ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ПРОЄКТУВАННЯ ТА РОЗРОБКА МЕНЕДЖЕРА ЗАДАЧ

Виконав: здобувач першого (бакалаврського)
рівня вищої освіти
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"
_____ Шестаков Костянтин Олегович
(підпис)

Науковий керівник: ст. викл.
_____ Плеша Василь Іванович
(підпис)

Львів - 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Освітньо-професійна програма "Комп'ютерні науки"
Спеціальність 122 "Комп'ютерні науки"

"**ЗАТВЕРДЖУЮ**"
Завідувач кафедри комп'ютерних наук,
прикладної та вищої математики
_____ Новосад З. Г.
(підпис) (прізвище та ініціали)

13.10.2025 р.

ЗАВДАННЯ
до кваліфікаційної роботи здобувачу
Шестаков Костянтин Олегович

1. Тема кваліфікаційної роботи "Проєктування та розробка менеджера задач"; керівник кваліфікаційної роботи Плеша Василь Іванович, старший викладач; затверджена наказом по Університету ***№ 992/03 від 06 жовтня 2025 р.*** (заочн)
2. Термін здачі здобувачем вищої освіти кваліфікаційної роботи 12 червня 2026 р.
3. Вхідні дані до роботи: література, документація, матеріали та сайти Internet, що пов'язані з розробкою та менеджерами задач .
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
   1) Аналіз предметної області; 2) Проєктування застосунку та інструментальні засоби реалізації; 3) Розробка застосунку
5. Перелік графічного матеріалу до 12 слайдів у форматі PowerPoint
6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 13.10.2025 р.

Календарний план

№ з/п	Найменування етапів робіт	Термін виконання	Примітка
1.	Отримання завдання і аналіз інформаційних джерел за темою роботи	до 11.12.2025	
2.	Дослідження джерел	до 13.01.2026	
3.	Аналіз предметної області	до 07.02.2026	
4.	Проєктування застосунку та інструментальні засоби реалізації	до 01.04.2026	
5.	Розробка застосунку	до 15.04.2026	
6.	Тестування	до 22.04.2026	
7.	Оформлення кваліфікаційної роботи. Подання роботи на перевірку керівнику	до 16.05.2026	
8.	Доопрацювання кваліфікаційної роботи з врахуванням зауважень керівника. Подання роботи для перевірки на рівень запозичень	до 20.05.2026	
9.	Подання кваліфікаційної роботи керівнику для отримання відгуку	до 27.05.2026	
10.	Представлення кваліфікаційної роботи на зовнішнє рецензування. Подання кваліфікаційної роботи на кафедру комп'ютерних наук	до 5.06.2026	
11.	Попередній захист кваліфікаційної роботи	05.06.2026 - 06.06.2026	
12.	Захист кваліфікаційної роботи на засіданні екзаменаційної комісії	15.06.2026 - 19.06.2026 (згідно графіку роботи ЕК)	

Здобувач вищої освіти _____ Шестаков К. О.

Керівник роботи _____ Плеша В. І.

Реферат

Робота складається із вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. Основний зміст роботи викладено на 46 сторінках і містить 8 рисунків, 1 таблицю, 3 додатки.

Об'єктом дослідження є процес управління проєктами, розподілу завдань та моніторингу їх виконання в межах командної роботи.

Предметом дослідження є програмні засоби, алгоритми, методи та вебтехнології (зокрема React, TypeScript та архітектура REST API), що використовуються для розробки клієнт-серверних систем управління завданнями на базі методології Kanban.

Метою дослідження є проєктування та розробка вебзастосунку менеджера задач (Task Manager) з інтерактивною Kanban-дошкою та системою рольового доступу, що забезпечує підвищення ефективності командної роботи, зручність управління проєктами та надійність збереження даних.

В першому розділі розглянуто теоретичні основи предметної області, досліджено еволюцію підходів до управління завданнями та принципи методології Kanban, а також проведено порівняльний аналіз існуючих рішень (Trello, Jira, Miro).

В другому розділі обґрунтовано вибір технологічного стеку, присвячено увагу проєктуванню модульної архітектури застосунку, розробці нереляційної схеми бази даних (JSON) та проєктуванню адаптивного користувацького інтерфейсу.

В третьому розділі детально описано процес програмної реалізації всіх модулів, імплементацію бізнес-логіки управління станом за допомогою Zustand, налаштування захищених маршрутів (RBAC), реалізацію інтерактивної Kanban-дошки з підтримкою механіки Drag-and-Drop та розгортання готового продукту у хмарному середовищі.

Перелік ключових слів: менеджер задач, Kanban-дошка, React, TypeScript, Zustand, REST API, json-server, контроль доступу на основі ролей (RBAC).

Abstract

The thesis consists of an introduction, three main chapters, conclusions, a list of references, and appendices. The main body of the thesis comprises 46 pages and includes 8 figures, 1 table, and 3 appendices.

The object of the study is the process of project management, task allocation, and monitoring of task execution within collaborative teamwork.

The subject of the study is software tools, algorithms, methods, and web technologies (including React, TypeScript, and REST API architecture) used for the development of client-server task management systems based on the Kanban methodology.

The purpose of the study is to design and develop a Task Manager web application with an interactive Kanban board and a role-based access control system, aimed at improving team collaboration efficiency, enhancing project management convenience, and ensuring reliable data storage.

The first chapter examines the theoretical foundations of the subject area, explores the evolution of task management approaches and the principles of the Kanban methodology, and provides a comparative analysis of existing solutions (Trello, Jira, and Miro).

The second chapter justifies the selection of the technology stack, focuses on the design of the application's modular architecture, the development of a non-relational database schema (JSON), and the design of a responsive user interface.

The third chapter provides a detailed description of the software implementation process for all modules, including the implementation of state management business logic using Zustand, the configuration of protected routes based on Role-Based Access Control (RBAC), the development of an interactive Kanban board with Drag-and-Drop functionality, and the deployment of the finished product in a cloud environment.

Keywords: task manager, Kanban board, React, TypeScript, Zustand, REST API, json-server, Role-Based Access Control (RBAC).

Зміст

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Дослідження предметної області та визначення ключових сутностей менеджера задач 11
1.2 Задачі програмної реалізації менеджера задач та сфера її застосування ... 16
1.3 Аналіз існуючих рішень 18
2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РЕАЛІЗАЦІЇ 23
2.1 Мова програмування та вибір середовища для реалізації 25
2.2 Інструменти реалізації бази даних 27
2.3 Інші допоміжні інструменти 28
2.4. Проєктування застосунку та бази даних для менеджера задач 31
3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ МЕНЕДЖЕРА ЗАДАЧ 35
3.1 Реалізація бази даних 36
3.2 Реалізація застосунку для менеджера задач 38
3.3 Огляд використання менеджера задач 40
ВИСНОВКИ 45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 47
Додатки 49

ВСТУП

У сучасному світі інформаційних технологій ефективна організація робочого процесу та управління завданнями відіграють ключову роль в успішності будь-якого проєкту. З переходом багатьох компаній на віддалений та гібридний формати роботи, а також із глобалізацією ринку праці, потреба в надійних, швидких та зручних системах колаборації зросла експоненціально. Особливе значення набувають системи управління проєктами (Task Managers), які здатні забезпечити прозорість процесів, чіткий розподіл обов'язків та моніторинг виконання завдань у реальному часі. Успіх командної роботи безпосередньо залежить від інструментарію, що використовується, що створює підвищений попит на вебзастосунки, здатні оптимізувати ці процеси.

Одним із найефективніших сучасних підходів до візуалізації робочого процесу є методологія Kanban. Вона дозволяє розбити складні проєкти на невеликі, керовані задачі та відстежувати їхній рух по етапах виконання. У поєднанні із сучасними технологіями розробки односторінкових застосунків (Single Page Applications, SPA) на базі бібліотеки React, методологія Kanban дозволяє створювати високоінтерактивні інтерфейси. Використання таких архітектурних підходів забезпечує:

- високу швидкість взаємодії користувача з інтерфейсом без перезавантаження сторінок;
- гнучкість управління станом застосунку завдяки сучасним менеджерам стану (наприклад, Zustand);
- надійність та передбачуваність коду за рахунок суворої типізації за допомогою TypeScript.

Актуальність цієї теми підтверджується загальними тенденціями цифровізації бізнесу. Існуючі на ринку рішення (такі як Jira або Trello) часто є крос-компромісними: вони або занадто перевантажені складним функціоналом, що вимагає тривалого навчання персоналу, або, навпаки, занадто прості й не забезпечують належного рівня безпеки та розмежування прав доступу (Role-Based Access Control — RBAC). Розробка власного кастомізованого рішення

дозволяє уникнути цих недоліків, створивши продукт, що ідеально балансує між зручністю (з використанням механік Drag-and-Drop) та суворою бізнес-логікою управління ролями.

Крім того, сучасні інвестори, власники бізнесу та тімліди очікують від систем не лише стабільної роботи бази даних, але й сучасного, адаптивного користувацького інтерфейсу (UI/UX). Використання utility-first CSS фреймворків, таких як Tailwind CSS у поєднанні з Flowbite, дозволяє створити масштабовану дизайн-систему, яку можна легко адаптувати під брендбук будь-якого замовника.

Таким чином, розробка застосунку для управління проєктами є актуальним і необхідним завданням, яке відповідає потребам сучасного ринку розробки програмного забезпечення.

Об’єктом дослідження є процес управління проєктами, розподілу завдань та моніторингу їх виконання в межах командної роботи.

Предметом дослідження є програмні засоби, алгоритми, методи та вебтехнології (зокрема React, TypeScript та архітектура REST API), що використовуються для розробки клієнт-серверних систем управління завданнями на базі методології Kanban.

Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку менеджера задач (Task Manager) з інтерактивною Kanban-дошкою та системою рольового доступу, що забезпечує підвищення ефективності командної роботи, зручність управління проєктами та надійність збереження даних.

Завданням кваліфікаційної роботи є:

1. Провести ґрунтовний аналіз предметної області управління завданнями, дослідити існуючі методології (зокрема Kanban) та визначити ключові сутності системи.
2. Виконати порівняльний аналіз існуючих на ринку систем управління проєктами (Trello, Jira, Miro) для виявлення їхніх переваг та недоліків.

3. Сформувати чіткі вимоги до функціональності, безпеки та користувацького інтерфейсу застосунку.

4. Обґрунтувати вибір технологічного стеку для клієнтської та серверної частин (React, TypeScript, Zustand, Tailwind CSS, json-server).

5. Спроєктувати архітектуру бази даних та ієрархію React-компонентів.

6. Реалізувати основний функціонал застосунку: автентифікацію, CRUD-операції для проєктів та користувачів, рольову модель доступу (Admin, Project Manager, Team Member) та інтерактивну дошку з підтримкою Drag-and-Drop.

7. Забезпечити розгортання (деплой) розробленого застосунку та його серверної частини у хмарному середовищі (Render).

Теоретичне значення. Робота систематизує та розширює знання щодо застосування сучасних патернів проєктування фронтенд-застосунків. Зокрема, досліджено ефективність використання легкої бібліотеки Zustand як альтернативи традиційному Redux для управління глобальним станом, а також проаналізовано підходи до реалізації складних інтерактивних елементів (Drag-and-Drop) у суворо типізованому середовищі TypeScript.

