Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка вебзастосунку для оптимізації процесів управління проєктами»

Виконав: студентка IV курсу
групи 1ПІ-21б (д/н) спеціальності
121 – Інженерія програмного забезпечення
А. В. Барцицька
(прізвище та ініціали)

Керівник: д-р. філос., асист. каф. ПЗ Карась О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Черняк О. І.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О. Н. Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Барцицької Анни Володимирівни

1. Тема роботи – Розробка вебзастосунку для оптимізації процесів управління проєктами.

Керівник роботи: Карась Олександр Володимирович, д.ф., асист. каф. ПЗ Карась О.В., затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 97.

2. Строк подання студентом роботи: 2 червня 2025.

3. Базові технології веброзробки – фреймворк Next.js 14 (React), серверна платформа Appwrite; база даних – Appwrite Databases; автентифікація – OAuth 2.0 (Google, GitHub); вхідні дані – інформація про користувачів, задачі, робочі простори, календарні події; вимоги до інтерфейсу – адаптивний дизайн, Tailwind CSS, інтерактивні компоненти, модальні вікна; алгоритми – створення, фільтрація, сортування задач; синхронізація в реальному часі; вихідні дані – багатокористувацький вебзастосунок із функціями керування задачами, ролями, робочими просторами та календарем.

4. Зміст текстової частини: вступ; аналіз предметної області, сучасних підходів та інструментів до організації проєктної діяльності; аналіз існуючих рішень (Jira, Trello, Asana); обґрунтування вибору методів реалізації та постановка задач; розробка архітектури та алгоритмів функціонування системи; побудова діаграм та інтерфейсів; реалізація модулів авторизації, керування задачами, робочими просторами та членами команди; модульне тестування застосунку, опис

сценаріїв тестування та результати; розробка інструкції користув
формулювання висновків щодо ефективності запропонованого рішення.

5. Перелік ілюстративного матеріалу: титульний слайд — автор, науко
керівник бакалаврської кваліфікаційної роботи, тема; актуальність розро
мета; об'єкт та предмет дослідження; постановка задач; наукова нови
практична цінність; аналіз аналогів; обґрунтування вибору техноло
архітектура вебзастосунку; модель роботи системи; загальний алгоритм роб
схема взаємодії модулів; діаграми станів; прототип інтерфейсу користув
реалізовані модулі (авторизація, задачі, робочі простори, команда); прикл
фрагментів коду; результати тестування; інструкція користувача; апроб
результатів; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Карась О. В., д.ф., асист. каф. ПЗ	24.03.2025	01.06.2025

7. Дата видачі завдання 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз стану розвитку програм для роботи над проєктами в ІТ-командах	25.03.2025-08.04.2025	виконано
3	Розробка методів, моделі та алгоритмів роботи програми	09.04.2025-18.04.2025	виконано
3	Розробка програмного забезпечення	19.04.2025-07.05.2025	виконано
4	Тестування програми	08.05.2025-16.05.2025	виконано
5	Оформлення матеріалів до захисту БКР	17.05.2025-30.05.2025	виконано

Студент _____ **Барцицька**
(підпис) (прізвище та іні

Керівник бакалаврської кваліфікаційної роботи _____ **Карась О**
(підпис) (прізвище та іні

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота містить 67 сторінок формату А4, на яких наведено 45 рисунків та 6 таблиць, а список використаних джерел налічує 23 найменувань.

У бакалаврській кваліфікаційній роботі розроблено вебзастосунок для оптимізації процесів управління проєктами. Проведено аналіз сучасних систем управління проєктами, визначено їх переваги та недоліки, а також обґрунтовано необхідність створення власного рішення.

У роботі обґрунтовано вибір архітектури системи, мови програмування, інструментів та фреймворків, що були використані під час розробки. Реалізовано функціональність створення та керування робочими просторами, задачами, командами, ролями користувачів і календарем у межах єдиного інтерфейсу. Застосунок підтримує багатокористувацьку взаємодію, drag-and-drop Kanban-дошку, аналітику прогресу та сучасні механізми авторизації.

Для реалізації вебзастосунку використано стек технологій: Next.js 14, Appwrite, Tailwind CSS, React Query, Zod. Розроблене рішення може бути використано в IT-компаніях, стартапах і фріланс-командах для підвищення ефективності управління проєктною діяльністю.

Ключові слова: вебзастосунок, управління проєктами, командна робота, Kanban, Next.js, Appwrite, Tailwind CSS.

ABSTRACT

The bachelor's thesis comprises 67 A4-sized pages, containing 45 pictures and 6 tables. The list of references includes 23 sources.

This bachelor's thesis presents the development of a web application aimed at optimizing the project work of IT teams. An analysis of modern project management systems was conducted, their advantages and disadvantages were identified, and the need to develop a custom solution was substantiated.

The thesis justifies the choice of system architecture, programming languages, tools, and frameworks used during development. The application implements functionality for creating and managing workspaces, tasks, teams, user roles, and a calendar within a unified interface. The system supports multi-user interaction, a drag-and-drop Kanban board, progress analytics, and modern authentication mechanisms.

The web application was developed using the following technology stack: Next.js 14, Appwrite, Tailwind CSS, React Query, and Zod. The resulting solution can be used by IT companies, startups, and freelance teams to enhance the efficiency of project management.

Keywords: web application, project management, teamwork, Kanban, Next.js, Appwrite, Tailwind CSS.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку інформаційних технологій характеризується активним впровадженням цифрових рішень у всі сфери діяльності, зокрема в управління проєктами. Зростаюча складність проєктів, особливо в IT-сфері, потребує високого рівня координації між учасниками команди, оперативного обміну інформацією, контролю виконання завдань та своєчасного виявлення ризиків [1]. Віддалені команди також пропонують значні переваги, такі як економія коштів, гнучкий графік, зниження тривожності та стресу, покращений баланс між роботою та особистим життям та усунення необхідності поїздок на роботу та з роботи [2]. У цих умовах особливої актуальності набувають вебсистеми, які забезпечують ефективну організацію командної роботи над проєктами.

Наявні інструменти для керування проєктами (такі як Trello, Asana, Jira) є потужними, проте часто мають надлишкову функціональність або складний інтерфейс, що може знижувати ефективність роботи команд. Також не всі системи забезпечують належний рівень адаптивності до специфічних потреб окремої команди або проєкту. У зв’язку з цим виникає потреба у створенні веборієнтованого програмного рішення для оптимізації процесів управління проєктами, яка дозволить автоматизувати планування завдань, відслідковувати статуси виконання, забезпечити ефективну комунікацію та централізоване управління робочими просторами.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є оптимізації командної роботи над проєктами за рахунок розробки системи для автоматизації планування, обліку виконаних завдань та візуалізації прогресу.

Відповідно до поставленої мети потрібно вирішити такі завдання:

- розробити зручний, адаптивний та інтуїтивно зрозумілий інтерфейс вебзастосунку для керування проєктами, що підтримує сучасні UX/UI практики та адаптацію до різних пристроїв;
- забезпечити повноцінне тестування вебзастосунку з метою перевірки стабільності, надійності та безпеки;
- розробити систему для оптимізації процесів управління проєктами, що включає:

1) реєстрацію облікового запису члена команди;
2) функціонал для керування робочими просторами, включаючи можливість керувати членами команди;
3) функціонал для керування проєктами ;
4) функціонал для керування задачми, включати вибір статусів, вкинуючих та виставлення дедлайнів виконання;
5) можливість переглядати та керувати задачами в трьох різних варіантах, а саме у вигляду таблиці, дошки та календаря;
6) розробка дашборду для перегляду інформації по обраному проєкту з аналітичними даними.

Об’єктом дослідження є процес розробки програмного продукту для організації командної роботи в рамках реалізації проєктів.

Предметом дослідження є методи та засоби розробки програмного продукту для оптимізації процесів управління проєктами.

Методи дослідження. У процесі дослідження використовувалися наступні методи:

- компонентність підходу в розробці інтерфейсу на базі Next.js та Tailwind CSS;
- розробка REST-інтерфейсів і логіки взаємодії клієнта з сервером за допомогою Hono та Appwrite;
- методи валідації вхідних даних із застосуванням Zod;

- методи тестування системи із застосуванням сучасних засобів перевірки користувацьких сценаріїв;
- методи автентифікації з використанням OAuth.

Новизна отриманих результатів.

1. Розроблено вебзастосунок для оптимізації процесів управління проєктами, який, на відміну від відомих рішень, поєднує базові функції керування проєктами (Kanban, календар, таблиця) з підтримкою ролей та робочих просторів у межах єдиної системи.

2. Запропоновано спрощену модульну архітектуру, що полегшує масштабування та додавання нових функцій без суттєвих змін у кодовій базі.

Практичне значення проєкту полягає у можливості використання створенні зручного та ефективного інструменту для малих і середніх команд, який дозволяє поліпшити організацію роботи, зменшити кількість помилок, пов’язаних із людським фактором, а також підвищити прозорість та контроль виконання проєктних завдань. Система може бути використана як у комерційних, так і в навчальних або дослідницьких цілях.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У науковій праці [3], опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, перелік розроблених функцій.

Апробація матеріалів. Основні положення роботи доповідалися на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» (Вінниця, 2025).

Публікації. Результати досліджень опубліковано в 1 науковій статті у матеріалах конференції [3].

1 АНАЛІЗ ПІДХОДІВ ТА ІНСТРУМЕНТІВ ОРГАНІЗАЦІЇ ПРОЄКТНОЇ ДІЯЛЬНОСТІ І ПОСТАНОВКА ЗАДАЧ

1.1 Огляд предметної області

Сучасний розвиток цифрових технологій кардинально змінює підходи до організації командної роботи над проєктами. Зі зростанням обсягів інформації, збільшенням складності задач та розподілом команд у просторі та часі, виникає потреба в ефективних інструментах управління проєктами, які здатні забезпечити контроль, планування, комунікацію та звітність у єдиному середовищі.

Традиційні методи організації командної роботи, що базуються на фізичних зустрічах, ручному плануванні або обміні інформацією через електронну пошту, більше не задовольняють потреб сучасних проєктів. Вони часто призводять до втрати важливої інформації, зниження продуктивності та неузгодженості дій команди. У цьому контексті веборієнтовані системи управління проєктами стають невід’ємною частиною робочого процесу.

Існуючі рішення, такі як Trello, Asana, Jira, ClickUp та інші, надають можливість ставити задачі, відслідковувати прогрес, забезпечувати взаємодію між учасниками та автоматизувати певні процеси. Вони пропонують візуальні інтерфейси у вигляді канбан-дошок, діаграм Ганта або календарів, що допомагає краще зрозуміти стан проєкту. Проте, незважаючи на широкий спектр функцій, багато з цих систем не підходять для всіх типів команд або проєктів. Їх використання може бути занадто складним з превантаженим функціоналом.

Особливо важливим стає підхід до створення модульних вебсистем, які дозволяють формувати рішення відповідно до конкретних потреб. Така система може включати окремі модулі для управління завданнями, контролю дедлайнів, аналітики виконання, комунікацій та інтеграцій із зовнішніми сервісами. Модульність сприяє адаптивності системи, зменшує навантаження на користувача та забезпечує масштабованість у разі зростання команди або ускладнення структури проєкту [4].

Окрім функціональності, важливими аспектами є також інтерфейс користувача, зручність взаємодії та доступність платформи на різних пристроях. Успішні вебсистеми поєднують гнучку архітектуру бекенду (часто з використанням REST або GraphQL API) із сучасними фронтенд-фреймворками (такими як React чи Vue.js), що дозволяє забезпечити високу швидкість реакції, реальновременні оновлення та підтримку різних форматів відображення.

Важливо також відзначити роль таких систем у віддаленій роботі, яка стала нормою після глобальної пандемії COVID-19. Команди, що працюють із різних куточків світу, потребують спільного простору для взаємодії, і саме вебсистеми відіграють ключову роль у підтримці цього процесу [5].

Отже, предметна область розробки вебсистем для оптимізації командної роботи охоплює безліч аспектів: від проектування архітектури та вибору технологій до UX-дизайну, аналітики та інтеграції з іншими сервісами. Успішна реалізація програмних модулів у цій галузі не лише покращує управління проєктами, але й підвищує загальну ефективність роботи команди, зменшує ймовірність помилок і втрати даних, а також забезпечує зручну та зрозумілу взаємодію користувачів із системою.

1.2 Аналіз існуючих рішень

У світі, де IT-проєкти стають дедалі складнішими, а команди працюють на глобальному рівні, використання спеціалізованих вебзастосунків для організації командної роботи стало справжньою необхідністю. Головна мета таких систем – забезпечити ефективну комунікацію, прозорість завдань, контроль термінів і результатів, що дозволяє командам працювати злагоджено, незалежно від того, де знаходяться їх учасники. Сучасні рішення для командної роботи активно поєднують функції управління проєктами, відстеження задач, хмарного зберігання даних, інтеграції з месенджерами та інструментами для розробки програмного забезпечення.

До найвідоміших рішень у цій сфері належать:

- Trello;

- Asana;
- Jira Software;
- ClickUp;
- Monday.com.

Одним із популярних інструментів є Trello (рис. 1.1) – візуальний менеджер завдань на основі методології Kanban. Він дозволяє створювати дошки, списки та картки для структурування задач у межах проєктів. Система підтримує додавання тегів, чеклістів, термінів виконання та дозволяє здійснювати колективну роботу в реальному часі [6].

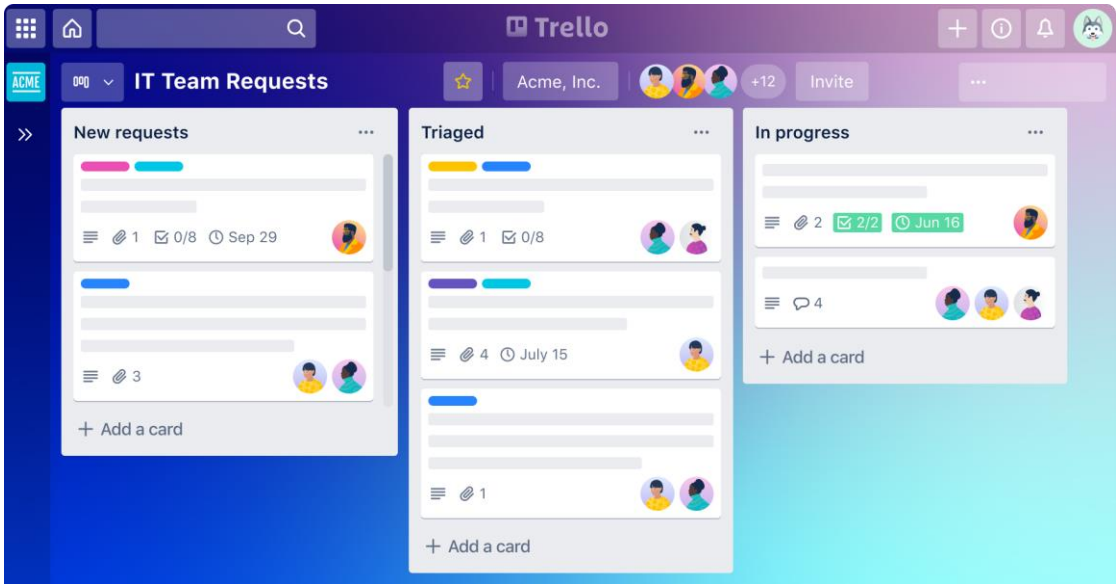


Рисунок 1.1 – Інтерфейс застосунку Trello

Інше рішення – Asana (рис. 1.2) – потужний інструмент для планування, відстеження та керування командною діяльністю. Він надає ширші можливості, ніж Trello, зокрема таймлайни, залежності задач, автоматизації та інтеграції з понад 100 іншими сервісами [7].

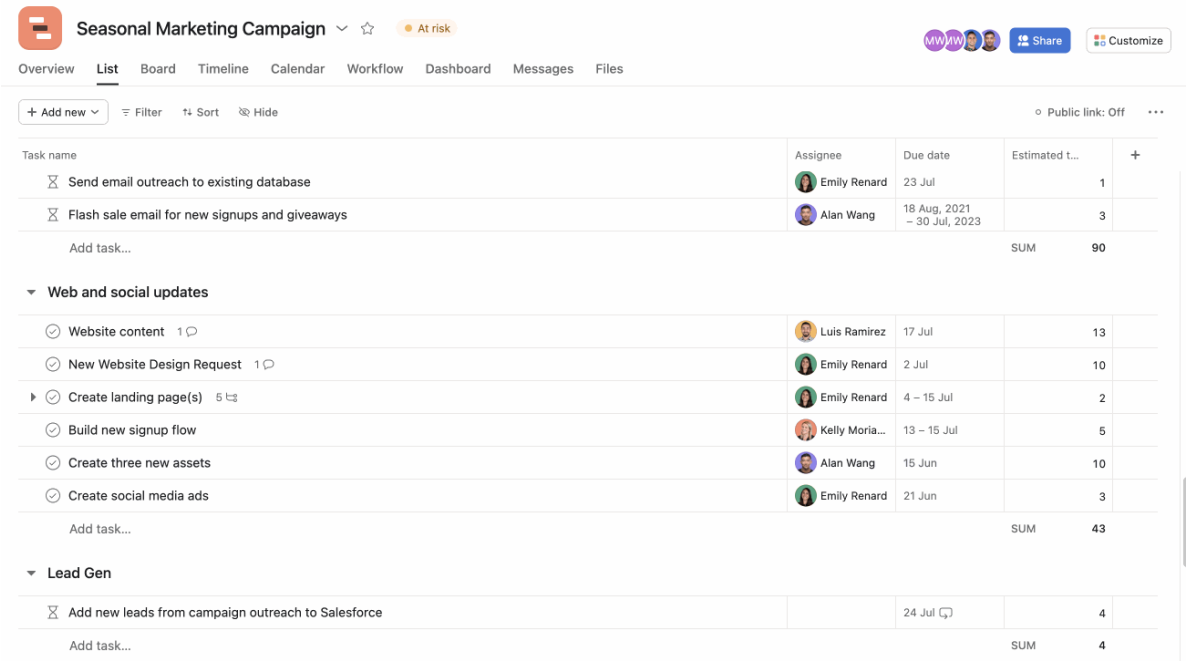


Рисунок 1.2 – Інтерфейс застосунку Asana

Jira Software (рис. 1.3), розроблений компанією Atlassian, є найбільш розповсюдженим рішенням у сфері розробки програмного забезпечення. Він дозволяє здійснювати гнучке керування проєктами за методологіями Scrum, Kanban або гібридними підходами, відстежувати баги, релізи, спринти, метрики продуктивності та інтегрується з системами контролю версій [8].

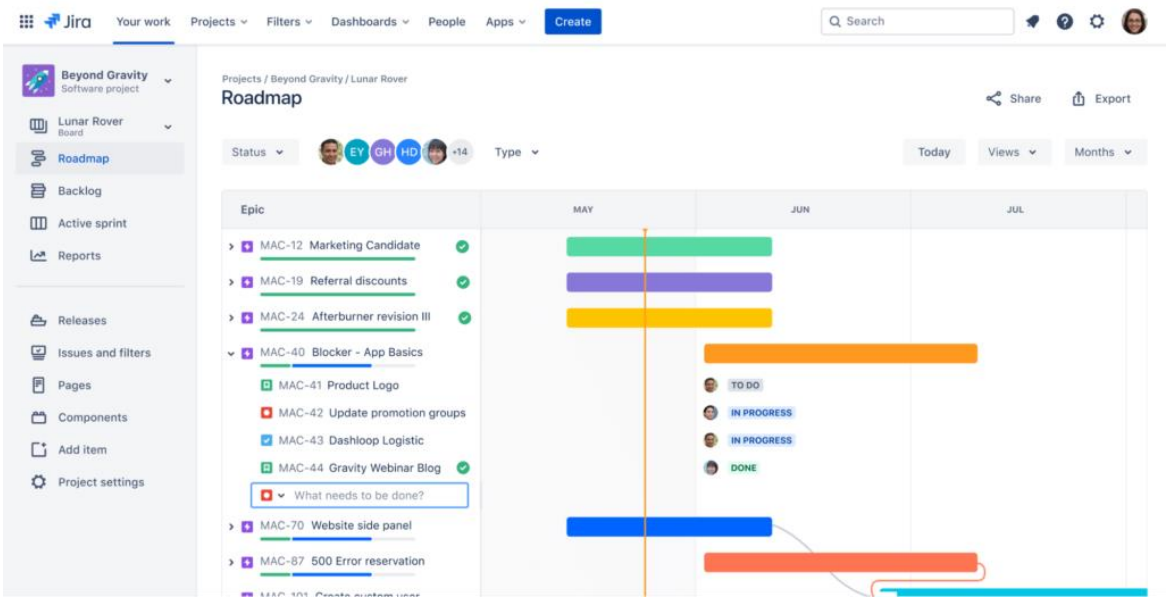


Рисунок 1.3 – Інтерфейс застосунку Jira Software

Система ClickUp (рис. 1.4) пропонує універсальне середовище для організації командної роботи, яке об’єднує керування завданнями, документацією, постановку цілей, тайм-трекінг, звітність і автоматизацію процесів.

ClickUp підтримує кілька типів візуалізації, зокрема Kanban-дошки, таблиці, календарі, діаграми Ганта, а також має інструменти фільтрації й аналітики. Завдяки інтеграціям із такими сервісами, як Slack, Google Calendar, GitHub, Zoom тощо, платформа забезпечує взаємодію в межах одного інтерфейсу. Її основна перевага – гнучкість налаштування інтерфейсу під потреби конкретної команди [9].

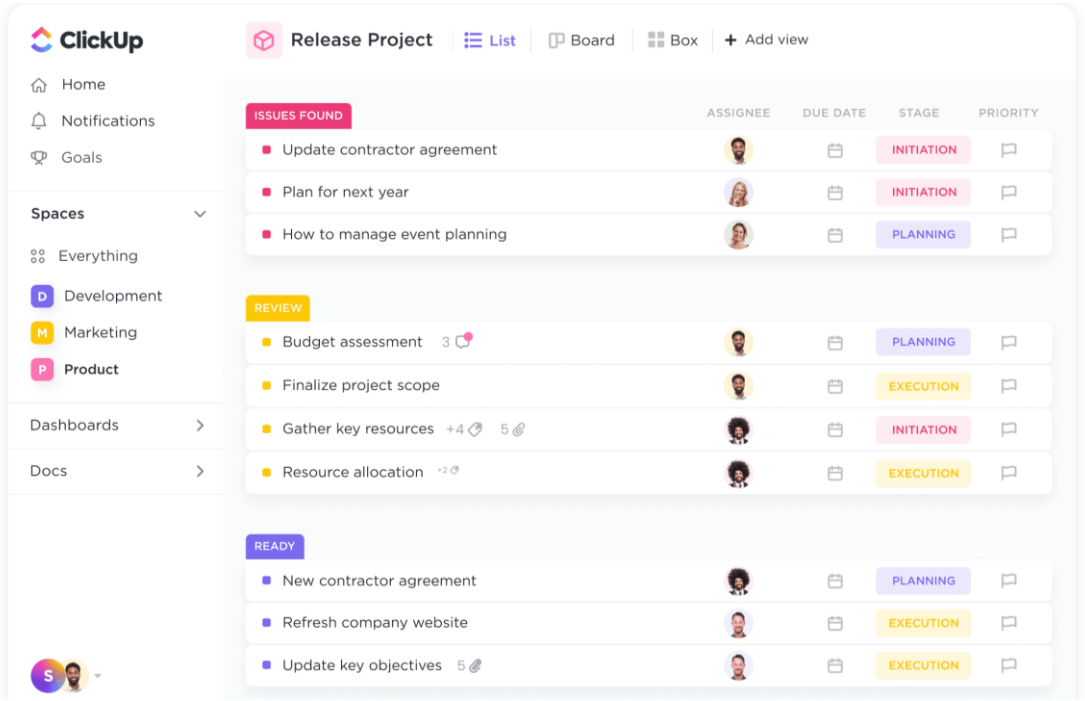


Рисунок 1.4 – Інтерфейс застосунку ClickUp

Ще одним рішенням є Monday.com (рис. 1.5) – платформа для організації командної роботи, яка підтримує адаптацію до різних сфер діяльності. Вона дозволяє створювати робочі простори з можливістю налаштування полів, статусів, дедлайнів та автоматизації робочих процесів. Платформа підтримує різні візуальні формати, а саме таблиці, дошки, календарі та діаграми.

Серед ключових переваг Monday.com є інтеграція з такими інструментами, як Google Workspace, Microsoft 365, Slack, Zoom, GitHub, що дозволяє централізувати управління завданнями [10].

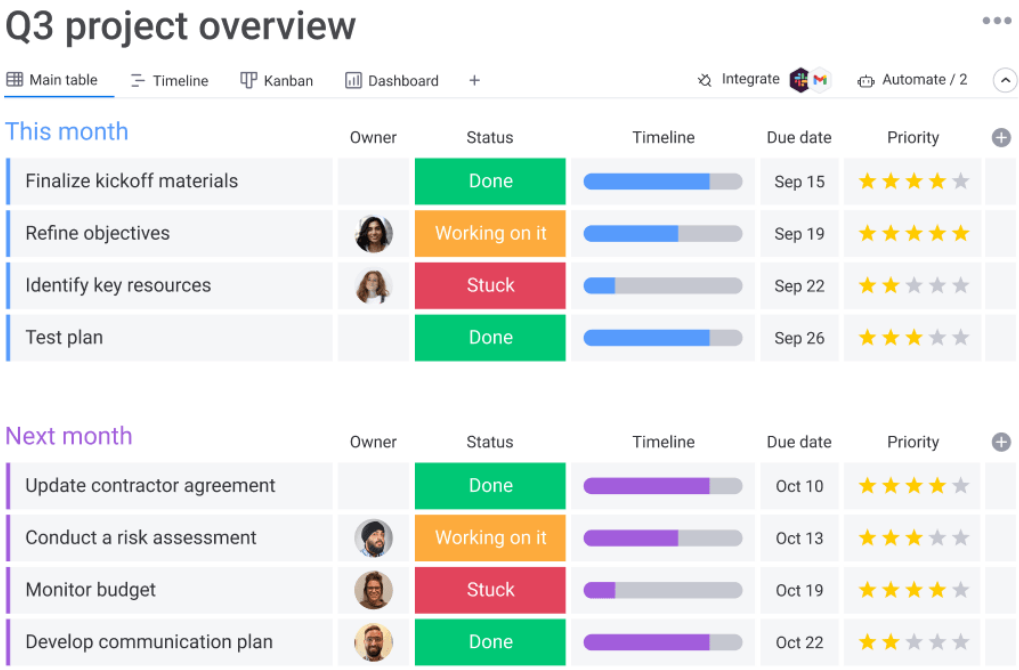


Рисунок 1.5 – Інтерфейс застосунку Monday.com

Результати порівняння аналогів зведено в таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Trello	Asana	Jira	ClickUp	Moday.com	Власна розробка
Візуалізація завдань	+	+	-	+	+	+
Гнучкість налаштувань	-	+	+	+	-	+
Підтримка Kanban	+	+	+	+	+	+
Підтримка календаря для роботи із задачами	-	-	-	-	-	+
Інтеграція з іншими сервісами	+	+	+	+	+	+
Простота використання	+	-	-	-	+	+
Загальна оцінка	66,6%	66,6%	50%	66,6%	66,6%	100%

Згідно з таблицею порівняльних характеристик, існуючі інструменти в основному відповідають загальним потребам команд у керуванні завданнями, але вони мають свої обмеження в гнучкості та можуть бути складними для освоєння. Це особливо стосується великих систем, таких як Jira.

Розробка власного програмного модуля дає можливість врахувати специфіку внутрішніх процесів, створити простий і зрозумілий інтерфейс, налаштувати логіку роботи під конкретні ролі в команді. Це забезпечить максимальну ефективність і зручність у командній роботі над проєктами. Отже, розробка вебсистеми для оптимізації процесів управління проєктами є актуальним, що дозволяє підвищити ефективність командної роботи, автоматизувати ключові процеси управління проєктами та забезпечити прозорість виконання завдань. Аналіз існуючих рішень засвідчив, що хоча такі системи, як Trello, Asana, Jira, ClickUp та Monday.com, пропонують широкий функціонал, вони не завжди повністю відповідають специфічним вимогам окремих команд.

1.3 Обґрунтування методів розв’язання задач

Проєктування та реалізація вебсистеми для оптимізації командної роботи над проєктами вимагає сучасних підходів, які об'єднують ефективні інструменти для створення інтерфейсу, управління даними та забезпечення взаємодії між користувачами.

Один із способів реалізації системи полягає у використанні готових інструментів, таких як Appwrite, для створення бекенд-частини. Цей підхід дозволяє швидко налаштувати автентифікацію користувачів, зберігання даних та управління доступом, що суттєво скорочує час розробки. Проте, використання готових рішень може обмежити гнучкість системи та ускладнити її масштабування в майбутньому.

Інший варіант – це розробка власного бекенду, що дає змогу повністю контролювати архітектуру системи та її компоненти. Цей метод забезпечує високу гнучкість і можливість масштабування, але вимагає значних ресурсів для розробки та підтримки, а також може призвести до збільшення часу на реалізацію проєкту.

Для створення інтерфейсу користувача розглядається використання сучасних фреймворків, таких як Next.js, який базується на React. Цей фреймворк поєднує переваги клієнтського та серверного рендерингу, що дозволяє зменшити час першого завантаження сторінки та покращити SEO-оптимізацію. Завдяки вбудованій підтримці динамічної маршрутизації, автоматичному розділенню коду та можливостям попереднього рендерингу, Next.js забезпечує стабільну, масштабовану та швидкодіючу роботу вебінтерфейсу [11].

Для стилізації інтерфейсу можна розглянути використання утилітної CSS-бібліотеки, такої як Tailwind CSS. Ця бібліотека дозволяє швидко та ефективно оформлювати елементи інтерфейсу без необхідності створення окремих CSS-файлів. Tailwind забезпечує зручну підтримку та адаптацію вигляду інтерфейсу для різних розмірів екранів, що сприяє створенню адаптивного та візуально привабливого дизайну.

