. Приклад тестів наведено на рисунку 4.7.



```
import { MOCK_OBJECT_ONE } from '../../../mocs/state';
import { Status } from '../../../types/shared';

import { profileReducer } from './profile';
import {
  changeAvatar,
  changeName,
  changePass,
  deleteAccount,
  fetchUserProfile,
  getStats,
} from './thunk';

describe('Testing profile slice extra reducers', () => {
  describe('fetchUserProfile extra reducers:', () => {
    it('should return updated profile when fetchUserProfile fulfilled', () => {
      const PROFILE_META = {
        _id: '431ggd91gdb9',
        createdAt: '2020',
        email: 'testtest@gmail.com',
        token: 'yes',
        updatedAt: '2023',
        username: 'kill',
      };

      const action = {
        type: fetchUserProfile.fulfilled.type,
        payload: PROFILE_META,
      };
      const result = profileReducer(MOCK_OBJECT_ONE.profile, action);
      expect(result).toEqual({
        message: '',
        data: PROFILE_META,
        status: Status.LOADING,
        stats: [],
      });
    });
```

Рисунок 4.7 – Тестування асинхронних санок

4.2.2. Юніт-тестування основних React-компонент

У кожної компоненти в React є власний стан та параметри, що передаються як атрибути у цю компоненту. Тести виглядають наступним чином: рендериться компонента, імітується якась дія (наприклад натискання миші), після чого тестується чи змінився стан компоненти або чи відбулися якісь події. Приклад такого тесту наведено на рисунку 4.8.

```
import { fireEvent,render, screen } from "@testing-library/react";

import Button from "./Button";

describe("Button", () => {
  const mockCallback = jest.fn();

  it("renders button with text", () => {
    render(<Button text="Click me" class="primary" callback={mockCallback} />);
    const buttonElement = screen.getByText("Click me");
    expect(buttonElement).toBeInTheDocument();
  });

  it("applies the correct CSS class", () => {
    render(<Button text="Click me" class="primary" callback={mockCallback} />);
    const buttonElement = screen.getByText("Click me");
    expect(buttonElement).toHaveClass("button");
    expect(buttonElement).toHaveClass("primary");
  });

  it("calls the callback function when clicked", () => {
    render(<Button text="Click me" class="primary" callback={mockCallback} />);
    const buttonElement = screen.getByText("Click me");
    fireEvent.click(buttonElement);
    expect(mockCallback).toHaveBeenCalledTimes(1);
  });
});
```

Рисунок 4.8 – Тестування компонент

Також компонента може бути обгорнута у контекст та мати доступ до глобального стану, у цьому випадку необхідно обгорнути компоненту, що рендериться, у провайдер контексту, але самі тести виглядають так само. Приклад наведено на рисунку 4.9.

```
describe('Footer', () => {
  describe('links', () => {
    describe('there are correct amount of links', () => {
      MOCK_FOR_AMOUNT_OF_LINKS_CHECKING.forEach(({ active, links }) => {
        it(`should render ${links.length} links if we are pass array of ${links.length} links`, () => {

          const linksElements = screen.getAllByRole('link');

          expect(linksElements.length).toBe(links.length);
        });

        it(`active link to "${active}" should have active class and other links shouldn't`, () => {
          render(
            <MemoryRouter initialEntries={[active]}>
              <Provider store={store}>
                <Footer links={links} />
              </Provider>
            </MemoryRouter>,
          );

          const linksElements = screen.getAllByRole('link');
          const activeLink = linksElements.find((link) =>
            link.classList.contains('isActive'),
          );
          const inActiveLinks = linksElements.filter(
            (link) => !link.classList.contains('isActive'),
          );

          expect(activeLink).toHaveAttribute('href', active);
          expect(inActiveLinks.length).toBe(links.length - 1);
        });
      });

      it('renders no links if we pass an empty array', () => {
        render(
          <MemoryRouter>
            <Provider store={store}>
              <Footer links={[]} />
            </Provider>
          </MemoryRouter>,
        );
```

Рисунок 4.9 – Тестування компонент з контекстом

4.3. Перевірка адаптивності інтерфейсу та підтримки тем

Адаптивність перевірялася вручну на усіх сторінках. На рисунку 4.10 наведено скріншот головної сторінки з відкритим модальним вікном з фільтрами пошуку при відкритті з телефону. На рисунку 4.11 наведено скріншот цієї ж сторінки але з закритим модальним вікном.

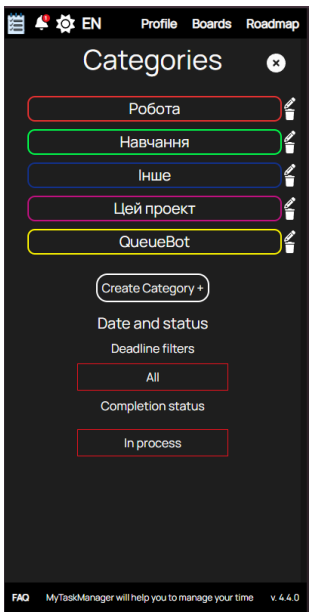


Рисунок 4.10 – Головна сторінка з відкритим модальним вікном

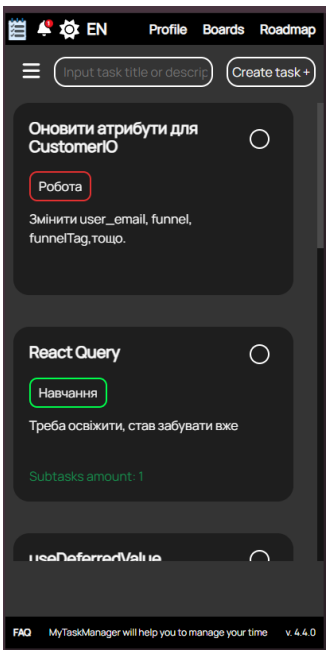


Рисунок 4.11 – Головна сторінка з закритим модальним вікном

На рисунку 4.12 наведено скріншот профілю користувача. Для телефонів всі компоненти розташовуються у колонку замість рядка та зменшуються у розмірах.

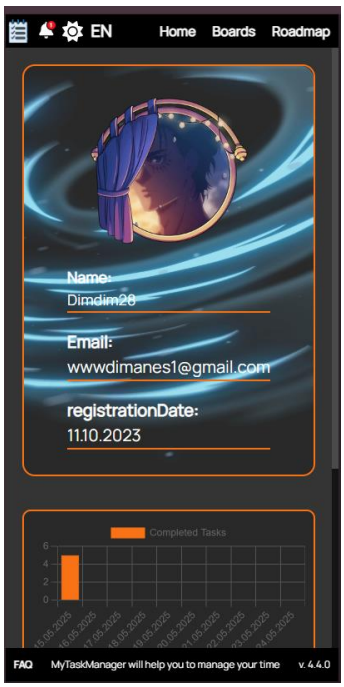


Рисунок 4.12 – Головна сторінка з закритим модальним вікном

На рисунку 4.13 наведено скріншот сторінки з усіма дорожніми картами, аналогічно виглядає сторінка з усіма канбан дошками. Для телефонів сітка проєктів замінена колонкою, у якій знаходяться всі проєкти, один під одним.

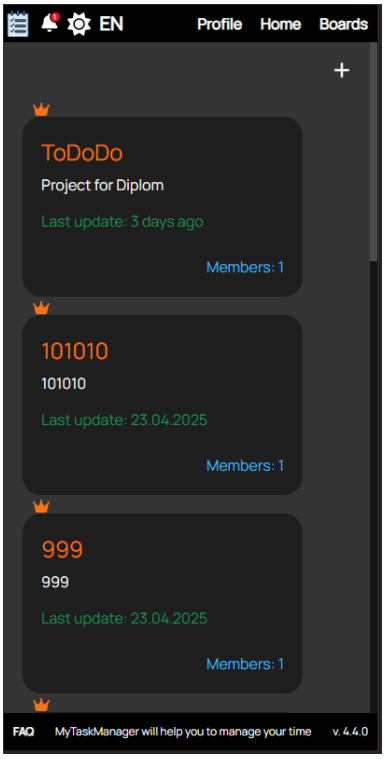


Рисунок 4.13 – Сторінка усіх проєктів

На рисунку 4.14 наведено скріншот канбан-проєкту. Оскільки сторінка має горизонтальну прокрутку, ніяких складнощів у адаптивній верстці немає. Бокова панель з інформацією про задачу відкривається на мобільних пристроях на всю ширину екрану. Скріншот сторінки наведено на рисунку 4.15.