Практичне значення. Розроблений прототип є повністю функціональним продуктом, який може бути впроваджений у реальні робочі процеси невеликих IT-команд, стартапів або фрілансерів. Гнучкість Kanban-дошки дозволяє адаптувати її не лише для циклу розробки програмного забезпечення, але й для виробничих підприємств, де процеси поділені на чіткі конвеєрні етапи (від підготовки та сортування матеріалів до фінального збирання букетів та пакування продукції). Крім того, винесена в окремі SCSS-файли конфігурація стилів робить застосунок готовим до швидкого редизайну під потреби конкретного бізнесу. Версія застосунку успішно розгорнута на платформі Render: https://task-manager-frontend-okw6.onrender.com/#/login; сам проєкт розміщений на платформі Github: https://github.com/s-kostiantyn/task-manager-frontend/tree/main та https://github.com/s-kostiantyn/task-manager-backend

Структура роботи. Робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи предметної області, проведено аналіз існуючих рішень та визначено головні завдання системи. Другий розділ присвячений проєктуванню архітектури застосунку, вибору інструментальних засобів (React, TypeScript, бази даних) та розробці користувацького інтерфейсу. У третьому розділі детально описано процес програмної реалізації всіх модулів, імплементацію бізнес-логіки, розмежування прав доступу та розгортання готового продукту на сервері. У висновках підведено підсумки дослідження та наведено перспективи подальшого масштабування застосунку.

У процесі дослідження та проєктування вебзастосунку було опрацьовано низку наукових, технічних джерел та офіційних документацій, які формують теоретичну і практичну базу сучасної веброзробки. Основним джерелом для розуміння принципів побудови користувацьких інтерфейсів стала офіційна документація екосистеми React [11]. Вона детально описує життєвий цикл компонентів, використання хуків (Hooks) та принципи рендерингу, що стало основою для побудови архітектури проєкту. Додатково було досліджено документацію мови TypeScript, яка визначає найкращі практики суворої типізації даних у JavaScript-додатках, що є критично важливим для уникнення помилок на етапі виконання. Для розуміння принципів візуалізації робочого процесу було опрацьовано праці, присвячені гнучким методологіям розробки (Agile) та безпосередньо Kanban. Зокрема, досліджено принципи обмеження кількості незавершеної роботи (WIP - Work in Progress) та візуального менеджменту, які лягли в основу логіки роботи розробленої Kanban-дошки. Значну увагу також було приділено сучасним підходам у UI/UX дизайні, що описуються в документаціях CSS-фреймворку Tailwind CSS та бібліотеки компонентів Flowbite. Вони розкривають концепцію utility-first дизайну, що дозволяє створювати складні та адаптивні інтерфейси без написання великих обсягів кастомного CSS-коду.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження предметної області та визначення ключових сутностей менеджера задач

Теоретичні основи управління задачами

Управління проєктами та задачами (Task Management) є фундаментальною складовою організації будь-якого сучасного робочого процесу. Історично підходи до управління роботою еволюціонували від простих паперових списків справ (To-Do lists) та лінійних діаграм Ганта до складних ітеративних методологій. У класичних підходах (наприклад, Waterfall або "Каскадна модель") управління задачами було жорстко регламентованим: кожен етап мав завершитися до початку наступного. Проте в умовах сучасної динамічної економіки та швидкої зміни ринкових вимог такий підхід виявився неефективним. Це призвело до популяризації сімейства гнучких методологій розробки (Agile), які ставлять на перше місце адаптивність, комунікацію та візуалізацію процесів.

Однією з найефективніших систем управління робочим процесом, яка ідеально вписується в парадигму Agile, є методологія Kanban. Слово "Kanban" (яп. 看板) буквально перекладається як "видима дошка" або "сигнальна картка". Ця концепція була вперше розроблена та впроваджена інженером Таїті Оно в корпорації Toyota у 1950-х роках для оптимізації виробничих процесів та реалізації принципу "точно в строк" (Just-in-Time). Початково Kanban використовувався виключно на заводах: працівники передавали фізичні картки між цехами, сигналізуючи про потребу в нових деталях [1].

З розвитком інформаційних технологій Девід Андерсон адаптував цю концепцію для сфери розробки програмного забезпечення. У цифровому вигляді Kanban-дошка являє собою простір, логічно розділений на вертикальні колонки. Кожна колонка відповідає певному статусу життєвого циклу задачі. Картки задач, що містять інформацію про необхідну роботу, переміщуються між цими колонками зліва направо, що дозволяє візуалізувати так званий "потік створення

цінності" (Value Stream).

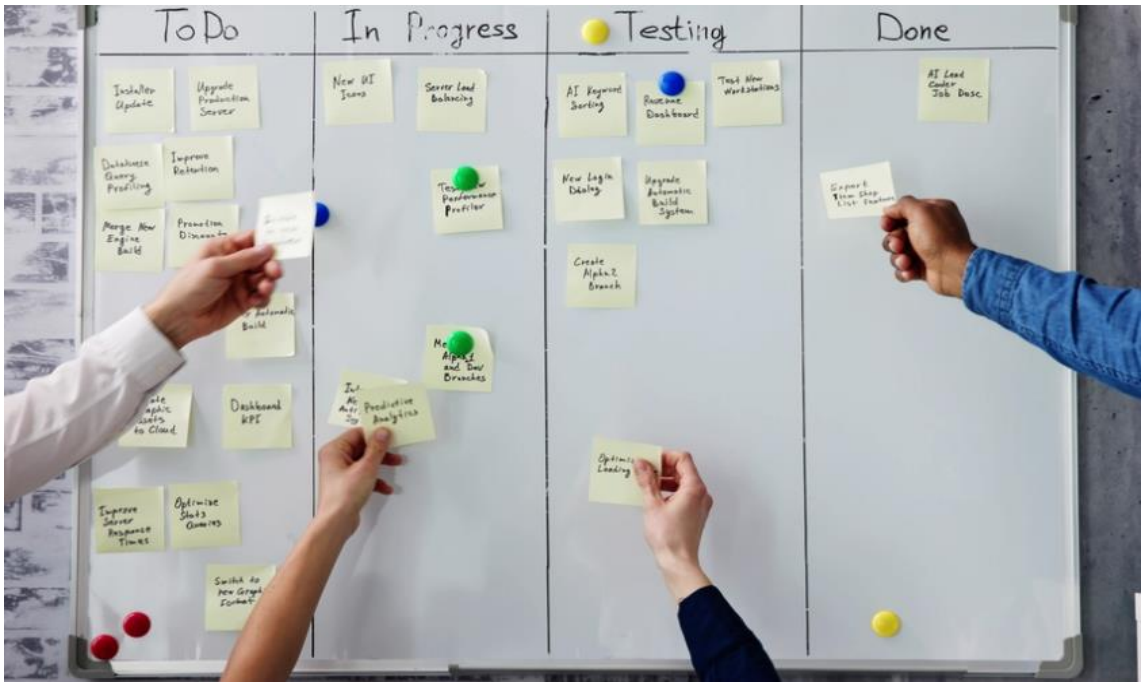


Рисунок 1.1 - Приклад Kanban-дошки

Ключовими принципами застосування методології Kanban у програмних застосунках є:

1. **Візуалізація робочого процесу:** Кожен учасник системи повинен мати можливість у будь-який момент часу побачити, над чим працює команда, які завдання заплановані, а які вже завершені.

2. **Обмеження незавершеної роботи (Limit Work in Progress - WIP):** Система стимулює команди не розпочинати нові завдання доти, доки не будуть завершені поточні. Це запобігає розпорошенню уваги та перевантаженню виконавців.

3. **Управління потоком (Manage Flow):** Аналіз швидкості проходження карток через колонки дозволяє виявляти "вузькі місця" (bottlenecks) у робочому процесі.

4. **Зрозумілі правила та критерії (Make Policies Explicit):** Кожен статус (колонка) має чіткі критерії того, коли задача може бути до нього переміщена.

Адаптація предметної області до потреб бізнесу

Проєктування інформаційної системи вимагає чіткого розуміння того, як

теоретична модель буде використовуватися на практиці. Розроблюваний Task Manager спроєктовано таким чином, щоб задовольнити потреби не лише класичних IT-команд (де розробники, тестувальники та спеціалісти з UI/UX дизайну передають завдання етапами), але й фахівців інших галузей.

Універсальність візуального трекінгу дозволяє застосовувати застосунок у процесах з чітко вираженою конвеєрною структурою. Наприклад, у сфері флористичного виробництва та тепличних господарств управління процесами має критичне значення через короткий термін придатності продукції. Використання електронної Kanban-дошки дозволяє керівникам виробництва візуалізувати кожен етап: від первинного сортування та нарізки троянд до збирання готових букетів і підготовки їх до логістичного відвантаження. Працівники цеху (Team Members) можуть відзначати зміну статусу кожної партії в системі, тоді як керівники (Project Managers) відстежують загальну ефективність зміни.

Визначення ключових сутностей системи

Для побудови реляційної (або в даному випадку псевдореляційної, на базі JSON) моделі даних необхідно виділити базові сутності предметної області, їхні атрибути та зв'язки. У розроблюваній системі управління завданнями виокремлено три фундаментальні сутності, які формують ядро бізнес-логіки:

1. Сутність "Проєкт" (Project) Проєкт є глобальним контекстом та макро-контейнером для всіх інших елементів. З точки зору архітектури бази даних, проєкт об'єднує метадані про ініціативу та встановлює зв'язки з виконавцями і конкретними кроками реалізації. Атрибути сутності "Проєкт" включають:

- id (унікальний ідентифікатор типу ID) — первинний ключ для точної ідентифікації проєкту в базі даних.
- title (рядок) — коротка та ємна назва проєкту.
- description (рядок) — розгорнутий опис цілей та вимог до проєкту.
- date та dateEnd (рядки у форматі дати ISO) — хронологічні межі проєкту, які дозволяють системі контролювати терміни (дедлайни) та в перспективі реалізувати механізми сповіщень про прострочені завдання.

- status (перелічуваний тип Statuses) — поточний глобальний стан усього проєкту (наприклад, активний, на паузі, завершений).
- peoples (масив об'єктів People) — реалізація зв'язку "багато до багатьох" (Many-to-Many). Цей масив визначає, хто з зареєстрованих користувачів має доступ до читання або модифікації даного проєкту.
- tasks (масив об'єктів Task) — реалізація зв'язку "один до багатьох" (One-to-Many). Вкладення задач безпосередньо у структуру проєкту в форматі JSON є архітектурним рішенням, яке зменшує кількість мережевих запитів під час рендерингу сторінки Kanban-дошки.


```
{
  "projects": [
    {
      "id": 1,
      "title": "Projec1",
      "description": "Some desc",
      "status": "Planned",
      "peoples": [],
      "tasks": [],
      "date": "2025-04-28",
      "dateEnd": "2025-04-30"
    },
  ],
  "peoples": [
    {
      "id": 0,
      "firstName": "admin",
      "lastName": "admin",
      "login": "admin",
      "password": "admin",
      "role": "Admin",
      "projects": [],
      "task": []
    },
  ]
}
```


2. Сутність "Користувач" (People / User) Система управління проєктами не може існувати без ідентифікації суб'єктів, які виконують роботу. Сутність користувача забезпечує не лише авторизацію в системі, але й реалізацію політик безпеки на основі ролей (Role-Based Access Control — RBAC). Атрибути сутності:

- id (унікальний ідентифікатор).
- firstName та lastName (рядки) — персональні дані для відображення в інтерфейсі (наприклад, на картках задач).
- login та password (рядки) — облікові дані для автентифікації. У

промислових масштабах паролі обов'язково підлягають хешуванню (наприклад, за допомогою алгоритму bcrypt), проте на етапі прототипування через json-server вони можуть зберігатися у відкритому вигляді для зручності тестування.