Не менш важливим є питання автентифікації користувачів. Один із способів – це використання OAuth-протоколу для авторизації через сторонні сервіси, такі як Google або GitHub. Це дозволяє користувачам входити в систему, використовуючи вже наявні акаунти, що спрощує процес реєстрації та підвищує безпеку системи.

Для організації роботи команди можна застосувати методологію, що базується на принципах Kanban. Це гнучкий підхід, який дозволяє візуально відстежувати статус завдань, серед переваг застосування канбану в програмній інженерії найчастіше згадуються підвищена видимість роботи**,** контроль процесів**,** покращення потоку завдань та скорочення часу виходу програмного продукту на ринок. Візуалізація статусів, обмеження WIP і моніторинг потоків сприяють підвищенню продуктивності команд [12].

Після аналізу переваг та недоліків кожного з підходів було вирішено обрати стратегію, яка поєднує використання готових інструментів для швидкої розробки з можливістю подальшого масштабування та адаптації системи до змінюваних вимог. Це дозволить забезпечити ефективну та гнучку роботу команди, зменшити час на розробку та підвищити якість кінцевого продукту.

1.4 Постановка задач розробки

Після аналізу сучасних інструментів для управління проєктами та вивчення потреб IT-команд, виявилено кілька недоліків у наявних рішеннях: занадто складні налаштування, перевантажений інтферфес та складність роботи з ним, відсутність адаптивного інтерфейсу для мобільних пристроїв та потребу у додаткових інстументах для роботи із задачами як представлення їх у календарі. В результаті, ми сформулювали основні задачі, які потрібно реалізувати для створення вебсистеми.

Зокрема, у рамках проєкту необхідно виконати такі завдання:

- розробити архітектуру вебсистеми, яка включатиме клієнтську та серверну частини, забезпечуючи модульність і можливість подальшого розвитку функціоналу;
- реалізувати авторизацію та автентифікацію користувачів із використанням сучасних методів захисту даних, зокрема інтеграцію з сервісами OAuth (Google, GitHub) через платформу Appwrite;
- створити програмні модулі для реєстрації та авторизації, керування робочими просторами, проєктами, завданнями, з можливістю призначення відповідальних осіб, встановлення дедлайнів, зміни статусів, контролю прогресу та управління членами комнади;
- реалізувати графічний інтерфейс користувача, орієнтований на простоту використання, адаптивність та швидкість взаємодії;
- впровадити механізми фільтрації та сортування завдань за різними критеріями (дата, статус, відповідальний, пріоритет);
- забезпечити зберігання даних про користувачів, проєкти та задачі із урахуванням політик доступу та розмежування ролей;
- реалізувати можливість колективної роботи над проєктами: запрошення учасників до команд та редагування задач у режимі реального часу;
- провести тестування системи, що охоплює функціональне тестування окремих модулів, перевірку роботи авторизації, інтерфейсу та збереження даних.

Таким чином, реалізація вищезазначених задач допоможе створити сучасний вебінструмент, який буде зосереджений на командній взаємодії в рамках проєктної діяльності. Цей інструмент буде гнучким, розширюваним і зручним у використанні для команд різного розміру.

1.5 Висновки

У першому розділі було проаналізовано сучасні вебсистеми для керування командною роботою над проєктами. Було визначено, що, незважаючи на їхню популярність і широкий функціонал, ці сервіси мають свої недоліки. Серед них – перевантаженість інтерфейсу, складність налаштування, обмежені можливості кастомізації під специфічні потреби команд, а також залежність від платних функцій.

На основі цього аналізу переваг і недоліків існуючих систем було обґрунтовано доцільність розробки власної вебсистеми. Вона буде спрямована на оптимізацію командної роботи над проєктами, враховуючи вимоги до простоти використання, адаптивності, функціональності та можливості масштабування. Було визначено основні задачі, які потрібно реалізувати в рамках проєкту: розробка архітектури системи, створення модулів для реєстрації та авторизації, управління проєктами, задачами та управління членами комнади реалізація зручного інтерфейсу, забезпечення авторизації та безпеки, а також проведення тестування функціональності.

2 РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз завдання та формалізація вхідних даних

Для реалізації вебзастосунку з метою оптимізації проєктної діяльності IT-команд, будуть використані сучасні вебтехнології, в тому числі React Query та Hono для взаємодії між клієнтом і сервером, серверну платформу Appwrite для збереження даних і автентифікації, бібліотеку компонентів TailwindCSS і фреймворк Next.js. Застосунок реалізовано зі використанням мови програмування TypeScript. Це дозволило додати типізацію та покращувати надійність коду.

Однією з важливих складових частин вебзастосунку є аналіз вхідних даних, що формуються автоматично внаслідок внутрішніх процесів, надходять від користувачів чи зовнішніх сервісів. Ці дані визначають логіку взаємодії користувачів, поведінку системи, а ще збереження результатів у базі даних й обробку запитів.

Умовний поділ вхідних даних можливо здійснити на декілька категорій. До них відносяться дані від користувача, системні вхідні дані й дані вже із зовнішніх джерел.

Дані, котрі вводяться користувачами напряму через інтерфейс системи, – це вхідні дані від користувача. Модулі керування проєктами, завданнями, робочими просторами та автентифікації розроблено для зручної взаємодії з системою.

Дані для автентифікації (через стандартну реєстрацію або OAuth Google та GitHub):

- email;
- пароль;
- ім’я користувача;
- токени OAuth.

Дані для створення та редагування робочих прсоторів:

- назва;
- учасники (ID користувачів);

- проєкти, які належать до прсотору.

Дані для створення та редагування завдань:

- назва;
- опис;
- статус (Backlog, To Do, In Progress, In Review, Done);
- виконавець (учасник команди);
- термін виконання;
- проєкт.

Ці дані вводяться через спеціалізовані UI-компоненти, такі як projects.tsx, analytics.tsx, modal-provider.tsx та date-picker.tsx, і далі обробляються через відповідні хуки (use-confirm.tsx, use-debounce.ts) та API-функції у файлах api/[[...route]] і features/.

Системні вхідні дані – це дані, що передаються в систему автоматично або через URL-параметри, middleware або API-запити. До таких даних входить:

- сесійні токени (через middleware) забезпечують збереження авторизації користувача;
- URL-параметри, які дозволяють передавати ідентифікатори проєктів, завдань, сторінок (наприклад: /project/[id]);
- API-запити до Appwrite, які виконуються з метою отримання, оновлення, видалення чи створення записів у базі даних;
- дані з OAuth (наприклад, email користувача з GitHub) обробляються через oauth.ts для реєстрації або входу.

Система підтримує взаємодію із зовнішніми сервісами. А саме OAuth-провайдери (GitHub та Google). Обробка таких даних реалізована через компонент query-provider.tsx, які взаємодіють із системою аналітики через API.

Отже, аналіз вхідних даних системи показує, що вебзастосунок працює з різними типами даних, такими як токени, об’єкти, дати, текстові поля та параметри URL. Основна обробка даних зосереджена у функціональних модулях системи (auth, members, projects, tasks, workspaces), які відповідають за управління відповідними об’єктами.

2.2 Розробка архітектури програмного продукту

Архітектура програмного продукту формує логічну структуру системи, визначає, як її компоненти взаємодіють, а також забезпечує масштабованість і зручність у підтримці. У рамках проєкту була реалізована повноцінна full-stack архітектура, яка об'єднує серверну та клієнтську частини, хмарну базу даних, авторизацію, асинхронну взаємодію та модульну структуру. Такий підхід гарантує розширюваність, повторне використання коду та ефективну взаємодію користувачів у спільному середовищі.

Застосунок поділено на фронтенд (Next.js 14) і бекенд (Appwrite). Взаємодія між клієнтом і сервером відбувається через HTTP-запити з використанням API. Структура коду побудована на принципах поділу відповідальності, з окремими модулями для логіки автентифікації, управління робочими просторами, тасками, ролями тощо.

Хмарна база даних і система авторизації для зберігання даних та обробки запитів реалізовані на платформі Appwrite. Вона дозволяє створювати, читати, оновлювати та видаляти записи, а також підтримує автентифікацію через OAuth. Тоюто можна увійти за допомогою електронної пошти, Google або GitHub. Всі вхідні дані перевіряються, зберігається токен сесії, а доступ до сторінок обмежується в залежності від ролі користувача.

Для управління станом використовується React Query, що забезпечує асинхронне отримання, кешування та оновлення даних. Також реалізована валідація: введення користувача перевіряється за допомогою бібліотеки Zod, що гарантує достовірність даних.

Інтерфейс реалізовано за допомогою Next.js 14 із підтримкою App Router і серверних компонентів (Server Components). Для стилізації використано Tailwind CSS – utility-first фреймворк, який дозволяє швидко створювати адаптивний дизайн без потреби у написанні великої кількості власних CSS-стилів. Tailwind забезпечує високу гнучкість оформлення компонентів і підтримку адаптивності для різних типів пристроїв [13].

Доступна мультирольова підтримка, де користувачі можуть мати різні ролі, такі як учасник або адмін, що визначає їхні можливості в межах робочого простору. Інтерактивна взаємодія через React Query дозволяє оновлювати інтерфейс без перезавантаження сторінки, що забезпечує зручну роботу з тасками, календарем, дошками Kanban та іншими елементами.

Календар і дошка завдань для візуального керування завданнями (drag-and-drop Kanban), планування дедлайнів через календар та фільтрацію/пошук.

Взаємодія компонентів відбувається наступним чином. Спочатку відбувається запит від клієнта – користувач виконує дію у браузері, наприклад, додає задачу. Потім на клієнтській стороні форма перевіряється за допомогою бібліотеки Zod, і дані відправляються через запит React Query. Далі відправляється POST, PUT або GET-запит до Appwrite API, наприклад, для створення документа в колекції tasks. На сервері Appwrite перевіряється авторизація, після чого дані зберігаються або повертаються. На завершальному етапі клієнт отримує відповідь і, використовуючи React Query, оновлює інтерфейс без необхідності перезавантажувати сторінку.

Переваги архітектури:

- Next.js 14 та Appwrite дозволяє швидко розробляти з мінімальними налаштуваннями;
- інтерактивність системи забезпечує миттєве відображення змін для користувачів завдяки React Query та серверним компонентам;
- OAuth дозволяє швидко входити без необхідності створення нових облікових записів.

Архітектура програмного продукту реалізована з урахуванням сучасних підходів до створення вебзастосунків, що забезпечує масштабованість, безпеку та зручність для користувачів. Завдяки використанню стеку Next.js 14, Appwrite, Tailwind CSS, React Query та інших технологій, вдалося створити гнучку модульну систему, яка підтримує автентифікацію та функціонал в реальному часі.

2.3 Розробка загального алгоритму роботи системи та діаграм станів

Загальний алгоритм роботи застосунку (рис. 2.1) слугує базовою основою для формування логіки програми. Він відображає взаємозв’язок між модулями всієї системи та загальну її поведінку.

Алгоритм взаємодії користувача з вебзастосунком починається з запуску системи. Після цього перевіряється, чи користувач уже автентифікований. Якщо ні, його перенаправляють на сторінку авторизації, де він може або увійти, якщо вже має обліковий запис, або перейти до сторінки реєстрації для створення нового акаунта. У процесі реєстрації користувач вводить необхідні дані, які система перевіряє на валідність.

Після успішної автентифікації користувача система перевіряє, чи доданий він до робочого простору. Якщо ні, користувачу пропонується створити новий робочий простір. У разі наявності доступу до робочого простору система перенаправляє користувача на головну сторінку застосунку, де він може взаємодіяти з основним меню.

У меню користувач може обрати кілька функціональних пунктів. Якщо обрано пункт «Home», система перевіряє, чи є задачі, пов’язані з проєктами робочого простору. Якщо задачі наявні, відображається відповідна інформація. Якщо ні, виводиться порожній стан екрану. Якщо обрано пункт «My Tasks», система перевіряє, чи є задачі, призначені безпосередньо цьому користувачу. У разі наявності таких задач вони виводяться на екран, у протилежному випадку показується порожній стан.

При виборі пункту меню «Settings» користувач може переглянути доступні налаштування робочого простору. Обравши пункт «Members», користувач переглядає список доданих учасників команди. У розділі «Профіль» доступна кнопка «Sign Out», яка завершує роботу користувача в системі.

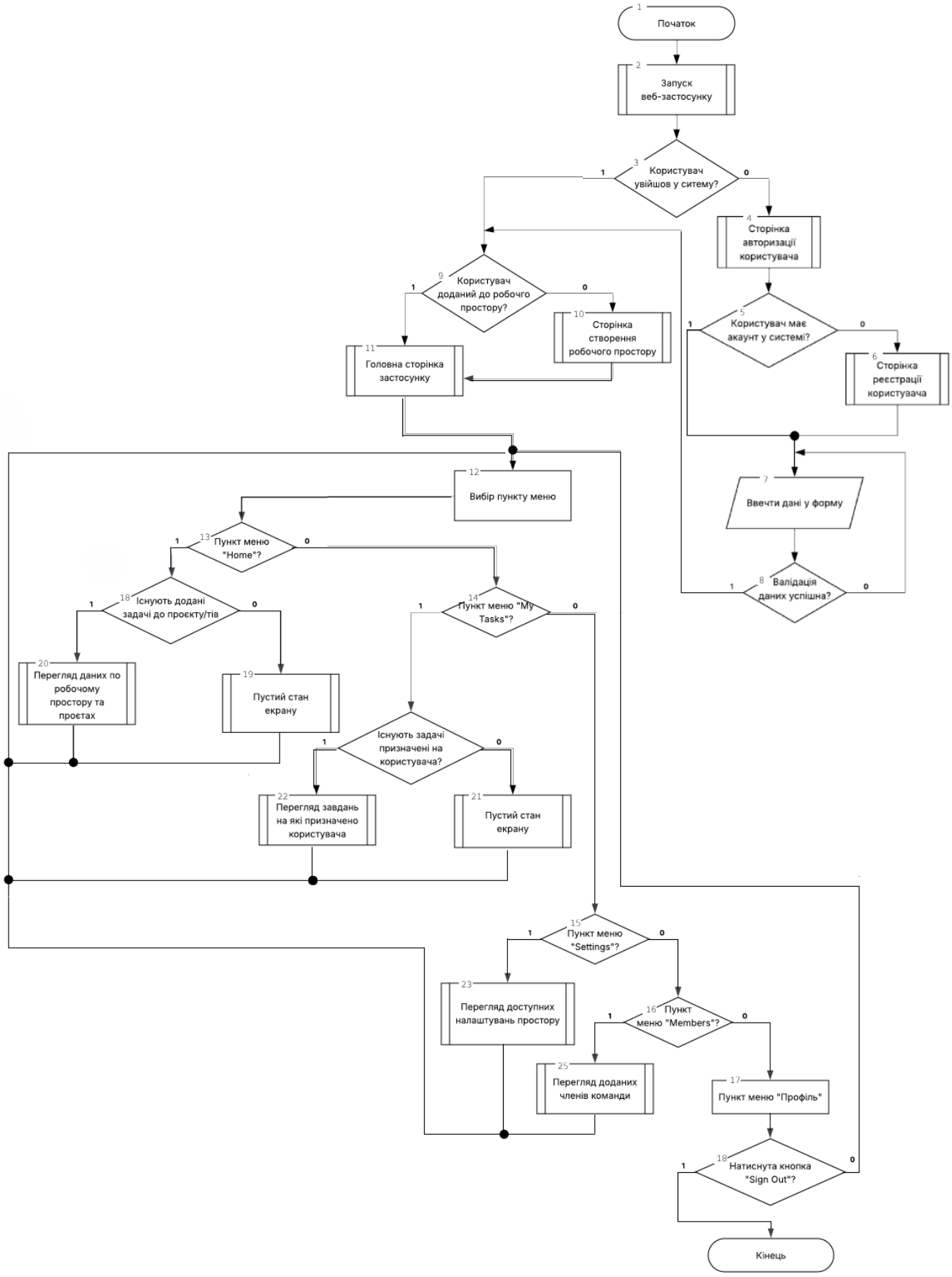


Рисунок 2.1 – Загальний алгоритм роботи вебсистеми

Діаграми станів [14] – це графічне представлення всіх можливих станів об'єкта та переходів між цими станами, спричинених подіями.

Основні елементи діаграми станів:

- стан, що відображає певний момент у життєвому циклі об'єкта, під час якого він виконує дії або чекає події;
- перехід, що є стрілкою, яка показує зміну стану в результаті події;
- подія, що є зовнішнім або внутрішнім тригером, що викликає перехід;
- дія як процес, який виконується під час переходу;
- початковий стан, а саме чорна точка, з якої починається життєвий цикл;
- кінцевий стан, а саме подвійне коло, яке позначає завершення життєвого циклу.

Діаграму станів модуля авторизації та реєстрації користувача наведено на рисунку 2.2.

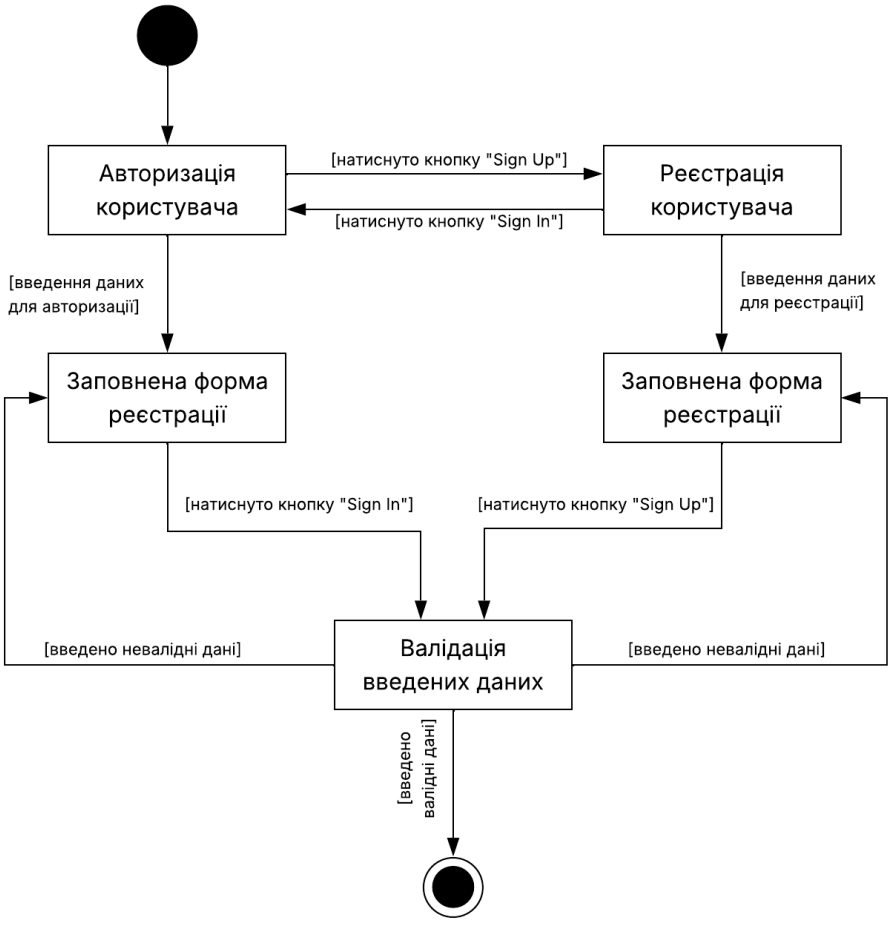


Рисунок 2.2 – Діаграма станів для модуля авторизації та реєстрації користувача

Діаграму станів модуля керування робочими просторами наведено на рисунку 2.3.

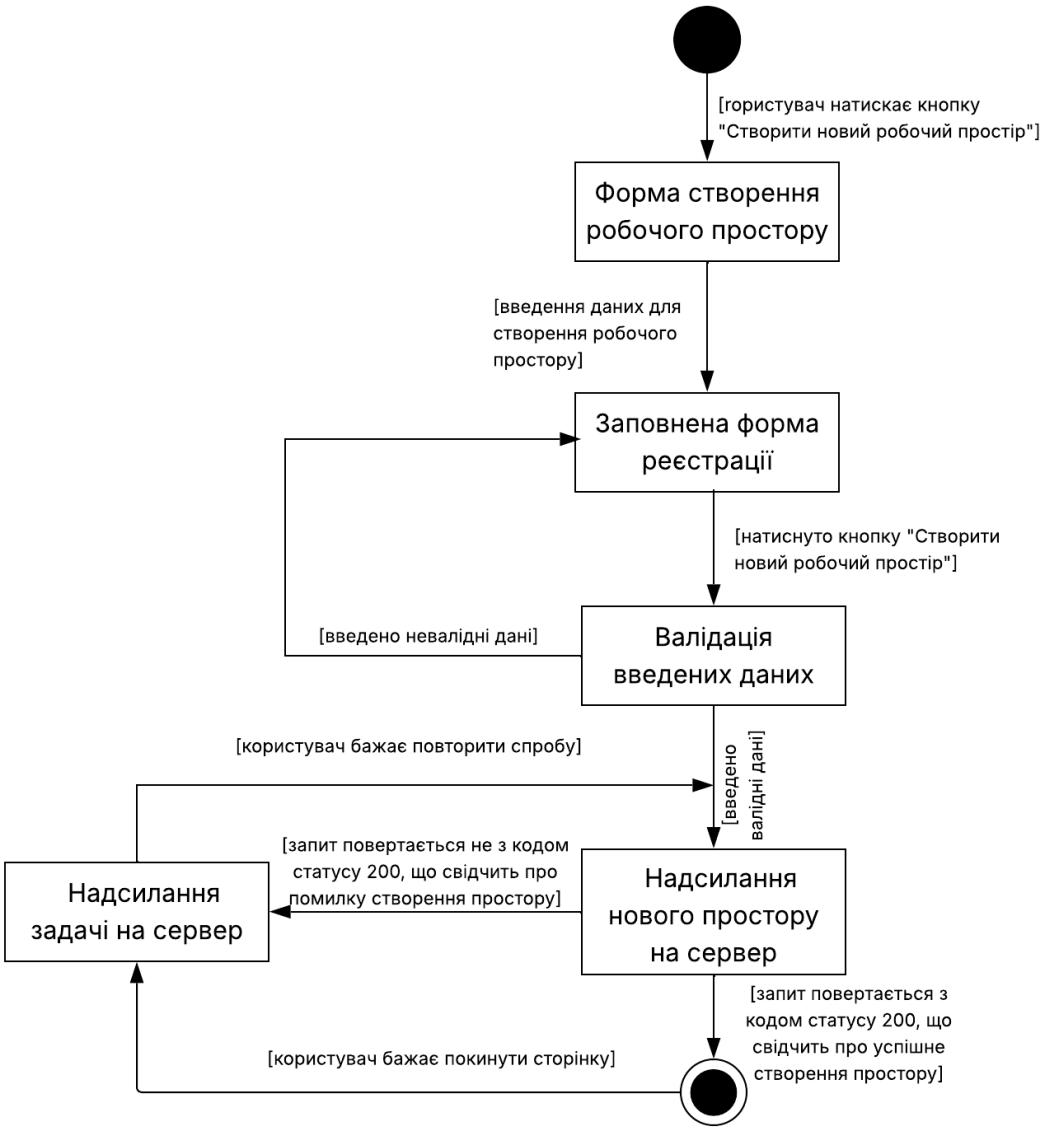


Рисунок 2.3 – Діаграма станів модуля створення робочого простору

Для того аби створити задачу користувач має обрати робочий простір та проект. Далі після натиснення кнопки для створення задачі відкривається відповідна форма. Наступним кроком після її заповнення, є перевірка заповнення обов’язкових полів. Якщо відсутній хоча б один критично важливий елемент, система повідомить про помилку, і користувач зможе повернутися до введення даних та виправити недоліки.Також, якщо не було обрано виконавця і дедлайн, то

буде названечено автоматично “Не визначено” та “Без дедлайну” відповідно. Діаграму станів модуля керування задачами наведено на рисунку 2.4.

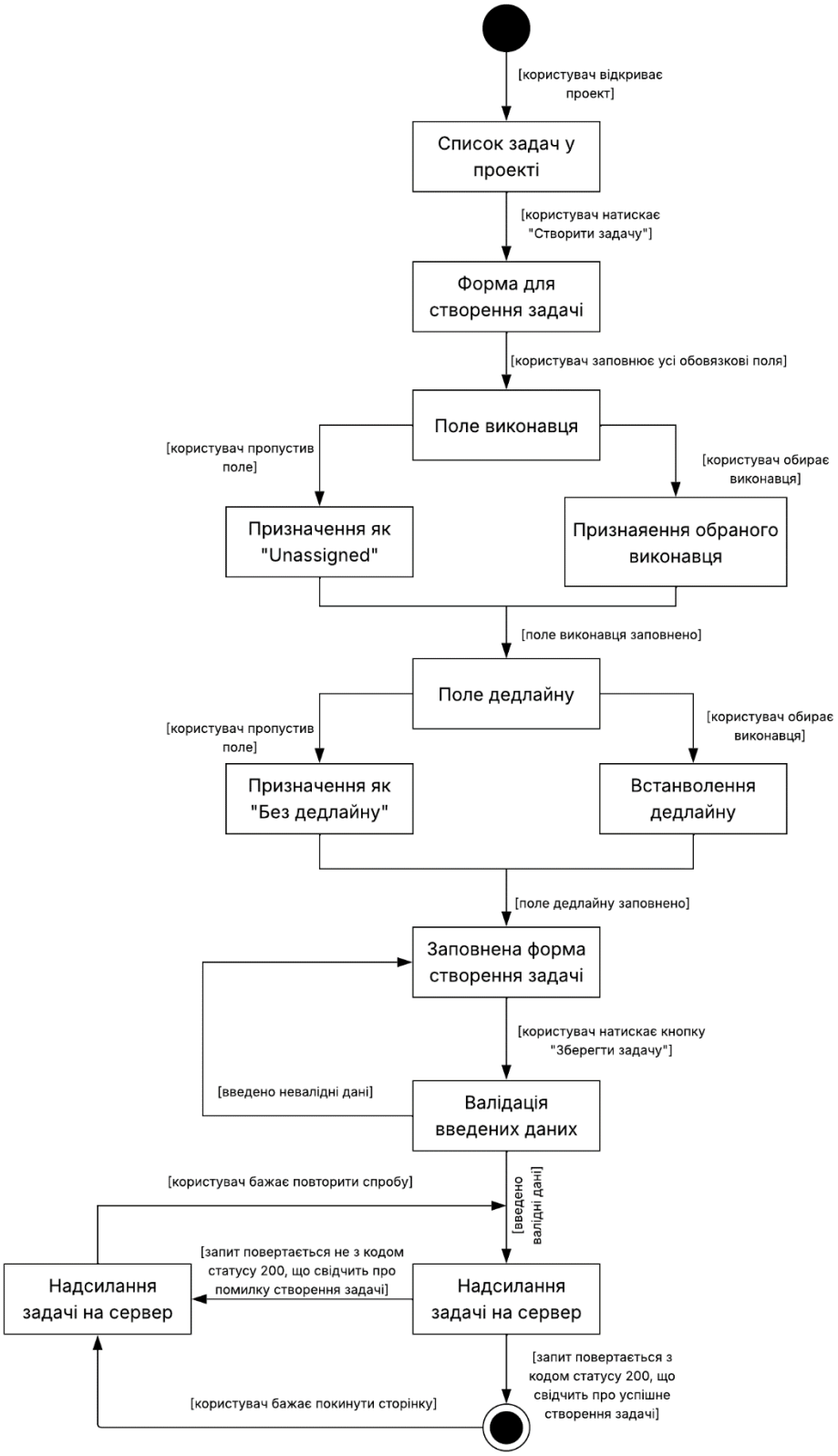


Рисунок 2.4 – Діаграма станів модуля додавання задач

Діаграму станів модуля дешборду користувача наведено на рисунку 2.5.

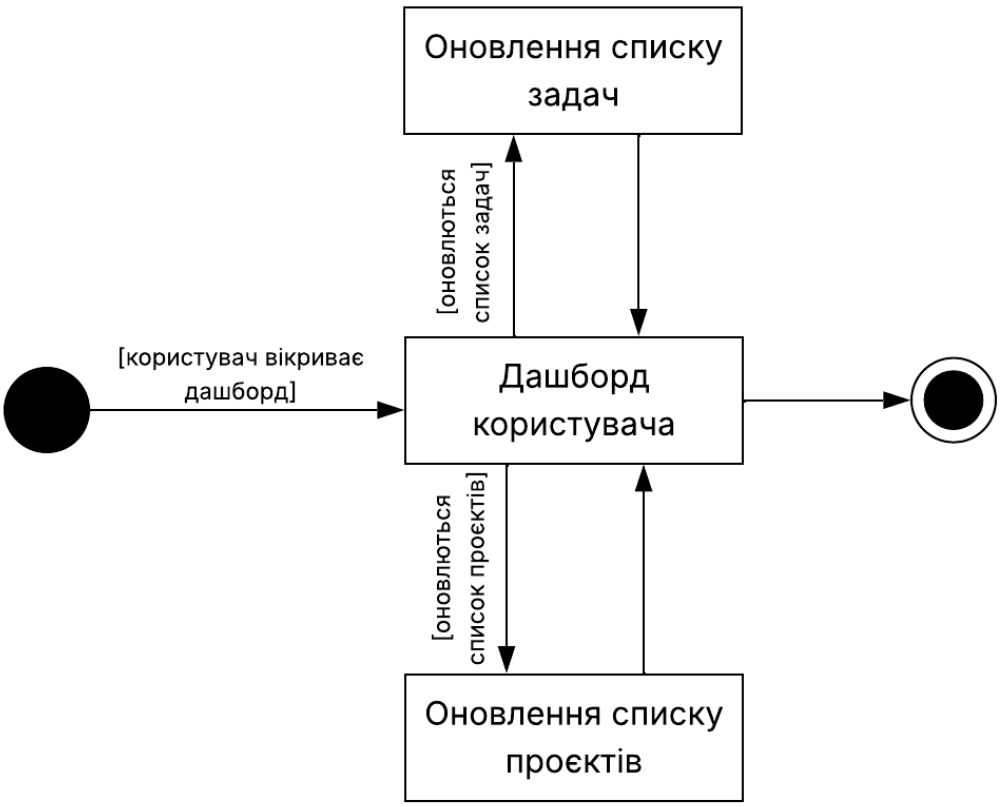


Рисунок 2.5 – Діаграма станів модуля дешборду користувача

2.4 Розробка інтерфейсу користувача

Інтерфейс користувача (UI) вебзастосунку розроблено з урахуванням принципів зручності, інтуїтивності та ефективності взаємодії. Основна мета – надати користувачам змогу швидко орієнтуватися у функціональності, створювати та керувати робочими просторами, задачами, командами та календарями в межах єдиного інтерфейсу.