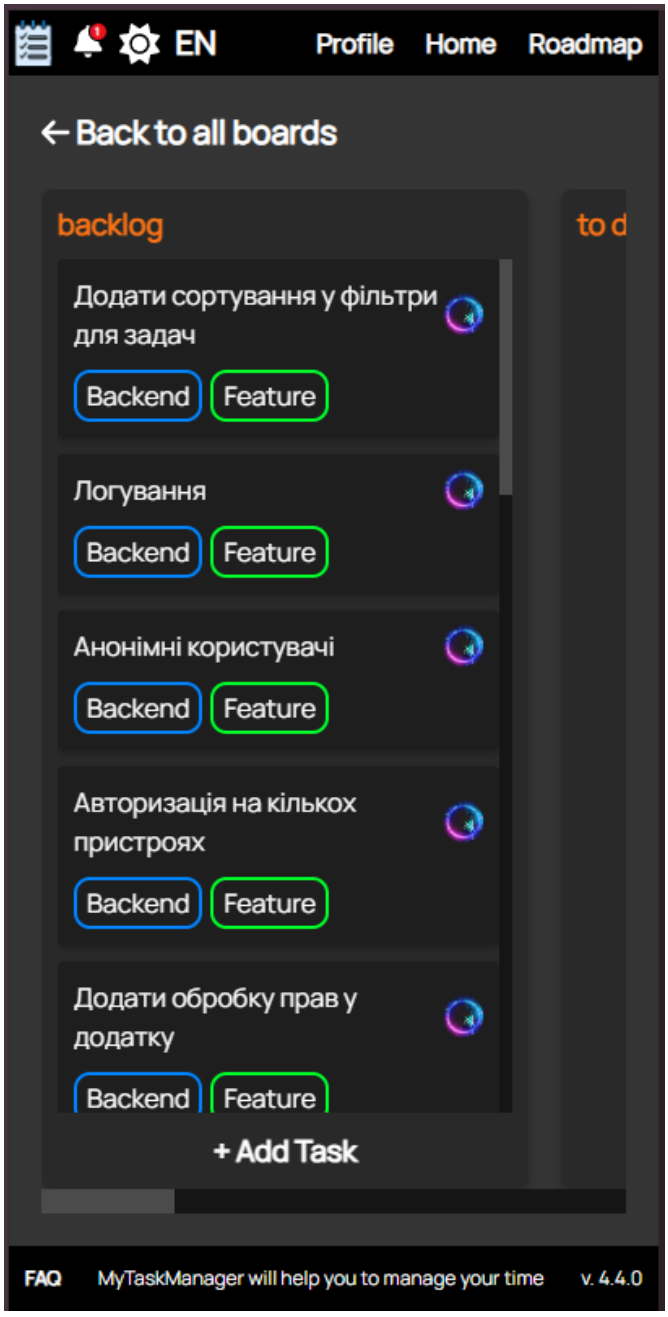


Рисунок 4.14 – Сторінка канбан дошки

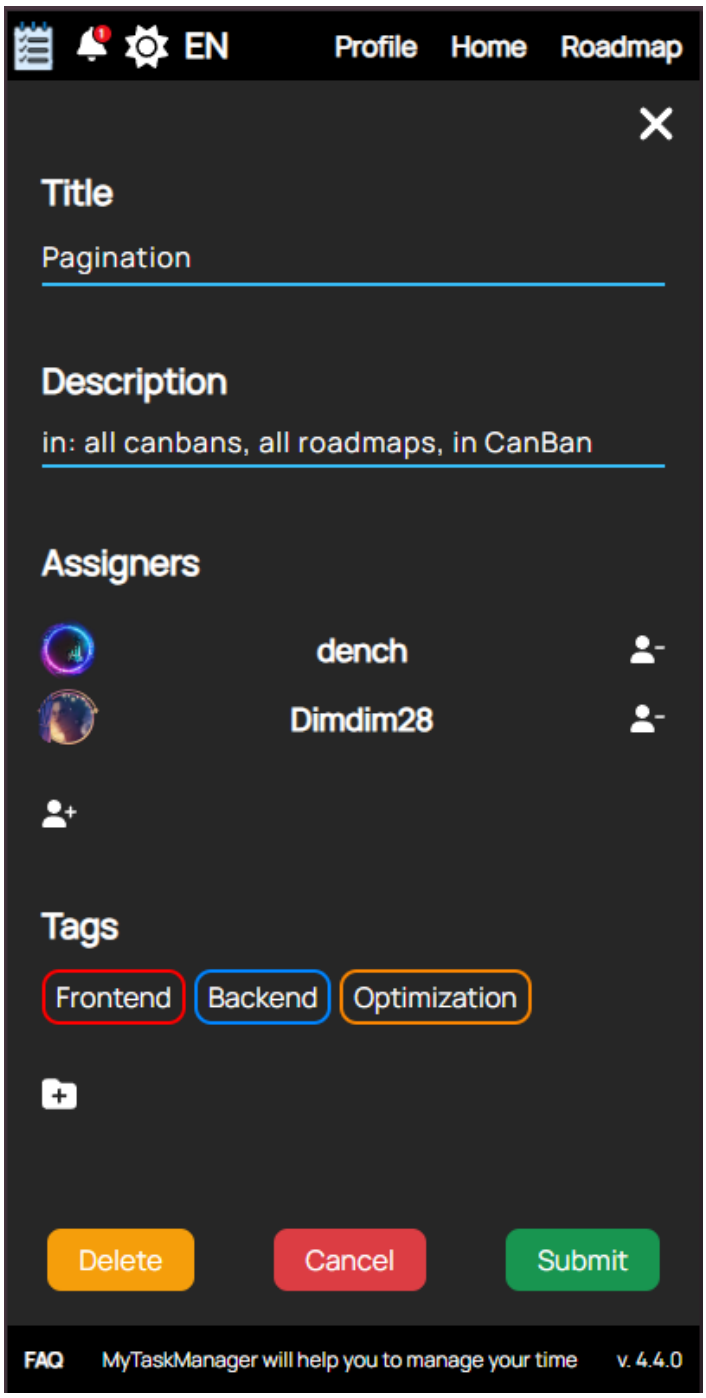


Рисунок 4.15 – Бокова панель канбан дошки

На рисунку 4.16 зображено дорожню карту, оскільки тут з самого початку закладено горизонтальну та вертикальну прокрутки, ніяких проблем з адаптивністю тут немає. Достатньо лише було зменшити текст та відступи.

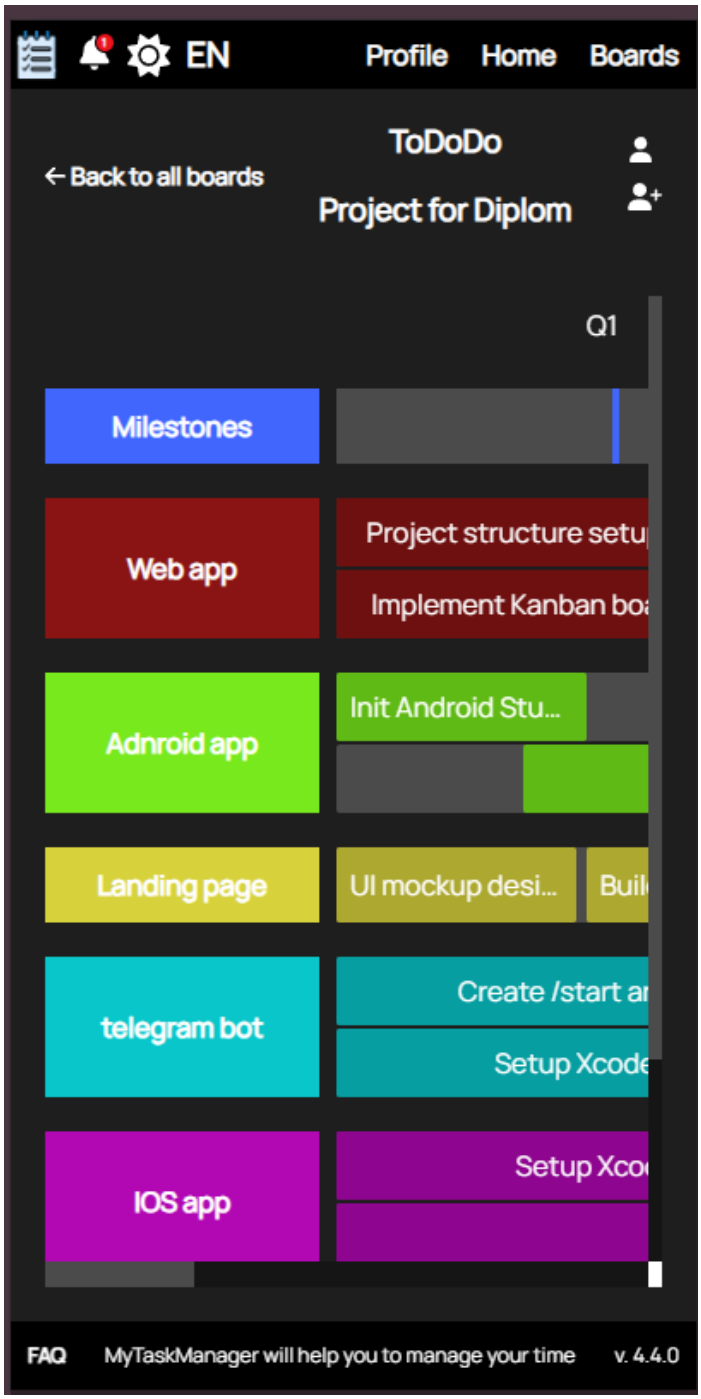


Рисунок 4.16 – Дорожня карта

Тестування світлої теми та локалізації також проводилося у ручному режимі. На рисунку 4.17 зображено сторінку дорожньої карти з включеною світлою темою і обраною українською мовою.

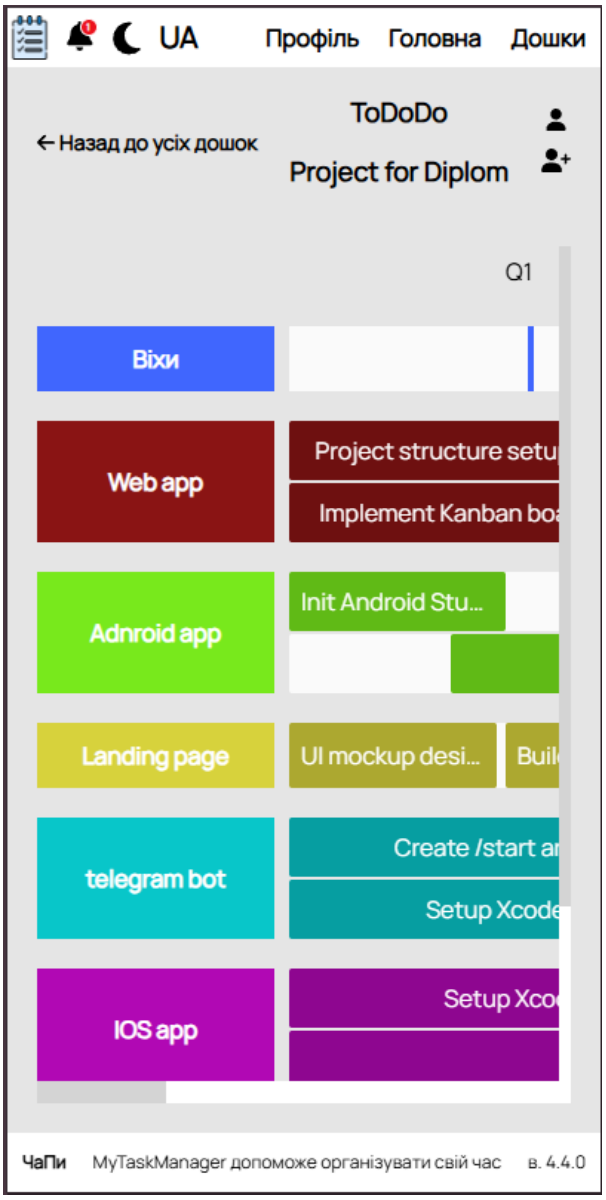


Рисунок 4.17 – Перевірка теми та локалізації для дорожньої карти

ВИСНОВОК ДО РОЗДІЛУ 4

Додаток стабільно працює і на телефонах, і на комп’ютерах. По всім 4 характеристикам (SEO, продуктивність, best practices, доступність) аналіз додатку показує стабільно високі значення на усіх пристроях. Усі ці значення більше 90. Якщо SEO досить легко підняти до цього рівня, то продуктивність та доступність підвищити до таких значень важко, для цього було проведено велику роботу.