- role (перелічуваний тип Roles) — ключовий атрибут бізнес-логіки. Може набувати значень Admin, Project Manager або Team Member. Саме цей атрибут перевіряється клієнтським застосунком (через глобальний стан Zustand) перед тим, як надати доступ до певних маршрутів (Routes) або показати кнопки редагування/видалення.

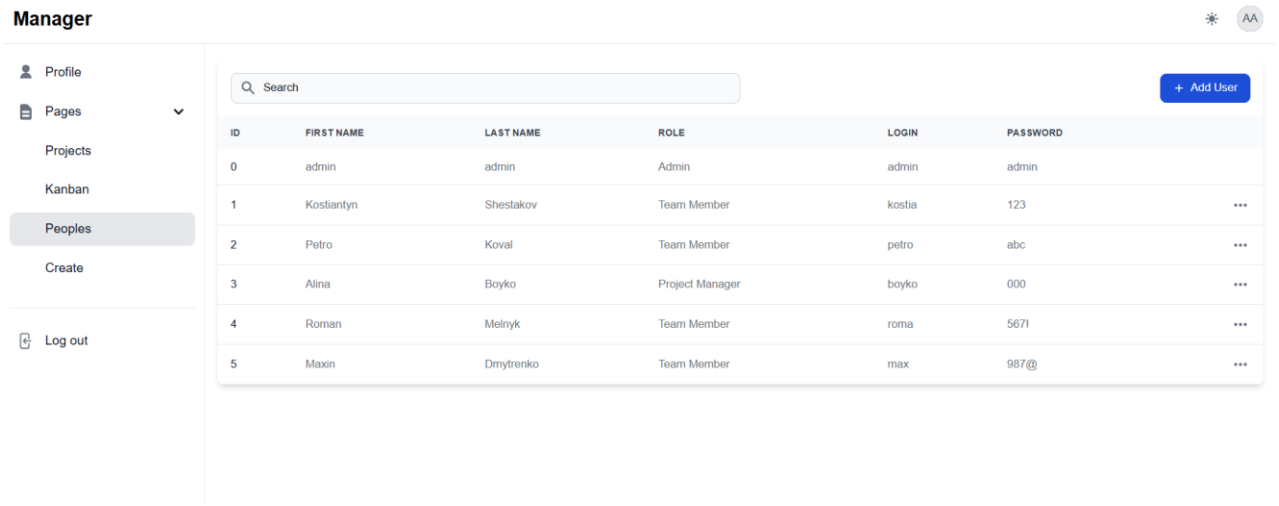


Рисунок 1.2 - Користувачі

3. Сутність "Задача" (Task) Задача є найменшою неподільною одиницею (атомом) в екосистемі розробленого застосунку. Робота із задачами становить близько 80% усього часу взаємодії користувача з інтерфейсом системи. Головним атрибутом задачі, окрім її назви та ідентифікатора виконавця, є status (статус). Зміна цього статусу є тригером для оновлення візуального представлення дошки. Завдяки імплементації бібліотеки @hello-pangea/dnd, процес зміни статусу відбувається максимально природно для користувача: картка фізично перетягується вказівником миші. При відпусканні картки в новій колонці (подія onDragEnd), система зчитує новий статус, моментально оновлює локальний стан та асинхронно відправляє оновлений об'єкт задачі на бекенд для постійного збереження.

Коректне виділення та проєктування цих трьох сутностей на етапі аналізу

предметної області створило надійний фундамент для подальшої розробки бази даних та архітектури React-компонентів.

1.2 Задачі програмної реалізації менеджера задач та сфера її застосування

Основні задачі програмної реалізації

Розробка сучасного вебзастосунку для управління проєктами вимагає вирішення комплексу інженерних, архітектурних та дизайнерських завдань. Головною метою програмної реалізації є створення надійного мосту між базою даних (де зберігається інформація) та кінцевим користувачем (який з нею взаємодіє), забезпечуючи при цьому високу швидкість відгуку та безпеку даних.

Для досягнення цієї мети в межах дипломного проєкту було визначено та реалізовано такі ключові задачі:

1. **Імплементація повного життєвого циклу даних (CRUD-операції).** Система повинна забезпечувати безперебійне виконання чотирьох базових функцій персистентного сховища: створення (Create), читання (Read), оновлення (Update) та видалення (Delete). У контексті застосунку це означає можливість створювати нові проєкти та задачі, переглядати їх у вигляді списків або на Kanban-дошці, редагувати описи та терміни, а також видаляти неактуальні елементи. Оскільки бекенд реалізовано за допомогою json-server, клієнтська частина (React) повинна коректно формувати HTTP-запити (GET, POST, PUT, PATCH, DELETE) за допомогою Fetch API або бібліотеки Axios, та обробляти відповіді сервера.

2. **Реалізація системи контролю доступу на основі ролей (Role-Based Access Control — RBAC).** В умовах командної роботи критично важливо розмежувати права користувачів. Задача полягає в тому, щоб на рівні глобального стану застосунку (через бібліотеку Zustand) зчитувати профіль авторизованого користувача та динамічно адаптувати інтерфейс.

   - *Admin (Адміністратор)* отримує повний доступ до сторінки PeoplesPage для управління персоналом та може втручатися в будь-який проєкт.

- *Project Manager (Менеджер проєкту)* отримує права на створення та конфігурацію проєктів на сторінці ProjectsPage, але його доступ до глобальних налаштувань системи обмежень.
- *Team Member (Учасник команди)* отримує мінімалістичний інтерфейс, де приховані кнопки видалення проєктів чи додавання нових користувачів. Його головна задача — працювати з картками на TasksPage.

3. **Синхронізація клієнтського стану та бази даних при Drag-and-Drop операціях.** Це одне з найскладніших технічних завдань. Коли користувач візуально перетягує картку задачі з однієї колонки в іншу (наприклад, з *Planned* у *In Progress*), система повинна миттєво відмалювати цю зміну на екрані (Оптимістичний рендеринг UI), щоб уникнути відчуття затримки. Одночасно у фоновому режимі система повинна відправити PATCH-запит до json-server для оновлення поля status відповідної задачі. У разі помилки мережі система має передбачити механізм повернення картки на попереднє місце та сповіщення користувача.

```
const onDragEnd = (result: DropResult) => {
  const { destination, source, draggableId } = result;
  if (!destination) return;

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

  const [projectIdStr, taskIdStr] = draggableId.split("-");
  const projectId = +projectIdStr;
  const taskId = +taskIdStr;

  const currentProject = useProjectStore.getState().projects.find((p) => p.id === projectId);
  if (!currentProject) return;

  const updatedTasks = Array.isArray(currentProject.tasks) ? [...currentProject.tasks] : [];
  const draggedTaskIndex = updatedTasks.findIndex((t) => t.id === taskId);
  const draggedTask = updatedTasks[draggedTaskIndex];
  if (!draggedTask) return;

  if (sourceStatus === destStatus) {
    const tasksInStatus = updatedTasks.filter((t) => t.status === sourceStatus);
    tasksInStatus.splice(source.index, 1);
    tasksInStatus.splice(destination.index, 0, draggedTask);

    updatedTasks.forEach((task, idx) => {
      if (task.status === sourceStatus) {
        updatedTasks[idx] = tasksInStatus.shift()!;
      }
    });
  } else {
    updatedTasks[draggedTaskIndex] = { ...draggedTask, status: destStatus as TaskStatuses };
  }

  updateTaskOrderInProject(currentProject, updatedTasks)
    .then(() => toast.success("Task updated successfully!"))
    .catch(() => toast.error("Failed to update task"));
};
```

Сфера застосування застосунку

Універсальність архітектури розробленого застосунку дозволяє його інтеграцію в різноманітні сфери діяльності.

По-перше, головною цільовою аудиторією є малі та середні IT-компанії, стартапи та команди розробників. Для них Kanban-дошка є природним середовищем, що дозволяє організовувати спринти, відстежувати баги та контролювати процес розробки нових можливостей.

По-друге, інструмент є надзвичайно корисним для незалежних підрядників та фрілансерів. Вони можуть використовувати застосунок як персональну CRM-систему, де кожен "проєкт" — це окремий замовник, а "задачі" — етапи виконання замовлення (від отримання технічного завдання до фінальної оплати).

Крім того, застосунок ідеально підходить для індивідуального планування, зокрема для осіб, які практикують підходи цифрового мінімалізму. Замість використання десятків різних мобільних додатків та громіздких корпоративних систем, що перевантажують увагу та викликають когнітивну втому, єдина централізована Kanban-дошка з чистим, ненав'язливим інтерфейсом дозволяє сфокусуватися виключно на поточних завданнях. Видалення зайвих цифрових подразників і концентрація на одному зручному інструменті сприяє суттєвому підвищенню особистої продуктивності та зменшенню прокрастинації.

1.3 Аналіз існуючих рішень

На сучасному ринку програмного забезпечення (SaaS) представлено сотні систем для управління проєктами. Для обґрунтування технічної та бізнес-цінності розробленого Task Manager було проведено детальний порівняльний аналіз трьох лідерів галузі: Trello, Jira та Miro.

Аналіз системи Trello (Atlassian) Trello — це один із перших застосунків, який масово популяризував концепцію Kanban-дошок для широкого загалу. Його архітектура побудована навколо трьох базових елементів: Дошка (Board), Список (List) та Картка (Card) [16].

- *Переваги:* Беззаперечною перевагою Trello є низький поріг входження (low barrier to entry). Користувацький інтерфейс є інтуїтивно зрозумілим. Процес реєстрації та створення першої дошки займає лічені хвилини.
- *Недоліки:* Trello має серйозні обмеження при спробі масштабування на рівень компанії. У системі відсутня строга реляційна прив'язка між різними проєктами. Найбільшим мінусом є слабка рольова модель: в базовій версії користувач є або адміністратором дошки, або звичайним учасником. Тут неможливо гнучко налаштувати права доступу так, щоб учасник міг перетягувати картки, але не міг редагувати їхній опис. Крім того, компанії, які мають жорсткі вимоги до конфіденційності (NDA), часто не можуть зберігати свої дані на публічних серверах Trello.