Інтерфейс реалізовано за допомогою фреймворку Next.js 14**,** з використанням Tailwind CSS для адаптивного дизайну та shadcn/ui як бібліотеки UI-компонентів.

Інтерфейс побудовано за принципом SPA (Single Page Application), де основні модулі завантажуються динамічно через маршрутизацію, а зміна стану відбувається без перезавантаження сторінки [15].

Розробка інтерфейсу почалась з сторінки автентифікації та реєстрації такі як Login та Register (рис. 2.6). Login має поля введення імені, емейлу і пароля з

валідацією, кнопка входу. У разі помилки – toast.error(), у разі успіху – toast.success(). Також наявна інтеграція з Google та GitHub, що можна побачити унизу форми.

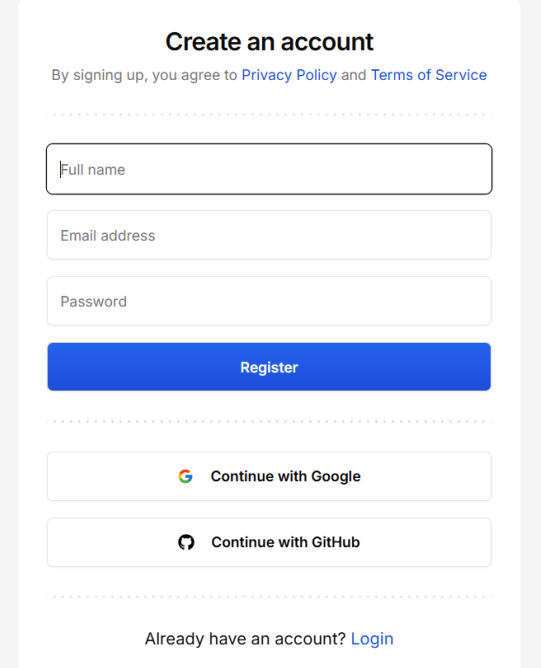


Рисунок 2.6 – Форма реєстрації

Після реєстації користувач має створити робочий простір (рис. 2.7), а саме заповнивши просту форму з наступними полями та кнопками:

1. Текстове поле для назви робочого простору.
2. Поле для додання фото.
3. Кнопка відміни дії.
4. Кнопка підтвердження дії.

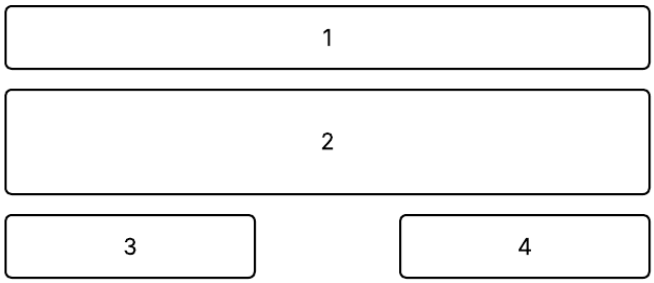


Рисунок 2.7 – Схема форми для створення нового робочого простору

Якщо користувач вже має створений робочий простір, то інтерфейс має мати бічну панель для навігації по основних сторінках (рис. 2.8) на десктоп пристороях або бургер-меню на мобільних пристороях. Містить основні розділи такі як Home, My Tasks, Settings, Members і Projects.

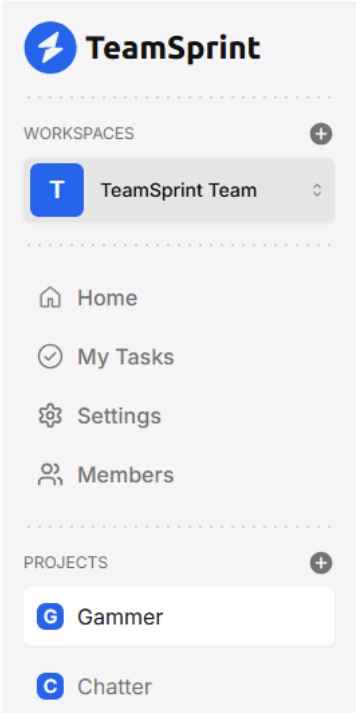


Рисунок 2.8 – Бічна панель

У розділі Workspaces користувач бачить перелік власних робочих просторів, які відображаються у вигляді випадаючого меню із назвою і фото. Передбачено можливість створення нового робочого простору на окремій сторінці, що відкривається після натискання кнопки «Create workspace» (рис. 2.8).

Основна контентна область динамічно відображає компоненти залежно від вибраного розділу для основних стоірнок. На рисунку 2.9 зображено схему сторінку «Home».

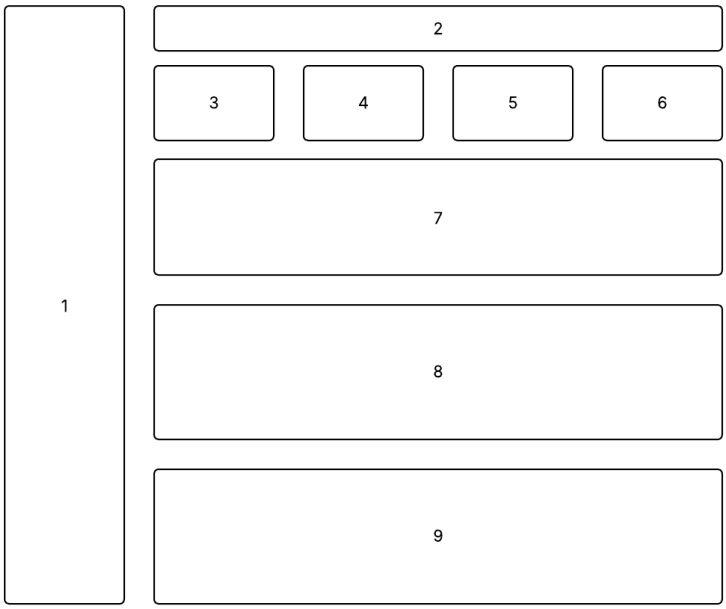


Рисунок 2.9 – Схема інтерфейсу сторінки «Home»

На схемі можна побачитинаступне:

1. Бічну панель.
2. Хедер, де можна побачити назву сторінки на якій знаходимось та іконку свого акаунта.
3. Блок з кількістю задач в робочому просторі.
4. Блок з кількістю задач, які призначені на користувача який володіє акаунтом, з якого переглядається.
5. Блок з кількістю завершених задач.
6. Блок з кількістю задач термін виконання яких прострокований.
7. Блок для перегляду задач.
8. Проекти, які додані до робочого простору з можливістю створювати нові натиснувши кнопку «+». Форма для створення виглядатиме так само як форма для створення робочого простору (рис. 2.7).
9. Список членів робочого простору.

Пункти 1-7 повторюються для сторінки проєкту та для сторінки «My Tasks» пункти 1-2 повторюється.

Якщо буде обрано конкретний проект з списку створених проектів, користувач буде перенаправлений на сторінку проєкту (рис. 2.10). Реалізовано

кілька варіантів візуалізації задач, а саме списком, Kanban-дошка та календар. Загалом стуктура сторінки повторює перші пункти зазначені вище при описі домашньої сторінки. Нижче представлені 3 таби, а саме «Table», «Kanban» та «Calendar». Кожна таба змінює наповнення блоку.

Під табами є фільтри, які дозволяють фільтрувати за стутусом, викоанвцем та дедлайном. Фільтри залишаться незмінними для всіх табів.

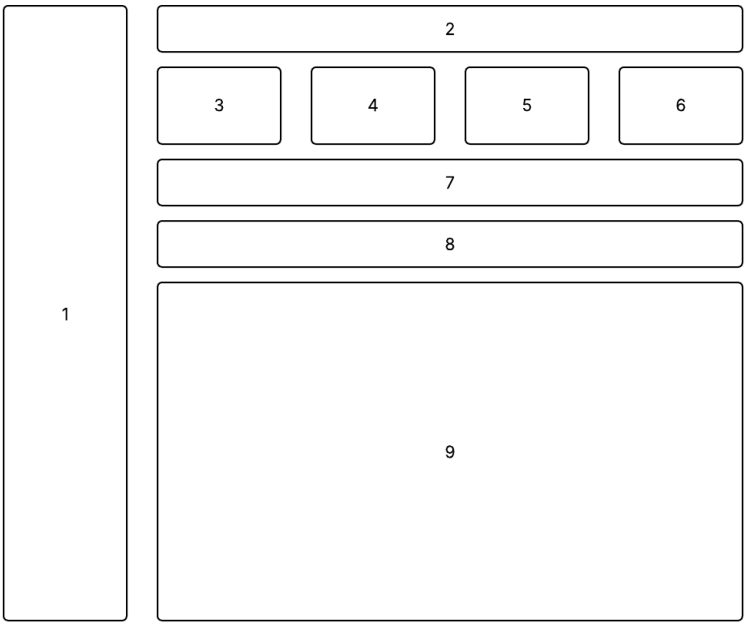


Рисунок 2.10 – Схема інтерфейсу сторінки обраного проєкту

Kanban-дошка поділяється на секції Backlog, To Do, In Progress, In Review та Done. Завдання відображаються як картки з назвою, дедлайном, та виконавцями. При натисканні на картку відкривається модальне вікно редагування задачі, у якому можна змінити її статус, опис, виконавця чи дедлайн.

Крім того, задачі можуть бути відображені у календарному вигляді за допомогою бібліотеки @fullcalendar/react. Подвійне натискання на задачу відкриває модальне вікно редагування, а зміна її статусу можлива шляхом перетягування події по календарю.

Ще один варіант представлення задач – у вигляді таблиці, де кожен рядок містить ключові параметри задачі:

- назву;
- відповідального;
- дедлайн;
- поточний статус.

Такий вигляд особливо зручний для швидкого перегляду великої кількості задач та фільтрації за певними критеріями.

Сторінка налаштувань має кілька блоків, які віповідають за різні аспекти робочого простору (рис. 2.11). На сторінці представлені наступні елементи:

1. Кнопка для повренення до домашньої сторінки та назва робочго простору.
2. Текстове поле для редагуввання назви та поле для завантаження фото. Наявна копка для збереження змін.
3. Посилання для запрошення членів команди. Можна скопіювати посилання
4. Зона для видалення робочого простору, попереджує про наслідки та має кнопку видалення.

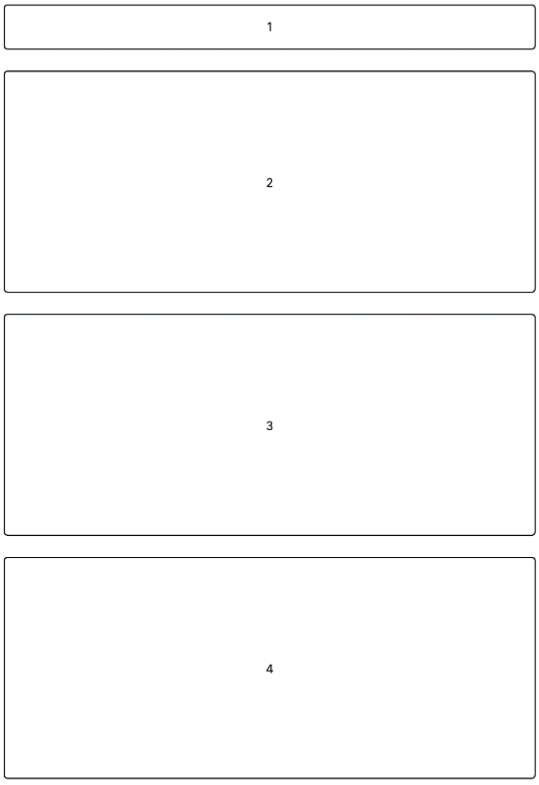


Рисунок 2.11 – Схема інтерфейсу сторінки налаштувань

Таблиця учасників робочого простору містить інформацію про ім’я користувача, електронну пошту та роль (рис. 2.12). Користувач із правами адміністратора може змінювати ролі учасників або видаляти їх із простору. На сторінці представлені наступні елементи:

1. Кнопка для повренення до домашньої сторінки та назва сторінки, на якій знаходиться користувач.
2. Список користувачів, які додані до робочого простору.

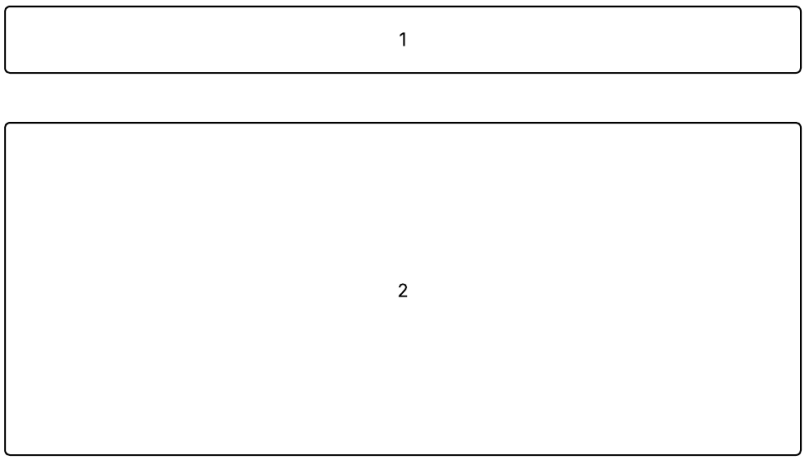


Рисунок 2.12 – Схема інтерфейсу сторінки зі списком членів робочого простору

Особливості UX:

- іконки з бібліотеки lucide-react, що забезпечує чистий мінімалістичний стиль;
- анімації відкриття/закриття модальних вікон, переміщення задач – через Framer Motion;
- адаптивність до різних розмірів екранів реалізована з Tailwind (grid, flex, sm: md: lg: брейки);
- toast-повідомлення для зворотного зв'язку, а саме успішні дії, помилки, попередження.

Таким чином, інтерфейс користувача вебзастосунку спроєктовано як інтуїтивно зрозумілий, візуально привабливий та технологічно сучасний

інструмент для спільної роботи над проєктами. Його реалізація враховує потреби як невеликих фріланс-команд, так і великих IT-компаній, які прагнуть підвищити ефективність внутрішніх процесів.

2.5 Висновки

У другому розділі було проаналізовано вхідні та вихідні дані вебсистеми, що призначена для оптимізації командної роботи над проєктами. Розглянуто основні компоненти інтерфейсу користувача та описано логіку взаємодії з ключовими модулями системи. Було побудовано загальний алгоритм роботи вебзастосунку, який охоплює процеси автентифікації, створення проєктів, призначення ролей та управління завданнями. Також розроблено діаграми станів для модулів авторизації та реєстрації користувача, створення робочого простору, додавання задач та дешборду користувача.

3. РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Варіантний аналіз засобів реалізації і обґрунтування вибору засобів для реалізації програмного забезпечення

Next.js 14 – сучасний фреймворк на базі React, який поєднує серверний і клієнтський рендеринг, що покращує продуктивність і SEO-оптимізацію. Завдяки вбудованій підтримці маршрутизації, SSR (server-side rendering) і гнучкій архітектурі компонентів, цей фреймворк чудово підходить для складних вебзастосунків із багатим інтерфейсом [11].

Альтернативами Next.js є Create React App (CRA) та Vue.js. CRA забезпечує простіший старт, але не підтримує SSR без додаткових налаштувань, що знижує SEO-можливості. Vue.js має нижчий поріг входу та простіший синтаксис, проте екосистема React та Next.js значно більша й активніша, що важливо для довготривалої підтримки.

Порівняння фреймворків для фронтенду представленно в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика фронтенд фреймворків.

Критерій	Next.js	React	Vue.js
Продуктивність	+	+/-	+
Наявність API-роутів	+	-	-
Висока підтримка SEO	+	-	+
Використання у великих проєктах	+	+	+
Висновок	100%	37.5%	75%

Для стилізації було застосовано Tailwind CSS, утилітну CSS-бібліотеку, яка дозволяє швидко створювати адаптивні та сучасні інтерфейси без потреби писати власні стилі з нуля. Завдяки класам, що відповідають окремим стилям, розробка інтерфейсу стає швидкою та гнучкою, а також легко підтримується. У порівнянні з Bootstrap, Tailwind є менш нав’язливим і дозволяє створювати унікальний дизайн

без перевизначення стилів. У той час як Bootstrap надає готові компоненти, Tailwind – це радше набір інструментів, що забезпечує повну гнучкість.

Порівняння інструментів стилізації інтерфейсу представленно в таблиці 3.2.

Таблиця 3.2 – Порівняльна характеристика інструментів стилізації інтерфейсу.

Критерій	Tailwind CSS	Bootstrap
Гнучкість дизайну	+	-
Малий розмір фінального CSS	+	-
Продуктивність	+	+/-
Легкість інтеграції з Next.js	+	-
Висновок	100%	17.5%

Як бекенд-рішення було обрано Appwrite – open-source платформа, яка надає повний стек для розробки застосунків, включаючи автентифікацію (через email, Google, GitHub тощо), базу даних та хмарне зберігання файлів та реального часу API [16]. Appwrite дозволяє зменшити кількість ручної реалізації бекенд-логіки та зосередитись на бізнес-логіці проєкту. Аналогічними рішеннями є Firebase та Supabase. Firebase є лідером у сфері BaaS, але має закритий код і платну модель. Supabase також є open-source, проте більше орієнтований на PostgreSQL. Appwrite надає більш гнучкий контроль над інфраструктурою, що було вирішальним фактором у виборі.

Порівняння BaaS-платформ представленно в таблиці 3.3.

Таблиця 3.3 – Порівняльна характеристика BaaS-платформ.

Критерій	Appwrite	Firebase	Supabase
Відкритий код	+	-	+
Розширюваність	+	-	+

Продовження таблиці 3.3

Критерій	Appwrite	Firebase	Supabase
Варіативність методів автентифікація	+	+	+
Власний UI-консоль серверу	+	-	+/-
Модульність компонентів	+	-	-
Висновок	100%	10%	70%

Для взаємодії з API та збереження стану використовувалась React Query – потужна бібліотека для управління запитами до серверу, кешуванням та оновленням даних у реальному часі. Це дозволяє уникнути зайвих перезапитів і зменшити навантаження на сервер, покращуючи загальну продуктивність застосунку [17]. У порівнянні з Redux, React Query не вимагає централізованого збереження всього стану, має простіший API для роботи з асинхронними даними та кращу інтеграцію з REST/GraphQL API.

Для валідація даних було вирішено виконати за допомогою Zod – бібліотеки, яка забезпечує типобезпечність і точну валідацію вхідних даних [18]. Альтернативами є Yup і Joi. Yup має простіший синтаксис, але гіршу інтеграцію з TypeScript. Joi –s більш потужна, але переважно використовується на сервері (Node.js). Zod було обрано через зручну підтримку TypeScript і декларативність.

Отже, на основі порівняльного аналізу було обґрунтовано вибір Next.js, Appwrite, Tailwind CSS, React Query та Zod як оптимальних технологій для реалізації системи управління проєктною діяльністю IT-команд. Їх комбінація забезпечує сучасну, масштабовану та зручну інфраструктуру з високою швидкістю розробки і підтримкою гнучких інтеграцій.

3.2 Розробка модуля авторизації та реєстрації користувача

Модуль автентифікації відповідає за надання користувачам можливості входу до системи та реєстрації нових облікових записів. Він є критично важливим

елементом вебзастосунку, що забезпечує безпечний доступ до персоніфікованого функціоналу системи управління проєктною діяльністю.

Фронтенд-частина модуля автентифікації реалізована з використанням React, React Query для обробки асинхронних запитів і Next.js Router для перенаправлення. Основна логіка авторизації інкапсульована в хук useLogin, який, у свою чергу, виконує POST-запит до /api/auth/login. У разі успішної автентифікації користувача, оновлюються дані про сесію, а інтерфейс перезавантажується для відображення оновленого стану (рис. 3.1).

```
export const useLogin = () => {
  const router = useRouter();
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ json }) => {
      const response = await client.api.auth.login['$post']({ json });

      if (!response.ok) throw new Error('Email or Password is incorrect!');

      return await response.json();
    },
    onSuccess: () => {
      router.refresh();

      queryClient.invalidateQueries({
        queryKey: ['current'],
      });
    },
    onError: () => {
      toast.error('Email or Password is incorrect!');
    },
  });

  return mutation;
```

Рисунок 3.1 – Хук useLogin для автентифікації користувача

Форма входу та реєстрації створюються як окремі компоненти, що дозволяє зручно перемикати режими автентифікації. У компоненті AuthForm реалізовано логіку збереження поточного стану (вхід або реєстрація) з використанням React state. Це забезпечує динамічне оновлення інтерфейсу без перезавантаження сторінки.

На бекенді реалізовано маршрути /login, /register, /logout та /current з використанням Hono.js, Appwrite SDK та middleware для управління сесіями. Запити валідуються за допомогою бібліотеки zod, а сам процес автентифікації інтегровано з Appwrite (рис. 3.2).

```
.post('/login', zValidator('json', signInFormSchema), async (ctx) => {
  const { email, password } = ctx.req.valid('json');

  const { account } = await createAdminClient();

  const session = await account.createEmailPasswordSession(email, password);

  setCookie(ctx, AUTH_COOKIE, session.secret, {
    path: '/',
    httpOnly: true,
    secure: true,
    sameSite: 'strict',
    maxAge: 60 * 60 * 24 * 30,
  });

  return ctx.json({ success: true });
})
```

Рисунок 3.2 – Обробка запитів автентифікації на сервері

Аналогічно, обробник /register створює нового користувача та одразу ініціює сесію. Збереження cookie забезпечує подальшу ідентифікацію користувача у запитах.

Реєстрація користувача реалізована у вигляді окремого користувацького хука useRegister, який інкапсулює логіку створення нового облікового запису через API. Цей хук використовує React Query для управління асинхронним запитом та оновлення кешу після успішної операції. На рисунку 3.3 зображено реалізацію функції реєстрації.

```
export const useRegister = () => {
  const router = useRouter();
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ json }) => {
      const response = await client.api.auth.register['$post']({ json });

      if (!response.ok) throw new Error('Failed to register!');

      return await response.json();
    },
    onSuccess: () => {
      router.refresh();

      queryClient.invalidateQueries({
        queryKey: ['current'],
      });
    },
    onError: (error) => {
      console.error('[REGISTER]: ', error);

      toast.error('Failed to register!');
    },
  });

  return mutation;
```

Рисунок 3.3 – Хук useRegister для створення нового користувача

Цей хук викликається у компоненті AuthForm, який залежно від стану перемикає між формами входу і реєстрації. Користувач заповнює поля з іменем, електронною поштою та паролем, після чого ці дані передаються у запит POST на /api/auth/register.

У разі успішної реєстрації:

- оновлюється кеш поточного користувача (['current']);
- виконується перезавантаження маршрутизатора для отримання нових даних з сервера;
- інтерфейс автоматично переходить до головної сторінки користувача або простору.

У разі помилки користувач отримує повідомлення через компонент toast, а також помилка фіксується у консолі для налагодження.

3.3 Розробка модуля керування задачами

Головною задачею модуля керування задачами є забезпечення створення, редагування, візуалізації, фільтрації та обробки задач у рамках конкретного проєкту. Модуль взаємодіє з базою даних Appwrite та реалізує динамічну взаємодію з користувачем через React-компоненти.

Кожна задача (task) у системі представлена як окремий документ у базі даних Appwrite зі визначеною структурою полів, що забезпечує її повноцінну інтеграцію в проєкт. Як показано на рисунку 3.4, структура задачі включає:

- name – назва задачі;
- status – статус виконання;
- assigneeId –ідентифікатор виконавця;
- projectId – ідентифікатор проєкту, до якого належить задача;
- workspaceId – зв'язок із робочим простором;
- position – порядковий номер задачі в списку;
- dueDate – дедлайн виконання задачі;
- description2 – додатковий опис задачі.

Ця структура дозволяє модулю задач гнучко керувати інформацією, підтримувати фільтрацію, сортування.

```
11 export type Task = Models.Document & {
12     name: string;
13     status: TaskStatus;
14     assigneeId: string;
15     projectId: string;
16     workspaceId: string;
17     position: number;
18     dueDate: string;
19     description?: string;
20 };
```

Рисунок 3.4 – Структура задачі

Створення задачі реалізується через кастомний хук useCreateTask, який використовує useMutation з бібліотеки React Query для обробки запиту на

створення. Запит надсилається до API клієнта через client.api.tasks['post'], а у випадку помилки створення викидається виняток із повідомленням.

У разі успішного створення виконується сповіщення користувача через toast.success, а також оновлення (інвалідація) відповідних кешованих запитів у React Query:

- workspace-analytics – аналітика для робочого простору;
- project-analytics – аналітика для проєкту;
- tasks – список задач у просторі.

У випадку помилки виводиться повідомлення про невдалу спробу через toast.error, а помилка логуються в консоль.

На рисунку 3.5 можна побачити логіку створення задачі.

```
export const useCreateTask = () => {
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ json }) => {
      const response = await client.api.tasks['$post']({ json });

      if (!response.ok) throw new Error('Failed to create task.');

      return await response.json();
    },
    onSuccess: ({ data }) => {
      toast.success('Task created.');

      queryClient.invalidateQueries({
        queryKey: ['workspace-analytics', data.workspaceId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['project-analytics', data.projectId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['tasks', data.workspaceId],
        exact: false,
      });
    },
    onError: (error) => {
      console.error('[CREATE_TASK]: ', error);

      toast.error('Failed to create task.');
    },
  });

  return mutation;
```

Рисунок 3.5 – Створення задачі

Задачі виводяться у вигляді Kanban-дошки, де кожна колонка відповідає певному статусу (todo, in-progress, done тощо). Приклад реалізації колонки в Kanban-картки задачі можна побачити на рисунку 3.6. Підхід Kanban-дошки дозволяє візуально відстежувати прогрес виконання завдань у межах проєкту.

```
export const KanbanCard = ({ task }: KanbanCardProps) => {
  return (
    <div className="mb-1.5 space-y-3 rounded bg-white p-2.5 shadow-sm">
      <div className="flex items-start justify-between gap-x-2">
        <p className="line-clamp-2 text-sm">{task.name}</p>

        <TaskActions id={task.$id} projectId={task.projectId}>
          <MoreHorizontal className="size-[18px] shrink-0 cursor-pointer stroke-1 text-neutral-700 transition hover:opacity-75" />
        </TaskActions>
      </div>

      <DottedSeparator />

      <div className="flex items-center gap-x-1.5">
        <MemberAvatar name={task.assignee.name} fallbackClassName="text-[10px]" />
        <div aria-hidden className="size-1 rounded-full bg-neutral-300" />
        <TaskDate value={task.dueDate} className="text-xs" />
      </div>

      <div className="flex items-center gap-x-1.5">
        <ProjectAvatar name={task.project.name} image={task.project.imageUrl} fallbackClassName="text-[10px]" />
        <span className="text-xs font-medium">{task.project.name}</span>
      </div>
    </div>
  );
```

Рисунок 3.6 – Kanban-картка задачі

Функціональність drag-and-drop реалізована через бібліотеку @hello-pangea/dnd. Вона дозволяє перетягувати задачі між колонками зі зміною статусу та позиції. Компонент DataKanban групує задачі за статусом (TaskStatus). Далі візуалізує задачі у колонках (Droppable) і самі задачі (Draggable). Після перетягування викликає функцію onChange з оновленими значеннями статусу та позиції. На рисунку 3.7 зображено ключова логіка переміщення задачі, а саме на кінець закінчення перетаскування.

```
61  const onDragEnd = useCallback(
62    (result: DropResult) => {
63      if (!result.destination) return;
64
65      const { source, destination } = result;
66      const sourceStatus = source.droppableId as TaskStatus;
67      const destStatus = destination.droppableId as TaskStatus;
68
69      let updatesPayload: { $id: string; status: TaskStatus; position: number }[] = [];
70
71      setTasks((prevTasks) => {
72        const newTasks = { ...prevTasks };
73
74        // Safely remove the task from the source column
75        const sourceColumn = [...newTasks[sourceStatus]];
76        const [movedTask] = sourceColumn.splice(source.index, 1);
77
78        // If there is no moved task, return the previous state
79        if (!movedTask) {
80          console.error('No task found at the source index.');
81          return prevTasks;
```

Рисунок 3.7 – Приклад ключової логіки переміщення задачі

Редагування задачі реалізується в окремому компоненті з передзаповненням форми. Воно виконуєтсья через модальне вікно, де дані форми автоматично заповнюються поточними значеннями.

На рисунку 3.8 можна побачити фрагмент коду, що демонструє логіку оновлення задачі за допомогою React Query. Для цього використовується хук useMutation, який надсилає запит до серверу з новими даними задачі. У разі успішного оновлення, користувач отримує повідомлення про успіх, а кешовані дані (аналітика, список задач, конкретна задача) оновлюються для відображення актуальної інформації.

Якщо під час запиту виникає помилка, система повідомляє про це користувача. Такий підхід дозволяє оновлювати дані без повного перезавантаження сторінки та підтримувати інтерфейс у актуальному стані.

```
export const useUpdateTask = () => {
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ json, param }) => {
      const response = await client.api.tasks[':taskId']['$patch']({ json, param });

      if (!response.ok) throw new Error('Failed to update task.');

      return await response.json();
    },
    onSuccess: ({ data }) => {
      toast.success('Task updated.');

      queryClient.invalidateQueries({
        queryKey: ['workspace-analytics', data.workspaceId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['project-analytics', data.projectId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['tasks', data.workspaceId],
        exact: false,
      });
      queryClient.invalidateQueries({
        queryKey: ['task', data.$id],
        exact: true,
      });
    },
    onError: (error) => {
      console.error('[UPDATE_TASK]: ', error);

      toast.error('Failed to update task.');
    },
  });

  return mutation;
```

Рисунок 3.8 – Функція оновлення задачі

Функціонал видалення задачі реалізовано окремою кнопкою зі спливаючим підтвердженням. Реалізація видалення задачі можна побачити на рисунку 3.9. Видалення задачі реалізовано з використанням React Query. Видалення виконується через API-запит методом DELETE. Якщо відповідь від серверу успішна, користувач бачить повідомлення про успішне видалення.