Різні кольорові теми і мови працюють стабільно, інтерфейс нормально виглядає і на великих, і на малих екранах. Додаток було також адаптовано для малих і дуже великих екранів. Усі сторінки було протестовано у ручному режимі, компоненти покриті юніт тестами, як і глобальний стан додатку: селектори, екстраредьюсери, дії. Світла тема та локалізації були також протестовані в ручному режимі, вони працюють правильно.

Під час ручного тестування було багато спроб зламати додаток, ввести неправильні далі, швидко робити хаотичні дії, спробувати навантажити сервер, часто натискаючи на кнопки, але ніщо з цього не створило проблем. В першу чергу через те, що всі введені дані валідуються і на сервері, і на клієнті, а при відправці запитів на сервер кнопки стають неактивними і відправити запит кілька разів неможливо.

ВИСНОВКИ

Додаток, що було створенно під час роботи над бакалаврським проєктом, допомагає користувачам управляти задачами, працювати разом з іншими користувачами над дорожніми картами та канбан дошками. Він реалізує функціонал, що має попит, та аудиторію, яка буде ним користуватися.

У першому розділі було розглянуто історію інструментів управління задачами та причини їх винекнення. Було розглянуто існуючі рішення та їх проблеми, які розроблений додаток вирішує. Якісь рішення недто важкі, деякі не реалізують весь необхідний функціонал, якісь додатки виглядають дуже просто, там немає персоналізації, анімацій і стилів.

У другому розділі було описано стек, проаналізовано технології, мови, бібліотеки, що мало сенс використати у написанні додатку. Було розглянуто стилізацію, бібліотеки для роботи з глобальним станом, тощо. Обраний стек актуальний та популярний серед розробників, він підтверджений часом та дозволяє зручно та швидко розробляти веб платформи, підтримувати їх та розширювати код.

У третьому розділі було розглянуто реалізацію додатку, проаналиізовано підходи при розробці та проаналізовано кожну сторінку, її код та алгоритм роботи. Було описано архітектуру, роботу зі стилями, глобальним станом, локальним станом, запитами на сервер, тощо. Було описано специфічні проблеми, як наприклад проблему з висотою додатку на мобільних браузерах типу Facebook, та методи їх вирішення.

У четвертому розділі було описано методики тестування додатку за допомогою Unit-тестів та скріншоти ручного тестування як цілих сторінок, так і окремих аспектів додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Asana – Getting Started Guide [Електронний ресурс] – Режим доступу до ресурсу: https://asana.com/guide.
2. Jira Software – What is Jira? [Електронний ресурс] // Atlassian – Режим доступу до ресурсу: https://www.atlassian.com/software/jira.
3. Aha! Roadmaps Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.aha.io/roadmapping/guide.
4. Zenkit – Project Management Tools [Електронний ресурс] – Режим доступу до ресурсу: https://zenkit.com/en/.
5. JavaScript.info: сучасний підручник з JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.javascript.info.
6. TypeScript Handbook [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/.
7. ECMAScript Modules: A Complete Guide [Електронний ресурс] // MDN Web Docs – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules.
8. React Documentation – Main Docs [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.
9. React.dev: JSX In-Depth [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn/writing-markup-with-jsx.
10. React.dev: Lazy Loading [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn/code-splitting.
11. DOM – Document Object Model Overview [Електронний ресурс] // MDN Web Docs – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.
12. Vue.js – офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/.

13. Angular – офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs.
14. Redux Toolkit Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redux-toolkit.js.org/.
15. Redux Fundamentals Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/tutorials/fundamentals/part-1-overview.
16. Flux проти Redux: Порівняння архітектур [Електронний ресурс] // Medium – Режим доступу до ресурсу: https://madfinn.medium.com/flux-проти-redux-порівняння-883c0932931a.
17. MVVM vs MVC vs MVP: Patterns Explained [Електронний ресурс] // Hackernoon – Режим доступу до ресурсу: https://hackernoon.com/mvvm-vs-mvc-vs-mvp.
18. React Documentation: useReducer [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/reference/react/useReducer/.
19. React Documentation: useContext [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/reference/react/useContext.
20. MobX Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://mobx.js.org/README.html.
21. MobX Concepts Overview [Електронний ресурс] – Режим доступу до ресурсу: https://mobx.js.org/observable-state.html.
22. Огляд популярних CSS фреймворків та методологій [Електронний ресурс] // ITRating UA – Режим доступу до ресурсу: https://it-rating.ua/.
23. Sass: Syntactically Awesome Style Sheets [Електронний ресурс] – Режим доступу до ресурсу: https://sass-lang.com/.
24. SCSS Language Basics [Електронний ресурс] – Режим доступу до ресурсу: https://sass-lang.com/guide.
25. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://tailwindcss.com/docs.

26. Tailwind Overview – W3Schools [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/tailwindcss/.
27. BEM Official Methodology [Електронний ресурс] – Режим доступу до ресурсу: https://en.bem.info/methodology/.
28. BEM 101 – CSS Tricks [Електронний ресурс] – Режим доступу до ресурсу: https://css-tricks.com/bem-101/.
29. Single-Page Applications (SPA): Concepts & Routing [Електронний ресурс] // freeCodeCamp – Режим доступу до ресурсу: https://www.freecodecamp.org/news/single-page-apps/.
30. i18next Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.i18next.com/.

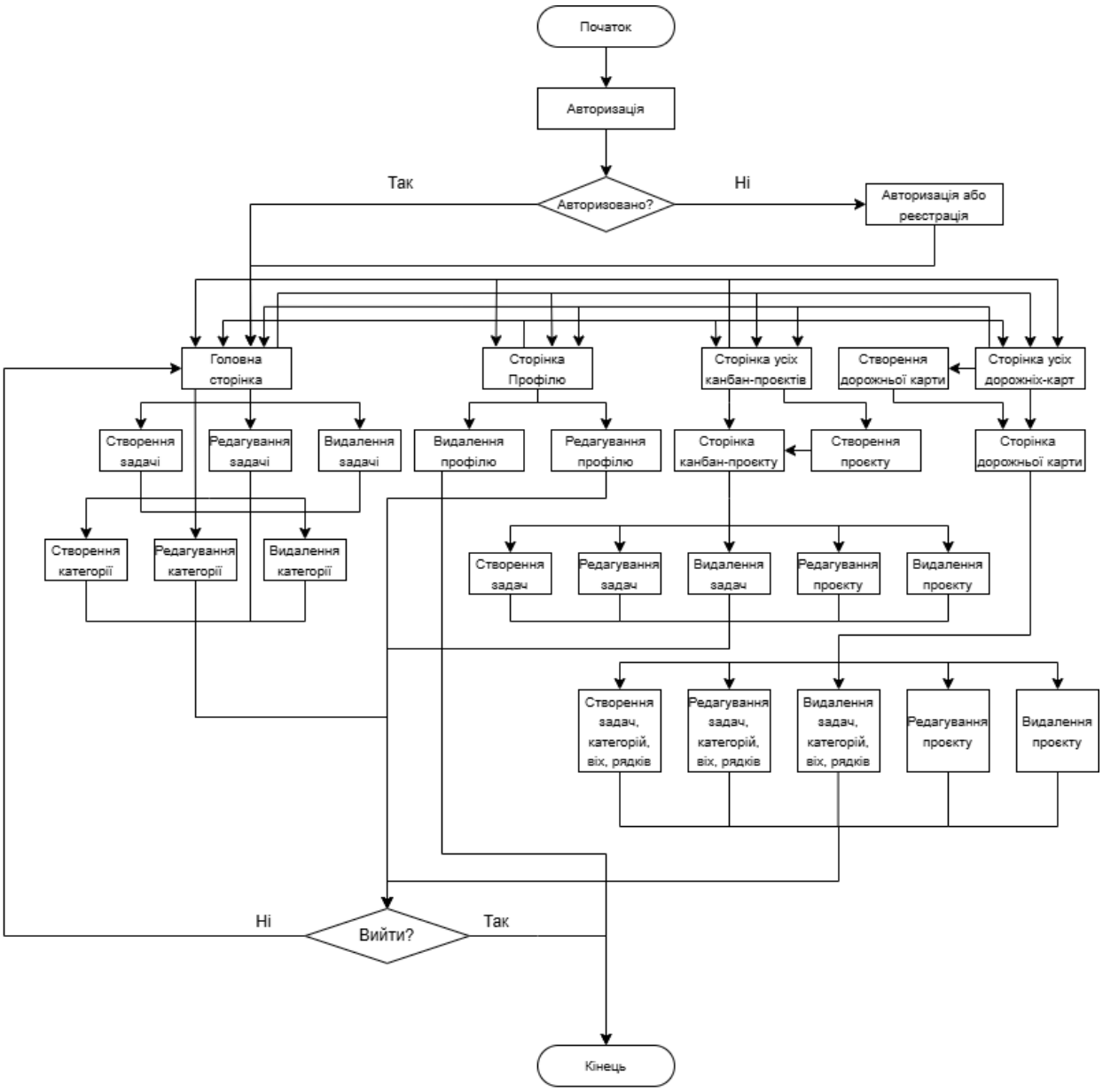
ДОДАТОК 1

Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map

Алгоритм дії веб-застосунку
(принципова схема)
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2025



					ІАЛЦ.467200.004 Д1	
Зм	***Арк.***	***№ докум.***	***Підпис***	***Дата***		
Розроб.		*Нестеров Д.В.*			*Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map* Принципова схема	***Літ.*** / ***Арк.*** 1 / ***Аркушів*** 1
Перевір.		*Шульга М.В.*				
Реценз.		*Матвійчук О.В.*				***КПІ ім. Ігоря Сікорського, ФІОТ ІМ-13***
Н. контр.		*Міщенко Л.Д.*				
Затверд.		*Новотарський М.А.*				