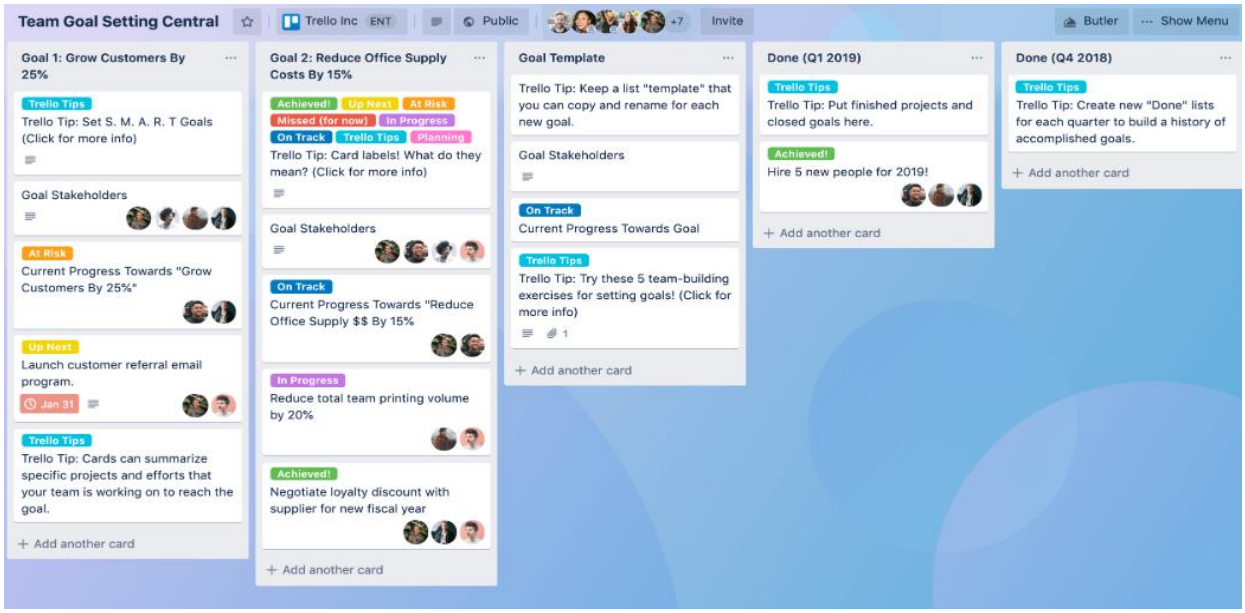


Рисунок 1.3 – Система Trello

Аналіз системи Jira (Atlassian) Jira була створена у 2002 році як система для відстеження помилок (bug tracker), але з часом еволюціонувала в найпотужніший корпоративний інструмент (Enterprise Level) для управління проєктами за методологіями Scrum та Kanban [17].

- *Переваги:* Jira пропонує безмежні можливості кастомізації. Адміністратори можуть створювати власні типи задач (Issue types), складні статуси, автоматизації (Workflows) та налаштовувати багаторівневу ієрархію

прав доступу. Jira інтегрується практично з усіма інструментами для розробки коду (GitHub, GitLab, CI/CD пайплайни).

- *Недоліки:* Головний недолік Jira — її надмірна складність та висока вартість. Інтерфейс системи перевантажений сотнями кнопок, фільтрів та меню. Налаштування нового проєкту в Jira часто вимагає залучення окремого спеціаліста (Jira Administrator). Для невеликої команди або стартапу такий інструмент створює зайве бюрократичне навантаження, уповільнюючи процес роботи замість його прискорення.

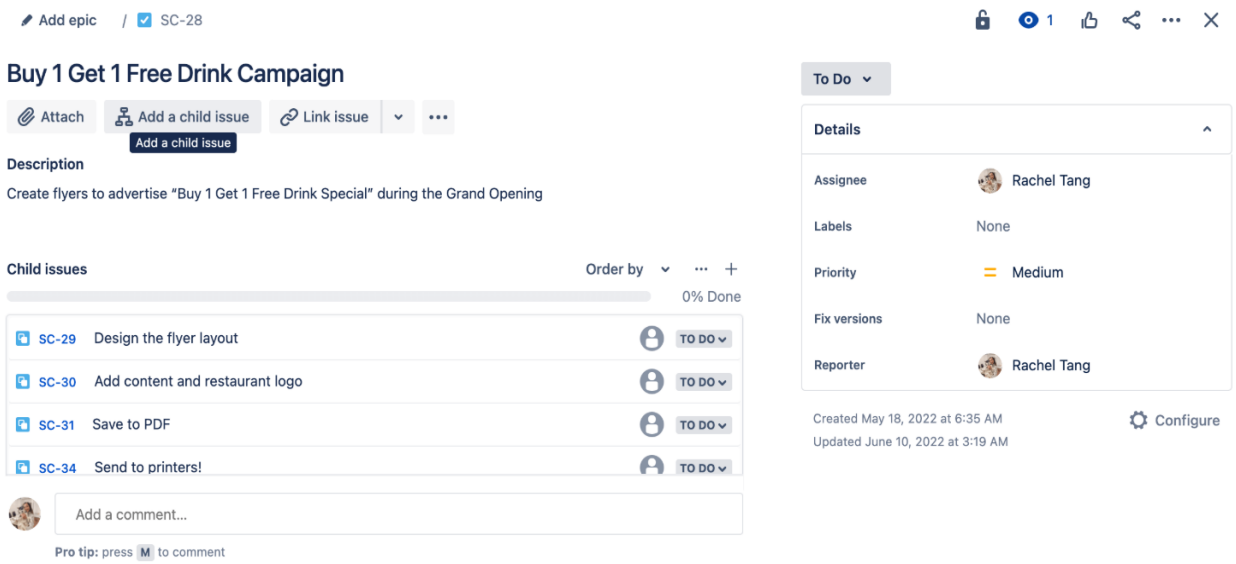


Рисунок 1.4 - системи Jira

Аналіз системи Miro Miro — це віртуальна дошка (online whiteboard), яка створена насамперед для спільної генерації ідей (brainstorming), проєктування архітектури та створення діаграм [18].

- *Переваги:* Безмежне полотно (infinite canvas) дозволяє малювати, прикріплювати стікери, документи та будувати зв'язки між будь-якими елементами абсолютно вільно. Це ідеальний інструмент для етапу проєктування продукту (Discovery phase).
- *Недоліки:* Хоча в Miro можна намалювати Kanban-дошку зі стікерів, вона не є системою баз даних. Картки в Miro не мають строгих атрибутів (терміни виконання, статуси в базі даних, автоматичні переходи). Це виключно візуальний інструмент, який не здатний генерувати аналітику по виконанню

проєкту, контролювати дедлайни чи обмежувати права на основі ролей (крім базових прав "редактор/глядач" на всю дошку цілком).

Функціональне порівняння систем управління проєктами

Перша таблиця відображає наявність та глибину реалізації базових можливостей, необхідних для повноцінного ведення проєктної діяльності в командах.

Таблиця 1.1 – Порівняння можливостей систем управління проектами

Критерій порівняння	**Trello**	**Jira**	**Miro**	**Розроблюваний Task Manager**
Візуалізація Kanban	Повна (базова механіка картка-колонка)	Повна (з підтримкою Agile-спринтів)	Часткова (у вигляді статичних стікерів)	Повна (інтерактивна Kanban-дошка)
Підтримка Drag-and-Drop	Реалізовано нативно	Реалізовано нативно	Реалізовано (вільне переміщення по площині)	Реалізовано нативно (@hello-pangea/dnd)
Рольова модель (RBAC)	Спрощена (Адмін / Учасник)	Високодеталізована (кастомні матриці прав)	Базова (Редактор / Коментатор / Глядач)	Строга трирівнева (Admin, PM, Team Member)
Управління користувачами	Централізовано через Atlassian ID	Глобальне (через Atlassian Access / Active Directory)	Базове управління запрошеннями	Виділений модуль (PeoplesPage) з повним CRUD
Контроль дедлайнів	Тільки текстове поле з нагадуванням	Автоматизований (діаграми, трекінг прострочень)	Відсутній (текстові мітки на стікерах)	Вбудований у сутності проєктів та задач

Обґрунтування доцільності розробленого застосунку

Проведений аналіз демонструє поляризацію ринку: існують або занадто прості рішення без корпоративного контролю (Trello), або надпотужні та дорогі системи з перевантаженим інтерфейсом (Jira).

Розроблений в рамках дипломної роботи **Task Manager** заповнює прогалину між цими крайнощами, пропонуючи оптимальний баланс функціональності та зручності:

1. **Зручність (як у Trello):** Застосунок має чистий, мінімалістичний інтерфейс без зайвих вікон. Використання бібліотеки Drag-and-Drop (@hello-pangea/dnd) забезпечує звичний та зручний користувацький досвід.

2. **Контроль та безпека (як у Jira):** Під капотом застосунку реалізовано строгу трирівневу систему ролей (Admin, Project Manager, Team Member). Доступ до маршрутів (react-router-dom) та елементів інтерфейсу чітко розмежований.

3. **Незалежність інфраструктури:** Використання архітектури клієнт-сервер (React + json-server) дозволяє розгорнути застосунок на власних приватних серверах організації (On-Premise), гарантуючи повний контроль над комерційною таємницею та даними співробітників.

4. **Масштабованість UI:** Завдяки використанню Tailwind CSS та виносу базових стилів у App.scss, зовнішній вигляд застосунку можна кардинально змінити за кілька хвилин для відповідності корпоративному стилю будь-якого замовника.

Таким чином, розробка власного вебзастосунку для управління проєктами є повністю обґрунтованим технічним та бізнес-рішенням, що дозволяє отримати сучасний, безпечний та адаптивний інструмент без переплат за ліцензії сторонніх корпорацій.

2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Будь-який процес створення програмного забезпечення, згідно з класичними та гнучкими методологіями розробки (SDLC — Software Development Life Cycle), неминуче проходить ключовий етап архітектурного проєктування. Перехід від концептуального розуміння предметної області, що було визначено в попередньому розділі, до безпосередньої програмної реалізації вимагає прийняття низки критичних інженерних рішень. На цьому етапі формується фундамент майбутньої інформаційної системи, від якого безпосередньо залежатиме її продуктивність, здатність до масштабування, рівень безпеки та легкість подальшого технічного обслуговування. Помилки на етапі проєктування або нераціональний вибір технологій неминуче призводять до накопичення так званого технічного боргу (technical debt), що в майбутньому може суттєво ускладнити або навіть унеможливити розвиток продукту.

Сучасний ринок веброзробки характеризується надзвичайною динамічністю та величезним розмаїттям інструментальних засобів. За останнє десятиліття архітектурні підходи зазнали значних трансформацій. На зміну класичним багатосторінковим застосункам (Multi-Page Applications), де більшість обчислень і процес рендерингу HTML-сторінок відбувалися виключно на стороні сервера, прийшла парадигма односторінкових застосунків (Single Page Applications — SPA). У такій архітектурі клієнтська частина (Frontend) бере на себе левову частку відповідальності: бізнес-логіку, складну маршрутизацію та багаторівневе управління глобальним станом. Це створює нові виклики для інженерів: необхідно обрати такий технологічний стек, який здатний ефективно обробляти великі масиви даних безпосередньо в браузері користувача, не допускаючи при цьому деградації продуктивності або втрати плавності інтерфейсу.

Вибір інструментарію для розробки менеджера задач (Task Manager) базувався на суворому аналізі нефункціональних вимог до створюваної системи. Серед них варто виділити наступні:

- Високий рівень інтерактивності (Responsiveness): Інтерфейс Kanban-дошки передбачає постійну, безперервную взаємодію користувача з елементами екрану. Операції перетягування карток (Drag-and-Drop), миттєва зміна статусів та анімації переміщення вимагають інструментів, здатних оптимізувати оновлення об'єктної моделі документа (DOM).
- Надійність та консистентність даних (Data Integrity): Система повинна забезпечувати строгу типізацію складних об'єктів (проєктів із вкладеними задачами та користувачів із різними правами доступу) для запобігання логічним помилкам на етапі виконання (runtime errors).
- Модульність та кастомізація (Modularity and Theming): Архітектура клієнтської частини має будуватися на незалежних компонентах, що підтримують повторне використання (reusability), а підходи до стилізації повинні гарантувати легку адаптацію візуального стилю системи під брендбук будь-якого кінцевого замовника.