Після цього інвалідуються (оновлюються) кешовані дані аналітики простору, проєкту, списку задач та окремої задачі, щоб інтерфейс відображав актуальну інформацію. Якщо ж трапляється помилка, вона логуються в консолі, а також виводиться повідомлення про невдале видалення.

```typescript
export const useDeleteTask = () => {
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ param }) => {
      const response = await client.api.tasks[':taskId']['$delete']({ param });

      if (!response.ok) throw new Error('Failed to delete task.');

      return await response.json();
    },
    onSuccess: ({ data }) => {
      toast.success('Task deleted.');

      queryClient.invalidateQueries({
        queryKey: ['workspace-analytics', data.workspaceId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['project-analytics', data.projectId],
        exact: true,
      });
      queryClient.invalidateQueries({
        queryKey: ['tasks', data.workspaceId],
        exact: false,
      });
      queryClient.invalidateQueries({
        queryKey: ['task', data.$id],
        exact: true,
      });
    },
    onError: (error) => {
      console.error('[DELETE_TASK]: ', error);

      toast.error('Failed to delete task.');
    },
  });

  return mutation;
};
```

Рисунок 3.9 – Функція видалення задачі

Фільтрація задач реалізована на клієнтському боці за допомогою React-змінних стану (рис. 3.10).

Для зручності користувача реалізовано фільтрацію задач за:

- проєктом;
- статусом;
- назвою;
- дедлайном;
- виконавцем.

Фільтрація відбувається локально після отримання задач з сервера.

```
export const useTaskFilters = () => {
  return useQueryStates({
    projectId: parseAsString,
    status: parseAsStringEnum(Object.values(TaskStatus)),
    assigneeId: parseAsString,
    search: parseAsString,
    dueDate: parseAsString,
  });
};
```

Рисунок 3.10 – Форма фільтрації задач

Модуль задач забезпечує повний цикл роботи з задачами від створення і фільтрації до оновлення та переміщення між статусами. Завдяки інтеграції Appwrite і React Query, дані автоматично синхронізуються після будь-яких CRUD-операцій, що значно підвищує зручність роботи користувача з Kanban-інтерфейсом. Компонентний підхід забезпечує повторне використання коду, а Zod гарантує безпечність і валідність введених даних.

3.4 Розробка модуля керування робочими просторами

Головною задачею функціонального модуля керування робочими просторами (Workspace) є організація та адміністрування простору для командної роботи над проєктами. Кожен робочий простір містить інформацію про учасників команди, ролі, пов’язані проєкти, дошки завдань та календарі.

Модуль реалізовано за допомогою React Query для керування станом і асинхронною логікою. На рисунку 3.11 представлено структуру основного компонента WorkspaceSwitcher, який відповідає за відображення списку робочих просторів для поточного користувача. Для кожного елемента списку створюється динамічне посилання на сторінку конкретного робочого простору, реалізоване за допомогою маршруту /workspace/[workspaceId]. У випадку, якщо користувач не автентифікований, здійснюється автоматичне перенаправлення на сторінку входу.

```tsx
export const WorkspaceSwitcher = () => {
  const router = useRouter();
  const workspaceId = useWorkspaceId();

  const { open } = useCreateWorkspaceModal();
  const { data: workspaces } = useGetWorkspaces();

  const onSelect = (id: string) => {
    router.push(`/workspaces/${id}`);
  };

  return (
    <div className="flex flex-col gap-y-2">
      <div className="flex items-center justify-between">
        <p className="text-xs uppercase text-neutral-500">Workspaces</p>
        <button onClick={open}>
          <RiAddCircleFill className="size-5 cursor-pointer text-neutral-500 transition hover:opacity-75" />
        </button>
      </div>
```

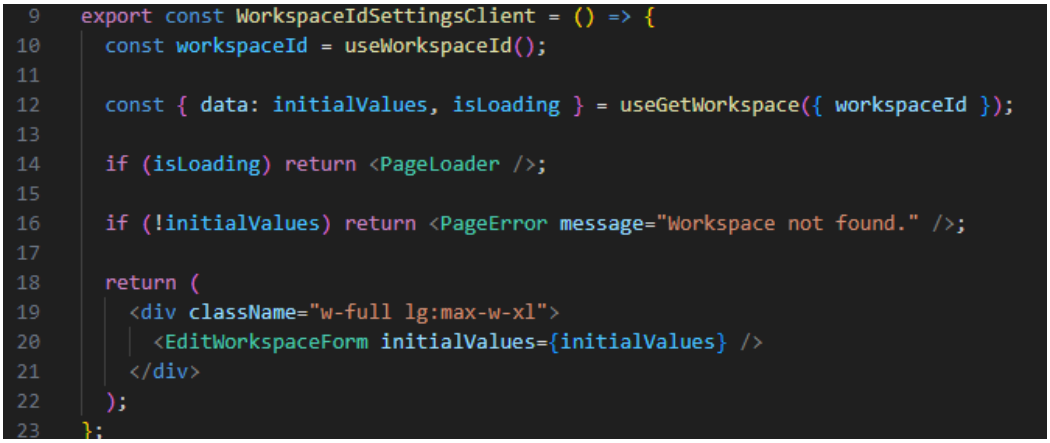
Рисунок 3.11 – Компонент для виведення доступних робочих просторів

Дані про робочі простори отримуються через функцію getWorkspaces, яка взаємодіє з сервером Appwrite через REST API. На рисунку 3.12 зображено виклик цієї функції та її внутрішню реалізацію.

```tsx
export const getWorkspaces = async () => {
  try {
    const { account, databases, storage } = await createSessionClient();

    const user = await account.get();
    const members = await databases.listDocuments(DATABASE_ID, MEMBERS_ID, [Query.equal('userId', user.$id)]);

    if (members.total === 0) return { documents: [], total: 0 };

    const workspaceIds = members.documents.map((member) => member.workspaceId);

    const workspaces = await databases.listDocuments(DATABASE_ID, WORKSPACES_ID, [
      Query.contains('$id', workspaceIds),
      Query.orderDesc('$createdAt'),
    ]);

    const workspacesWithImages: Models.Document[] = await Promise.all(
      workspaces.documents.map(async (workspace) => {
        let imageUrl: string | undefined = undefined;

        if (workspace.imageId) {
          const arrayBuffer = await storage.getFileView(IMAGES_BUCKET_ID, workspace.imageId);
          imageUrl = `${Buffer.from(arrayBuffer).toString('base64')}`;
        }

        return {
          ...workspace,
          imageUrl,
        };
      }),
    );

    return {
      documents: workspacesWithImages,
      total: workspaces.total,
    };
  } catch {
    return { documents: [], total: 0 };
  }
```

Рисунок 3.12 – Функція getWorkspaces для отримання робочих просторів

Наступним етапом є відображення детальної інформації про вибраний робочий простір, а також надання можливостей для взаємодії з його внутрішніми модулями: створення задач, перегляд календаря, призначення ролей тощо. На рисунку 3.13 показано компонент WorkspaceIdSettingsClient, який відображає дані конкретного простору. Спочатку компонент отримує workspaceId за допомогою відповідного хука. Далі відбувається запит до API для отримання даних простору. У разі завантаження або помилки виводяться відповідні компоненти (PageLoader або PageError). Якщо дані успішно отримані, вони передаються у форму редагування EditWorkspaceForm.



```
export const WorkspaceIdSettingsClient = () => {
  const workspaceId = useWorkspaceId();

  const { data: initialValues, isLoading } = useGetWorkspace({ workspaceId });

  if (isLoading) return <PageLoader />;

  if (!initialValues) return <PageError message="Workspace not found." />;

  return (
    <div className="w-full lg:max-w-xl">
      <EditWorkspaceForm initialValues={initialValues} />
    </div>
  );
};
```

Рисунок 3.13 – Компонент WorkspaceIdSettingsClient для перегляду одного робочого простору

Для можливості створення нового робочого простору використовується компонент модального вікна, який викликається з кнопки «+ New Workspace», і який збирає дані форми (назва, фото) та виконує запит до Appwrite для створення нового документа. Створення нового робочого простору відбувається за допомогою функції useCreateWorkspace (рис. 3.14). Вона використовує модульний підхід React, зокрема хук useMutation, що дозволяє керувати асинхронними запитами. При виклику функції збираються дані з форми (назва та фото), які надсилаються на сервер через клієнт Appwrite для створення нового документа в базі даних.

Якщо запит проходить успішно, користувач отримує повідомлення про створення простору, а інтерфейс оновлюється: виконується оновлення роутера та повторне завантаження пов’язаних даних (списку робочих просторів). Якщо ж виникає помилка, її фіксує консоль, і користувачу показується повідомлення про невдалу спробу створення. У такий спосіб забезпечується зручний і зрозумілий досвід для користувача при додаванні нового робочого простору.

```
11  export const useCreateWorkspace = () => {
12    const router = useRouter();
13    const queryClient = useQueryClient();
14
15    const mutation = useMutation<ResponseType, Error, RequestType>({
16      mutationFn: async ({ form }) => {
17        console.log('[FORM_DATA]:', form); //
18        const response = await client.api.workspaces['$post']({ form });
19
20        if (!response.ok) throw new Error('Failed to create workspace.');
21
22        return await response.json();
23      },
24      onSuccess: () => {
25        toast.success('Workspace created.');
26
27        router.refresh();
28        queryClient.invalidateQueries({
29          queryKey: ['workspaces'],
30        });
31      },
32      onError: (error) => {
33        console.error('[CREATE_WORKSPACE]: ', error);
34
35        toast.error('Failed to create workspace.');
36      },
37    });
38
39    return mutation;
```

Рисунок 3.14 – Функція для створення нового робочого простору

Таким чином, модуль Workspace забезпечує повний цикл роботи з робочими просторами: перегляд, створення, маршрутизацію та взаємодію з іншими модулями системи, такими як Kanban-дошки, календар, профілі учасників тощо. Кожен користувач отримує доступ лише до тих просторів, до яких він належить.

3.5 Розробка модуля управління членами команди

Модуль управління членами команди забезпечує отримання списку учасників простору, зміну ролей користувачів та видалення членів команди. На рисунках 3.15–3.17 наведено реалізацію відповідних хук-функцій.

Хук useGetMembers виконує GET-запит на маршрут /api/members, передаючи workspaceId як query-параметр. Серверна частина перевіряє автентифікацію користувача, а потім повертає список членів з додатковими полями name та email, отриманими з API Appwrite Users.

Запит на зміну ролі обробляється PATCH-методом. На сервері перевіряється, чи є ініціатор зміни адміністратором простору. Також враховується, що єдиний учасник не може бути знижений у правах.

На сервері при видаленні перевіряється, чи користувач є учасником простору, та чи має право видаляти інших. Видалення є неможливим, якщо учасник – єдиний у просторі.

```
export const useGetMembers = ({ workspaceId }: UseGetMembersProps) => {
  const query = useQuery({
    queryKey: ['members', workspaceId],
    queryFn: async () => {
      const response = await client.api.members.$get({ query: { workspaceId } });

      if (!response.ok) throw new Error('Failed to fetch members.');

      const { data } = await response.json();

      return data;
    },
  });

  return query;
};
```

Рисунок 3.15 – Хук useGetMembers для отримання списку учасників

```ts
export const useUpdateMember = () => {
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ param, json }) => {
      const response = await client.api.members[':memberId']['$patch']({ param, json });

      if (!response.ok) throw new Error('Failed to update member.');

      return await response.json();
    },
    onSuccess: ({ data }) => {
      toast.success('Member updated.');

      queryClient.invalidateQueries({
        queryKey: ['members', data.workspaceId],
        exact: true,
      });
    },
    onError: (error) => {
      console.error('[UPDATE_MEMBER]: ', error);

      toast.error('Failed to update member.');
    },
  });

  return mutation;
```

Рисунок 3.16 – Хук useUpdateMember для оновлення ролі користувача

```ts
export const useDeleteMember = () => {
  const queryClient = useQueryClient();

  const mutation = useMutation<ResponseType, Error, RequestType>({
    mutationFn: async ({ param }) => {
      const response = await client.api.members[':memberId']['$delete']({ param });

      if (!response.ok) throw new Error('Failed to delete member.');

      return await response.json();
    },
    onSuccess: ({ data }) => {
      toast.success('Member deleted.');

      queryClient.invalidateQueries({
        queryKey: ['members', data.workspaceId],
        exact: true,
      });
    },
    onError: (error) => {
      console.error('[DELETE_MEMBER]: ', error);

      toast.error('Failed to delete member.');
    },
  });

  return mutation;
```

Рисунок 3.17 – Хук useDeleteMember для видалення учасника

Бекенд реалізований з використанням Hono-фреймворку. Основні операції:

- GET /api/members – отримання списку учасників;

- PATCH /api/members/:memberId – оновлення ролі;
- DELETE /api/members/:memberId – видалення учасника.

Для автентифікації використовується sessionMiddleware, а перевірка доступу реалізована через функцію getMember, яка визначає роль користувача у просторі.

Також реалізовано перевірки на:

- наявність адміністративних прав;
- заборону змін ролей або видалення єдиного учасника простору;
- відповідність поточного користувача тому, кого він намагається змінити чи видалити.

3.6 Висновки

У третьому розділі було проведено варіантний аналіз та обґрунтовано вибір технологій для створення вебсистеми оптимізації командної роботи над проєктами. Було обрано сучасні фреймворки й бібліотеки – Next.js 14, Tailwind CSS, Appwrite, React Query та Zod, що забезпечують масштабованість, зручність розробки, продуктивність та безпеку. Реалізовано модулі реєстрації та авторизації, керування задачами, членами команди та робочими просторами, які дозволяють створювати, редагувати, фільтрувати та відображати задачі, а також організовувати взаємодію між користувачами в межах команд.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Вибір методу тестування

Для тестування програмних модулів вебсистеми оптимізації командної роботи над проєктами обрано комбінований підхід, що поєднує методи «чорної скриньки» та «білої скриньки» [19]. Такий вибір зумовлений комплексністю системи, яка включає клієнтську частину (інтерфейс Kanban-дошки, календар, форму створення задач, панель користувача) та серверну логіку (аутентифікація через OAuth, керування робочими просторами, збереження даних в Appwrite).

Метод «чорної скриньки» застосовується для перевірки функціональності з погляду кінцевого користувача без урахування внутрішньої реалізації [19]. Зокрема, тестуються такі сценарії:

- реєстрація та вхід користувача через емейл з паролем, Google або GitHub;
- створення та редагування задач;
- перетягування задач між колонками;
- фільтрація та пошук задач;
- створення команд і запрошення нових учасників.

Це дозволяє перевірити відповідність функціоналу технічним вимогам, обробку помилок (наприклад, некоректні логіни чи порожні поля), а також загальну стабільність системи.

Метод «білої скриньки» використовується для аналізу внутрішньої логіки програмного коду. З його допомогою проводиться:

- перевірка алгоритмів авторизації, ролей користувачів і контролю доступу;
- аналіз функцій CRUD-операцій для задач і проєктів;
- перевірка логіки оновлення статусу задач при перетягуванні;
- тестування інтеграцій із базою даних Appwrite та реакції системи на виключення чи збої з’єднання.

Також проводиться перевірка коректності використання React Query для асинхронного запиту даних, валідації форм за допомогою Zod та обробки помилок на клієнті.

Комбінація методів дозволяє досягти високого покриття функціоналу тестами, забезпечити виявлення як зовнішніх, так і внутрішніх помилок. Ручне тестування застосовується для UI-компонентів, а автоматизоване – для API-ендпоінтів і серверної логіки.

4.2 Опис процесу тестування

Процес тестування вебсистеми оптимізації командної роботи над проєктами проводився вручну з використанням комбінованого підходу «чорної скриньки» та «білої скриньки». Основний інструмент тестування – Postman, який використовувався для перевірки роботи API-ендпоінтів вебсистеми [20].

На етапі підготовки були визначені ключові функціональні можливості, що підлягали перевірці:

- реєстрація та автентифікація користувачів через сторонні сервіси (Google, GitHub);
- створення та управління робочими просторами;
- додавання, редагування, перегляд і видалення задач;
- керування статусами задач;
- інтеракція з календарем задач;
- призначення ролей та взаємодія між учасниками команд.

Для методу «чорної скриньки» перевірялися зовнішні функції API без аналізу внутрішньої логіки, а саме правильність відповіді на запити, обробка введених даних, повідомлення про помилки. Наприклад:

- тестування POST-запиту на створення задачі з валідними даними (очікувано – код відповіді 201);
- перевірка відповіді на запит із неповними чи неправильними даними (наприклад, порожнє поле назви задачі – очікувано помилка з кодом 400);

- тестування доступу до задач іншої команди – очікувана відмова в доступі (403).

У рамках методу «білої скриньки» аналізувалася логіка обробки даних у відповіді на запити: структура JSON, коректність і повнота інформації, яка повертається сервером, а також перевірка реакції системи на граничні або некоректні ситуації (наприклад, неіснуючі ID, порожні запити).

Усі тести проводилися вручну за допомогою Postman, шляхом надсилання HTTP-запитів (GET, POST, PUT, DELETE) до відповідних ендпоінтів. При цьому фіксувалися такі параметри:

- статус-коди відповіді (200, 201, 400, 403, 404);
- час обробки запитів;
- вміст відповіді (структура даних, ключі, повідомлення про помилки).

Результати тестування документувалися у вигляді таблиці, що включала: опис запиту, очікувану відповідь, фактичну поведінку системи та висновок. У процесі тестування було виявлено кілька незначних помилок, зокрема відсутність валідації певних полів при створенні задач та некоректне повідомлення про помилку при спробі доступу до чужого простору.

Ці недоліки були оперативно усунені. Завдяки системному підходу вдалося досягти стабільної та передбачуваної роботи API відповідно до визначених функціональних вимог.

4.3 Представлення результатів тестування

Важливим етапом проєкту є тестування розробленої вебсистеми. Процес тестування проведено з покроковим виконанням програми, від її запуску до завершення, з метою перевірки коректності функціонування всіх основних модулів. Кожен крок супроводжується поясненням отриманих результатів, що відображає реальний досвід взаємодії з системою.

Для тестування були використанні наступні інструменти тестування:

- Google Chrome (v123);
- локальний сервер через Live Server (Visual Studio Code);

- розширення DevTools;
- консоль розробника.

Тестування проводилося вручну.

Для початку тестування було запущено проєкт у браузері за допомогою розширення Live Server, що дозволяє оновлювати застосунок динамічно без неохідності перезапукати його кожен раз [21]. Сторінка логіну (/sign-in), що зображена на рисунку 4.1, завантажилася без помилок. У консолі браузера не було виявлено JavaScript-помилок, що підтверджує коректне підключення усіх скриптів, стилів і ресурсів. Після спроби логінізації було отримано помилку «Емейл або пароль неправильні», що є очікуваною і правильною поведінкою, адже користувач ще не зареєстрований.

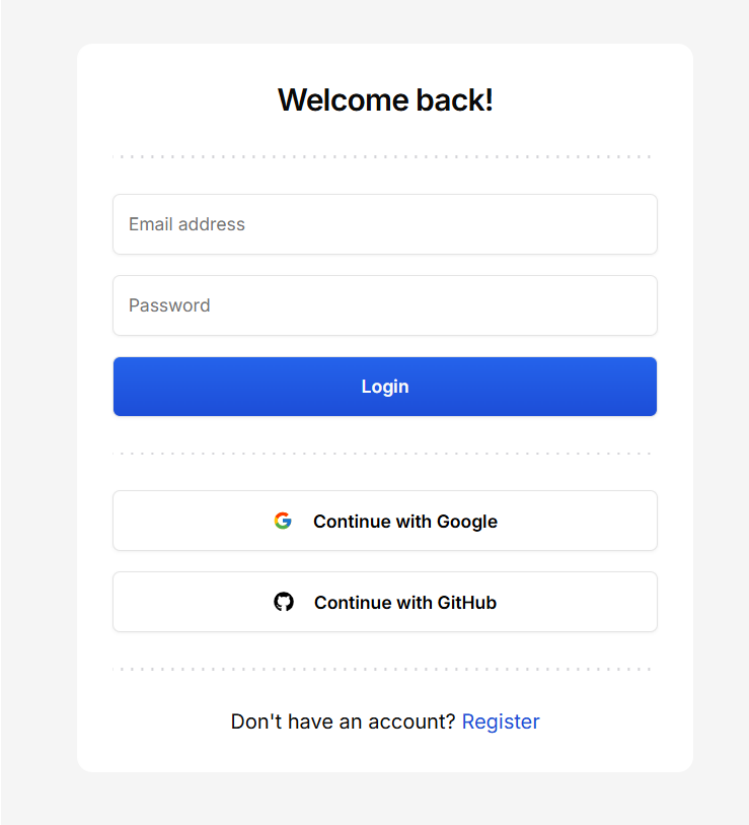


Рисунок 4.1 – Сторінка логіну

Щоб перейти на сторінку реєстрації (/sign-up), потрібно нажати кнопку «Register» внизу форми логіну та користувач буде перенаправлений на сторінку

реєстрації (рис. 4.2). На сторінці була протестована форма реєстрації, було введення ім’я, новий емейл й пароля. Після булоотримано повідомлення «Реєстрація успішна».

Отже, механізм авторизації та реєстрації працює коректно, повідомлення відображаються своєчасно, валідація – реалізована.

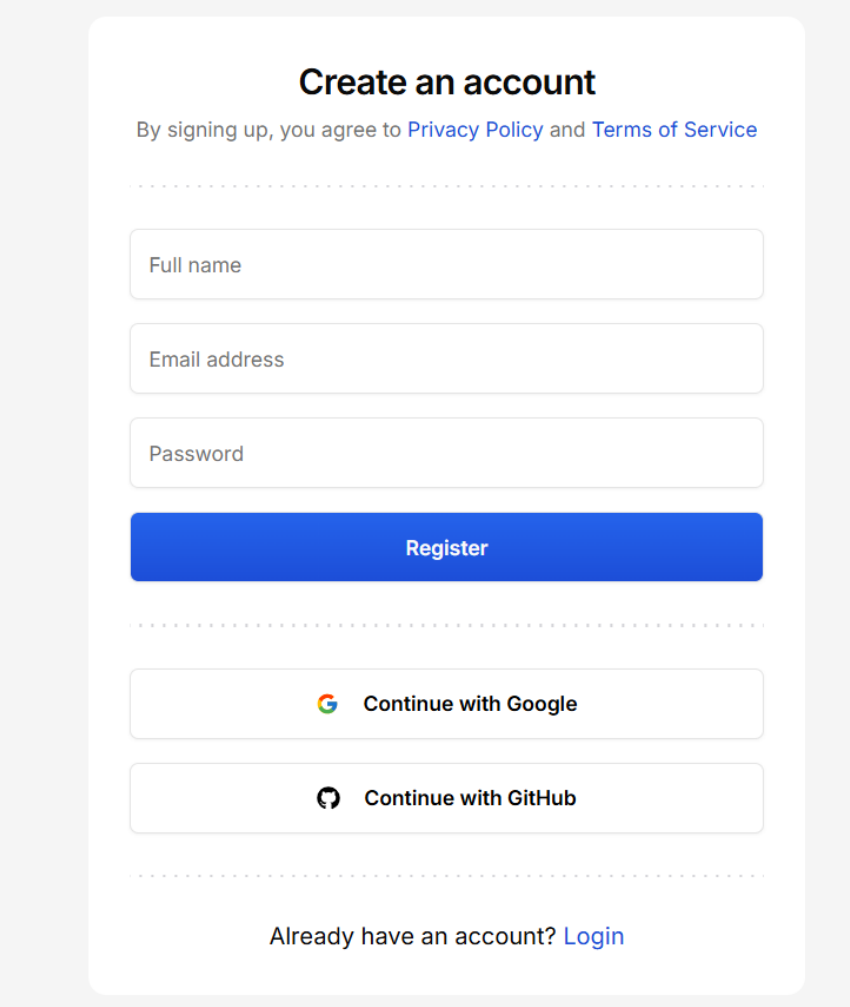


Рисунок 4.2 – Сторінка реєстрації

Після успішної реєстрації, якщо користувач не має створених робочих просторів, він буде направлений на сторінку створення нового робочого простору. У формі потрібно буде ввести назву робочого простору та опіонально можна додати фото (рис. 4.3). Після натискання кнопки «Create workspace» користувач буде направлено на сторінку дашборду створеного робочого простору.

Отже, створенн нового робочого простору працює, дані відображаються одразу, підтверджуючи правильність взаємодії з бекендом.

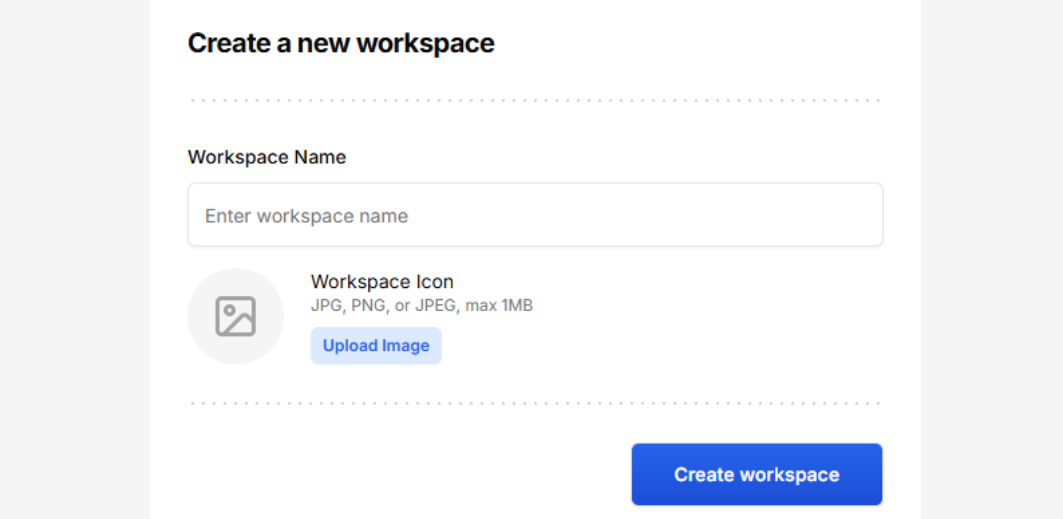


Рисунок 4.3 – Сторінка створення робочого простору

На лівій панелі у розіділ «Projects» при натисканні кнопки створення нового проєкту, буде відкрите модальне вікно (рис. 4.4). У модальній формі потрібно ввести назву проєкту та опційно фото. Після натискання кнопки «Create Project» проєкт з'явився у списку без оновлення сторінки (за допомогою AJAX/JS). Отже, додавання нового проєкту працює, дані відображаються одразу, підтверджуючи правильність взаємодії з бекендом.

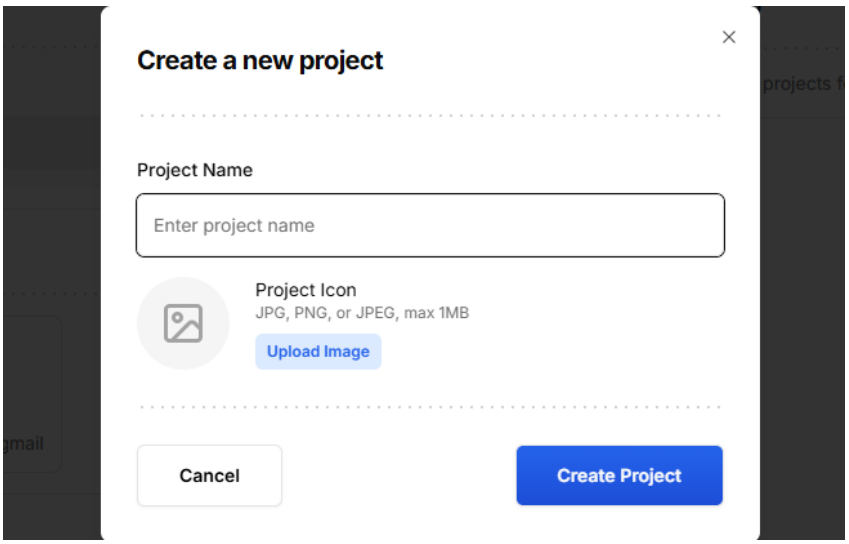


Рисунок 4.4 – Модальне вікно створення нового проєкту

На сторінці проєкту було перевірено роботу блоків із статистичними даним. Після створення проєкту та задач із різними статусами, визначенням дедлайнів та різних виконавців, дані оновлювалися відповідно до змін (рис. 4.5). Отже, аналітичні блоки працюють динамічно, дані оновлюються в реальному часі.