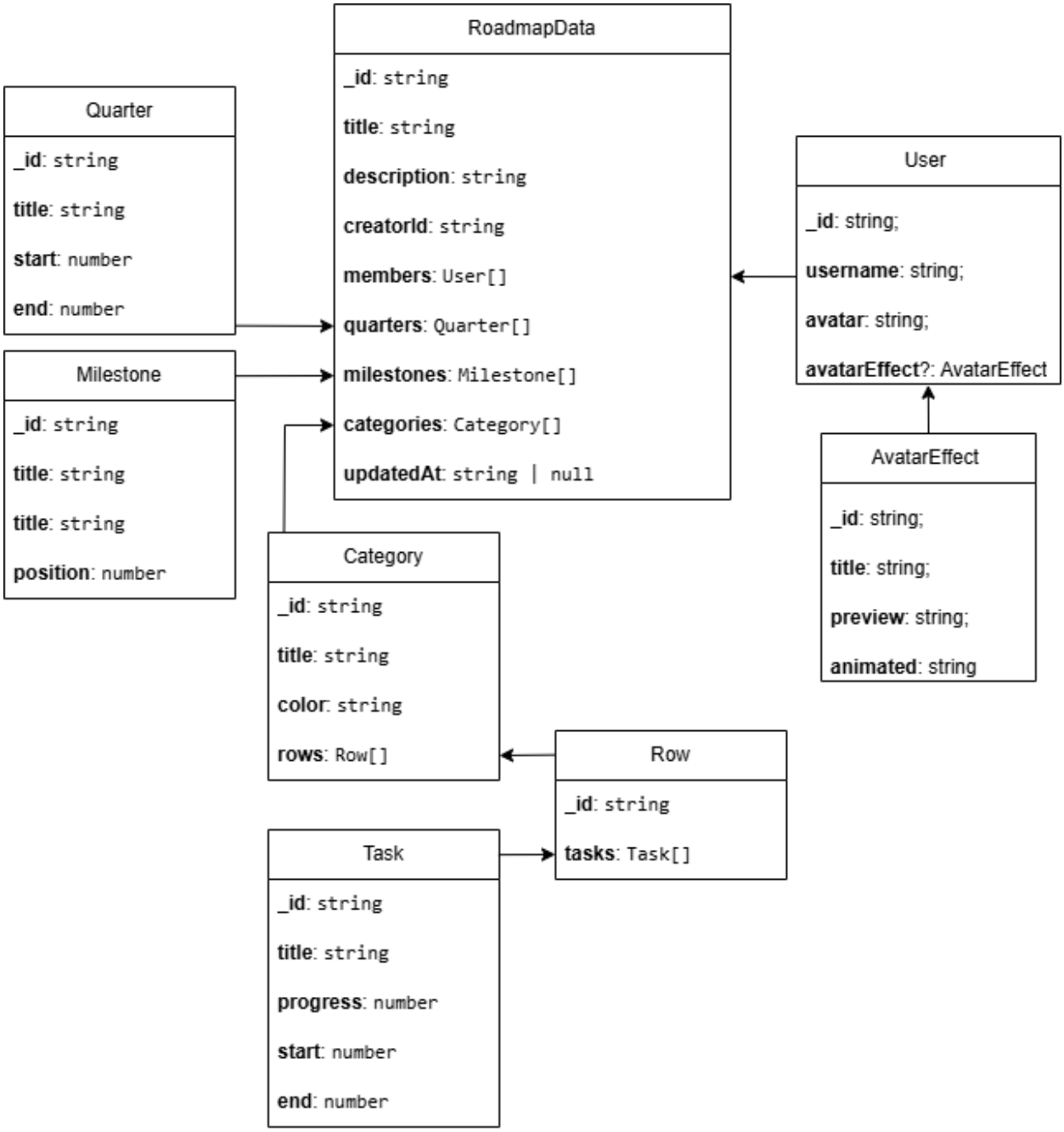
ДОДАТОК 2

Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map

Діаграма класів дорожньої карти
(функціональна схема)
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2025



					ІАЛЦ.467200.005 Д2	
Зм	***Арк.***	***№ докум.***	***Підпис***	***Дата***		
Розроб.		*Нестеров Д.В.*			*Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map* Функціональна схема	***Літ.*** / ***Арк.*** 1 / ***Аркушів*** 1
Перевір.		*Шульга М.В.*				
Реценз.		*Матвійчук О.В.*				
Н. контр.		*Міщенко Л.Д.*				
Затверд.		*Новотарський М.А.*				***КПІ ім. Ігоря Сікорського, ФІОТ ІМ-13***

ДОДАТОК З

Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map

Архітектура додатку
(структурна схема)
ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ – 2025

					ІАЛЦ.467200.006 ДЗ		
Зм	**Арк.**	**№ докум.**	**Підпис**	**Дата**			
Розроб.		*Нестеров Д.В.*			*Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map* Структурна схема	**Літ.**	**Арк.** 1 / **Аркушів** 1
Перевір.		*Шульга М.В.*					
Реценз.		*Матвійчук О.В.*				**КПІ ім. Ігоря Сікорського, ФІОТ ІМ-13**	
Н. контр.		*Міщенко Л.Д.*					
Затверд.		*Новотарський М.А.*					

ДОДАТОК 4

Інтерактивна платформа для організування проєктів на базі методології Kanban з використанням метрик та таблиць Road map