Враховуючи вищезазначені критерії, для реалізації дипломного проєкту було сформовано сучасний, збалансований стек технологій на базі екосистеми JavaScript. У даному розділі наведено глибоке теоретичне та практичне обґрунтування вибору кожної ключової технології: клієнтської бібліотеки для побудови інтерфейсу, мови для забезпечення суворої типізації, патернів управління глобальним станом, інструментів стилізації, а також засобів для швидкого розгортання REST API. Крім того, розділ містить детальний опис етапу проєктування структури бази даних, зв'язків між сутностями та ієрархії компонентів, що формують надійний архітектурний каркас розробленого програмного продукту.

2.1 Мова програмування та вибір середовища для реалізації

Вибір базової бібліотеки: React

Основним середовищем для реалізації клієнтської частини (Frontend) було обрано бібліотеку **React**, розроблену компанією Meta (Facebook). На відміну від монолітних фреймворків (наприклад, Angular), React є відносно легкою бібліотекою, сфокусованою виключно на відображенні користувацького інтерфейсу (View-шар у патерні MVC).

Вибір React для реалізації менеджера задач обґрунтовується такими технічними перевагами:

1. Компонентна архітектура. Інтерфейс Kanban-дошки розбивається на незалежні, ізольовані компоненти: Board (Дошка), Column (Колонка статусу), TaskCard (Картка задачі). Це дозволяє повторно використовувати код, полегшує його тестування та паралельну розробку.

2. Virtual DOM (Віртуальне дерево об'єктів документа). Операції перетягування карток (Drag-and-Drop) вимагають сотень перемальовувань інтерфейсу за секунду. Пряма маніпуляція реальним DOM є дуже повільною операцією. React створює віртуальну копію DOM у пам'яті, порівнює її з попереднім станом за допомогою алгоритму Reconciliation (узгодження) і вносить у реальний браузер лише мінімально необхідні зміни. Це гарантує плавність анімацій та відсутність "зависань" інтерфейсу під час роботи з великими проєктами.

3. Однонаправлений потік даних (One-way data binding). Дані в React передаються виключно зверху вниз (від батьківських компонентів до дочірніх). Це робить поведінку застосунку передбачуваною: зміна статусу задачі в одному місці гарантовано оновлює всі пов'язані з нею візуальні елементи.

Для ініціалізації проєкту було використано інструмент Create React App (CRA). CRA інкапсулює складні конфігурації збирачів (Webpack) та транспайлерів (Babel), дозволяючи зосередитися безпосередньо на написанні бізнес-логіки з першого дня розробки:

```
36      "scripts": {
37        "start": "react-scripts start",
38        "build": "react-scripts build",
39        "test": "react-scripts test",
40        "eject": "react-scripts eject",
41        "predeploy": "npm run build",
42        "deploy": "gh-pages -d build"
43      },
```

Вибір мови програмування: TypeScript

Хоча нативною мовою веббраузерів є JavaScript, для розробки застосунку було обрано TypeScript — суворо типізовану надмножину JavaScript від корпорації Microsoft. У застосунках класу Task Manager обсяг даних, що циркулюють між клієнтом та сервером, є значним і має складну структуру (наприклад, об'єкт Проєкту містить масиви Задач, які в свою чергу мають власні статуси та виконавців). Використання класичного JavaScript часто призводить до помилок під час виконання (runtime errors), пов'язаних із спробами звернення до неіснуючих властивостей об'єктів.

TypeScript вирішує цю проблему ще на етапі компіляції. У проєкті були задекларовані строгі інтерфейси (Interfaces) для всіх ключових сутностей бази даних:

- Project: строго визначає наявність полів id, title, status, масивів peoples та tasks.
- People: гарантує, що поле role може приймати лише визначені значення (наприклад, Admin, PM, Team Member) через використання перелічуваних типів (Enums) або Union-типів.
- Task: типізує статус задачі.

Завдяки TypeScript, середовище розробки (IDE) забезпечує розумне автодоповнення коду та миттєво підсвічує логічні помилки, що суттєво підвищує надійність фінального продукту:

```ts
import { ID } from "./ID";
import { People } from "./People";
import { Statuses } from "./Statuses";
import { Task } from "./Task";

export interface Project {
  id: ID;
  title: string;
  description: string;
  peoples: People[];
  tasks: Task[];
  date: string;
  dateEnd: string;
  status: Statuses;
}
```

```ts
import { ID } from "./ID";
import { Project } from "./Project";
import { Roles } from "./Roles";
import { Task } from "./Task";

export interface People {
  id: ID;
  firstName: string;
  lastName: string;
  role: Roles;
  login: string;
  password: string;
  projects: Project[];
  task?: Task[];
}
```

2.2 Інструменти реалізації бази даних

Дипломний проєкт сфокусований на реалізації складної клієнтської частини з багатою логікою взаємодії. Для того, щоб забезпечити повноцінну клієнт-серверну архітектуру без необхідності розробки окремого складного бекенду (наприклад, на Node.js або Python), як систему управління базами даних було використано пакет **json-server**.

json-server — це інструмент, який дозволяє за лічені секунди розгорнути повноцінний REST API на основі єдиного файлу db.json. Його використання в архітектурі розробленого менеджера задач має низку концептуальних переваг:

1. **Повна RESTful сумісність:** Інструмент автоматично генерує маршрути (Endpoints) для всіх колекцій у JSON-файлі. Клієнтський застосунок використовує стандартні HTTP-методи для маніпуляції даними:
   - GET /projects — для завантаження списку проєктів;
   - POST /tasks — для створення нової задачі;
   - PATCH /tasks/:id — для часткового оновлення задачі (наприклад, при зміні статусу після Drag-and-Drop);
   - DELETE /peoples/:id — для видалення користувача адміністратором.
2. **Ізоляція Frontend:** Такий підхід дозволив повністю абстрагувати клієнтську логіку. Якщо в майбутньому виникне потреба перевести проєкт на промислову базу даних (наприклад, PostgreSQL або MongoDB), клієнтський код (React) не потребуватиме жодних змін — достатньо буде лише змінити базову URL-адресу для API-запитів.

3. **Хмарне розгортання:** Серверна частина успішно контейнеризується та працює в хмарному середовищі (на платформі Render), що дозволяє демонструвати роботу проєкту онлайн, з будь-якого пристрою:


```
[
  {
    "title": "Task manager",
    "description": "Create own task manager app",
    "status": "Planned",
    "peoples": [
      {
        "id": 0,
        "firstName": "admin",
        "lastName": "admin",
        "login": "admin",
        "password": "admin",
        "role": "Admin",
        "projects": [],
        "task": []
      }
    ],
    "tasks": [
      {
        "id": 3,
        "description": "create UI/UX",
        "title": "UI/UX",
        "status": "To Do",
        "date": "06/01/2026",
        "deadline": "3 days",
        "peoples": [
          {
            "id": 0,
            "firstName": "admin",
            "lastName": "admin",
            "login": "admin",
            "password": "admin",
            "role": "Admin",
            "projects": [],
            "task": []
          }
        ]
      },
      {
```


2.3 Інші допоміжні інструменти

Для реалізації специфічного функціоналу сучасного Single Page Application (SPA) базових можливостей React недостатньо. До архітектури проєкту було інтегровано низку високопродуктивних бібліотек.

Управління глобальним станом: Zustand У великих React-застосунках передача даних (пропсів) через безліч рівнів компонентів (prop-drilling) стає проблемою. Історично стандартом для вирішення цієї проблеми був Redux. Однак Redux є надто багатослівним (boilerplate-heavy) і вимагає створення складних екосистем (Actions, Reducers, Dispatchers). Для розробленого застосунку було обрано сучасну альтернативу — Zustand [13]. Ця бібліотека базується на принципах хуків (Hooks) і дозволяє створити глобальне сховище

(Store) з мінімальною кількістю коду. У проєкті Zustand відповідає за збереження сесії авторизованого користувача: після успішного входу на LoginPage, дані користувача та його роль (role) зберігаються у Zustand-сховищі і стають миттєво доступними для будь-якого компонента системи. Це є основою роботи імплементованої Role-Based Access Control системи:

```
import { create } from "zustand";
import { persist } from "zustand/middleware";
import { usePeopleStore } from "./peoplesStore";
import { AuthState } from "../types/storeTypes/AuthState";

export const useAuthStore = create<AuthState>()(
  persist(
    (set) => ({
      user: null,
      token: null,
      loading: false,
      error: null,

      login: async (login, password) => {
        set({ loading: true, error: null });

        const {
          users,
          fetchUsers,
          loading: usersLoading,
        } = usePeopleStore.getState();

        if (users.length === 0 && !usersLoading) {
          await fetchUsers();
        }

        const updatedUsers = usePeopleStore.getState().users;
        const user = updatedUsers.find(
          (user) => user.login === login && user.password === password
        );

        if (user) {
          const token = "some-generated-token";
          set({
            user,
            token,
            loading: false,
          });

          localStorage.setItem("token", token);
          return Promise.resolve();
        } else {
          set({ error: "Invalid credentials", loading: false });
          return Promise.reject("Invalid credentials");
        }
      },

      logout: () => {
        set({ user: null, token: null });
        localStorage.removeItem("token");
      },
    }),
    { name: "auth-storage" }
  )
);
```

Маршрутизація: react-router-dom Менеджер задач є односторінковим застосунком (SPA). Це означає, що при переході між сторінками браузер не робить запит до сервера за новими HTML-документами. Замість цього використовується бібліотека react-router-dom, яка перехоплює зміну URL-адреси і динамічно підміняє React-компоненти. У проєкті налаштовано такі основні маршрути: /login, /, /projects, /tasks/:id та /peoples. Бібліотека також забезпечує захист маршрутів (Protected Routes): якщо користувач без прав адміністратора спробує перейти на /peoples, маршрутизатор автоматично перенаправить його на сторінку з помилкою або головну сторінку.

Механіка Kanban: @hello-pangea/dnd Ядром застосунку є Kanban-дошка на сторінці TasksPage. Для реалізації візуального перетягування задач була використана бібліотека @hello-pangea/dnd. На відміну від стандартного HTML5 Drag-and-Drop API, ця бібліотека створена спеціально для React. Вона гарантує плавну анімацію елементів, доступність (можливість переміщувати задачі за допомогою клавіатури для людей з обмеженими можливостями) та коректну роботу на мобільних пристроях із сенсорними екранами.

UI/UX Дизайн та стилізація: Tailwind CSS + SCSS + Flowbite Дизайн застосунку побудований на сучасній парадигмі Utility-First CSS за допомогою фреймворку Tailwind CSS [10]. Замість написання сотень рядків кастомного CSS, класи описуються безпосередньо в розмітці (наприклад, flex items-center justify-between p-4). Це суттєво пришвидшує розробку візуальної частини. Для прискорення створення стандартизованих елементів (кнопки, модальні вікна, форми вводу) використано бібліотеку компонентів Flowbite, яка ідеально інтегрується з Tailwind. Проте, для збереження гнучкості та підтримки корпоративної ідентичності, архітектура стилів була розширена за допомогою препроцесора SCSS. У файлі App.scss визначено глобальні змінні (CSS Variables) для основних кольорів, шрифтів та відступів. Tailwind CSS налаштовано таким чином, щоб споживати ці змінні з SCSS. Це архітектурне рішення дозволяє повністю змінити кольорову гаму застосунку (наприклад, створити темну тему

або адаптувати інтерфейс під брендбук клієнта), змінивши лише кілька рядків у App.scss.