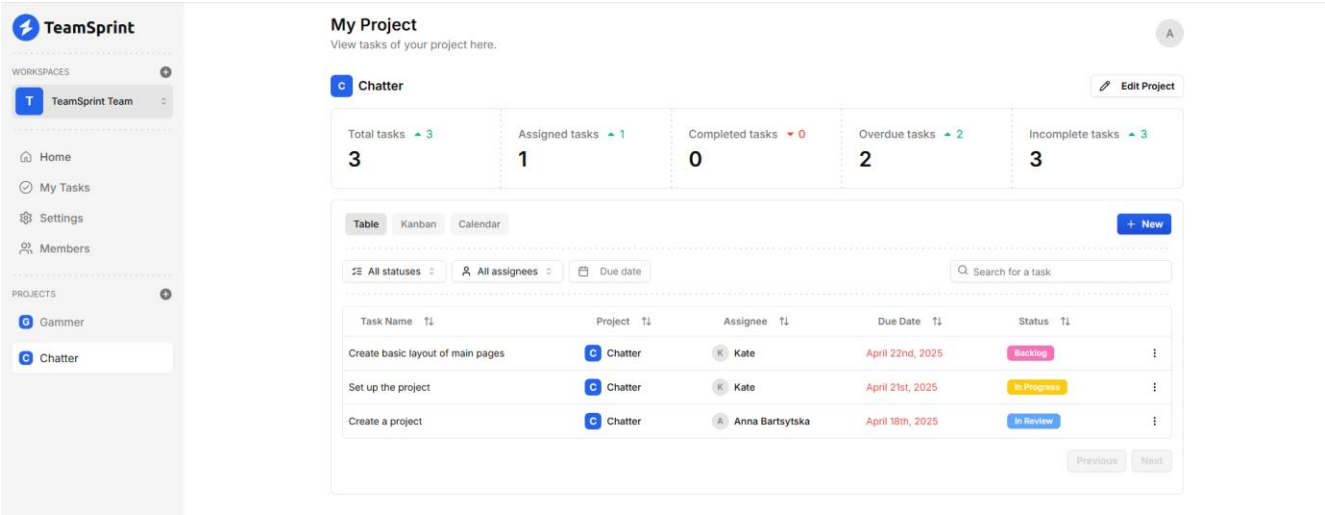


Рисунок 4.5 – Сторінка проекту

У межах створеного проєкту додано кілька завдань із різними пріоритетами та виконавцями. Завдання можна редагувати – змінювався статус, дедлайн, відповідальний. Зміни зберігалися без перезавантаження сторінки. Отже, логіка CRUD-операцій (створення, читання, оновлення, видалення) [22] для завдань реалізована стабільно.

На вкладці «Table» перевірено відображення завдань у вигляді таблиці, де видно назву задачі, назву проєкту, призначеного виконуючого, статус та дедлайн та можна в обрати із доступних дій натиснувши три крапки (рис. 4.6). Отже, відображення в таблиці працює коректно, усі задачі та їх деталі відображаються коректно.

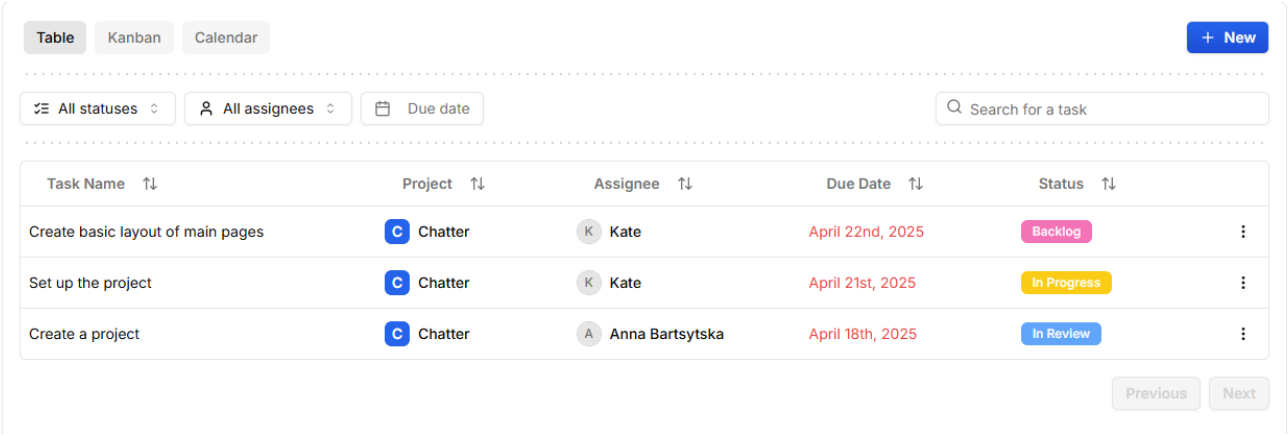


Рисунок 4.6 – Вкладка «Table»

При натискання на редагування задачі буде відкрита сторінка для редагування, де можна відредагувати базові дані та додати опис (рис. 4.7).

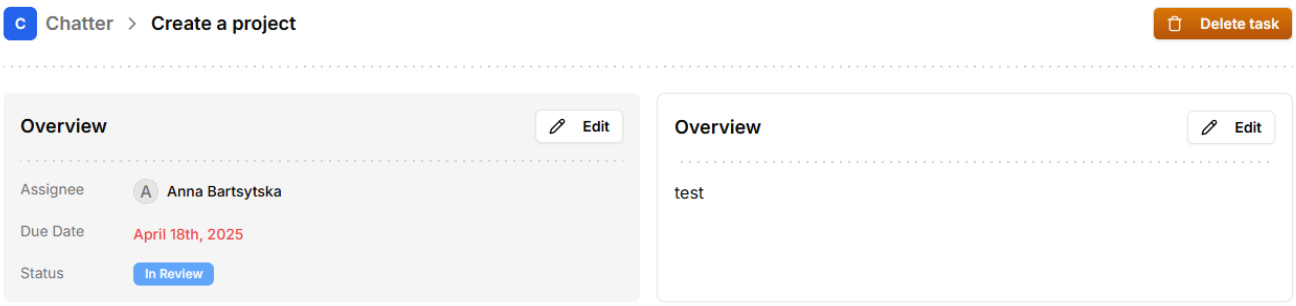


Рисунок 4.7 – Сторінка для огляду і редагування задачі

У системі реалізовано Канбан-дошку, яка дозволяє візуально відслідковувати статуси завдань за принципом «Backlog» → «To Do» → «In Progress» → «In review» → «Done» (рис. 4.8). Завдання можна було переміщувати між колонками – зі статусу «To Do» в «In Progress» або «Done» тощо, а також повертати назад. Під час переміщення статус завдання змінювався автоматично. Отже, канбан-дошка працює згідно з очікуванням, забезпечуючи зручний спосіб керування завданнями та візуалізацію робочого процесу. Функціональність drag-and-drop реалізована стабільно, статуси оновлюються коректно, візуальні маркери покращують зручність користування.

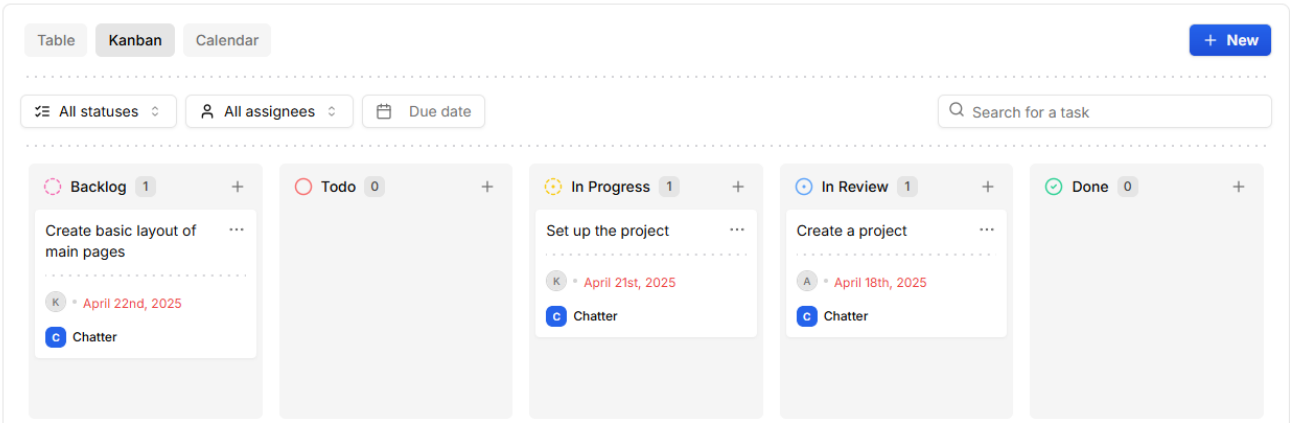


Рисунок 4.8 – Вкладка «Kanban»

На вкладці «Calendar» перевірено відображення завдань у вигляді подій у календарі (рис. 4.9). Було протестовано перемикання між днями, тижнями, місяцями. Всі завдання зі встановленим дедлайном відображалися у відповідні дати. Клік по події відкривається сторінку огляду задачі. Отже, інтеграція з календарем працює, події прив’язані до реального дедлайну.

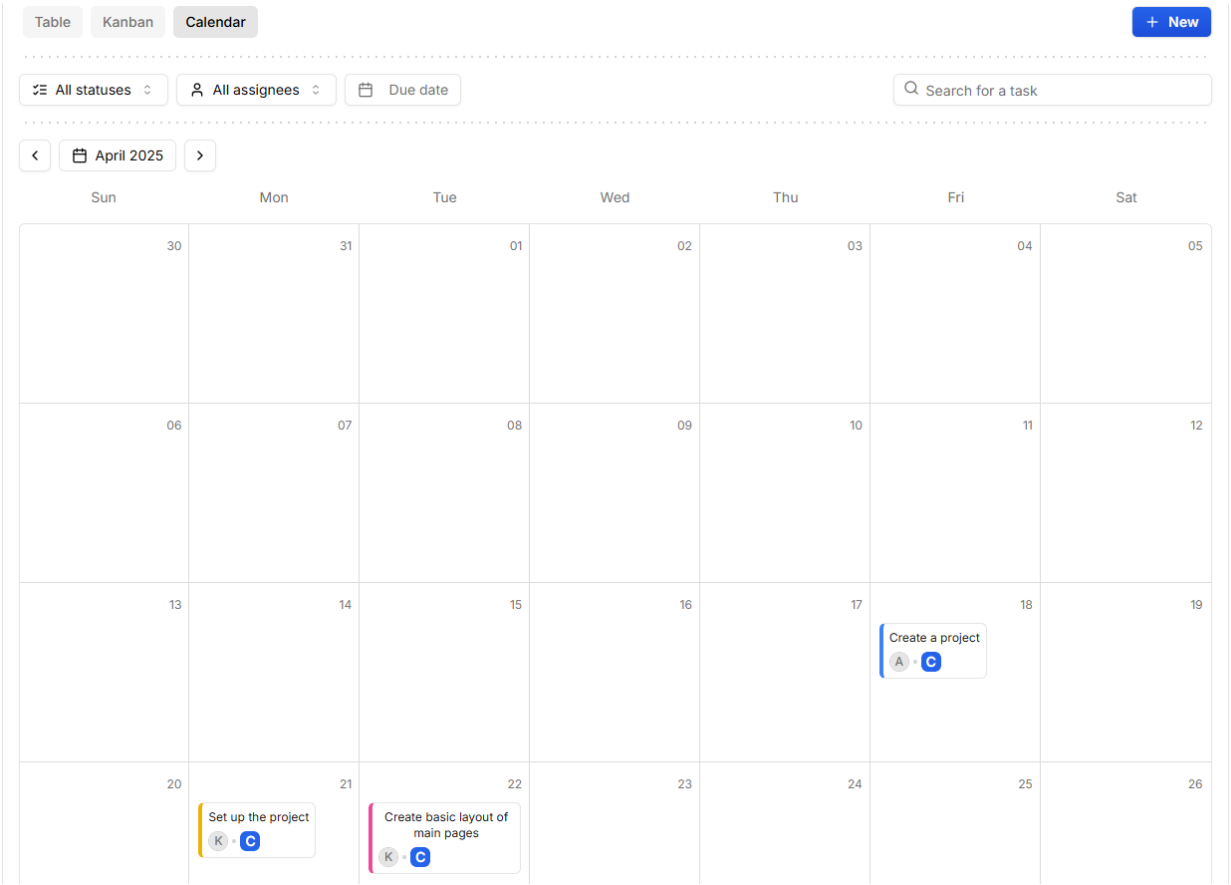


Рисунок 4.9 – Вкладка «Calendar»

Також, для всіх трьох вкладок «Table», «Kanban» та «Calendar» було протетсовано створення задач та фільтрація. Задачі створюються і миттєво з’являються у відповідному просторі. Протестовано фільтрації за виконавцем, статусом та дедлайном. Результати фільтрації відображались миттєво, без оновлення сторінки.

На сторінці «My Tasks» (рис. 4.10) було протестовано відображення та взаємодію з усіма основними функціями керування задачами . Інтерфейс дозволяє переглядати задачі у трьох вкладках: Table, Kanban та Calendar.

Кожен рядок таблиці містить назву задачі, назву проєкту, виконавця, дату дедлайну та статус. Система коректно відображає задачі, що належать до різних проєктів і мають різних виконавців. Була перевірена точність відображення дат та статусів – статуси автоматично підсвічуються відповідним кольором, що полегшує візуальне сприйняття.

Також було протестовано фільтрацію задач за статусом, виконавцем, проєктом та дедлайном. Усі фільтри працювали коректно, миттєво змінюючи вміст. Функція пошуку за ключовим словом у назві задачі успішно знаходила відповідні записи.

Кнопка «+ New» відкривала форму створення нової задачі, яка після збереження миттєво з’являлася в інтерфейсі.

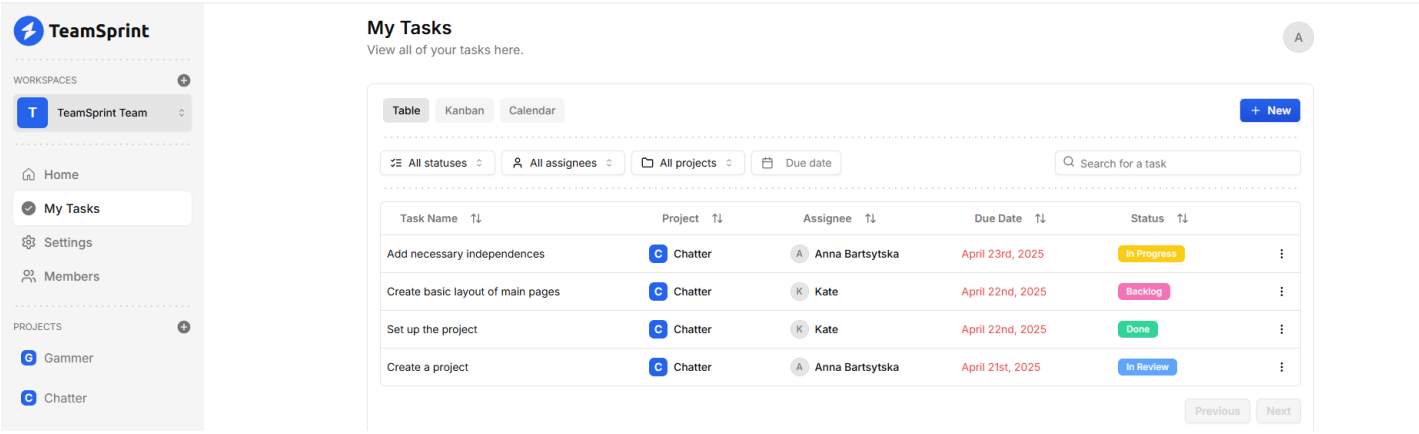


Рисунок 4.10 – Сторінка «My Tasks»

На сторінці налаштування робочого простору (рис. 4.11) було протестовано зміну назви простору та завантаження іконки. Після редагування назви та натискання кнопки «Save Changes», зміни миттєво застосовувались і залишались збереженими після оновлення сторінки. Завантаження зображення у форматах JPG та PNG розміром до 1MB також проходило успішно; некоректні формати або перевищення ліміту викликали повідомлення про помилку.

Було перевірено генерацію та використання інвайт-посилання для запрошення учасників. Скопійоване посилання коректно відкривалося у новому вікні, і система дозволяла приєднання нового користувача до простору. Функція «Reset invite link» формувала нове посилання, при цьому попереднє ставало неактивним, що також було підтверджено під час тестування.

У секції «Danger Zone» було перевірено процес видалення робочого простору. Натискання кнопки «Delete Workspace» супроводжувалось запитом підтвердження дії, після чого простір видалявся без можливості відновлення.

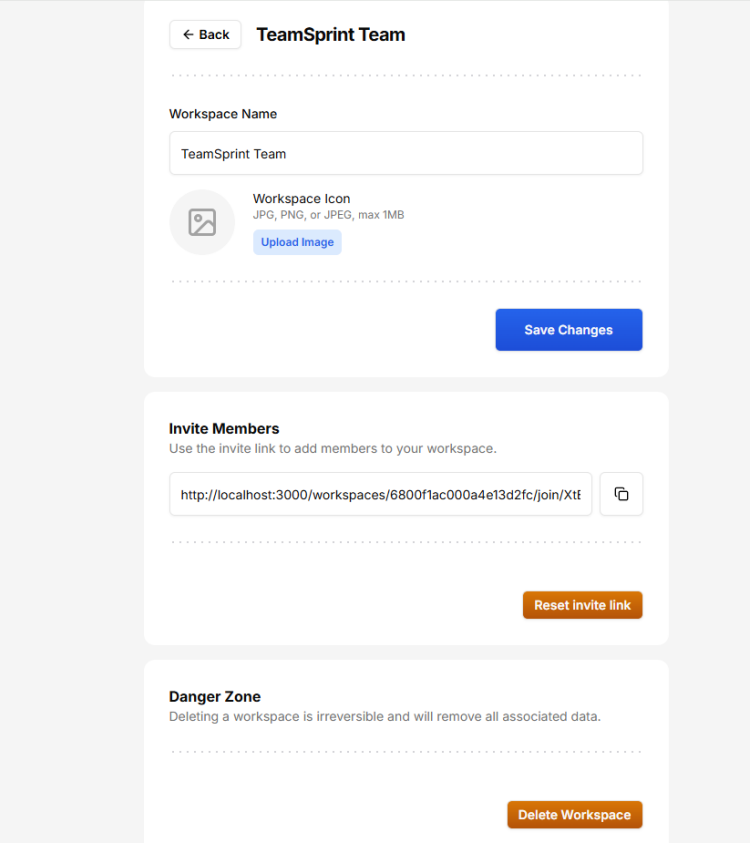


Рисунок 4.10 – Сторінка налаштувань робочого простору

Всі пов’язані дані (задачі, учасники, налаштування) були безповоротно видалені. Отже, сторінка налаштувань функціонує стабільно та забезпечує коректне виконання всіх передбачених дій.

На сторінці «Members» (рис. 4.12) було проведено тестування відображення та керування списком учасників робочого простору. Після приєднання користувачів до простору вони коректно з’являлися у списку, де відображалися їхні імена, email-адреси та аватари з ініціалами.

Було перевірено роботу кнопки з трьома крапками (контекстного меню) біля кожного учасника. Після натискання відкривалося меню з можливими діями (наприклад, видалення учасника або зміна ролі), які виконувалися без помилок і викликали відповідні зміни у складі команди. Видалення користувача зі списку супроводжувалося миттєвим оновленням інтерфейсу без необхідності перезавантаження сторінки. Також наявна кнопка «Back» повертає користувача на сторінку налаштувань робочого простору.

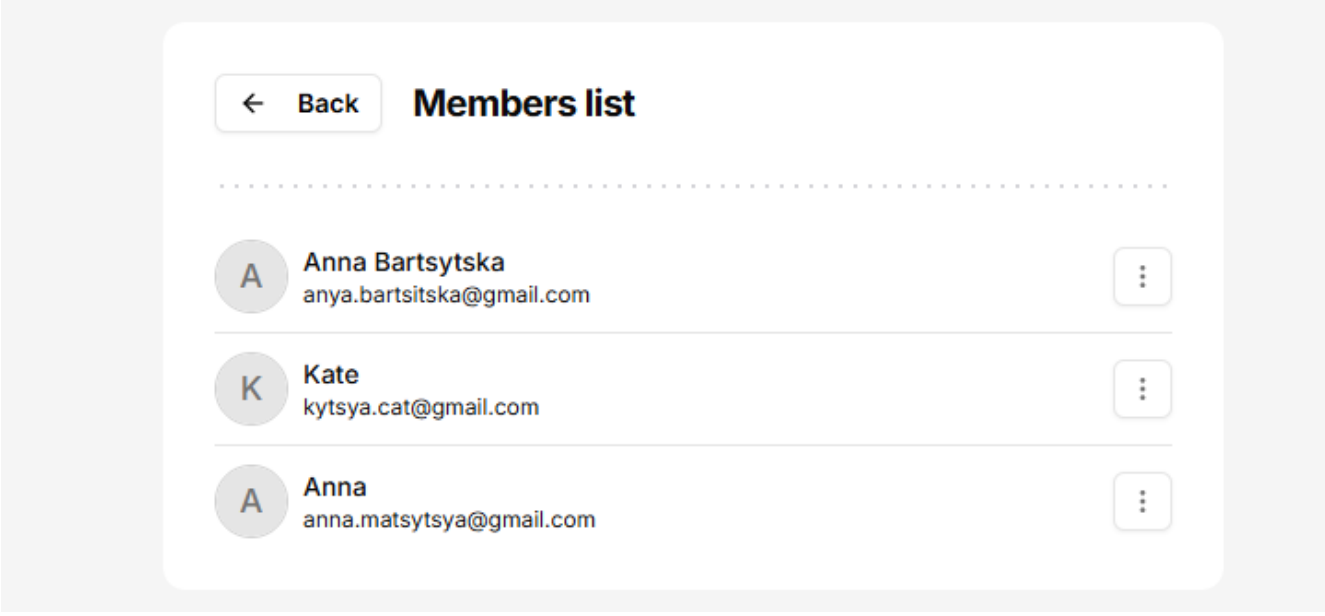


Рисунок 4.12 – Сторінка «Members»

Для тестування виходу із системи потрібно натиснути на іконку в правому верхньому кутку і з’явиться вікно з кнопкою для виходу. Після натискання кнопки «Sign out» користувача перенаправляє на сторінку входу (рис. 4.13). При спробі

вручну повернутися назад – сесія вже завершена, доступ до внутрішніх сторінок обмежено. Отже, механізм завершення сесії працює коректно.

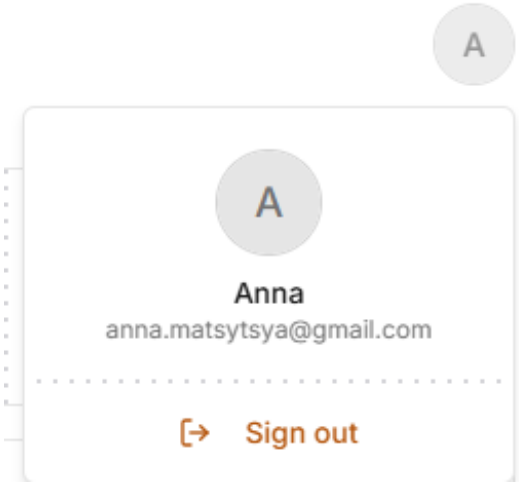


Рисунок 4.13 – Кнопка виходу

Отже, тестування показало, що вебсистема для оптимізації роботи IT-команд функціонує стабільно. Усі ключові модулі – автентифікація, управління проєктами, задачами, аналітика – працюють згідно з очікуваннями. Інтерфейс зручний, а взаємодія з користувачем не потребує перезавантаження сторінок.

4.4 Розробка інструкції користувача

Інструкція користувача визначає технічні вимоги для запуску вебсистеми оптимізації командної роботи над проєктами. У таблиці 4.1 наведено деталі щодо мінімальної та рекомендованої конфігурації апаратного забезпечення для комфортної роботи з програмним продуктом.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5 або аналогічний	Intel Core i3 або аналогічний

Продовження таблиці 4.1

Вимоги до конфігурування	Рекомендовано	Мінімально
Оперативна пам’ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	10 ГБ	2 ГБ
Відеокарта	NVIDIA GeForce GTX 1650 або аналогічна	Вбудована графіка (Intel HD Graphics 4000 або краще)
Операційна система	Windows 10, macOS Ventura або Ubuntu 20.04	Windows 7, macOS Mojave або Ubuntu 18.04
Монітор	Роздільна здатність 1920x1080 пікселів, діагональ 15" або більше	Роздільна здатність 1280x720 пікселів, діагональ 13" або більше

Для роботи з вебсистемою оптимізації командної роботи необхідно мати стабільне підключення до Інтернету зі швидкістю не менше 10 Мбіт/с. Це забезпечує швидке завантаження даних із бекенду.

Після встановлення необхідного середовища (Node.js, та залежностей через npm install) потрібно запустити весистему командою «npm run dev».

Після запуску система стає доступною у браузері за локальною адресою – http://localhost:3000. На початковому екрані з’являється сторінка для входу.

Першим кроком повноцінної роботи із системою є вхід до акаунта або реєстрація. На сторінці «Sign in» користувач вводить логін і пароль. Якщо обліковий запис ще не створено – доступна вкладка «Sign up», де необхідно ввести ім’я, e-mail і пароль.

Система підтримує авторизацію через Google або GitHub (OAuth 2.0), що спрощує вхід у корпоративному середовищі.

Після входу, якщо робочий простір вже створений, то відкривається основна панель управління (рис. 4.14), де доступні ключові функціональні модулі:

- Dashboard – огляд поточних активностей і завдань;

- Workspaces – створення та перегляд робочих просторів;
- My Tasks – взаємодія із задачами користувача;
- Members – управління ролями та членами команди;
- Settings – зміна параметрів робочого середовища;
- Projects – створення та керування проєктами.

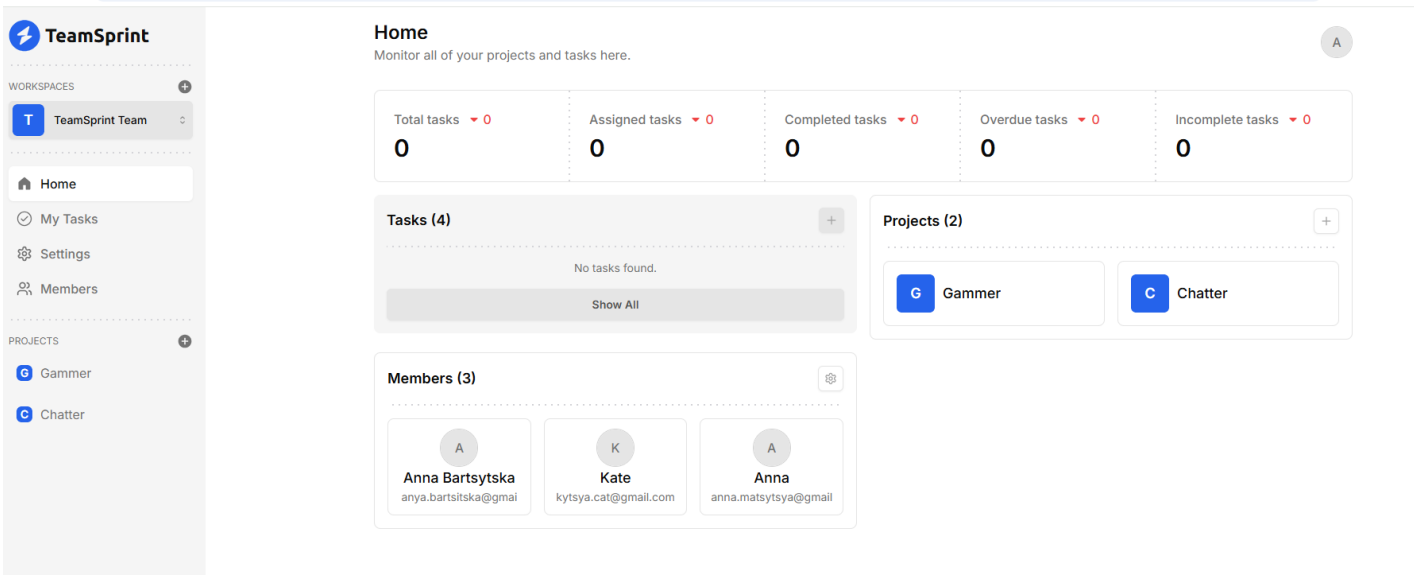


Рисунок 4.14 – Головна сторінка

Після завершення роботи із системою користувач може вийти з облікового запису через меню. Усі дані автоматично зберігаються і будуть доступні при наступному вході.

4.5 Висновки

Проведено тестування вебсистеми оптимізації командної роботи над проєктами з використанням комбінованого підходу – тестування «чорної» та «білої скриньки». Перевірено коректність процесу авторизації, створення та керування робочими просторами, задачами, проєктами та членами команди. Система продемонструвала стабільну роботу за рекомендованих технічних параметрів. Інструкція користувача містить опис вимог, процесу встановлення, входу в систему та навігації між основними модулями.

ВИСНОВКИ

У рамках проєкту була розроблена вебсистема для оптимізації командної роботи в управлінні проєктами, що включає програмні модулі для ефективного планування, контролю завдань, взаємодії між членами команди та моніторингу прогресу проєкту. Для розробки використовувалося середовище програмування Visual Studio Code. Реалізація проєкту відповідає методичним вказівкам [23].

Під час виконання проєкту було здійснено аналіз проблематики. Розглянуто провідні аналоги існуючих програмних рішень, проаналізовано їхні недоліки та проведено порівняння з власною розробкою. На основі отриманих результатів було сформульовано основні завдання проєкту.

У проєкті було зроблено наступне:

- визначено функціональні та нефункціональні вимоги до системи оптимізації командної роботи над проєктами;
- реалізовано архітектуру клієнт-серверної вебсистеми з використанням Next.js 14, Appwrite, Tailwind CSS, React Query, Zod і OAuth-автентифікації;
- розроблено модулі створення та керування задачами, робочими просторами (workspaces), дошками Kanban і календарем;
- забезпечено підтримку багатокористувацької взаємодії з розподіленням ролей (власник, учасник);
- створено зручний, адаптивний і мінімалістичний інтерфейс;
- розроблено модель системи за допомогою UML-діаграм;
- проведено тестування програмних модулів та підготовлено інструкції для користувачів.

Було створено та протестовано алгоритми обробки даних, а також розроблено UML-діаграми для відображення структури й процесів системи.

Тестування системи показало відповідність функціональним вимогам, зручність інтерфейсу та швидкість обробки даних. Створено інструкцію для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання сучасних інформаційних технологій управління проектами [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/318600116_Vikoristanna_sucasnih_informacijnih_tehnologij_upravlinna_proektami

2. Mwamba, M., & Malik, M. A. (2022). The Role of Digital and Virtual Teams in Project Management. Intl. Journal of Scientific Research and Management [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/361599582_The_Role_of_Digital_and_Virtual_Teams_in_Project_Management_Zambia_Centre_for_Communications

3. Карась О. В., Барцицька А. В. Розробка вебсистеми для оптимізації проєктної діяльності в IT-командах. Молодь в науці: дослідження, проблеми, перспективи (МН-2025) : зб. матеріалів доп. учасн. Міжнар. наук.-практ. конф. Вінниця : Вінницький національний технічний університет, 2025. С.3.