Текст програмного коду
ІАЛЦ.467200.007 Д4

Аркушів 37

Київ – 2025

Github репозиторій

https://github.com/Dimdim28/to_do_list_frontend

Roadmap.tsx

```
import { DragEvent, useEffect, useRef, useState } from 'react';
import { useTranslation } from 'react-i18next';
import { useParams } from 'react-router';
import { NavLink } from 'react-router-dom';
import {
  faArrowLeft,
  faPencil,
  faPlus,
  faTrash,
  faUser,
  faUserPlus,
} from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

import roadmapAPI from '../../api/roadmapApi';
import { Modal } from '../../components/common/Modal/Modal';
import Preloader from '../../components/Preloader/Preloader';
import withLoginRedirect from '../../hoc/withLoginRedirect';
import { useAppDispatch, useAppSelector } from '../../hooks';
import { selectProfile } from '../../redux/slices/auth/selectors';
import {
  addRoadmapNewLineToCategory,
  addRoadmapNewQuarter,
  moveRoadmapTask,
  setRoadmapClickPosition,
  setRoadmapCurrentCategory,
  setRoadmapCurrentMilestone,


  setRoadmapCurrentQuarter,
  setRoadmapCurrentRow,
  setRoadmapCurrentTask,
  setRoadmapIsDeletingCategoryOpened,
  setRoadmapIsDeletingMilestoneOpened,
  setRoadmapIsDeletingQuarterOpened,
  setRoadmapIsDeletingRowOpened,
  setRoadmapIsDeletingTaskOpened,
  setRoadmapIsEditingCategoryOpened,
  setRoadmapIsEditingMilestoneOpened,
  setRoadmapIsEditingTaskOpened,
  updateRoadmapData,
  updateRoadmapTaskInCategory,
} from '../../redux/slices/roadmap/roadmap';
import {
  selectRoadmapCreatorId,
```

```
  selectRoadmapData,
  selectRoadmapIsDeletingCategoryOpened,
  selectRoadmapIsDeletingMilestoneModalOpened,
  selectRoadmapIsDeletingQuarterOpened,
  selectRoadmapIsDeletingRowOpened,
  selectRoadmapIsDeletingTaskModalOpened,
  selectRoadmapIsEditingCategoryOpened,
  selectRoadmapIsEditingMilestoneModalOpened,
  selectRoadmapIsEditingTaskModalOpened,
} from '../../redux/slices/roadmap/selectors';
import {
  Category,
  Quarter,
  RoadmapData,
  Row,
} from '../../redux/slices/roadmap/type';
import ROUTES from '../../routes';
import { Status } from '../../types/shared';

import AddUserToProjectModal from './components/AddUserToProjectModal/AddUserToProjectModal';
import { CategoryDeleting } from './components/CategoryDeleting/CategoryDeleting';
import CategoryForm from './components/CategoryForm/CategoryForm';
import EditProjectInfo from './components/EditProjectInfo/EditProjectInfo';
import MilestoneComponent from './components/Milestone';
import { MilestoneDeleting } from './components/MilestoneDeleting/MilestoneDeleting';
import MilestoneForm from './components/MilestoneForm/MilestoneForm';
import { QuarterDeleting } from './components/QuarterDeleting/QuarterDeleting';
import { RowDeleting } from './components/RowDeleting/RowDeleting';
import TaskComponent from './components/Task';
import { TaskDeleting } from './components/TaskDeleting/TaskDeleting';
import TaskForm from './components/TaskForm/TaskForm';

import styles from './styles.module.scss';

const RoadMap = () => {
  const dispatch = useAppDispatch();
  const { t } = useTranslation();
  const { id: boardId } = useParams();

  const wrapperRef = useRef<HTMLDivElement>(null);
  const roadmapRef = useRef<HTMLDivElement>(null);

  const SCROLL_THRESHOLD = 50;
  const SCROLL_SPEED = 10;

  const data = useAppSelector(selectRoadmapData);

  const [isLoading, setIsLoading] = useState(true);
  const [errorMessage, setErrorMessage] = useState('');
  const [roadmapContentWidth, setRoadmapContentWidth] = useState<number>(0);

  const [isProjectSettingsOpened, setIsProjectSettingsOpened] =
    useState<boolean>(false);
  const [isAddUserToProjectModalOpened, setIsAddUserToProjectModalOpened] =
    useState<boolean>(false);

  const categoryEditing = useAppSelector(selectRoadmapIsEditingCategoryOpened);
  const categoryDeleting = useAppSelector(
```

```
  selectRoadmapIsDeletingCategoryOpened,
);
const rowDeleting = useAppSelector(selectRoadmapIsDeletingRowOpened);
const quarterDeleting = useAppSelector(selectRoadmapIsDeletingQuarterOpened);

const milestoneEditing = useAppSelector(
  selectRoadmapIsEditingMilestoneModalOpened,
);
const milestoneDeleting = useAppSelector(
  selectRoadmapIsDeletingMilestoneModalOpened,
);

const taskEditing = useAppSelector(selectRoadmapIsEditingTaskModalOpened);
const taskDeleting = useAppSelector(selectRoadmapIsDeletingTaskModalOpened);

const currentUserProfile = useAppSelector(selectProfile);
const creatorId = useAppSelector(selectRoadmapCreatorId);

const isCreator = currentUserProfile?._id === creatorId;

const updateWidth = () => {
  if (!roadmapRef.current) return;
  const fullWidth = roadmapRef.current.clientWidth;
  const contentWidth = fullWidth - 200 - 10;
  setRoadmapContentWidth(contentWidth);
};

useEffect(() => {
  const fetchBoardData = async (boardId: string) => {
    setIsLoading(true);
    const res = await roadmapAPI.getBoard(boardId);

    if (res.status === Status.SUCCESS) {
      dispatch(updateRoadmapData(res.data));
      setErrorMessage('');
      setIsLoading(false);
      requestAnimationFrame(updateWidth);
    } else {
      setErrorMessage(res.message);
      setIsLoading(false);
    }
  };

  if (boardId) {
    fetchBoardData(boardId);
  }
}, [boardId, dispatch]);

useEffect(() => {
  const frame = requestAnimationFrame(updateWidth);
  window.addEventListener('resize', updateWidth);

  return () => {
    cancelAnimationFrame(frame);
    window.removeEventListener('resize', updateWidth);
  };
}, []);
```

```
useEffect(() => {
  if (!isLoading && data?.quarters.length) {
    requestAnimationFrame(updateWidth);
  }
}, [isLoading, data?.quarters.length]);

if (errorMessage)
  return (
    <div className={styles.wrapper}>
      <div className={styles.error}>{errorMessage}</div>
    </div>
  );

if (isLoading || !data)
  return (
    <div className={styles.wrapper}>
      <Preloader />
    </div>
  );

const { title, categories, milestones, quarters, description } = data;

const totalQuarters = quarters.length;

const handleOpenEditColumnModal = (category: Category) => {
  dispatch(setRoadmapCurrentCategory(category));
  dispatch(setRoadmapIsEditingCategoryOpened(true));
};

const handleOpenDeleteColumnModal = (category: Category) => {
  dispatch(setRoadmapCurrentCategory(category));
  dispatch(setRoadmapIsDeletingCategoryOpened(true));
};

const handleEditProjectSettingsModal = () => {
  dispatch(setRoadmapCurrentCategory(null));
  dispatch(setRoadmapIsEditingCategoryOpened(true));
};

const handleCreateRow = async (category: Category, data: RoadmapData) => {
  const result = await roadmapAPI.createRow({
    categoryId: category._id,
    roadmapId: data._id,
  });

  if (result.status === Status.SUCCESS) {
    const newRow = result.data;
    dispatch(
      addRoadmapNewLineToCategory({
        rowId: newRow._id,
        categoryId: category._id,
      }),
    );
  }
};

const handleOpenDeleteRowModal = (row: Row, category: Category) => {
  dispatch(setRoadmapCurrentRow(row));
```

```
  dispatch(setRoadmapCurrentCategory(category));
  dispatch(setRoadmapIsDeletingRowOpened(true));
 };

 const handleAddNewQuarter = async (
  roadmapData: RoadmapData,
  totalQuarters: number,
 ) => {
  const result = await roadmapAPI.createQuarter({
   start: totalQuarters * 100,
   end: (totalQuarters + 1) * 100,
   roadmapId: roadmapData._id,
   title: `Q${totalQuarters + 1}`,
  });

  if (result.status === Status.SUCCESS) {
   dispatch(addRoadmapNewQuarter(result.data));
  }
 };

 const handleOpenDeleteQuarteerModal = (quarter: Quarter) => {
  dispatch(setRoadmapCurrentQuarter(quarter));
  dispatch(setRoadmapIsDeletingQuarterOpened(true));
 };

 const handleRowDoubleClick = (
  e: React.MouseEvent<HTMLDivElement>,
  row: Row,
  category: Category,
 ) => {
  const rect = e.currentTarget.getBoundingClientRect();
  const clickX = e.clientX - rect.left;
  const rowWidth = rect.width;

  const timelineWidth = totalQuarters * 100;
  const timelineValue = Math.round((clickX / rowWidth) * timelineWidth);

  const taskLength = 12;
  const buffer = 2;

  const newStart = timelineValue;
  const newEnd = newStart + taskLength;

  const hasOverlap = row.tasks.some(
   (task) =>
    !(task.end + buffer <= newStart || task.start >= newEnd + buffer),
  );

  const outOfBounds = newEnd > timelineWidth || newStart < 0;

  if (hasOverlap || outOfBounds) {
   return;
  }

  dispatch(setRoadmapClickPosition(newStart));
  dispatch(setRoadmapCurrentCategory(category));
  dispatch(setRoadmapCurrentRow(row));
  dispatch(setRoadmapCurrentTask(null));
```

```
  dispatch(setRoadmapIsEditingTaskOpened(true));
 };

 const handleDragOver = (e: DragEvent<HTMLDivElement>) => {
  if (!wrapperRef.current) return;
  e.preventDefault();

  const wrapper = wrapperRef.current;
  const { top, bottom } = wrapper.getBoundingClientRect();
  const mouseY = e.clientY;

  if (mouseY - top < SCROLL_THRESHOLD) {
   wrapper.scrollTop -= SCROLL_SPEED;
  } else if (bottom - mouseY < SCROLL_THRESHOLD) {
   wrapper.scrollTop += SCROLL_SPEED;
  }
 };

 const handleMilestoneDoubleClick = (e: React.MouseEvent<HTMLDivElement>) => {
  const target = e.currentTarget;
  const rect = target.getBoundingClientRect();

  const clickX = e.clientX - rect.left;
  const percent = clickX / rect.width;

  const timelineValue = Math.round(percent * totalQuarters * 100);
  const maxPosition = totalQuarters * 100;

  const minDistance = 10;

  const isTooClose = milestones.some(
   (m) => Math.abs(m.position - timelineValue) < minDistance,
  );

  const outOfBounds = timelineValue < 5 || timelineValue > maxPosition - 10;

  if (isTooClose || outOfBounds) {
   return;
  }

  dispatch(setRoadmapClickPosition(timelineValue));
  dispatch(setRoadmapCurrentMilestone(null));
  dispatch(setRoadmapIsEditingMilestoneOpened(true));
 };

 const handleTaskDrop = async (
  e: DragEvent<HTMLDivElement>,
  row: Row,
  category: Category,
 ) => {
  if (!e.dataTransfer) return;
  const raw = e.dataTransfer.getData('application/json');

  if (!raw) return;
  const data = JSON.parse(raw);

  console.log('Dropping data:', data);
```

```
const dropX = e.clientX;
const roadmapWidth = roadmapContentWidth;
const timelineWidth = totalQuarters * 100;

const deltaPx = dropX - data.offsetX;
const deltaValue = (deltaPx / roadmapWidth) * timelineWidth;

const newStart = Math.round(data.start + deltaValue);