2.4. Проєктування застосунку та бази даних для менеджера задач

Процес проєктування будь-якого сучасного програмного продукту вимагає створення чіткої внутрішньої структури (архітектури файлів та директорій), розробки схеми зберігання даних та визначення принципів побудови користувацького інтерфейсу. Правильно спроєктована архітектура гарантує, що застосунок буде легко підтримувати, тестувати та масштабувати в майбутньому при додаванні нового функціоналу або розширенні команди розробників.

Архітектура клієнтської частини (File Structure & Modularity)

Клієнтська частина Task Manager побудована за принципами компонентно-орієнтованої архітектури, характерної для екосистеми React. Для уникнення проблеми "монолітного фронтенду", коли весь код змішується в кількох великих файлах, було прийнято рішення використовувати модульну структуру директорій (Feature-Sliced Design або модульний підхід). Увесь вихідний код застосунку зосереджено в директорії src/, яка логічно поділена на спеціалізовані підкаталоги.

Такий поділ відповідає принципу єдиної відповідальності (Single Responsibility Principle) з абревіатури SOLID. Основна структура директорій має такий вигляд:

- src/modules/ (Модулі та Компоненти): Найбільша директорія проєкту. Тут містяться всі візуальні елементи та сторінки (Pages). Компоненти згруповані за їхнім логічним призначенням (наприклад, окремі папки для модулів авторизації, Kanban-дошки, управління користувачами). Це дозволяє інкапсулювати бізнес-логіку кожного розділу застосунку.
- src/services/ (Сервіси та API): Директорія для зберігання допоміжних функцій, які не відповідають за відображення інтерфейсу (UI-agnostic code). Тут зосереджені функції для виконання HTTP-запитів до json-server (методи GET,

POST, PUT, DELETE), обробки помилок мережі та форматування дат чи текстових даних.

- src/stores/ (Глобальний стан): Містить конфігураційні файли бібліотеки Zustand. Тут описані глобальні сховища, зокрема authStore (зберігає дані про поточну сесію, логін та роль авторизованого користувача).
- src/styles/ (Стилізація): Зберігає глобальні SCSS-файли та конфігурації для Tailwind CSS, що формують єдину дизайн-систему застосунку.
- src/types/ (Типізація): Директорія, яка є ключовою для TypeScript. Тут зібрані всі декларації інтерфейсів (Interfaces) та перелічуваних типів (Enums). Завдяки винесенню типів в окрему папку, їх можна легко імпортувати в будь-який компонент або сервіс, що усуває дублювання коду.
- Кореневі файли: App.tsx є головним компонентом-контейнером, Root.tsx відповідає за конфігурацію маршрутизації (React Router), а index.tsx є точкою входу (Entry point), яка монтує React-застосунок у реальний DOM браузера.

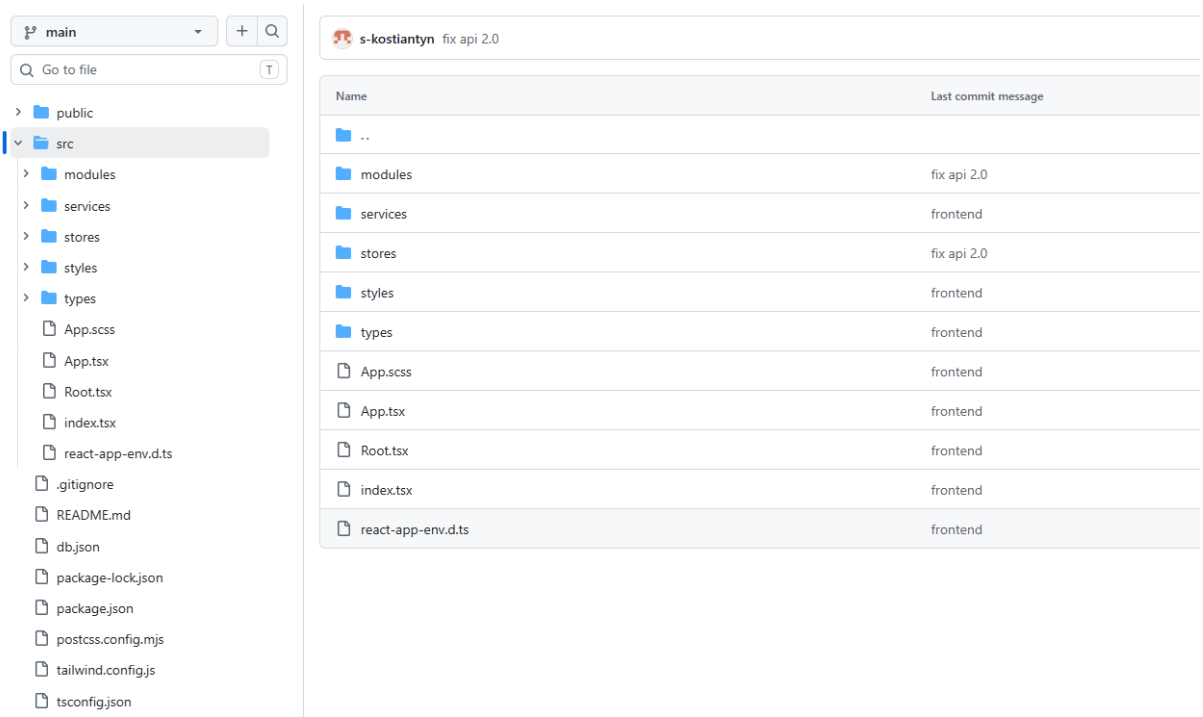


Рисунок 2.1 – Структура каталогів проєкту

Проєктування схеми бази даних (Data Modeling)

Оскільки проєкт використовує json-server як серверну базу даних, схема зберігання інформації побудована на принципах NoSQL (документо-орієнтованих) баз даних. Замість класичних суворих реляційних таблиць із зовнішніми ключами, дані зберігаються у вигляді ієрархічних JSON-об'єктів. Це рішення ідеально підходить для вебзастосунків на JavaScript, оскільки формат JSON є нативним для браузерів і не потребує складних перетворень при передачі даних (парсингу).

База даних db.json містить дві головні колекції: "peoples" та "projects".

1. Колекція "peoples" (Користувачі системи): Ця колекція зберігає облікові записи працівників. Кожен об'єкт містить ідентифікатор (id), персональні дані (firstName, lastName), дані для автентифікації (login, password) та найважливіше поле — role. Поле role (наприклад, "Admin", "Project Manager" або "Team Member") безпосередньо впливає на логіку відображення інтерфейсу та доступність маршрутів (Protected Routes). Також передбачені масиви projects та task для можливого розширення логіки двостороннього зв'язку (наприклад, щоб швидко отримати всі проєкти, за якими закріплений конкретний користувач).

2. Колекція "projects" (Проєкти та Задачі): Колекція проєктів є агрегатором робочих процесів. Найважливішим архітектурним рішенням на етапі проєктування бази даних стала денормалізація даних (вкладення колекції). Замість того, щоб створювати окрему глобальну колекцію "tasks" і пов'язувати кожну задачу з проєктом через projectId, було вирішено вкласти масив задач безпосередньо всередину об'єкта проєкту ("tasks": []). Такий підхід має критичну перевагу для продуктивності клієнтської частини: при переході на сторінку конкретного проєкту (TasksPage), застосунку достатньо виконати лише один GET-запит до API (/projects/:id), щоб миттєво отримати всю інформацію про проєкт та всі пов'язані з ним задачі для відмальовування Kanban-дошки. Це значно зменшує навантаження на мережу та час очікування для користувача (Latency).

Проєктування користувацького інтерфейсу (UI/UX Design)

З огляду на те, що Kanban-дошка — це візуальний інструмент управління, проєктуванню користувацького інтерфейсу (UI) та користувацького досвіду (UX) було приділено особливу увагу. Інтерфейс повинен бути не лише функціональним, але й відповідати сучасним стандартам вебдизайну: бути адаптивним, мати достатній контраст та забезпечувати швидкий зворотний зв'язок.

Для реалізації цих вимог було обрано комбінований підхід до стилізації:

- Tailwind CSS + Flowbite: Використання utility-first фреймворку дозволило проєктувати інтерфейс безпосередньо в JSX-розмітці компонентів. Бібліотека Flowbite надала набір заздалегідь спроєктованих, доступних (Accessible) компонентів, що значно прискорило розробку стандартних елементів інтерфейсу, таких як форми вводу, модальні вікна та кнопки.
- Глобальна конфігурація через SCSS: Оскільки Tailwind CSS генерує класичні стилі, виникла потреба в створенні централізованої системи управління темою (Theming). Для цього в директорії styles/ було створено файл App.scss. У цей файл винесено ключові візуальні змінні (CSS Custom Properties).

Такий архітектурний підхід робить дизайн застосунку максимально "відчуженим" (White-label ready). Якщо застосунок необхідно буде впровадити в іншій компанії з іншим корпоративним стилем, розробнику не потрібно буде переписувати сотні класів Tailwind у кожному React-компоненті. Достатньо лише оновити значення базових змінних (кольори акцентів, сітка відступів, шрифти) у файлі App.scss, і весь застосунок автоматично змінить свій візуальний вигляд. Цей принцип гарантує високу масштабованість та економічну доцільність продукту при комерційному розгортанні.

3. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ МЕНЕДЖЕРА ЗАДАЧ

Безпосередня програмна реалізація інформаційної системи є визначальним етапом життєвого циклу розробки програмного забезпечення (SDLC), на якому спроєктовані раніше архітектурні рішення, моделі даних та інтерфейсні макети трансформуються у функціональний програмний код. Практична імплементація вебзастосунку для управління проєктами вимагає комплексного підходу до інтеграції різноманітних технологій та забезпечення їхньої безперебійної взаємодії.

Розробка клієнт-серверних рішень у форматі Single Page Application (SPA) передбачає делегування значної частини обчислювального навантаження та бізнес-логіки на сторону клієнта (браузера користувача). У контексті менеджера задач (Task Manager) це зумовлює необхідність програмної реалізації складних механізмів управління динамічною маршрутизацією, обробки багаторівневої авторизації (Role-Based Access Control) та синхронізації глобального стану системи (State Management) у режимі реального часу. Відповідно, процес кодування базувався на оптимізації рендерингу віртуального дерева документа (Virtual DOM) за допомогою бібліотеки React, застосуванні строгої типізації даних через TypeScript та налаштуванні надійної асинхронної взаємодії з REST API сервером.

З метою мінімізації технічних ризиків та забезпечення стабільності програмного коду, процес розробки здійснювався за ітеративною моделлю. Практична імплементація системи охоплює наступні логічні етапи, які детально розглядаються у даному розділі:

1. Ініціалізація програмного середовища та розгортання бази даних. Налаштування конфігураційних файлів проєкту (зокрема tsconfig.json для TypeScript) та створення персистентного сховища даних (json-server) із процедурою первинного наповнення тестовими наборами даних (Database Seeding).