4. On Modular Architectures. What they are and why you should care life [Електронний ресурс]. – Режим доступу до ресурсу: https://medium.com/on-software-architecture/on-modular-architectures-53ec61f88ff4

5. How hybrid work has changed the way people work, live, and shop [Електронний ресурс]. – Режим доступу до ресурсу: https://www.mckinsey.com/mgi/our-research/empty-spaces-and-hybrid-places-chapter-1

6. Trello. Visual tool for organizing work and life [Електронний ресурс]. – Режим доступу до ресурсу: https://trello.com

7. Asana. Manage your team’s work, projects, & tasks online [Електронний ресурс]. – Режим доступу до ресурсу: https://asana.com

8. Jira Software – Project Management Software for Agile Teams [Електронний ресурс]. – Режим доступу до ресурсу: https://www.atlassian.com/software/jira

9. ClickUp. One app to replace them all [Електронний ресурс] – Режим доступу до ресурсу: https://clickup.com

10. Monday.com – Work Management Software [Електронний ресурс]. – Режим доступу до ресурсу: https://monday.com

11. What is Next.js? [Електронний ресурс]. – Режим доступу до ресурсу: https://nextjs.org/docs

12. Communication Patterns of Kanban Teams and their Impact on Iteration Performance and Quality ? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.es.mdh.se/pdf_publications/5539.pdf

13. Getting started with Tailwind CSS [Електронний ресурс]. – Режим доступу до ресурсу: https://v2.tailwindcss.com/docs (дата звернення: 01.04.2025)

14. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide (2nd Edition). Boston, MA: Addison-Wesley.

15. What Are Single Page Applications and Why Do People Like Them So Much? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.bloomreach.com/en/blog/what-is-a-single-page-application#:~:text=A%20single%20page%20application%20is,browser%20loading%20entire%20new%20pages

16. Learn how to build like a team of hundreds [Електронний ресурс]. – Режим доступу до ресурсу: https://appwrite.io/docs

17. Beginner's Guide to React Query [Електронний ресурс]. – Режим доступу до ресурсу: https://refine.dev/blog/react-query-guide/#performing-basic-data-fetching

18. Zod [Електронний ресурс]. – Режим доступу до ресурсу: https://zod.dev/

19. Бейзер Б. Техніка тестування програмного забезпечення / Б. Бейзер ; пер. з англ. – К. : Діалектика, 2004. – 512 с.

20. Postman documentation overview [Електронний ресурс]. – Режим доступу до ресурсу: https://learning.postman.com/docs/introduction/overview/

21. Live Server Web overview [Електронний ресурс]. – Режим доступу до ресурсу: https://chromewebstore.google.com/detail/live-server-web-extension/fiegdmejfepffgpnejdinekhfieaogmj

22. CRUD [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CRUD

23. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

72

Додаток А. Технічне завдання

Міністрерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу

«Розробка вебзастосунку для оптимізації процесів управління проєктами»

За спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.ф., асист. каф. ПЗ Карась О. В.
«21» березня 2025 р.

Виконала:
студентка гр. 1ПІ-21б Барцицька А. В
«21» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебзастосунку для оптимізації процесів управління проєктами».

Галузь застосування – інформаційні технології, управління проєктами в сфері розробки програмного забезпечення.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Мета дослідження – підвищення ефективності організації командної роботи в IT-проєктах шляхом створення вебзастосунку з функціями планування, контролю та спільної роботи.

Призначення роботи – розробка вебзастосунку, що забезпечує зручний інструмент для управління проєктами, задачами, командами та часовими ресурсами в IT-середовищі.

4. Вихідні дані для проведення НДР

1. On Modular Architectures. What they are and why you should care life [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/on-software-architecture/on-modular-architectures-53ec61f88ff4

2. What is Next.js? [Електронний ресурс] – Режим доступу до ресурсу: https://nextjs.org/docs

3. Бейзер Б. Техніка тестування програмного забезпечення / Б. Бейзер ; пер. з англ. – К. : Діалектика, 2004. – 512 с.

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 8 ГБ, розмір жорсткого диску 256 ГБ. Операційна система Windows 10 або 11.

6. Конструктивні вимоги.

Вебзастсоунок повинні відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програм для роботи над проєктами в ІТ-командах	25.03.2025-08.04.2025
3	Розробка методів, моделі та алгоритмів роботи програми	09.04.2025-18.04.2025
3	Розробка програмного забезпечення	19.04.2025-07.05.2025
4	Тестування програми	08.05.2025-16.05.2025
5	Оформлення матеріалів до захисту БКР	17.05.2025-30.05.2025

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

76

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка вебзастосунку для оптимізації процесів управління проєктами»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, ФІТКІ, 1ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,7 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Карась О. В., д-р. філос., асист. каф. ПЗ
(прізвище, ініціали, посада)

Здобувач (підпис) Барцицька А. В.
(прізвище, ініціали)

Додаток В. Лістинг програми

Sign-in.tsx

```
import { redirect } from 'next/navigation';

import { SignInCard } from '@/features/auth/components/sign-in-card';
import { getCurrent } from '@/features/auth/queries';

const SignInPage = async () => {
  const user = await getCurrent();

  if (user) redirect('/');

  return <SignInCard />;
};

export default SignInPage;
```

Sign-up.tsx

```
import { redirect } from 'next/navigation';

import { SignUpCard } from '@/features/auth/components/sign-up-card';
import { getCurrent } from '@/features/auth/queries';

const SignUpPage = async () => {
  const user = await getCurrent();

  if (user) redirect('/');

  return <SignUpCard />;
};

export default SignUpPage;
```

auth-layout.tsx

```
'use client';

import Link from 'next/link';
import { usePathname } from 'next/navigation';
import type { PropsWithChildren } from 'react';
```

```
import { Logo } from '@/components/logo';
import { Button } from '@/components/ui/button';

const AuthLayout = ({ children }: PropsWithChildren) => {
  const pathname = usePathname();
  const isSignIn = pathname === '/sign-in';

  return (
    <main className="min-h-screen bg-neutral-100">
      <div className="mx-auto max-w-screen-2xl p-4">
        <nav className="flex items-center justify-between">
          <Logo />

          <div className="flex items-center gap-x-2.5">
            <Button variant="secondary" asChild>
              <Link href={isSignIn ? '/sign-up' : 'sign-in'}>{isSignIn ? 'Register' : 'Login'}</Link>
            </Button>
          </div>
        </nav>

        <div className="flex flex-col items-center justify-center p-4 md:pt-14">{children}</div>
      </div>
    </main>
  );
};

export default AuthLayout;
```

Loading.tsx

```
import { Loader2 } from 'lucide-react';

const AuthLoadingPage = () => {
  return (
    <div className="flex h-screen items-center justify-center">
      <Loader2 className="size-6 animate-spin text-muted-foreground" />
    </div>
  );
};
export default AuthLoadingPage;
```

dashboard.tsx

```
import { redirect } from 'next/navigation';
```

```
import { getCurrent } from '@/features/auth/queries';
import { getWorkspaces } from '@/features/workspaces/queries';

const HomePage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');
  const workspaces = await getWorkspaces();

  if (workspaces.total === 0) redirect('/workspaces/create');

  redirect(`/workspaces/${workspaces.documents[0].$id}`);
};

export default HomePage;
```

dashboard-layout.tsx

```
import type { PropsWithChildren } from 'react';

import { ModalProvider } from '@/components/modal-provider';
import { Navbar } from '@/components/navbar';
import { Sidebar } from '@/components/sidebar';

const DashboardLayout = ({ children }: PropsWithChildren) => {
  return (
    <div className="min-h-screen">
      <ModalProvider />

      <div className="flex size-full">
        <div className="fixed left-0 top-0 hidden h-full overflow-auto lg:block lg:w-[264px]">
          <Sidebar />
        </div>

        <div className="w-full lg:pl-[264px]">
          <div className="mx-auto h-full max-w-screen-xl">
            <Navbar />

            <main className="flex h-full flex-col px-6 py-8">{children}</main>
          </div>
        </div>
      </div>
    </div>
```

```
  );
};
export default DashboardLayout;
```

workspace-client.tsx

```
'use client';

import { Pencil } from 'lucide-react';
import Link from 'next/link';

import { Analytics } from '@/components/analytics';
import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { Button } from '@/components/ui/button';
import { useGetProject } from '@/features/projects/api/use-get-project';
import { useGetProjectAnalytics } from '@/features/projects/api/use-get-project-analytics';
import { ProjectAvatar } from '@/features/projects/components/project-avatar';
import { useProjectId } from '@/features/projects/hooks/use-project-id';
import { TaskViewSwitcher } from '@/features/tasks/components/task-view-switcher';

export const ProjectIdClient = () => {
  const projectId = useProjectId();

  const { data: project, isLoading: isLoadingProject } = useGetProject({ projectId });
  const { data: projectAnalytics, isLoading: isLoadingProjectAnalytics } = useGetProjectAnalytics({ projectId });

  const isLoading = isLoadingProject || isLoadingProjectAnalytics;

  if (isLoading) return <PageLoader />;

  if (!project) return <PageError message="Project not found." />;

  return (
    <div className="flex flex-col gap-y-4">
      <div className="flex items-center justify-between">
        <div className="flex items-center gap-x-2">
          <ProjectAvatar name={project.name} image={project.imageUrl} className="size-8" />

          <p className="text-lg font-semibold">{project.name}</p>
        </div>

        <div>
          <Button variant="secondary" size="sm" asChild>
```

```
          <Link href={`/workspaces/${project.workspaceId}/projects/${project.$id}/settings`}>
            <Pencil className="mr-2 size-4" />
            Edit Project
          </Link>
        </Button>
      </div>
    </div>

    {projectAnalytics && <Analytics data={projectAnalytics} />}

    <TaskViewSwitcher projectId={projectId} hideProjectFilter />
  </div>
 );
};
```

workspace-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';

import { WorkspaceIdClient } from './client';

const WorkspaceIdPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <WorkspaceIdClient />;
};
export default WorkspaceIdPage;
```

client-projects.tsx

```
'use client';

import { Pencil } from 'lucide-react';
import Link from 'next/link';

import { Analytics } from '@/components/analytics';
import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { Button } from '@/components/ui/button';
```

```
import { useGetProject } from '@/features/projects/api/use-get-project';
import { useGetProjectAnalytics } from '@/features/projects/api/use-get-project-analytics';
import { ProjectAvatar } from '@/features/projects/components/project-avatar';
import { useProjectId } from '@/features/projects/hooks/use-project-id';
import { TaskViewSwitcher } from '@/features/tasks/components/task-view-switcher';

export const ProjectIdClient = () => {
  const projectId = useProjectId();

  const { data: project, isLoading: isLoadingProject } = useGetProject({ projectId });
  const { data: projectAnalytics, isLoading: isLoadingProjectAnalytics } = useGetProjectAnalytics({ projectId });

  const isLoading = isLoadingProject || isLoadingProjectAnalytics;

  if (isLoading) return <PageLoader />;

  if (!project) return <PageError message="Project not found." />;

  return (
    <div className="flex flex-col gap-y-4">
      <div className="flex items-center justify-between">
        <div className="flex items-center gap-x-2">
          <ProjectAvatar name={project.name} image={project.imageUrl} className="size-8" />

          <p className="text-lg font-semibold">{project.name}</p>
        </div>

        <div>
          <Button variant="secondary" size="sm" asChild>
            <Link href={`/workspaces/${project.workspaceId}/projects/${project.$id}/settings`}>
              <Pencil className="mr-2 size-4" />
              Edit Project
            </Link>
          </Button>
        </div>
      </div>

      {projectAnalytics && <Analytics data={projectAnalytics} />}

      <TaskViewSwitcher projectId={projectId} hideProjectFilter />
    </div>
  );
};
```

Projects-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';

import { ProjectIdClient } from './client';

const ProjectIdPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <ProjectIdClient />;
};
export default ProjectIdPage;
```

tasks-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';
import { TaskViewSwitcher } from '@/features/tasks/components/task-view-switcher';

const TasksPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return (
    <div className="flex h-full flex-col">
      <TaskViewSwitcher />
    </div>
  );
};
export default TasksPage;
```

taskId-client.tsx

```
'use client';

import { DottedSeparator } from '@/components/dotted-separator';
import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { useGetTask } from '@/features/tasks/api/use-get-task';
import { TaskBreadcrumbs } from '@/features/tasks/components/task-breadcrumbs';
import { TaskDescription } from '@/features/tasks/components/task-description';
```

```
import { TaskOverview } from '@/features/tasks/components/task-overview';
import { useTaskId } from '@/features/tasks/hooks/use-task-id';

export const TaskIdClient = () => {
  const taskId = useTaskId();

  const { data: task, isLoading } = useGetTask({ taskId });

  if (isLoading) return <PageLoader />;

  if (!task) return <PageError message="Task not found." />;

  return (
    <div className="flex flex-col">
      <TaskBreadcrumbs project={task.project} task={task} />

      <DottedSeparator className="my-6" />

      <div className="grid grid-cols-1 gap-4 lg:grid-cols-2">
        <TaskOverview task={task} />
        <TaskDescription task={task} />
      </div>
    </div>
  );
};
```

taskId-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';

import { TaskIdClient } from './client';

const TaskIdPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <TaskIdClient />;
};
export default TaskIdPage;
```

standalone-layout.tsx

```
import type { PropsWithChildren } from 'react';

import { Logo } from '@/components/logo';
import { UserButton } from '@/features/auth/components/user-button';

const StandaloneLayout = ({ children }: PropsWithChildren) => {
  return (
    <main className="min-h-screen bg-neutral-100">
      <div className="mx-auto max-w-screen-2xl px-4">
        <nav className="flex h-[73px] items-center justify-between">
          <Logo />

          <div className="flex items-center gap-x-2.5">
            <UserButton />
          </div>
        </nav>

        <div className="flex flex-col items-center justify-center py-4">{children}</div>
      </div>
    </main>
  );
};
export default StandaloneLayout;
```

createWorkspace-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';
import { CreateWorkspaceForm } from '@/features/workspaces/components/create-workspace-form';

const WorkspaceCreatePage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return (
    <div className="w-full lg:max-w-xl">
      <CreateWorkspaceForm />
    </div>
  );
};
export default WorkspaceCreatePage;
```

workspaceIdJoin-client.tsx

```
'use client';

import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { useGetWorkspaceInfo } from '@/features/workspaces/api/use-get-workspace-info';
import { JoinWorkspaceForm } from '@/features/workspaces/components/join-workspace-form';
import { useWorkspaceId } from '@/features/workspaces/hooks/use-workspace-id';

export const WorkspaceIdJoinClient = () => {
  const workspaceId = useWorkspaceId();
  const { data: initialValues, isLoading } = useGetWorkspaceInfo({ workspaceId });

  if (isLoading) return <PageLoader />;
  if (!initialValues) return <PageError message="Workspace not found." />;

  return (
    <div className="w-full lg:max-w-xl">
      <JoinWorkspaceForm initialValues={initialValues} />
    </div>
  );
};
```

workspaceIdJoin-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';

import { WorkspaceIdJoinClient } from './client';

const WorkspaceIdJoinPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <WorkspaceIdJoinClient />;
};

export default WorkspaceIdJoinPage;
```

workspaceIdMembers-page.tsx

```
import { redirect } from 'next/navigation';
```

```
import { getCurrent } from '@/features/auth/queries';
import { MembersList } from '@/features/workspaces/components/members-list';

const WorkspaceIdMembersPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return (
    <div className="w-full lg:max-w-xl">
      <MembersList />
    </div>
  );
};
export default WorkspaceIdMembersPage;
```

projectIdSettings-client.tsx

```
'use client';

import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { useGetProject } from '@/features/projects/api/use-get-project';
import { EditProjectForm } from '@/features/projects/components/edit-project-form';
import { useProjectId } from '@/features/projects/hooks/use-project-id';

export const ProjectIdSettingsClient = () => {
  const projectId = useProjectId();
  const { data: initialValues, isLoading } = useGetProject({ projectId });

  if (isLoading) return <PageLoader />;
  if (!initialValues) return <PageError message="Project not found." />;

  return (
    <div className="w-full lg:max-w-xl">
      <EditProjectForm initialValues={initialValues} />
    </div>
  );
};
```

projectIdSettings-page.tsx

```
import { redirect } from 'next/navigation';
```

```
import { getCurrent } from '@/features/auth/queries';

import { ProjectIdSettingsClient } from './client';

const ProjectIdSettingsPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <ProjectIdSettingsClient />;
};
export default ProjectIdSettingsPage;
```

workspaceIdSettings-client.tsx

```
'use client';

import { PageError } from '@/components/page-error';
import { PageLoader } from '@/components/page-loader';
import { useGetWorkspace } from '@/features/workspaces/api/use-get-workspace';
import { EditWorkspaceForm } from '@/features/workspaces/components/edit-workspace-form';
import { useWorkspaceId } from '@/features/workspaces/hooks/use-workspace-id';

export const WorkspaceIdSettingsClient = () => {
  const workspaceId = useWorkspaceId();

  const { data: initialValues, isLoading } = useGetWorkspace({ workspaceId });

  if (isLoading) return <PageLoader />;

  if (!initialValues) return <PageError message="Workspace not found." />;

  return (
    <div className="w-full lg:max-w-xl">
      <EditWorkspaceForm initialValues={initialValues} />
    </div>
  );
};
```

workspaceIdSettings-page.tsx

```
import { redirect } from 'next/navigation';

import { getCurrent } from '@/features/auth/queries';

import { WorkspaceIdSettingsClient } from './client';

const WorkspaceIdSettingsPage = async () => {
  const user = await getCurrent();

  if (!user) redirect('/sign-in');

  return <WorkspaceIdSettingsClient />;
};
export default WorkspaceIdSettingsPage;
```

Api.ts

```
import { Hono } from 'hono';
import { handle } from 'hono/vercel';

import auth from '@/features/auth/server/route';
import members from '@/features/members/server/route';
import projects from '@/features/projects/server/route';
import tasks from '@/features/tasks/server/route';
import workspaces from '@/features/workspaces/server/route';

export const runtime = 'nodejs';

const app = new Hono().basePath('/api');

const routes = app
  .route('/auth', auth)
  .route('/members', members)
  .route('/projects', projects)
  .route('/tasks', tasks)
  .route('/workspaces', workspaces);

export type AppType = typeof routes;

export const GET = handle(app);
export const POST = handle(app);
export const PATCH = handle(app);
export const DELETE = handle(app);
```