const newEnd = Math.round(data.end + deltaValue);

const isSameRow =
  data.rowId === row._id && data.categoryId === category._id;

if (newStart < 0 || newEnd > timelineWidth) return;

const overlap = row.tasks.some(
  (t) =>
    t._id !== data.taskId &&
    !(newEnd + 2 <= t.start || newStart >= t.end + 2),
);

if (overlap) return;

if (isSameRow) {
  await roadmapAPI.updateTask({
    roadmapId: data.roadmapId,
    categoryId: data.categoryId,
    rowId: data.rowId,
    taskId: data.taskId,
    start: newStart,
    end: newEnd,
  });

  dispatch(
    updateRoadmapTaskInCategory({
      categoryId: category._id,
      rowId: row._id,
      taskId: data.taskId,
      updates: {
        start: newStart,
        end: newEnd,
      },
    }),
  );
} else {
  const task = {
    _id: data.taskId,
    title: data.title,
    start: newStart,
    end: newEnd,
    progress: data.progress,
    status: data.status,
  };

  await roadmapAPI.moveTask({
    roadmapId: data.roadmapId,
    categoryId: data.categoryId,
    rowId: data.rowId,
```

```
      taskId: data.taskId,
      toCategoryId: category._id,
      toRowId: row._id,
      start: newStart,
      end: newEnd,
    });

    dispatch(
      moveRoadmapTask({
        fromCategoryId: data.categoryId,
        fromRowId: data.rowId,
        toCategoryId: category._id,
        toRowId: row._id,
        task,
      }),
    );
  }
};

return (
  <div
    ref={wrapperRef}
    className={styles.wrapper}
    onDragOver={handleDragOver}
  >
    {isCreator ? (
      <div className={styles.projectLine}>
        <div className={styles.options}>
          <NavLink to={ROUTES.ROADMAP} className={styles.backToAllBoards}>
            <FontAwesomeIcon icon={faArrowLeft} /> {t('backToAllBoards')}
          </NavLink>
        </div>

        <div className={styles.options}>
          <h1 className={styles.projectTitle}>{title}</h1>
          <h2 className={styles.projectDescription}>{description}</h2>
        </div>

        <div className={styles.options}>
          <FontAwesomeIcon
            className={styles.gear}
            onClick={() => {
              setIsProjectSettingsOpened(true);
            }}
            fontSize="20px"
            icon={faUser}
          />
          <FontAwesomeIcon
            className={styles.user}
            onClick={() => {
              setIsAddUserToProjectModalOpened(true);
            }}
            fontSize="20px"
            icon={faUserPlus}
          />
        </div>
      </div>
    ) : (
```

```
  <NavLink to={ROUTES.ROADMAP} className={styles.backToAllBoards}>
    <FontAwesomeIcon icon={faArrowLeft} /> {t('backToAllBoards')}
  </NavLink>
)}

<div className={styles.roadmap}>
  <div
    ref={roadmapRef}
    className={styles.roadmapContent}
    style={{ minWidth: `${210 + totalQuarters * 300}px` }}
  >
    <div className={styles.quarterLines}>
      {quarters.map((quarter, index) =>
        index > 0 ? (
          <div
            key={quarter._id}
            className={styles.quarterLine}
            style={{
              left: `${(index / totalQuarters) * 100}%`,
            }}
          />
        ) : null,
      )}
    </div>
    <div className={styles.lineWrapper}>
      <div className={styles.category}></div>
      <div className={styles.blocks}>
        <div className={styles.quarteers}>
          {quarters.map((quarteer, id) => (
            <div
              key={quarteer._id}
              className={styles.quarteer}
              style={{ width: `${100 / totalQuarters}%` }}
            >
              {quarteer.title}

              {id === quarters.length - 1 ? (
                <FontAwesomeIcon
                  fontSize="15px"
                  icon={faTrash}
                  className={styles.removeQuarterIcon}
                  onClick={(e) => {
                    handleOpenDeleteQuarteerModal(quarteer);
                    e.stopPropagation();
                  }}
                />
              ) : null}
            </div>
          ))}

          <FontAwesomeIcon
            className={styles.addQuarterIcon}
            onClick={(e) => {
              e.stopPropagation();
              handleAddNewQuarter(data, totalQuarters);
            }}
            fontSize="20px"
            icon={faPlus}
```

```
      />
    </div>
  </div>
</div>

<div className={styles.lineWrapper}>
  <div
    className={styles.category}
    style={{
      backgroundColor: '#4066FE',
    }}
  >
    {t('milestones')}
  </div>
  <div className={styles.blocks}>
    <div
      className={styles.milestonesRow}
      onDoubleClick={handleMilestoneDoubleClick}
    >
      {milestones.map((milestone) => (
        <MilestoneComponent
          roadmapId={data._id}
          totalQuarters={totalQuarters}
          milestone={milestone}
          key={milestone._id}
          roadmapContentWidth={roadmapContentWidth}
        />
      ))}
    </div>
  </div>
</div>

{categories.map((category) => (
  <div key={category._id} className={styles.lineWrapper}>
    <div
      className={styles.category}
      style={{
        backgroundColor: category.color,
      }}
    >
      {category.title}

      <div className={styles.icons}>
        <FontAwesomeIcon
          className={`${styles.icon} ${styles.pencil}`}
          onClick={(e) => {
            handleOpenEditColumnModal(category);
            e.stopPropagation();
          }}
          fontSize="15px"
          icon={faPencil}
        />
        <FontAwesomeIcon
          fontSize="15px"
          icon={faTrash}
          className={`${styles.icon} ${styles.trash}`}
          onClick={(e) => {
            handleOpenDeleteColumnModal(category);
```

```
            e.stopPropagation();
          }}
        />
      </div>
    </div>

    <div className={`${styles.blocks} ${styles.rows}`}>
      {category.rows.map((row) => (
        <div
          onDoubleClick={(e) =>
            handleRowDoubleClick(e, row, category)
          }
          key={row._id}
          className={styles.row}
          onDragOver={(e) => e.preventDefault()}
          onDrop={(e) => handleTaskDrop(e, row, category)}
        >
          {row.tasks.map((task) => (
            <TaskComponent
              roadmapContentWidth={roadmapContentWidth}
              allTasksInRow={row.tasks}
              task={task}
              totalQuarters={totalQuarters}
              key={task._id}
              category={category}
              row={row}
              roadmapId={data._id}
            />
          ))}

          <FontAwesomeIcon
            fontSize="15px"
            icon={faTrash}
            className={styles.removeRowIcon}
            onClick={(e) => {
              e.stopPropagation();
              handleOpenDeleteRowModal(row, category);
            }}
          />
        </div>
      ))}

      <FontAwesomeIcon
        className={styles.addRowIcon}
        onClick={(e) => {
          e.stopPropagation();
          handleCreateRow(category, data);
        }}
        fontSize="20px"
        icon={faPlus}
      />
    </div>
  </div>
))}
<div className={styles.line}>
  <FontAwesomeIcon
    className={styles.addIcon}
    onClick={handleEditProjectSettingsModal}
```

```
          fontSize="20px"
          icon={faPlus}
        />
      </div>
    </div>
  </div>

  <Modal
    active={isProjectSettingsOpened}
    setActive={() => {
      setIsProjectSettingsOpened(false);
    }}
    ChildComponent={EditProjectInfo}
    childProps={{ members: data.members, projectId: data._id }}
  />
  <Modal
    active={isAddUserToProjectModalOpened}
    setActive={() => {
      setIsAddUserToProjectModalOpened(false);
    }}
    ChildComponent={AddUserToProjectModal}
    childProps={{
      isOpened: isAddUserToProjectModalOpened,
      projectId: data._id,
    }}
  />

  <Modal
    active={categoryEditing}
    setActive={() => {
      dispatch(setRoadmapCurrentCategory(null));
      dispatch(setRoadmapIsEditingCategoryOpened(false));
      dispatch(setRoadmapCurrentCategory(null));
    }}
    ChildComponent={CategoryForm}
    childProps={{ roadmapId: data._id }}
  />
  <Modal
    active={categoryDeleting}
    setActive={() => {
      dispatch(setRoadmapCurrentCategory(null));
      dispatch(setRoadmapIsDeletingCategoryOpened(false));
    }}
    ChildComponent={CategoryDeleting}
    childProps={{ roadmapId: data._id }}
  />

  <Modal
    active={rowDeleting}
    setActive={() => {
      dispatch(setRoadmapCurrentRow(null));
      dispatch(setRoadmapIsDeletingRowOpened(false));
    }}
    ChildComponent={RowDeleting}
    childProps={{ roadmapId: data._id }}
  />

  <Modal
```

```
        active={quarterDeleting}
        setActive={() => {
          dispatch(setRoadmapCurrentQuarter(null));
          dispatch(setRoadmapIsDeletingQuarterOpened(false));
        }}
        ChildComponent={QuarterDeleting}
        childProps={{ roadmapId: data._id }}
      />

      <Modal
        active={milestoneEditing}
        setActive={() => {
          dispatch(setRoadmapCurrentMilestone(null));
          dispatch(setRoadmapIsEditingMilestoneOpened(false));
          dispatch(setRoadmapClickPosition(0));
        }}
        ChildComponent={MilestoneForm}
        childProps={{ roadmapId: data._id }}
      />

      <Modal
        active={milestoneDeleting}
        setActive={() => {
          dispatch(setRoadmapCurrentMilestone(null));
          dispatch(setRoadmapIsDeletingMilestoneOpened(false));
        }}
        ChildComponent={MilestoneDeleting}
        childProps={{ roadmapId: data._id }}
      />

      <Modal
        active={taskEditing}
        setActive={() => {
          dispatch(setRoadmapCurrentTask(null));
          dispatch(setRoadmapIsEditingTaskOpened(false));
          dispatch(setRoadmapCurrentCategory(null));
          dispatch(setRoadmapCurrentRow(null));
          dispatch(setRoadmapClickPosition(0));
        }}
        ChildComponent={TaskForm}
        childProps={{ roadmapId: data._id }}
      />

      <Modal
        active={taskDeleting}
        setActive={() => {
          dispatch(setRoadmapCurrentTask(null));
          dispatch(setRoadmapIsDeletingTaskOpened(false));
          dispatch(setRoadmapCurrentCategory(null));
          dispatch(setRoadmapCurrentRow(null));
        }}
        ChildComponent={TaskDeleting}
        childProps={{ roadmapId: data._id }}
      />
    </div>
  );
};
```

```
export default withLoginRedirect(RoadMap);
```