2. Імплементація підсистеми безпеки та маршрутизації. Побудова базового архітектурного каркаса клієнтської частини, налаштування глобального

сховища стану (за допомогою бібліотеки Zustand) для обробки сесій авторизованих користувачів та розробка захищених маршрутів (Protected Routes) для розмежування прав доступу.

3. Реалізація функціонального ядра системи. Програмування основної бізнес-логіки застосунку, що включає розробку сторінки управління проєктами та інтерактивної Kanban-дошки. Імплементація алгоритмів обробки подій перетягування об'єктів (Drag-and-Drop) із застосуванням методів оптимістичного оновлення інтерфейсу (Optimistic UI update) для синхронізації з базою даних.

4. Оптимізація користувацького інтерфейсу та розгортання. Інтеграція стилістичних рішень за допомогою CSS-фреймворків, розробка системи зворотного зв'язку (сповіщень) для користувача та підготовка програмного коду до розгортання (Deployment) у хмарному середовищі.

Детальний опис алгоритмів та фрагментів коду, наведених у даному розділі, слугує підтвердженням працездатності розробленої системи та обґрунтовує коректність інженерних рішень, прийнятих на етапі архітектурного проєктування.

3.1 Реалізація бази даних

Програмна реалізація серверної частини та бази даних для застосунку базується на використанні пакета json-server, який дозволяє швидко розгорнути повноцінний REST API з використанням формату JSON для зберігання даних. Цей підхід забезпечує незалежність клієнтської частини (Frontend) і дозволяє повністю зосередитися на розробці користувацького інтерфейсу та бізнес-логіки.

Формування структури зберігання даних Основою бази даних є файл db.json, який виконує роль персистентного нереляційного сховища. У файлі задекларовано дві основні колекції (масиви об'єктів): "peoples" (користувачі) та "projects" (проєкти).

Колекція "peoples" спроєктована для забезпечення функціонування модуля авторизації та системи контролю доступу. Кожен об'єкт користувача містить унікальний ідентифікатор (id), текстові поля для збереження імені та прізвища

(firstName, lastName), облікові дані (login, password) та критично важливе поле ролі (role). Додатково передбачено порожні масиви projects та task для можливого встановлення зворотних зв'язків з боку користувача (наприклад, для формування персонального дашборду співробітника).

Колекція "projects" містить інформацію про макрозадачі системи. З метою оптимізації кількості мережевих запитів під час роботи застосунку було застосовано метод денормалізації даних. Замість винесення задач в окрему колекцію з посиланням на ідентифікатор проєкту (як це робиться в класичних SQL-базах), масив задач ("tasks") фізично вкладено всередину об'єкта відповідного проєкту. Це архітектурне рішення гарантує, що при відкритті Kanban-дошки клієнтський застосунок здійснює лише один HTTP-запит (GET) до ендпоінту /projects/:id, миттєво отримуючи всю необхідну структуру для рендерингу карток.

Ініціалізація початкових даних (Seeding) та налаштування REST API Для забезпечення можливості тестування функціоналу на етапі розробки, базу даних було проініціалізовано початковими наборами даних (mock data). Було створено три тестові облікові записи, що репрезентують різні рівні доступу (Admin, Project Manager, Team Member).

Завдяки використанню json-server, базові маршрути API генеруються автоматично на основі структури JSON-файлу. Клієнтська частина застосунку взаємодіє з сервером через такі основні ендпоінти:

- GET /peoples — для завантаження списку персоналу (доступно лише адміністратору);
- GET /projects — для отримання списку всіх ініціатив;
- POST /projects — для збереження новоствореного проєкту в базі;
- PUT /projects/:id або PATCH /projects/:id — для оновлення стану проєкту, зокрема для збереження нового порядку та статусів задач після їх переміщення на Kanban-дошці;
- DELETE /peoples/:id — для видалення облікового запису користувача.

З метою забезпечення постійного доступу до даних, серверну частину (json-server) було контейнеризовано та розгорнуто на хмарній платформі Render.

3.2 Реалізація застосунку для менеджера задач

Реалізація клієнтської частини застосунку здійснена за допомогою бібліотеки React та мови суворої типізації TypeScript. Процес програмування було поділено на кілька взаємопов'язаних модулів: управління глобальним станом, налаштування маршрутизації та реалізація інтерактивної бізнес-логіки.

Імплементація управління станом (State Management) Для забезпечення консистентності даних між різними компонентами системи інтегровано бібліотеку Zustand. Створено глобальне сховище стану (store), яке відповідає за управління сесією користувача. Після успішної валідації облікових даних на сторінці LoginPage, об'єкт користувача передається до сховища Zustand. Це дозволяє уникнути передачі даних через багаторівневу ієрархію компонентів (усунення проблеми prop-drilling) і забезпечує миттєвий доступ до поля role для перевірки прав доступу на будь-якому рівні застосунку.

Реалізація системи маршрутизації та захисту сторінок Навігація в межах односторінкового застосунку (SPA) реалізована за допомогою пакету react-router-dom. У кореневому файлі Root.tsx описано дерево маршрутів системи. Окремим архітектурним завданням стала реалізація механізму захисту маршрутів (Protected Routes). Створено спеціальний компонент-обгортку (Higher-Order Component), який перехоплює процес рендерингу закритої сторінки. Компонент зчитує поточну роль користувача зі сховища Zustand і порівнює її з масивом дозволених ролей для даного маршруту. Якщо перевірка не пройдена, ініціюється примусове перенаправлення (Redirect) на безпечну сторінку (наприклад, HomePage або вікно помилки доступу). Це блокує можливість перегляду сторінки PeoplesPage користувачами з правами Project Manager або Team Member.

Програмування інтерактивної Kanban-дошки Сторінка TasksPage є ключовим модулем системи, що вимагав реалізації складної алгоритмічної

логіки. Забезпечення візуального перетягування об'єктів виконано з використанням бібліотеки @hello-pangea/dnd.

Алгоритм обробки зміни статусу задачі складається з наступних кроків:

1. Компонент <DragDropContext> перехоплює подію завершення перетягування (onDragEnd), генеруючи об'єкт із даними про початкову та кінцеву координати картки (ідентифікатори колонок).

2. Якщо картка була переміщена за межі активних колонок, операція скасовується, і картка повертається на початкову позицію.

3. Якщо картка переміщена в межах тієї ж колонки, виконується перерахунок індексів елементів масиву для збереження нового порядку сортування.

4. Якщо картка перенесена до іншої колонки (зміна статусу задачі), алгоритм знаходить потрібну задачу в масиві tasks поточного проєкту і змінює значення її поля status.

5. Для забезпечення високої чутливості інтерфейсу застосовується патерн "Оптимістичне оновлення" (Optimistic UI). Зміни миттєво відмальовуються в браузері, після чого ініціюється асинхронний PATCH-запит до бекенду для постійного збереження нових даних у db.json. У випадку збою мережі (catch-блок у запиті), система генерує попередження через бібліотеку react-toastify та відкочує локальний стан до попереднього (синхронізація з сервером).

Стилізація та формування інтерфейсу Інтерфейсна складова реалізована на базі компонентів Flowbite та утилітарних класів Tailwind CSS. Для забезпечення масштабованості дизайну, ключові параметри стилістики (кольорова палітра, сітка відступів) винесені у конфігураційний файл App.scss. Це дозволило створити цілісну дизайн-систему, яка адаптується під різні розміри екранів пристроїв (Responsive Web Design) і легко піддається модифікації.

Версія застосунку розгорнута на платформі Render https://task-manager-frontend-okw6.onrender.com/#/login; код проєкту розміщений на платформі

Github: https://github.com/s-kostiantyn/task-manager-frontend/tree/main та https://github.com/s-kostiantyn/task-manager-backend

3.3 Огляд використання менеджера задач

Експлуатаційна модель розробленого застосунку передбачає чітку послідовність взаємодії користувача з інтерфейсом, яка адаптується залежно від рівня наданих прав (авторизаційної ролі).

Робочий сеанс починається зі сторінки LoginPage. Після коректного введення логіну та пароля система перенаправляє користувача на головну сторінку (HomePage). Ця сторінка виконує функцію інформаційного дашборду, де відображаються загальні дані про обліковий запис.

Подальша взаємодія безпосередньо залежить від ролі:

Користувач із роллю "Team Member" (Виконавець): Має доступ до обмеженого переліку функцій. При переході на сторінку ProjectsPage він бачить лише ті проєкти, до яких його було залучено. Інтерфейс створення нових проєктів або їх видалення приховано. Основне робоче середовище виконавця — сторінка TasksPage, де він візуально відслідковує свої задачі та переміщує їхні картки по мірі виконання роботи.

Користувач із роллю "Project Manager" (Менеджер проєкту): Отримує розширені права на сторінці проєктів. Він може ініціювати створення нових проєктних просторів, редагувати їхні описи та терміни завершення (dateEnd), а також додавати задачі на дошку. Однак спроба перейти за маршрутом /peoples призведе до появи вікна з помилкою 404 або повідомленням про обмежений доступ.

Користувач із роллю "Admin" (Адміністратор): Має абсолютний контроль над системою. Адміністратору доступні всі CRUD-операції в межах будь-якого проєкту. Крім того, він має ексклюзивний доступ до сторінки PeoplesPage, де реалізовано повноцінний інтерфейс для управління персоналом: реєстрація нових співробітників, зміна їхніх персональних даних та призначення ролей.

Усі критичні дії користувачів (успішна авторизація, видалення проєкту, зміна статусу задачі або помилки з'єднання з сервером) супроводжуються виведенням спливаючих сповіщень у куті екрану, що забезпечує належний рівень зворотного зв'язку між системою та оператором. Фінальна версія застосунку успішно розгорнута на платформі Render, забезпечуючи стабільний доступ до Kanban-дошки з будь-якої точки світу (https://task-manager-frontend-okw6.onrender.com/#/login).

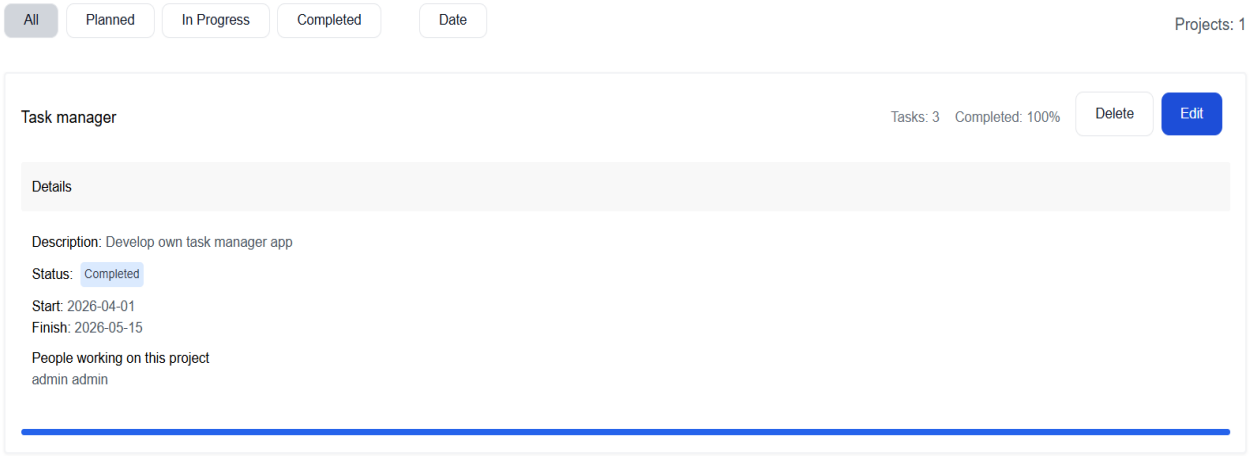


Рисунок 3.1 – Приклад вікна менеджера задач

У вікні інтерфейсу для управління проєктами відображається картка конкретного проєкту під назвою "Task manager". Основні деталі, які можна побачити:

Статус та прогрес. Статус проєкту: Позначений як "Completed" (Завершено). Завдання: Вказано, що всього було 3 завдання, і всі вони виконані (100%). Часові межі: Проєкт тривав з 1 квітня 2026 року по 15 травня 2026 року.