Додаток Г. Ілюстрована частина

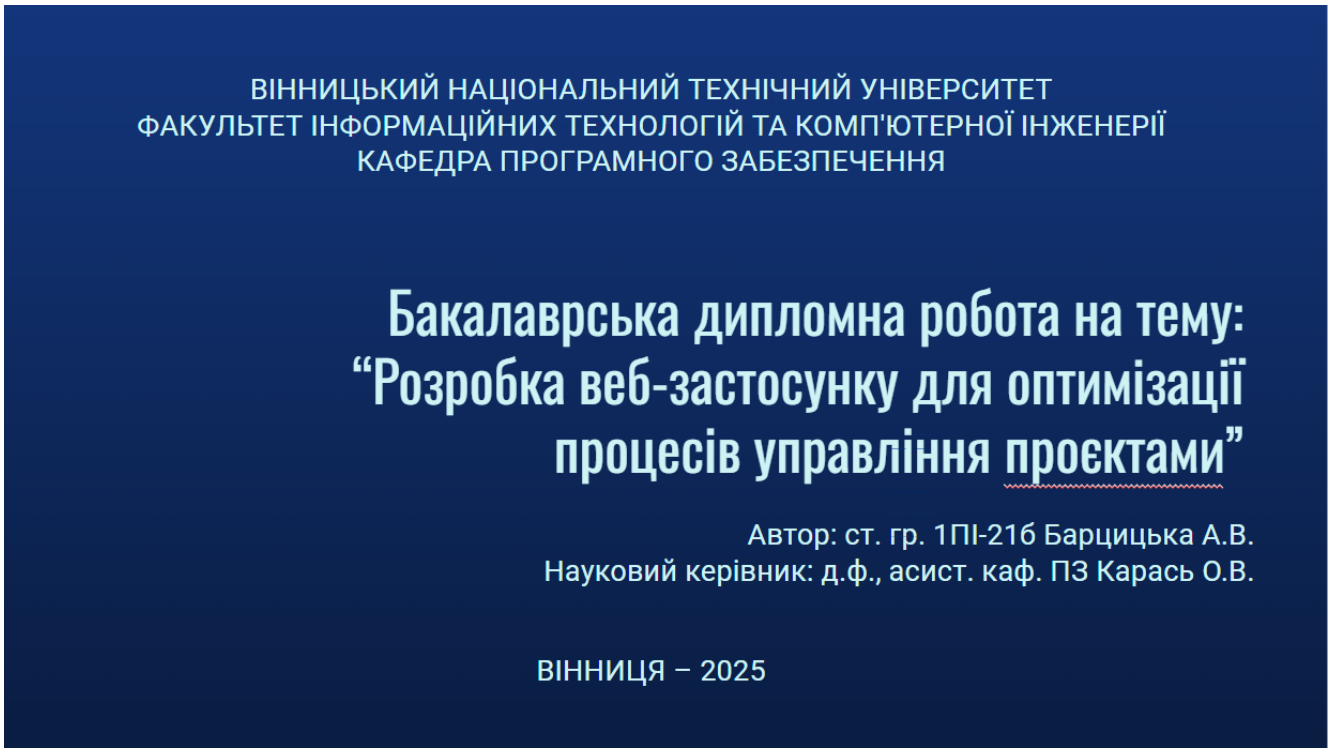


Рисунок Г.1 – Титульний слайд

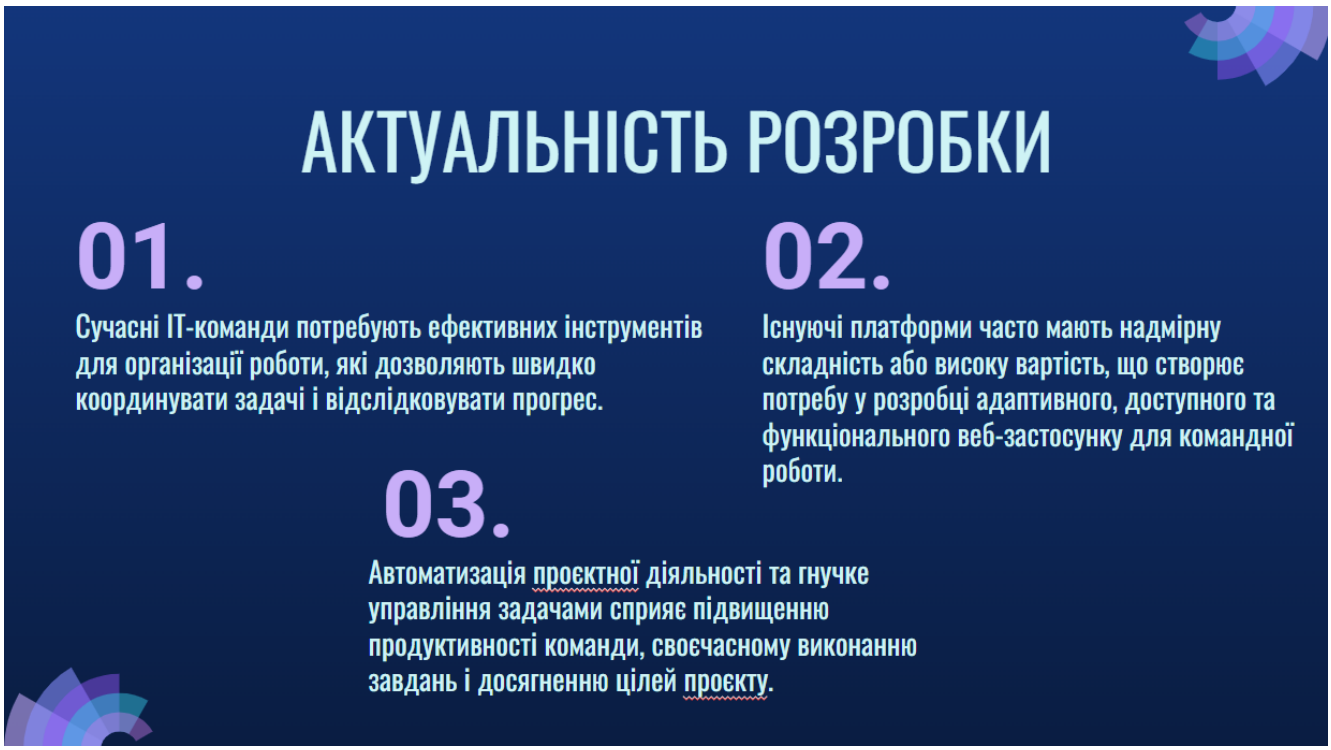


Рисунок Г.2 – Актуальність розробки

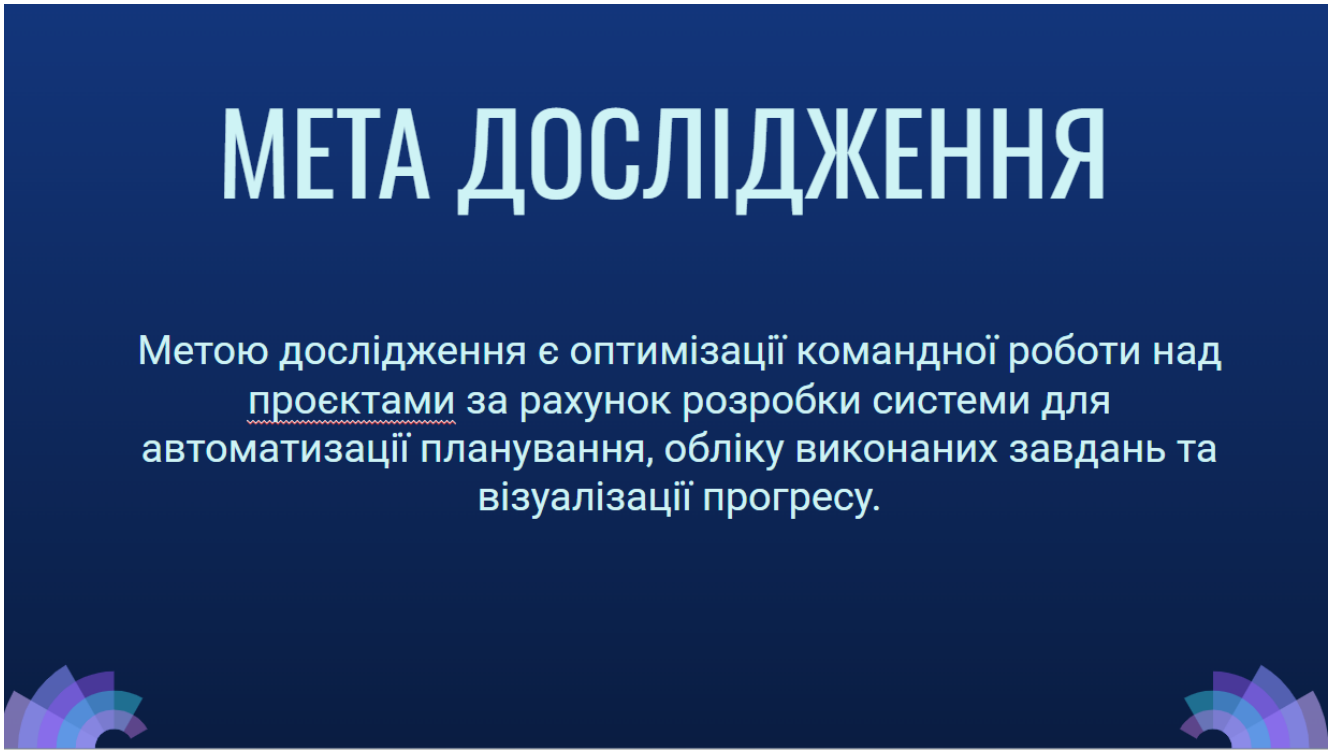


Рисунок Г.3 – Мета дослідження

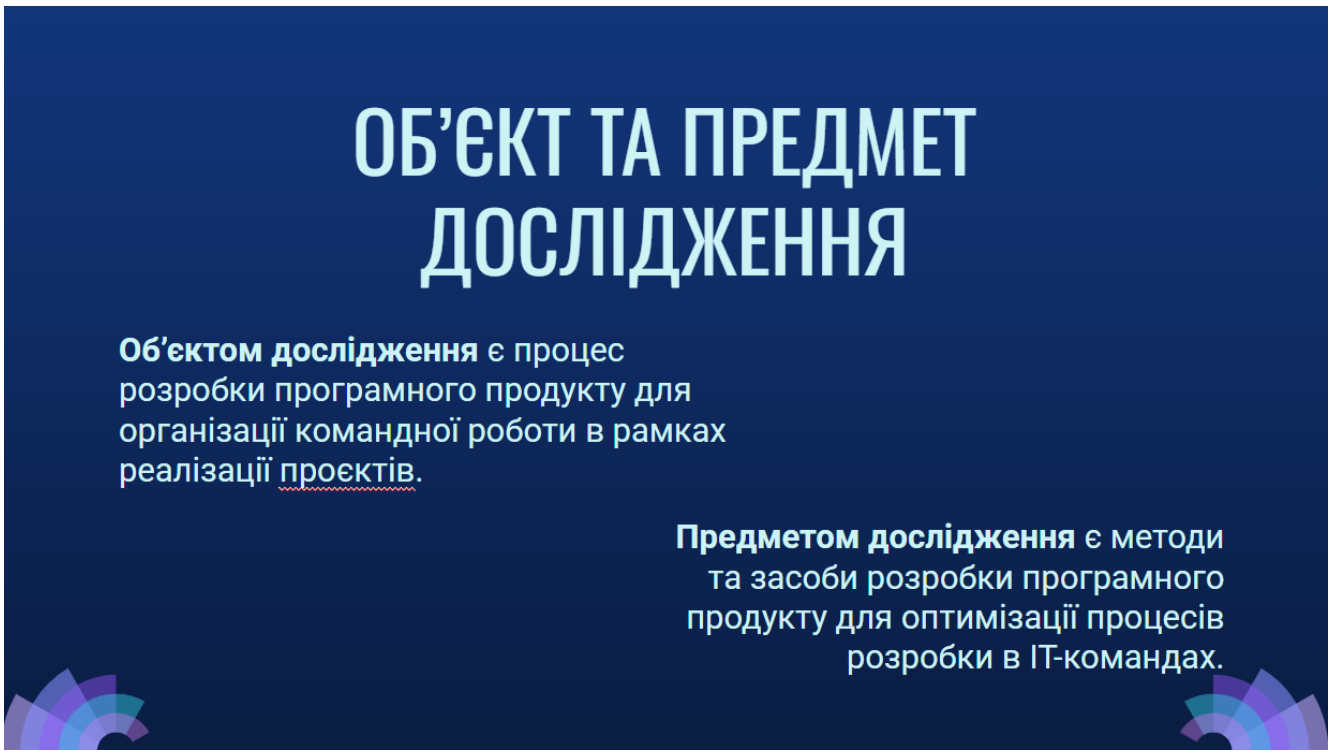


Рисунок Г.4 – Об'єкт та предмет дослідження

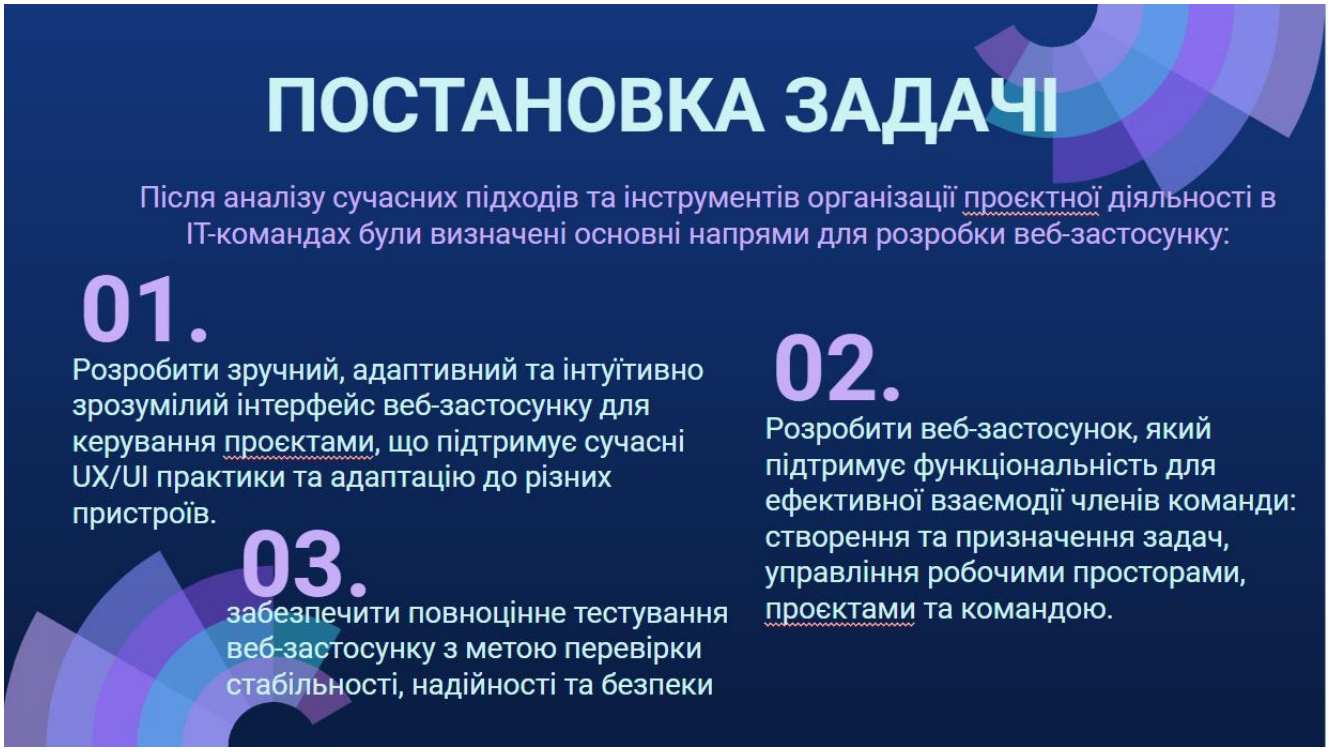


Рисунок Г.5 – Постановка задачі

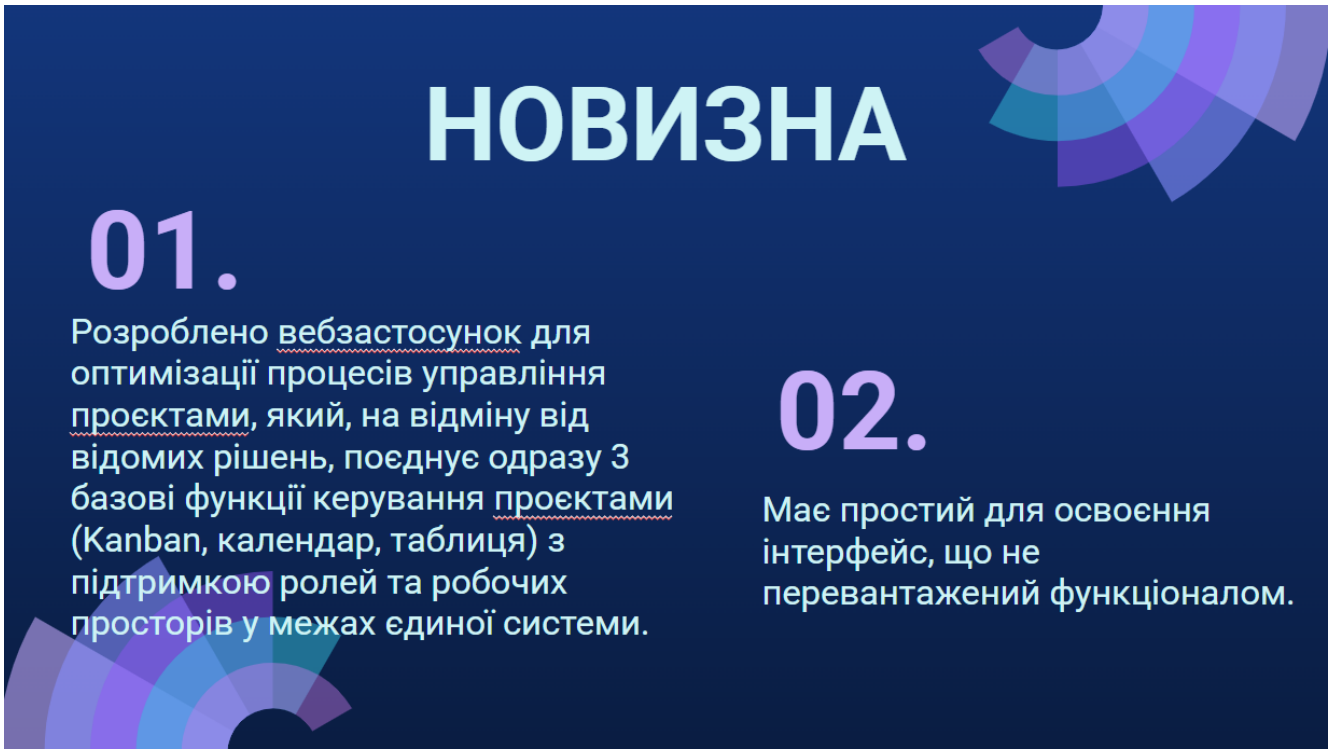


Рисунок Г.6 – Новизна розробленого застосунку

Рисунок Г.7 – Практична цінність

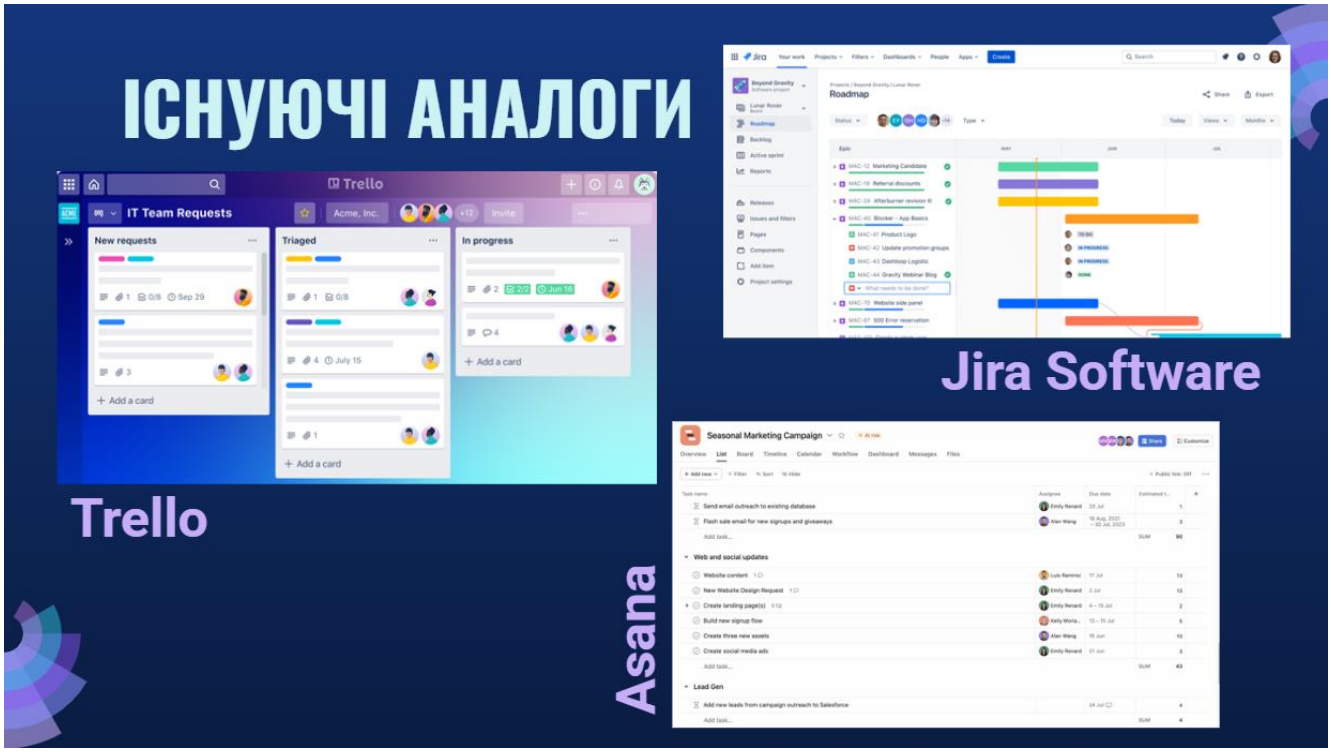


Рисунок Г.8 – Існуючі аналоги як Trello, Jira та Asana

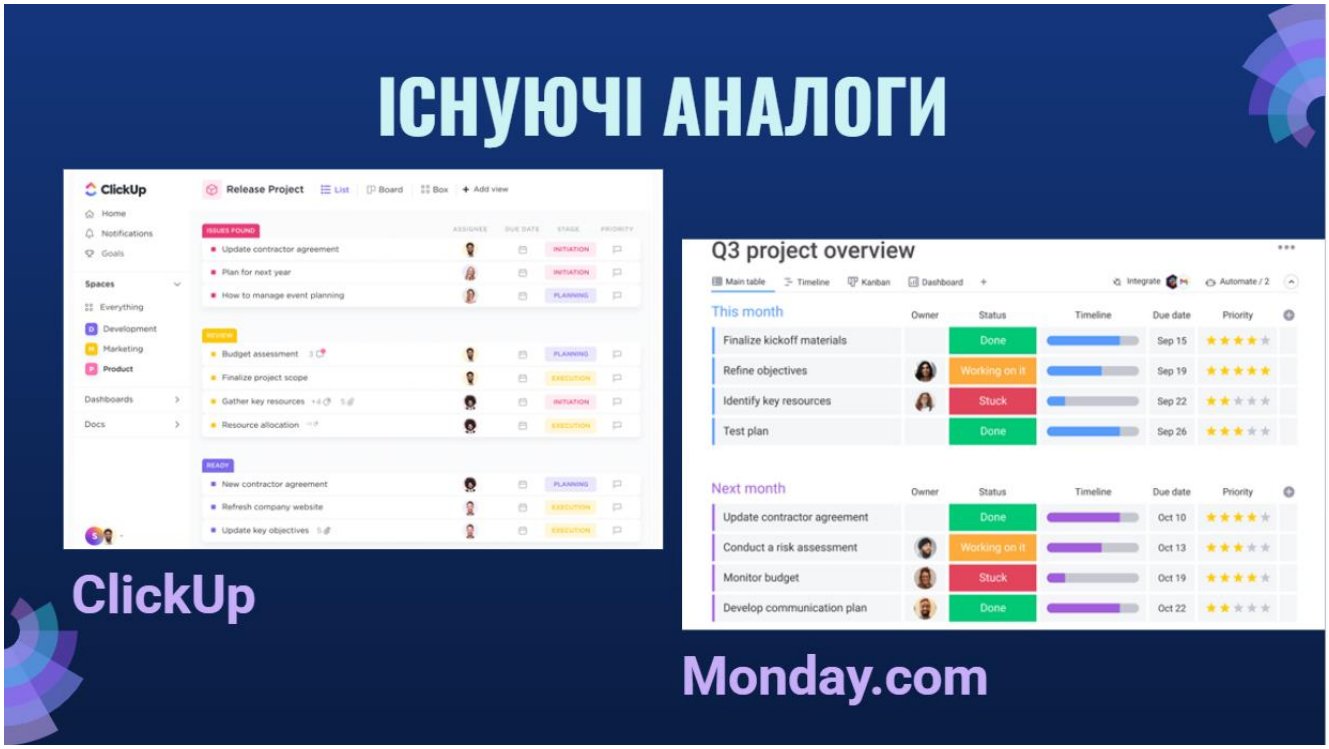


Рисунок Г.9 – Існуючі аналоги як ClickUp та Moday.com

Критерії	Trello	Asana	Jira	ClickUp	Moday.com	Власна розробка
Візуалізація завдань	✓	✓	✗	✓	✓	✓
Гнучкість налаштувань	✗	✓	✓	✓	✗	✓
Підтримка Kanban	✓	✓	✓	✓	✓	✓
Підтримка календаря для роботи із задачами	✗	✗	✗	✗	✗	✓
Інтеграція з іншими сервісами	✓	✓	✓	✓	✓	✓
Простота використання	✓	✗	✗	✗	✓	✓
Загальна оцінка	66,6%	66,6%	50%	66,6%	66,6%	100%

Рисунок Г.10 – Аналіз аналогів

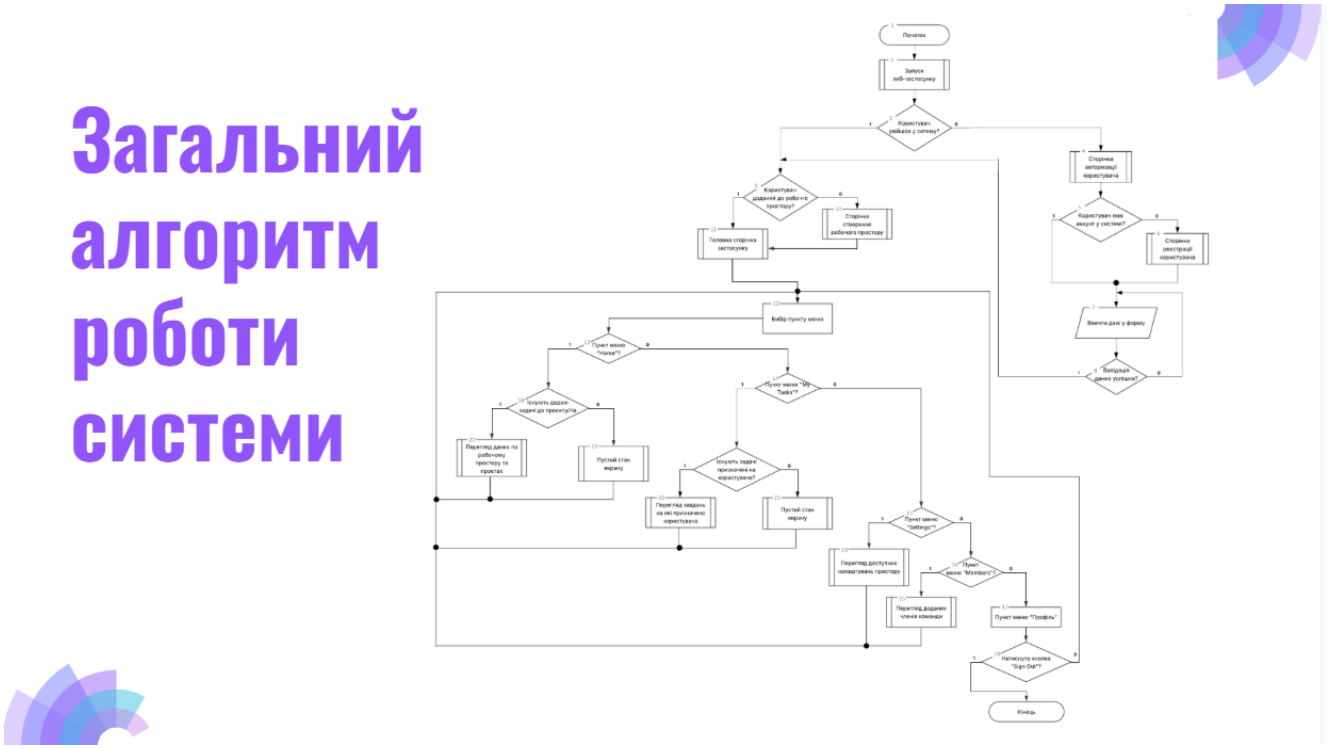


Рисунок Г.11 – Загальний алгоритм роботи системи

Рисунок Г.12 – Діаграми станів

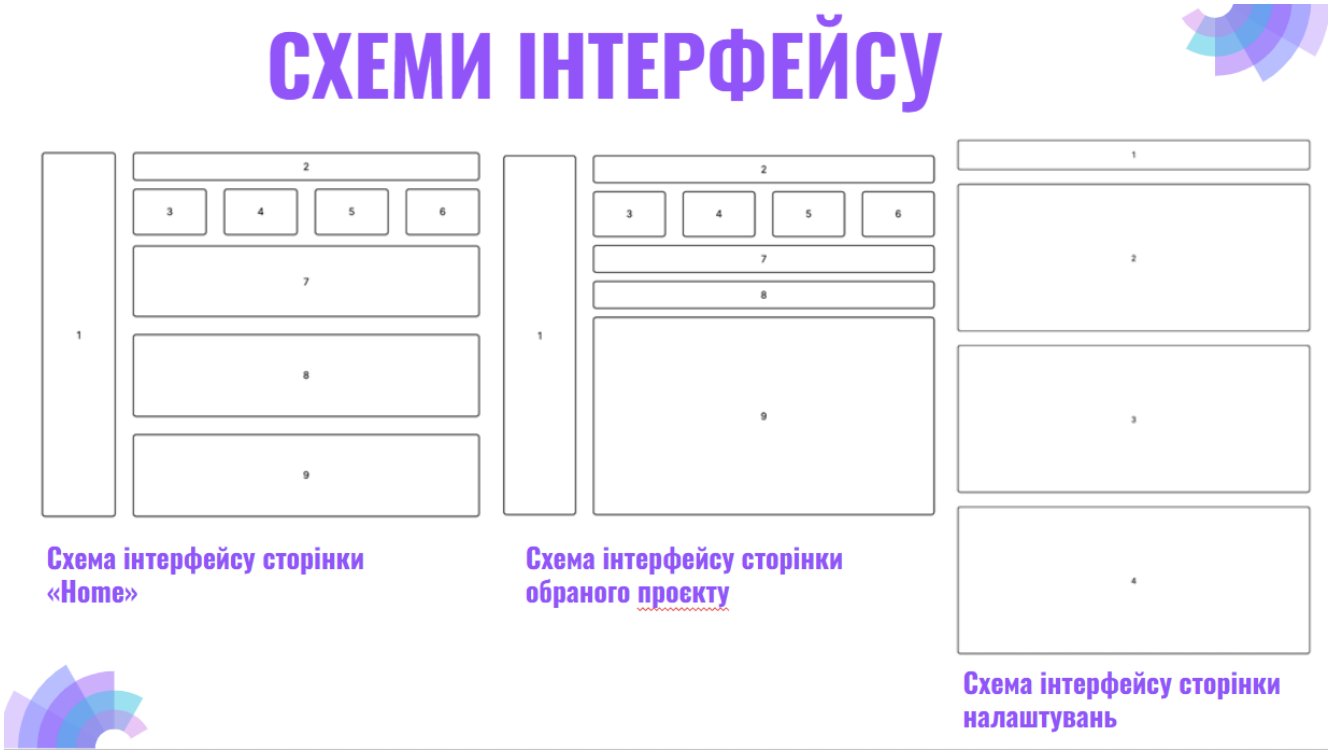


Рисунок Г.13 – Схеми інтерфейсу

Рисунок Г.14 – Використані технології

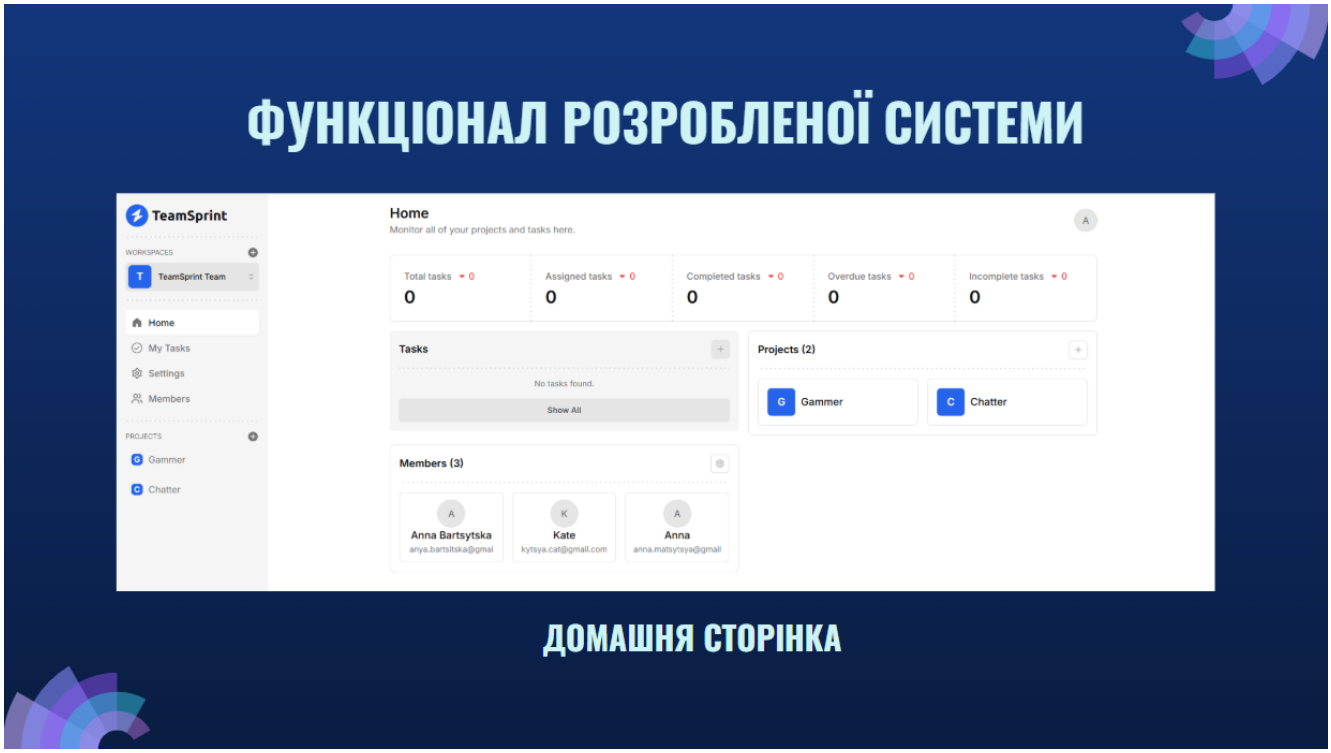


Рисунок Г.15 – Функціонал розробленої системи (домашня сторінка)

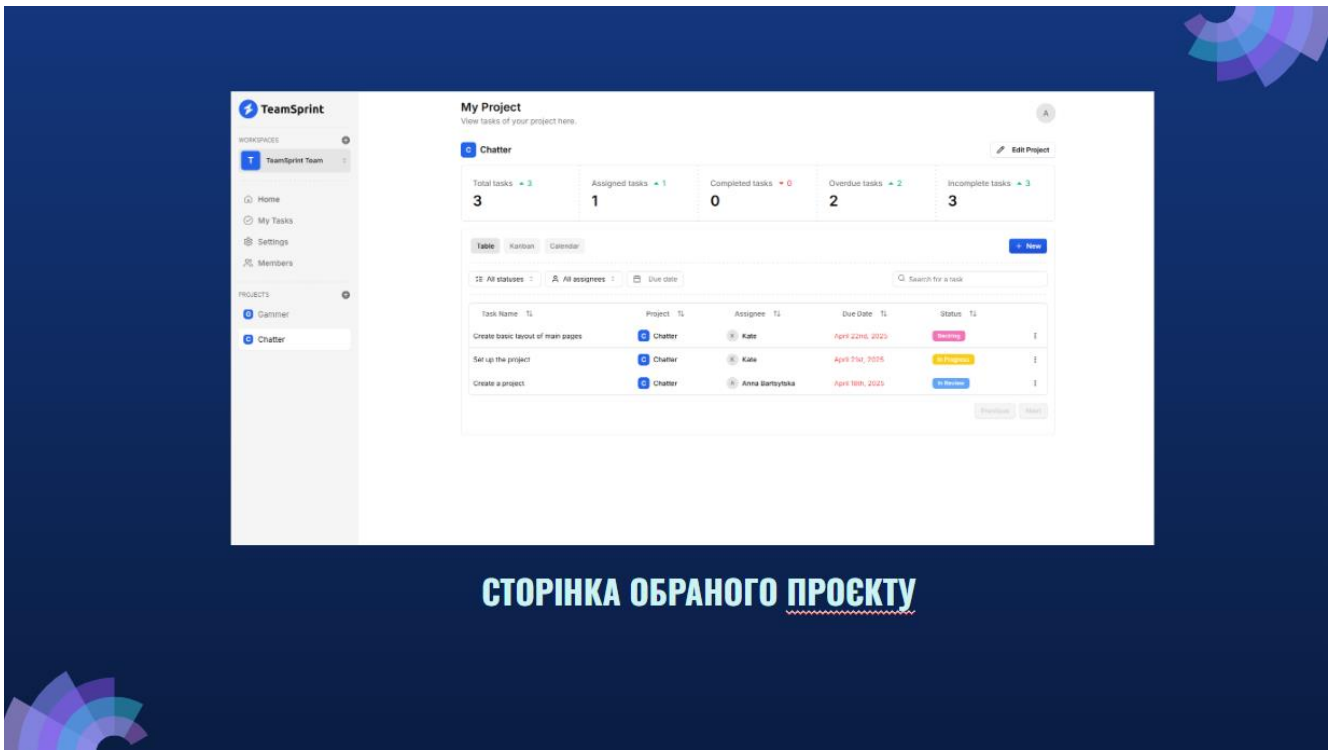


Рисунок Г.16 – Функціонал розробленої системи (обраний проєкт)

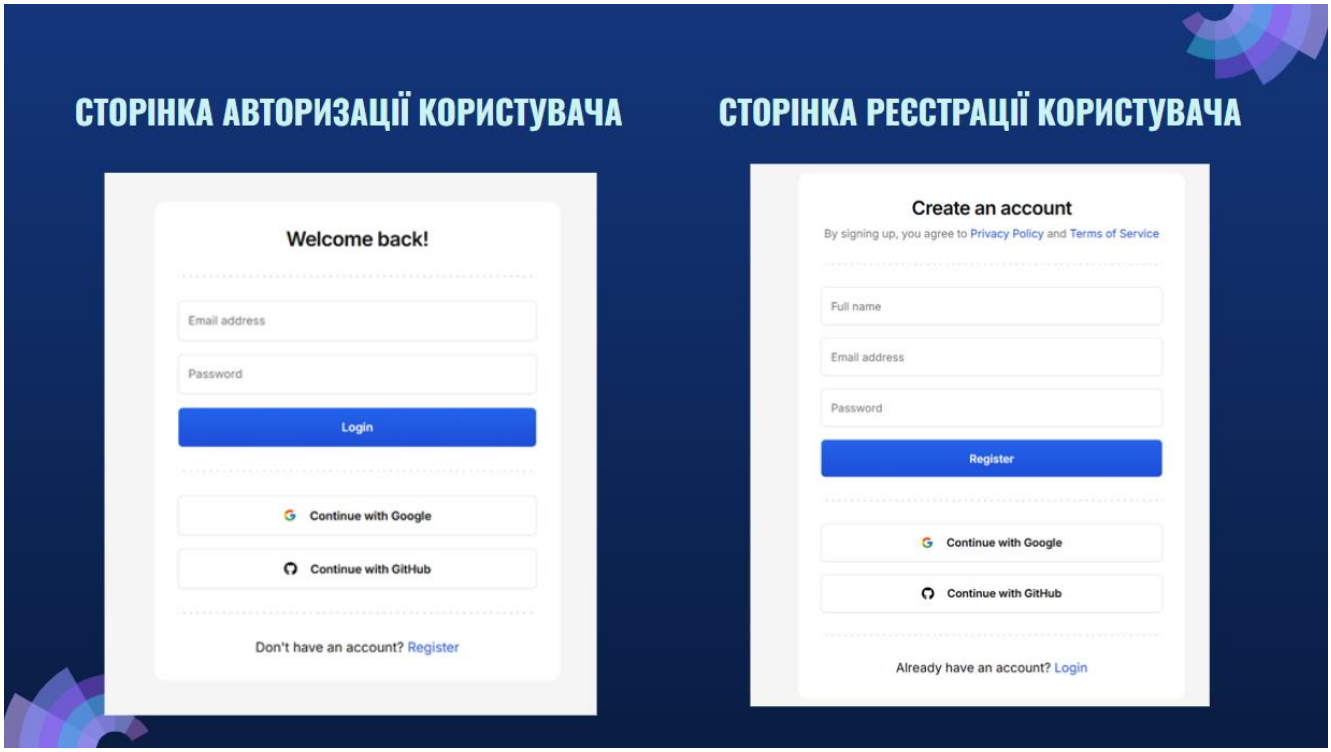


Рисунок Г.17 – Функціонал розробленої системи
(авторизація та реєстрація)