RoadmapApi.ts

```
import instanse from '../axios';
import {
  Category,
  Milestone,
  Quarter,
  Row,
  Task,
} from '../redux/slices/roadmap/type';
import { Status } from '../types/shared';

import { RoadmapData } from './../redux/slices/roadmap/type';

export type RoadMapProjectShortInfo = {
  _id: string;
  title: string;
  description: string;
  creatorId: string;
  membersCount: number;
  updatedAt: string;
};

type ProjectFullInfo = RoadmapData;

type GetBoardsApiResponse = {
  data: {
    results: RoadMapProjectShortInfo[];
    page: number;
    totalPages: number;
  };
  status: number;
  statusText: string;
};

type GetBoardsResponse = {
  data: GetBoardsApiResponse['data'];
  message?: string;
  status: Status;
};

type GetBoardApiResponse = {
  data: ProjectFullInfo;
  status: number;
  statusText: string;
};

type GetBoardResponseSuccess = {
  status: Status.SUCCESS;
  data: ProjectFullInfo;
};

type GetBoardResponseFail = {
  status: Status.ERROR;
  message: string;
```

```
};

type CreateBoardPayload = {
  title: string;
  description: string;
};

type CreateBoardApiResponse = {
  data: RoadMapProjectShortInfo;
  status: number;
  statusText: string;
};

type CreateBoardResponseSuccess = {
  status: Status.SUCCESS;
  data: RoadMapProjectShortInfo;
};

type CreateBoardResponseFail = {
  status: Status.ERROR;
  message: string;
};

type UpdateBoardPayload = {
  boardId: string;
  title: string;
  description: string;
};

type UpdateBoardApiResponse = {
  data: RoadMapProjectShortInfo;
  status: number;
  statusText: string;
};

type UpdateBoardResponseSuccess = {
  status: Status.SUCCESS;
  data: RoadMapProjectShortInfo;
};

type UpdateBoardResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteBoardPayload = string;

type DeleteBoardApiResponse = {
  data: RoadMapProjectShortInfo;
  status: number;
  statusText: string;
};

type DeleteBoardResponseSuccess = {
  status: Status.SUCCESS;
  data: RoadMapProjectShortInfo;
};
```

```
type DeleteBoardResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type CreateCategoryPayload = {
  roadmapId: string;
  title: string;
  color: string;
};

export type UpdateCategoryPayload = {
  roadmapId: string;
  categoryId: string;
  title?: string;
  color?: string;
};

export type DeleteCategoryPayload = {
  roadmapId: string;
  categoryId: string;
};

type CreateCategoryApiResponse = {
  data: Category;
  status: number;
  statusText: string;
};

type CreateCategoryResponseSuccess = {
  status: Status.SUCCESS;
  data: Category;
};

type CreateCategoryResponseFail = {
  status: Status.ERROR;
  message: string;
};

type UpdateCategoryResponseSuccess = {
  status: Status.SUCCESS;
};

type UpdateCategoryResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteCategoryResponseSuccess = {
  status: Status.SUCCESS;
};

type DeleteCategoryResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type CreateRowPayload = {
```

```
  roadmapId: string;
  categoryId: string;
};

export type DeleteRowPayload = {
  roadmapId: string;
  categoryId: string;
  rowId: string;
};

type CreateRowApiResponse = {
  data: Row;
  status: number;
  statusText: string;
};

type CreateRowResponseSuccess = {
  status: Status.SUCCESS;
  data: Row;
};

type CreateRowResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteRowResponseSuccess = {
  status: Status.SUCCESS;
};

type DeleteRowResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type CreateQuarterPayload = {
  roadmapId: string;
  title: string;
  start: number;
  end: number;
};

export type UpdateQuarterPayload = {
  roadmapId: string;
  quarterId: string;
  title?: string;
  start?: number;
  end?: number;
};

export type DeleteQuarterPayload = {
  roadmapId: string;
  quarterId: string;
};

type CreateQuarterApiResponse = {
  data: Quarter;
  status: number;
```

```
  statusText: string;
};

type CreateQuarterResponseSuccess = {
  status: Status.SUCCESS;
  data: Quarter;
};

type CreateQuarterResponseFail = {
  status: Status.ERROR;
  message: string;
};

type UpdateQuarterResponseSuccess = {
  status: Status.SUCCESS;
};

type UpdateQuarterResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteQuarterResponseSuccess = {
  status: Status.SUCCESS;
};

type DeleteQuarterResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type CreateMilestonePayload = {
  roadmapId: string;
  title: string;
  position: number;
};

export type UpdateMilestonePayload = {
  roadmapId: string;
  milestoneId: string;
  title?: string;
  position?: number;
};

export type DeleteMilestonePayload = {
  roadmapId: string;
  milestoneId: string;
};

type CreateMilestoneApiResponse = {
  data: Milestone;
  status: number;
  statusText: string;
};

type CreateMilestoneResponseSuccess = {
  status: Status.SUCCESS;
  data: Milestone;
```

```
};

type CreateMilestoneResponseFail = {
  status: Status.ERROR;
  message: string;
};

type UpdateMilestoneResponseSuccess = {
  status: Status.SUCCESS;
};

type UpdateMilestoneResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteMilestoneResponseSuccess = {
  status: Status.SUCCESS;
};

type DeleteMilestoneResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type CreateTaskPayload = {
  roadmapId: string;
  categoryId: string;
  rowId: string;
  title: string;
  progress?: number;
  start?: number;
  end?: number;
};

export type UpdateTaskPayload = {
  roadmapId: string;
  categoryId: string;
  rowId: string;
  taskId: string;
  title?: string;
  progress?: number;
  start?: number;
  end?: number;
};

export type DeleteTaskPayload = {
  roadmapId: string;
  categoryId: string;
  rowId: string;
  taskId: string;
};

type CreateTaskApiResponse = {
  data: Task;
  status: number;
  statusText: string;
};
```

```
type CreateTaskResponseSuccess = {
  status: Status.SUCCESS;
  data: Task;
};

type CreateTaskResponseFail = {
  status: Status.ERROR;
  message: string;
};

type UpdateTaskResponseSuccess = {
  status: Status.SUCCESS;
};

type UpdateTaskResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type MoveTaskPayload = {
  roadmapId: string;
  categoryId: string;
  rowId: string;
  taskId: string;
  toCategoryId: string;
  toRowId: string;
  start: number;
  end: number;
};

type MoveTaskResponseSuccess = {
  status: Status.SUCCESS;
};

type MoveTaskResponseFail = {
  status: Status.ERROR;
  message: string;
};

type DeleteTaskResponseSuccess = {
  status: Status.SUCCESS;
};

type DeleteTaskResponseFail = {
  status: Status.ERROR;
  message: string;
};

export type AddUserToRoadmapPayload = {
  roadmapId: string;
  userId: string;
};

export type RemoveUserFromRoadmapPayload = {
  roadmapId: string;
  userId: string;
};
```

```
export type LeaveRoadmapPayload = string;

type RoadmapUserActionResponseSuccess = {
  status: Status.SUCCESS;
};

type RoadmapUserActionResponseFail = {
  status: Status.ERROR;
  message: string;
};

class roadmapAPIClass {
  public async getBoards(
    page?: number,
    limit = 10,
  ): Promise<GetBoardsResponse> {
    try {
      const response: GetBoardsApiResponse = await instanse.get('roadmap', {
        params: {
          ...(page ? { page } : {}),
          ...(limit ? { limit } : {}),
        },
      });

      return { status: Status.SUCCESS, data: response.data };
    } catch (err: any) {
      return {
        data: { results: [], page: 0, totalPages: 0 },
        message: err?.response?.data?.message || 'Error',
        status: Status.ERROR,
      };
    }
  }

  public async getBoard(
    boardId: string,
  ): Promise<GetBoardResponseSuccess | GetBoardResponseFail> {
    try {
      const response: GetBoardApiResponse = await instanse.get(
        `roadmap/${boardId}`,
      );

      return {
        status: Status.SUCCESS,
        data: response.data,
      };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error',
      };
    }
  }

  public async createBoard(
    payload: CreateBoardPayload,
  ): Promise<CreateBoardResponseSuccess | CreateBoardResponseFail> {
```

```
  try {
    const response: CreateBoardApiResponse = await instanse.post(
      `roadmap`,
      payload,
    );
    return { status: Status.SUCCESS, data: response.data };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error',
    };
  }
}

public async updateBoard(
  payload: UpdateBoardPayload,
): Promise<UpdateBoardResponseSuccess | UpdateBoardResponseFail> {
  try {
    const response: UpdateBoardApiResponse = await instanse.patch(
      `roadmap/${payload.boardId}`,
      { title: payload.title, description: payload.description },
    );
    return { status: Status.SUCCESS, data: response.data };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error',
    };
  }
}

public async deleteBoard(
  payload: DeleteBoardPayload,
): Promise<DeleteBoardResponseSuccess | DeleteBoardResponseFail> {
  try {
    const response: DeleteBoardApiResponse = await instanse.delete(
      `roadmap/${payload}`,
    );
    return { status: Status.SUCCESS, data: response.data };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error',
    };
  }
}

public async createCategory(
  payload: CreateCategoryPayload,
): Promise<CreateCategoryResponseSuccess | CreateCategoryResponseFail> {
  try {
    const response: CreateCategoryApiResponse = await instanse.post(
      `roadmap/${payload.roadmapId}/category`,
      {
        title: payload.title,
        color: payload.color,
      },
    );
```

```
    return {
      status: Status.SUCCESS,
      data: response.data,
    };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error creating category',
    };
  }
}

public async updateCategory(
  payload: UpdateCategoryPayload,
): Promise<UpdateCategoryResponseSuccess | UpdateCategoryResponseFail> {
  try {
    await instanse.patch(
      `roadmap/${payload.roadmapId}/category/${payload.categoryId}`,
      {
        title: payload.title,
        color: payload.color,
      },
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error updating category',
    };
  }
}

public async deleteCategory(
  payload: DeleteCategoryPayload,
): Promise<DeleteCategoryResponseSuccess | DeleteCategoryResponseFail> {
  try {
    await instanse.delete(
      `roadmap/${payload.roadmapId}/category/${payload.categoryId}`,
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error deleting category',
    };
  }
}

public async createRow(
  payload: CreateRowPayload,
): Promise<CreateRowResponseSuccess | CreateRowResponseFail> {
  try {
    const response: CreateRowApiResponse = await instanse.post(
      `roadmap/${payload.roadmapId}/category/${payload.categoryId}/row`,
    );
    return {
      status: Status.SUCCESS,
      data: response.data,
```

```
    };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error creating row',
    };
  }
}

public async deleteRow(
  payload: DeleteRowPayload,
): Promise<DeleteRowResponseSuccess | DeleteRowResponseFail> {
  try {
    await instanse.delete(
      `roadmap/${payload.roadmapId}/category/${payload.categoryId}/row/${payload.rowId}`,
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error deleting row',
    };
  }
}

public async createQuarter(
  payload: CreateQuarterPayload,
): Promise<CreateQuarterResponseSuccess | CreateQuarterResponseFail> {
  try {
    const response: CreateQuarterApiResponse = await instanse.post(
      `roadmap/${payload.roadmapId}/quarter`,
      {
        title: payload.title,
        start: payload.start,
        end: payload.end,
      },
    );
    return {
      status: Status.SUCCESS,
      data: response.data,
    };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error creating quarter',
    };
  }
}

public async updateQuarter(
  payload: UpdateQuarterPayload,
): Promise<UpdateQuarterResponseSuccess | UpdateQuarterResponseFail> {
  try {
    await instanse.patch(
      `roadmap/${payload.roadmapId}/quarter/${payload.quarterId}`,
      {
        title: payload.title,
        start: payload.start,
```

```
        end: payload.end,
      },
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error updating quarter',
    };
  }
}

public async deleteQuarter(
  payload: DeleteQuarterPayload,
): Promise<DeleteQuarterResponseSuccess | DeleteQuarterResponseFail> {
  try {
    await instanse.delete(
      `roadmap/${payload.roadmapId}/quarter/${payload.quarterId}`,
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error deleting quarter',
    };
  }
}

public async createMilestone(
  payload: CreateMilestonePayload,
): Promise<CreateMilestoneResponseSuccess | CreateMilestoneResponseFail> {
  try {
    const response: CreateMilestoneApiResponse = await instanse.post(
      `roadmap/${payload.roadmapId}/milestone`,
      {
        title: payload.title,
        position: payload.position,
      },
    );
    return {
      status: Status.SUCCESS,
      data: response.data,
    };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error creating milestone',
    };
  }
}

public async updateMilestone(
  payload: UpdateMilestonePayload,
): Promise<UpdateMilestoneResponseSuccess | UpdateMilestoneResponseFail> {
  try {
    await instanse.patch(
      `roadmap/${payload.roadmapId}/milestone/${payload.milestoneId}`,
      {
```

```
        title: payload.title,
        position: payload.position,
      },
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error updating milestone',
    };
  }
}

public async deleteMilestone(
  payload: DeleteMilestonePayload,
): Promise<DeleteMilestoneResponseSuccess | DeleteMilestoneResponseFail> {
  try {
    await instanse.delete(
      `roadmap/${payload.roadmapId}/milestone/${payload.milestoneId}`,
    );
    return { status: Status.SUCCESS };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error deleting milestone',
    };
  }
}

public async createTask(
  payload: CreateTaskPayload,
): Promise<CreateTaskResponseSuccess | CreateTaskResponseFail> {
  try {
    const response: CreateTaskApiResponse = await instanse.post(
      `roadmap/${payload.roadmapId}/category/${payload.categoryId}/row/${payload.rowId}/task`,
      {
        title: payload.title,
        progress: payload.progress,
        start: payload.start,
        end: payload.end,
      },
    );
    return {
      status: Status.SUCCESS,
      data: response.data,
    };
  } catch (err: any) {
    return {
      status: Status.ERROR,
      message: err?.response?.data?.message || 'Error creating task',
    };
  }
}

public async updateTask(
  payload: UpdateTaskPayload,
): Promise<UpdateTaskResponseSuccess | UpdateTaskResponseFail> {
  try {
```

```
      await instanse.patch(

`roadmap/${payload.roadmapId}/category/${payload.categoryId}/row/${payload.rowId}/task/${payload.taskId
}`,
        {
          title: payload.title,
          progress: payload.progress,
          start: payload.start,
          end: payload.end,
        },
      );
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error updating task',
      };
    }
  }

  public async moveTask(
    payload: MoveTaskPayload,
  ): Promise<MoveTaskResponseSuccess | MoveTaskResponseFail> {
    try {
      await instanse.patch(

`roadmap/${payload.roadmapId}/category/${payload.categoryId}/row/${payload.rowId}/task/${payload.taskId
}/move`,
        {
          toCategoryId: payload.toCategoryId,
          toRowId: payload.toRowId,
          start: payload.start,
          end: payload.end,
        },
      );
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error moving task',
      };
    }
  }

  public async deleteTask(
    payload: DeleteTaskPayload,
  ): Promise<DeleteTaskResponseSuccess | DeleteTaskResponseFail> {
    try {
      await instanse.delete(

`roadmap/${payload.roadmapId}/category/${payload.categoryId}/row/${payload.rowId}/task/${payload.taskId
}`,
      );
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error deleting task',
```

```
      };
    }
  }

  public async addUserToRoadmap(
    payload: AddUserToRoadmapPayload,
  ): Promise<RoadmapUserActionResponseSuccess | RoadmapUserActionResponseFail> {
    try {
      await instanse.post(
        `roadmap/${payload.roadmapId}/add-user/${payload.userId}`,
      );
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error adding user',
      };
    }
  }

  public async removeUserFromRoadmap(
    payload: RemoveUserFromRoadmapPayload,
  ): Promise<RoadmapUserActionResponseSuccess | RoadmapUserActionResponseFail> {
    try {
      await instanse.delete(
        `roadmap/${payload.roadmapId}/remove-user/${payload.userId}`,
      );
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error removing user',
      };
    }
  }

  public async leaveRoadmap(
    roadmapId: LeaveRoadmapPayload,
  ): Promise<RoadmapUserActionResponseSuccess | RoadmapUserActionResponseFail> {
    try {
      await instanse.delete(`roadmap/${roadmapId}/leave`);
      return { status: Status.SUCCESS };
    } catch (err: any) {
      return {
        status: Status.ERROR,
        message: err?.response?.data?.message || 'Error leaving roadmap',
      };
    }
  }
}

const roadmapAPI = new roadmapAPIClass();
export default roadmapAPI;
```