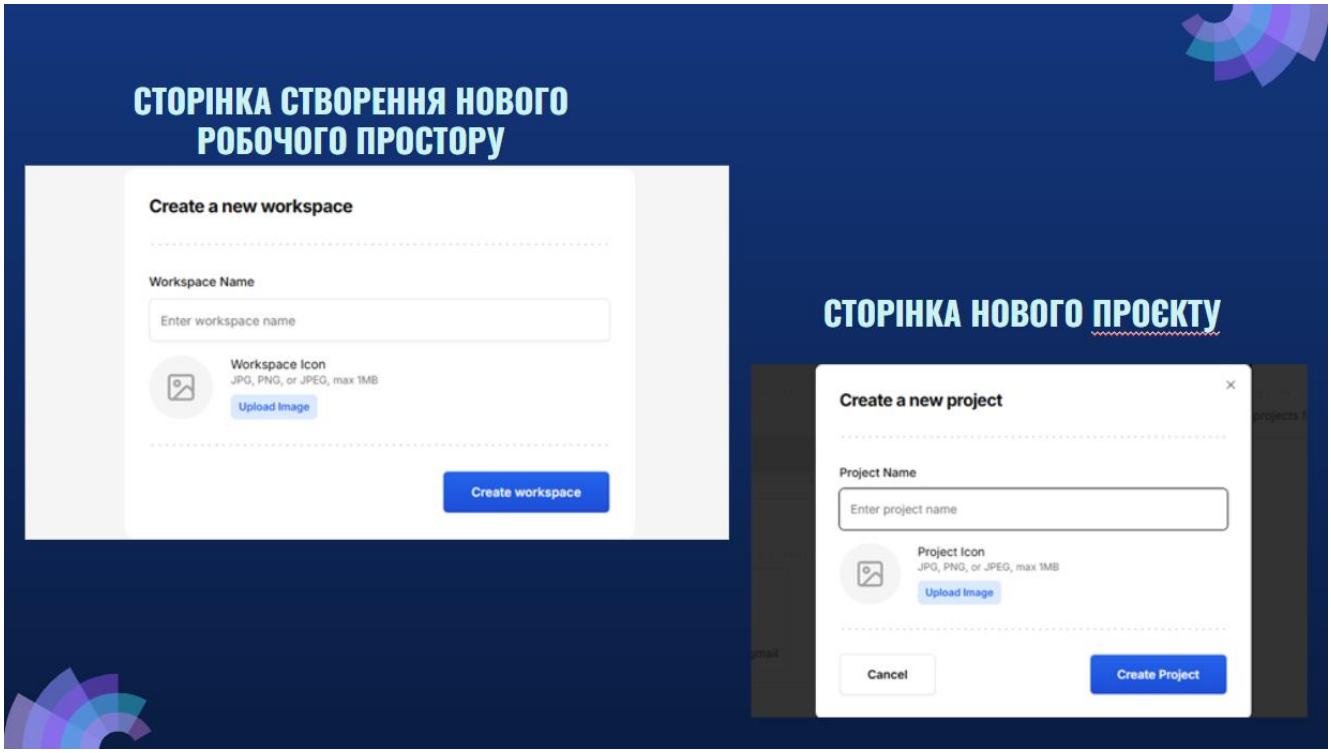


Рисунок Г.18 – Функціонал розробленої системи
(створення новго робочго простору та проєкту)

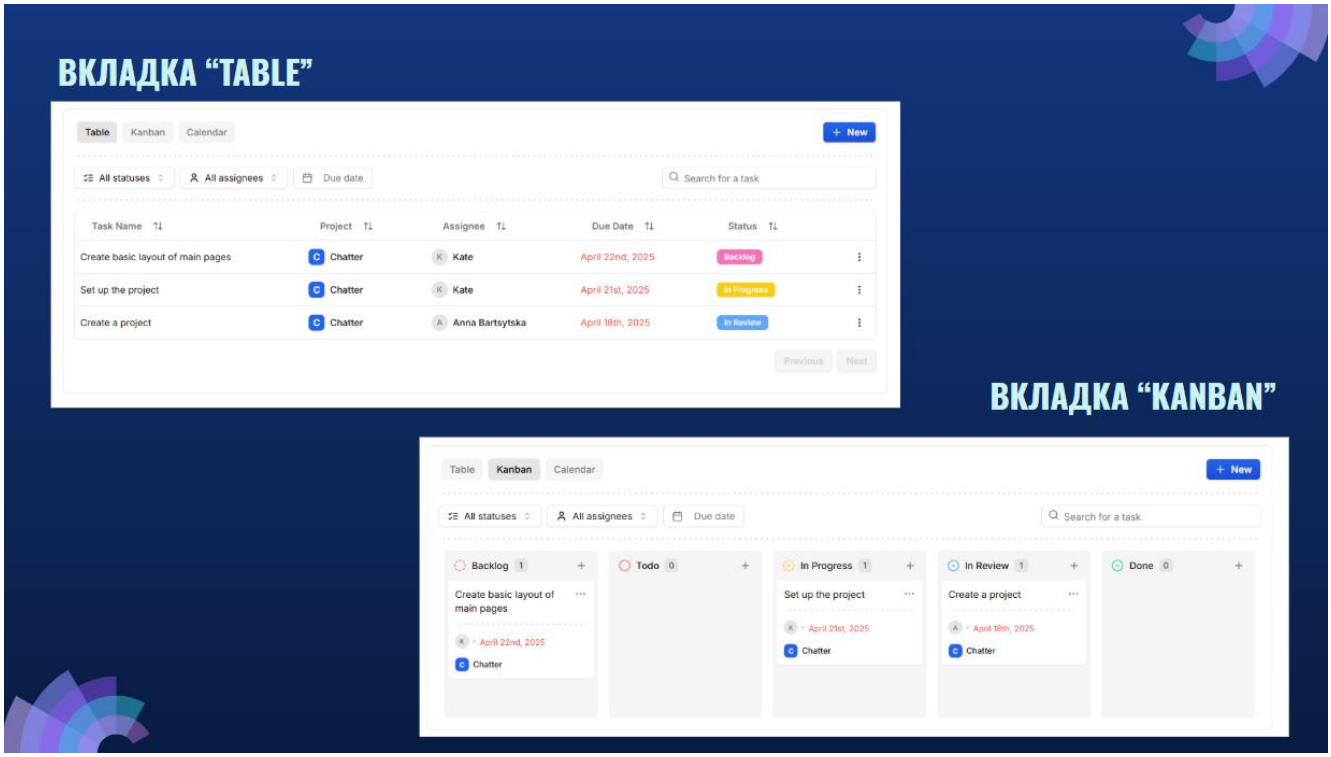


Рисунок Г.19 – Функціонал розробленої системи

(вкладки «Table» та «Kanban»)

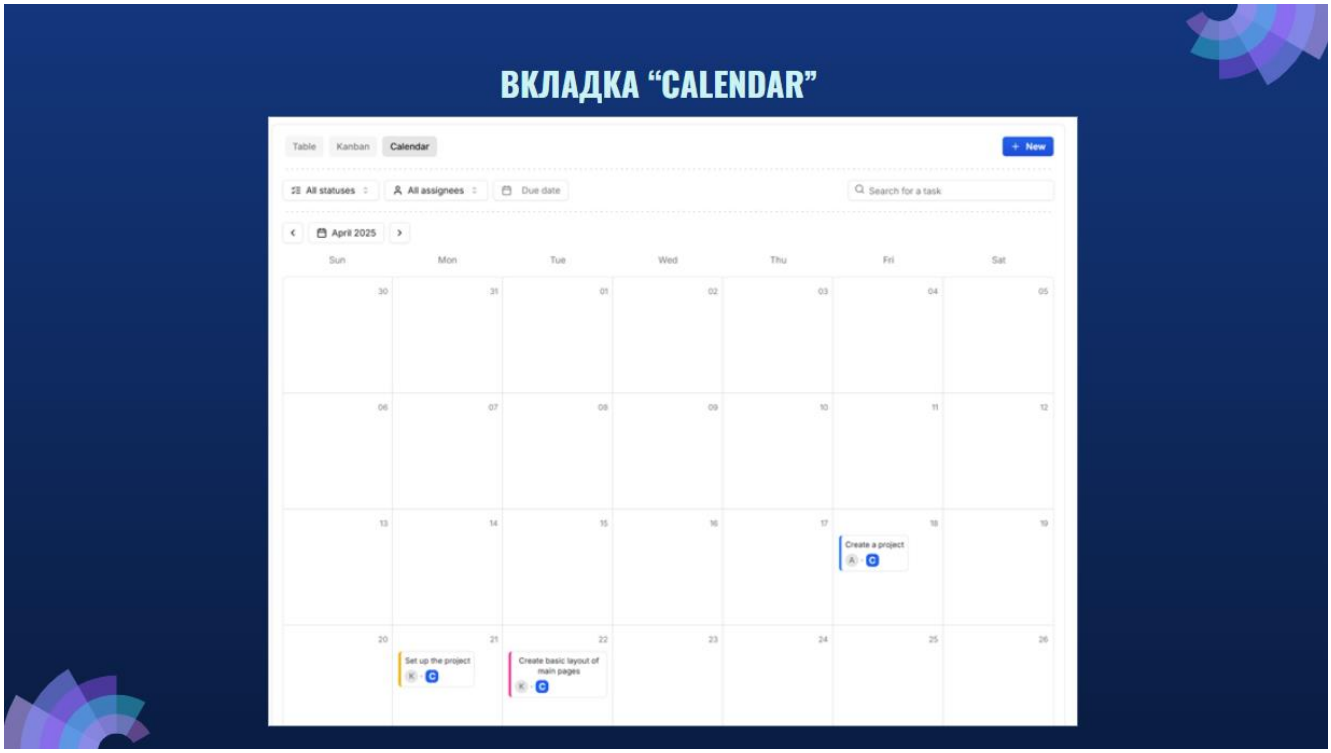


Рисунок Г.20 – Функціонал розробленої системи

(вкладки «Calendar»)

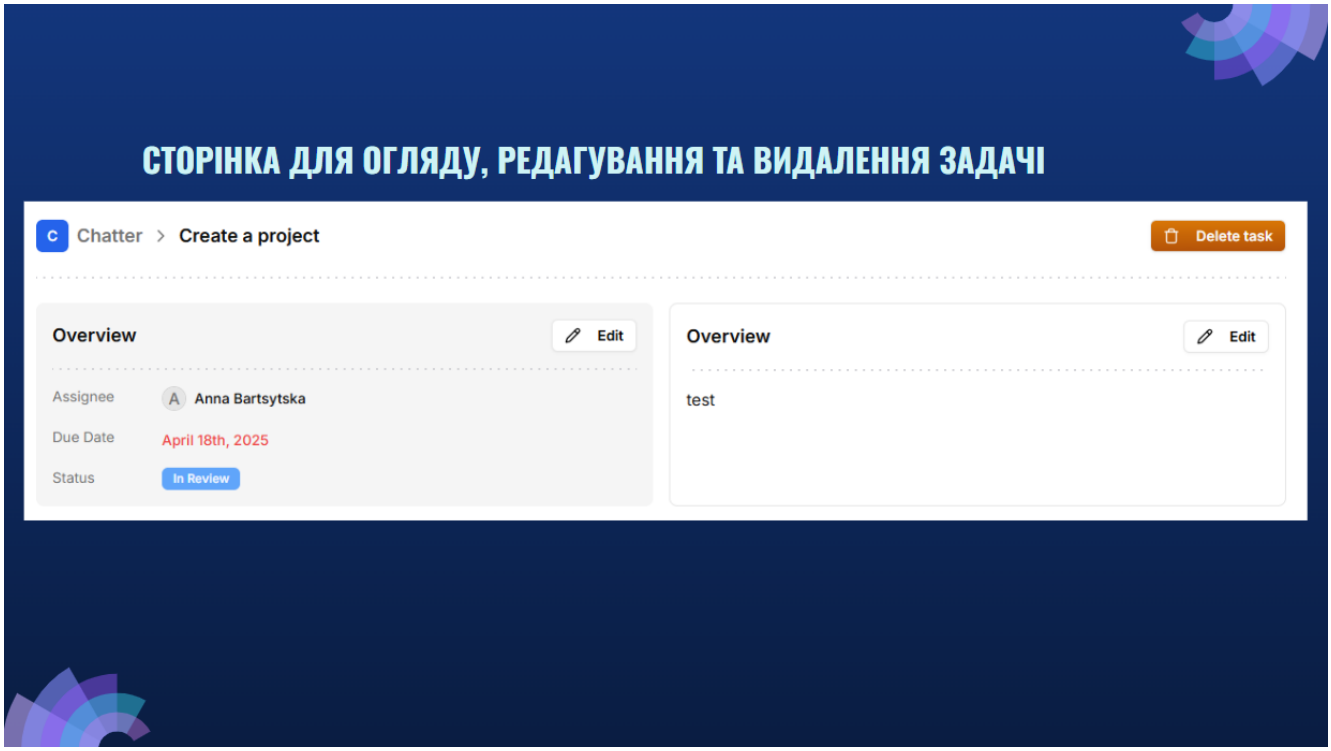


Рисунок Г.21 – Сторінка для огляду, редагування та видалення задачі

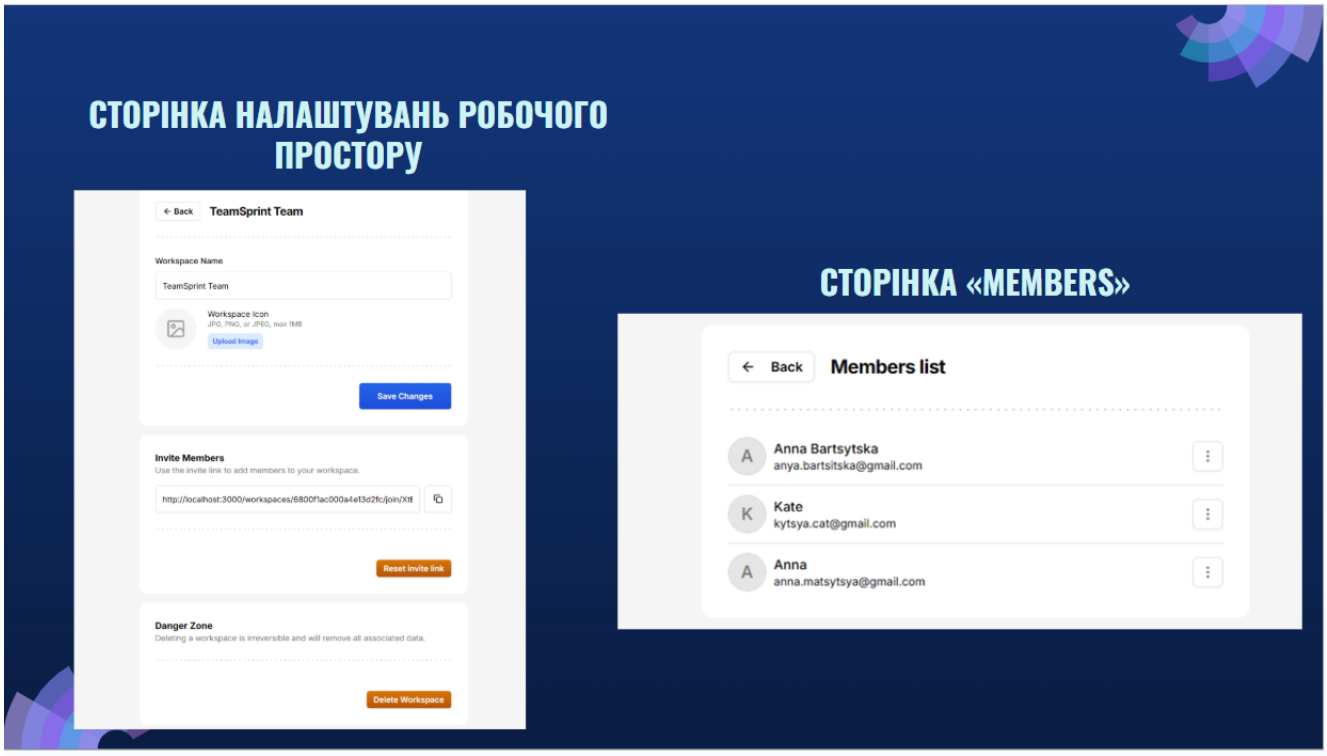


Рисунок Г.22 – Функціонал розробленої системи
(налаштування робочого простору та сторінка «Members»)

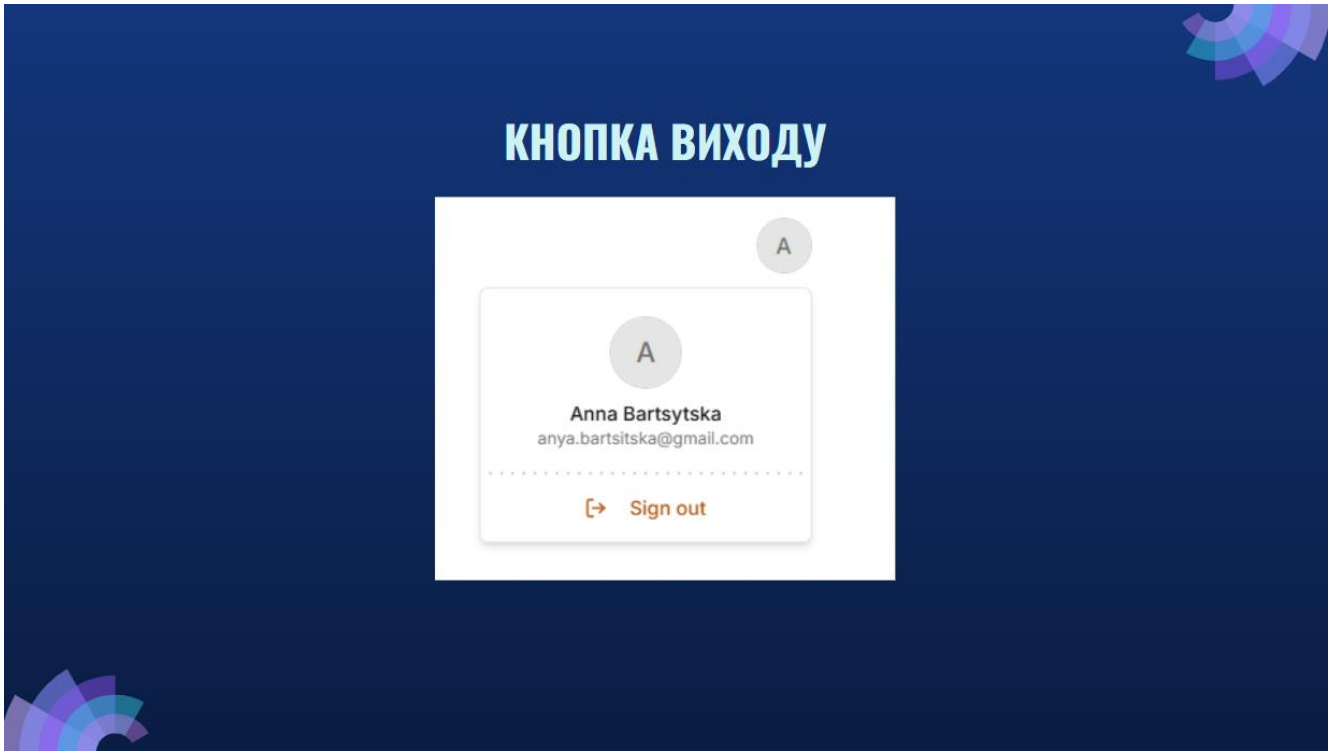


Рисунок Г.23 – Кнопка виходу із системи

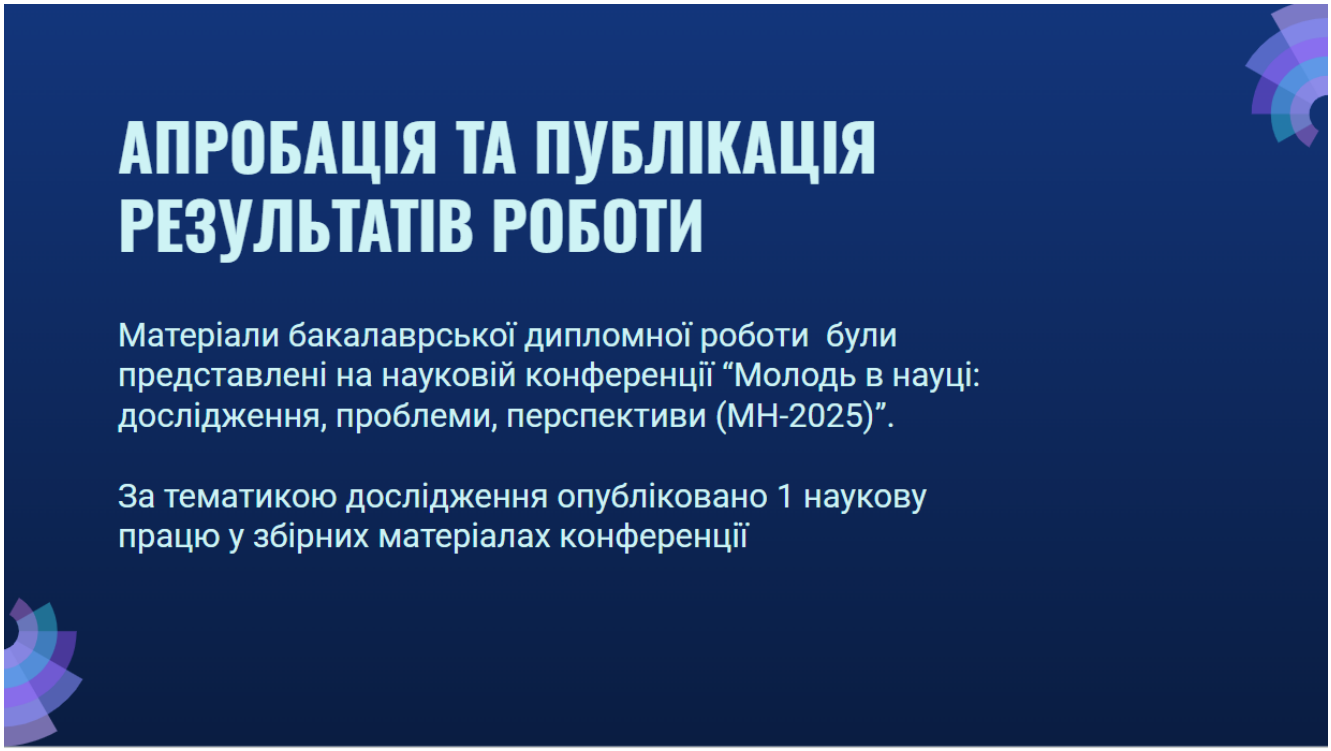


Рисунок Г.24 – Апробація та публікація результатів роботи

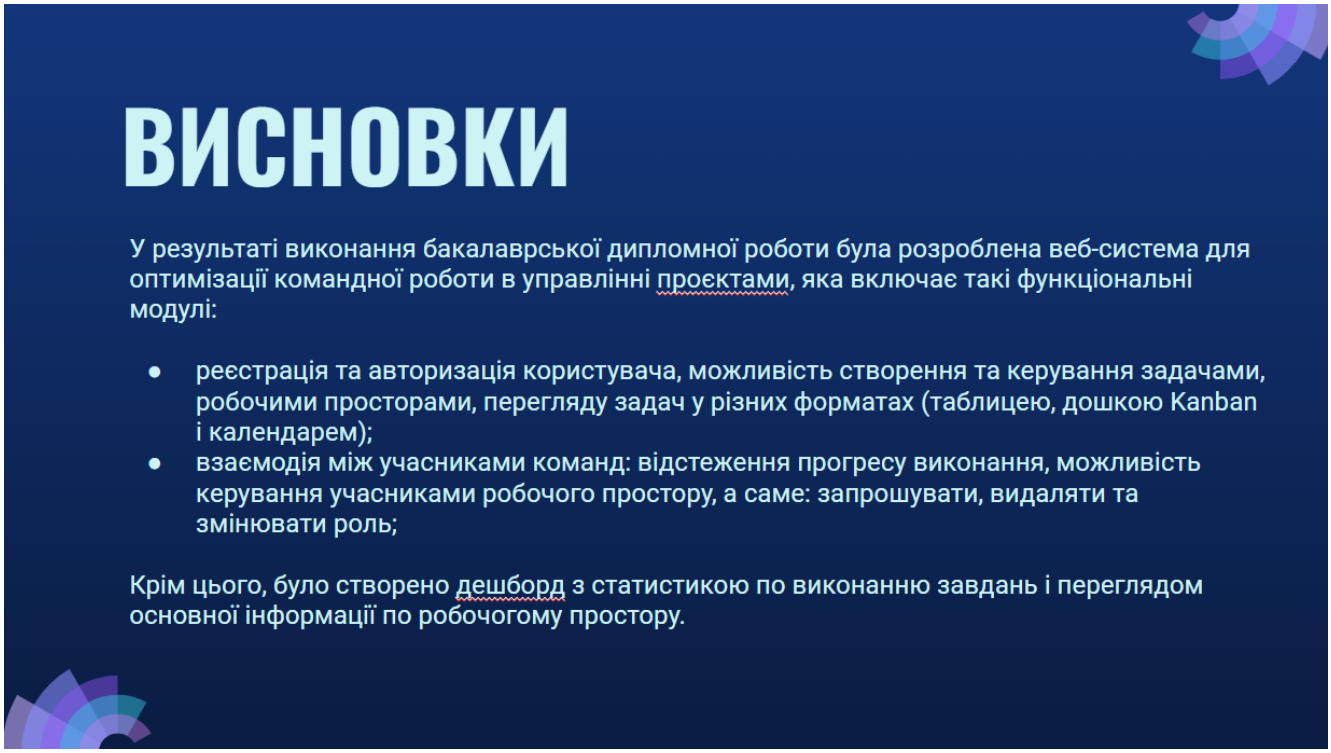


Рисунок Г.25 – Висновки

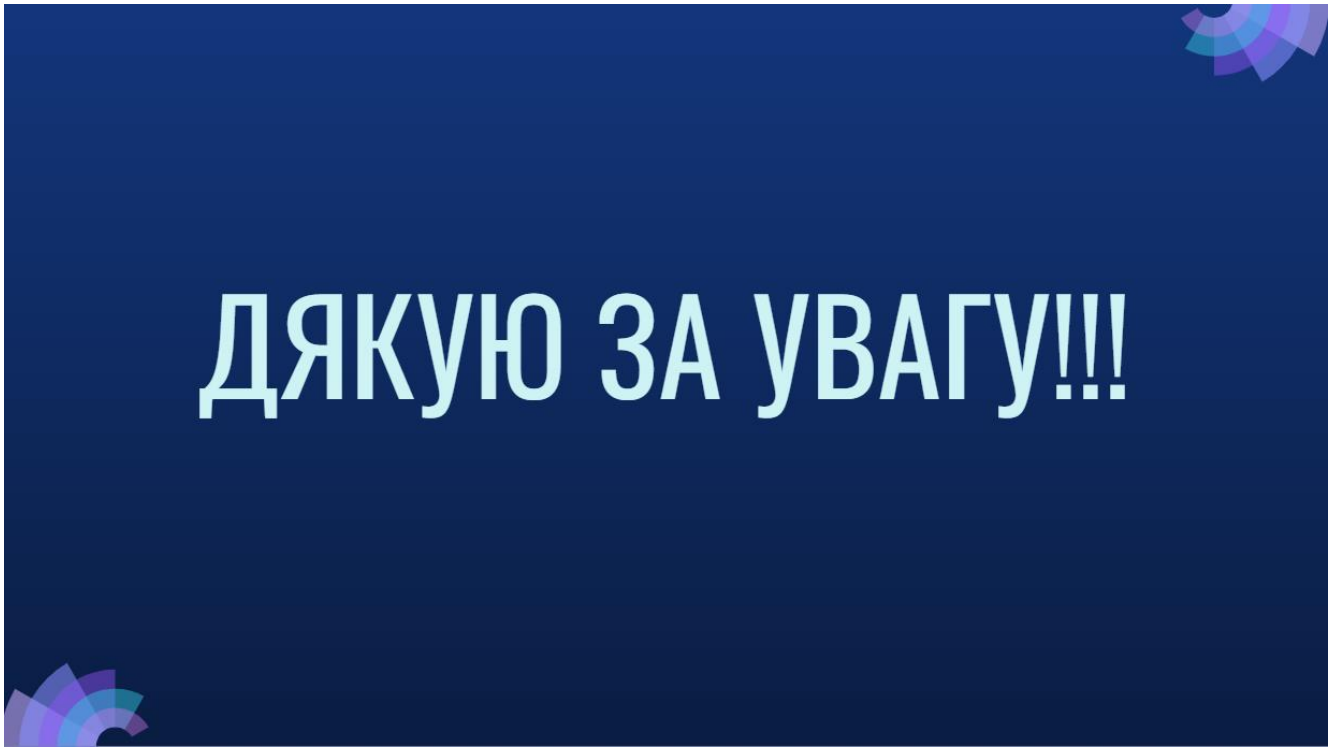


Рисунок Г.26 – Фінальний слайд