Roadmap.tsx

```
@import 'src/styles/partials/theme';
```

```
.wrapper {
  display: flex;
  flex-direction: column;
  width: 100%;
  height: calc(100% - 80px);
  overflow-y: auto;
  overflow-x: hidden;
  padding: 20px;
  color: white;
  transition: color $transition-delay;
}

.projectLine {
  display: flex;
  align-items: center;
  justify-content: space-between;
  width: 100%;
  margin-bottom: 40px;
  position: sticky;
  left: 0;
}

.addColumnButton {
  margin-top: 10px;
  align-self: flex-start;
  background-color: $info-200;
  color: $white;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color $transition-delay;
  font-weight: bold;
  font-size: 18px;

  &:hover {
    background-color: $amber-300;
  }
}

.options {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.gear {
  color: white;
  width: 20px;
  height: 20px;
  z-index: 3;
  cursor: pointer;
  transition: color $transition-delay;

  &:hover {
    color: $warning-200;
  }
}
```

```
.user {
  color: white;
  width: 20px;
  height: 20px;
  z-index: 3;
  cursor: pointer;
  transition: color $transition-delay;

  &:hover {
    color: $info-200;
  }
}

.columns {
  display: flex;
  gap: 20px;
  height: calc(100% - 44px);
  min-height: 0;
  width: 100%;
  overflow-x: auto;
  overflow-y: hidden;
  position: relative;
  scroll-behavior: smooth;
}

.backToAllBoards {
  width: fit-content;
  color: white;
  font-size: 20px;
  font-weight: 600;
  transition: color $transition-delay;

  &:hover {
    color: $warning-200;
  }
}

.projectTitle {
  text-align: center;
  font-size: 24px;
  font-weight: bold;
  margin-bottom: 10px;
  line-height: 30px;
}

.projectDescription {
  text-align: center;
  font-size: 18px;
  font-weight: bold;
  line-height: 30px;
}

.roadmap {
  overflow-x: auto;
  overflow-y: visible;
  width: 100%;
}
```

```
.roadmapContent {
  display: flex;
  flex-direction: column;
  gap: 20px;
  min-width: fit-content;
  position: relative;
}

.lineWrapper {
  display: flex;
  width: 100%;
  gap: 10px;
}

.category {
  overflow: hidden;
  display: flex;
  width: 200px;
  align-items: center;
  justify-content: center;
  padding: 10px;
  flex-shrink: 0;
  font-weight: 700;
  color: white;
  position: sticky;
  left: 0;
  z-index: 3;

  &:hover {
    .icons {
      opacity: 1;
      pointer-events: all;
    }
  }
}

.icons {
  display: flex;
  align-items: center;
  gap: 5px;
  position: absolute;
  right: 0;
  top: 0;
  display: flex;
  flex-direction: column;
  padding: 5px;
  transition: opacity $transition-delay;
  opacity: 0;
  pointer-events: none;
}

.icon {
  width: 15px;
  height: 15px;
  z-index: 3;
  cursor: pointer;
  transition: color $transition-delay;
```

```
}

.pencil {
  color: $white;

  &:hover {
    color: $black;
  }
}

.trash {
  color: $white;

  &:hover {
    color: $black;
  }
}

.blocks {
  display: flex;
  flex-grow: 1;
}

.quarteers {
  display: flex;
  width: 100%;
  flex-grow: 1;
  position: relative;
}

.addQuarterIcon {
  width: 20px;
  height: 20px;
  z-index: 3;
  cursor: pointer;
  transition: color $transition-delay;

  &:hover {
    color: $warning-200;
  }
  position: absolute;
  right: 0;
  top: 50%;
  transform: translateY(-50%);
}

.removeQuarterIcon {
  cursor: pointer;
  position: absolute;
  right: 20%;
  top: 50%;
  opacity: 0;
  pointer-events: none;
  color: $white;
  transform: translateY(-50%);
  transition:
    opacity $transition-delay,
    color $transition-delay;
```

```
  &:hover {
    color: $error-200;
  }
}

.quarteer {
  padding: 5px;
  text-align: center;
  display: flex;
  align-items: center;
  justify-content: center;
  position: relative;

  &:hover {
    .removeQuarterIcon {
      opacity: 1;
      pointer-events: all;
    }
  }
}

.milestonesRow {
  overflow-x: hidden;
  display: flex;
  background-color: $background-dark-100;
  width: 100%;
  position: relative;
  transition: background-color $transition-delay;
}

.rows {
  display: flex;
  gap: 2px;
  width: 100%;
  flex-direction: column;
  position: relative;

  .row {
    background-color: $background-dark-100;
    height: 40px;
    width: 100%;
    flex-grow: 1;
    display: flex;
    position: relative;
    border-radius: 2px;
    transition: background-color $transition-delay;
  }
}

.addRowIcon {
  width: 20px;
  height: 20px;
  z-index: 3;
  cursor: pointer;
  transition:
    color $transition-delay,
    opacity $transition-delay;
```

```
  position: absolute;
  left: 0;
  bottom: 0;
  transform: translateY(50%);
  opacity: 0;
  pointer-events: none;

  &:hover {
    color: $warning-200;
  }
 }
 .rows {
  &:hover {
    .addRowIcon {
      opacity: 1;
      pointer-events: all;
    }
  }
 }

 .quarterLines {
  position: absolute;
  top: 0;
  left: 210px;
  width: calc(100% - 210px);
  height: calc(100% - 80px);
  pointer-events: none;
 }

 .quarterLine {
  position: absolute;
  top: 0;
  bottom: 0;
  width: 2px;
  background-color: white;
  transition: background-color $transition-delay;
 }

 .line {
  display: flex;
  justify-content: start;
  align-items: center;
  padding: 20px 0;
  position: relative;
 }

 .addIcon {
  position: sticky;
  left: 0px;
  width: 20px;
  height: 20px;
  z-index: 3;
  cursor: pointer;
  transition: color $transition-delay;

  &:hover {
    color: $warning-200;
  }
```

```
}

.removeRowIcon {
  cursor: pointer;
  position: absolute;
  right: 0;
  top: 50%;
  transform: translate(100%, -50%);
  opacity: 0;
  pointer-events: none;
  color: $white;
  transition:
    opacity $transition-delay,
    color $transition-delay;

  &:hover {
    color: $error-200;
  }
}

.error {
  color: $error-200;
  text-align: center;
  width: 100%;
}

:global(.light_theme) .removeRowIcon {
  color: $black;

  &:hover {
    color: $error-200;
  }
}

:global(.light_theme) .removeQuarterIcon {
  color: $black;

  &:hover {
    color: $error-200;
  }
}

.row:hover {
  .removeRowIcon {
    opacity: 1;
    pointer-events: all;
  }
}

:global(.light_theme) {
  .wrapper {
    color: black;
  }
  .rows {
    .row {
      background-color: $gray-600;
    }
  }
```

```
  .milestonesRow {
    background-color: $gray-600;

    .quarteer {
      border-color: black;
    }
  }
  .quarterLine {
    background-color: black;
  }
}

:global(.light_theme) {
  .wrapper {
    background: $gray-500;
    color: black;
  }

  .gear {
    color: black;
    width: 20px;

    &:hover {
      color: $warning-200;
    }
  }

  .user {
    color: black;

    &:hover {
      color: $info-200;
    }
  }

  .backToAllBoards {
    color: black;

    &:hover {
      color: $warning-200;
    }
  }
}
```