Опис та команда. Опис: Розробка власного додатка для менеджменту задач ("Develop own task manager app"). У розділі "People working on this project" вказано користувача з іменем "admin admin".

Елементи керування Верхня панель фільтрації: Дозволяє сортувати проєкти за категоріями: All, Planned, In Progress, Completed, а також за датою.

Кнопки дій: У правому куті картки є синя кнопка "Edit" (Редагувати) та кнопка "Delete" (Видалити).

Візуальний індикатор: Знизу картки розташована товста синя лінія, яка, ймовірно, дублює статус завершення проєкту.Загалом інтерфейс виглядає чистим і мінімалістичним, орієнтованим на швидкий перегляд ключових показників проєкту.

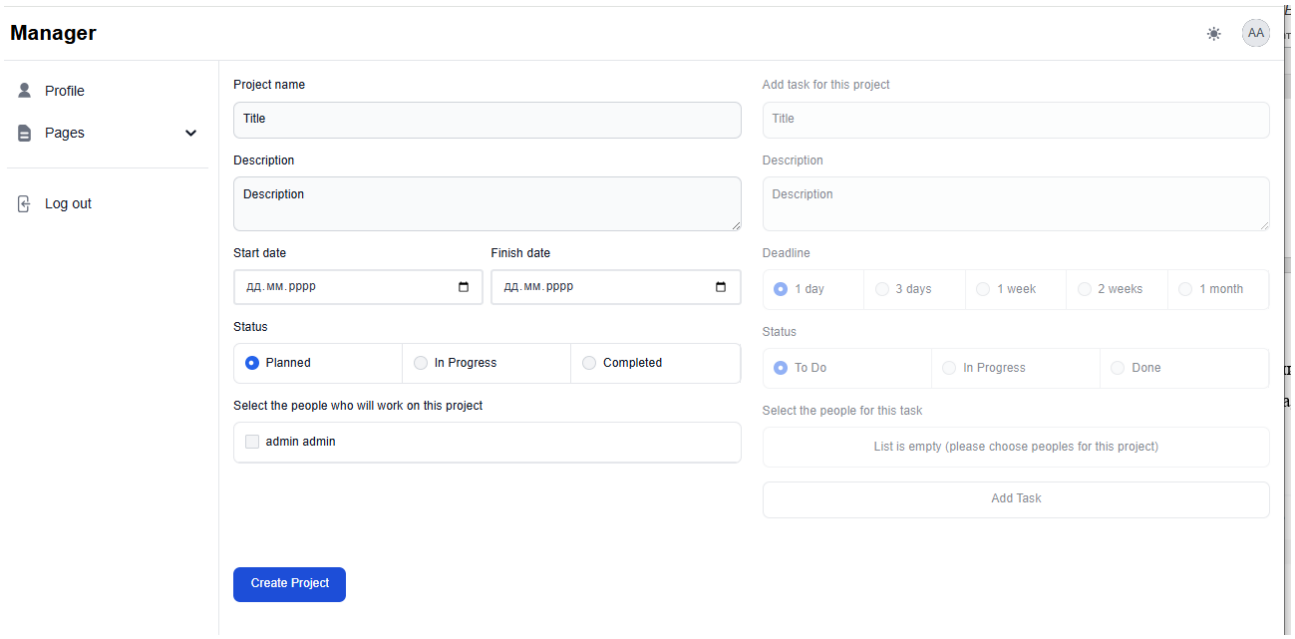


Рисунок 3.2 – Вікно додавання нового проєкту

Вікно додавання нового проєкту в цьому менеджері задач розділене на дві основні логічні зони: ліва частина присвячена загальним параметрам проєкту, а права — створенню першого завдання в межах цього проєкту. Назва та опис проєкту, Заголовок (Project name): Поле для введення назви проєкту. Опис (Description): Текстове поле для розширеної інформації про цілі або деталі проєкту. Дати (Start/Finish date): Два віконця з календарем для вибору дати початку та планованого завершення робіт. Статус (Status): Перемикачі (radio buttons) для вибору поточного стану: Planned (Заплановано), In Progress (В процесі) або Completed (Завершено).Команда (Select the people): Список користувачів з чекбоксами для призначення відповідальних (на скриншоті доступний лише "admin admin").

Додавання завдання. Ця частина дозволяє одразу додати першу задачу до новоствореного проєкту: Назва та опис задачі: Аналогічні поля для заголовка та деталей конкретного завдання.Дедлайн (Deadline): Швидкий вибір терміну виконання (1 день, 3 дні, 1 тиждень, 2 тижні або 1 місяць).Статус задачі: Варіанти

To Do, In Progress або Done. Вибір виконавців: Список людей для конкретної задачі (залежить від обраних людей для всього проєкту). Кнопка "Add Task": Підтверджує додавання задачі до форми.

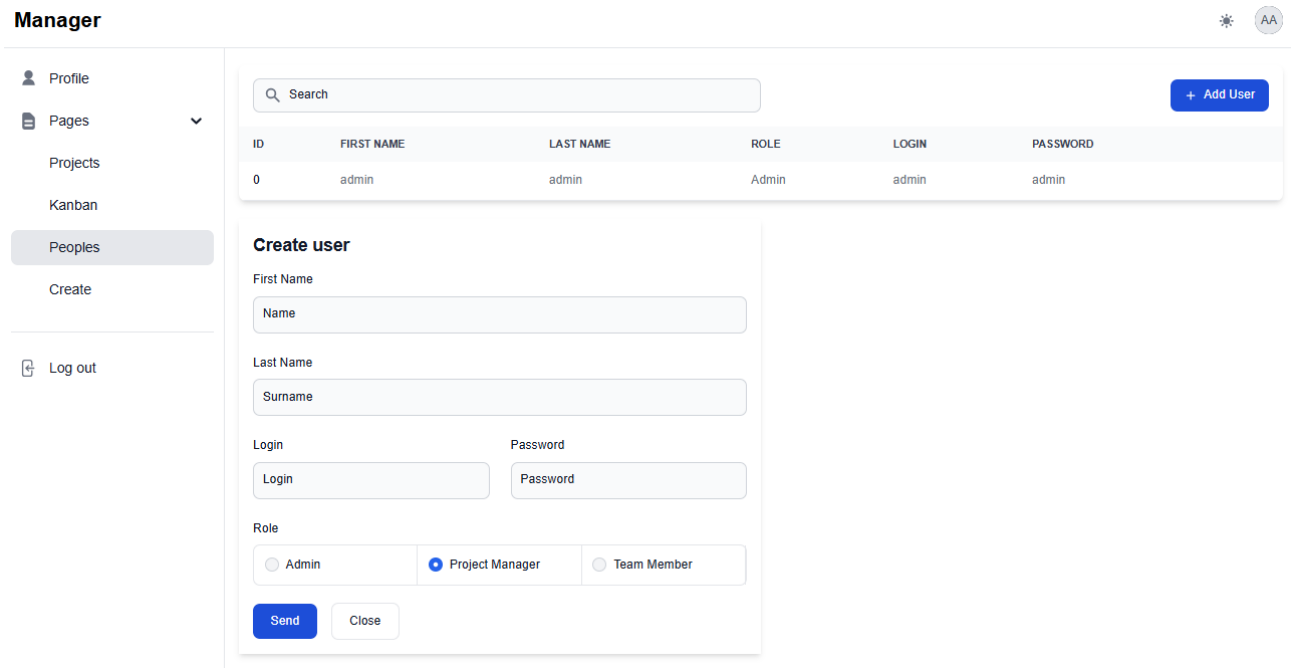


Рисунок 3.3 – Вікно переліку користувачів та додавання нового користувача

Інтерфейс розділу керування користувачами в системі менеджменту завдань. Вікно розділене на дві основні функціональні зони:

Перелік користувачів (Верхня частина)Це таблиця, яка відображає поточну базу даних користувачів: Пошук: Зверху розташоване поле Search для швидкої фільтрації списку. Таблиця: Має стовпці: ID, First Name (Ім'я), Last Name (Прізвище), Role (Роль), Login (Логін) та Password (Пароль).Дані: Наразі в базі є один запис — адміністратор із роллю Admin. Кнопка дії: Синя кнопка + Add User у правому верхньому куті, активує форму створення.

Форма створення нового користувача (Нижня частина)Блок Create user містить наступні поля для заповнення: Ім'я та Прізвище: Текстові поля First Name та Last Name. Облікові дані: Поля для введення Login та Password. Вибір ролі (Role): Перемикачі (radio buttons) для призначення прав доступу: AdminProject Manager (обрано за замовчуванням)Team Member

Кнопки керування:Send (синя) — для збереження даних та створення аккаунту.Close (біла) — для скасування дії або закриття форми.

ВИСНОВКИ

У ході виконання дипломної роботи було вирішено актуальну науково-практичну задачу — спроєктовано та програмно реалізовано клієнт-серверний вебзастосунок (Task Manager) для управління проєктами та задачами на базі методології Kanban із впровадженою системою контролю доступу.

В результаті проведеного дослідження та практичної розробки було досягнуто наступних результатів:

1. Проведено ґрунтовний аналіз предметної області. Досліджено еволюцію підходів до управління задачами та обґрунтовано ефективність використання методології Kanban для візуалізації робочого процесу. Проведено порівняльний аналіз існуючих на ринку комерційних рішень (Trello, Jira, Miro), який підтвердив доцільність створення власного, збалансованого продукту без зайвої функціональної перевантаженості.

2. Спроєктовано архітектуру системи. Сформовано об'єктну модель бази даних, виділено ключові сутності (Проєкт, Користувач, Задача) та визначено їхні атрибути. Обґрунтовано використання документо-орієнтованого підходу (JSON) для зберігання даних, що оптимізує швидкість взаємодії застосунку з сервером.

3. Обґрунтовано вибір інструментальних засобів. Доведено ефективність використання бібліотеки React та мови суворої типізації TypeScript для створення надійної клієнтської частини. Замість класичних складних патернів (на кшталт Redux) для управління глобальним станом успішно інтегровано легковагову бібліотеку Zustand.

4. Реалізовано систему безпеки (RBAC). Програмно імплементовано трирівневу модель доступу (Admin, Project Manager, Team Member). Забезпечено захист маршрутів на стороні клієнта, що гарантує конфіденційність корпоративних даних та унеможливлює несанкціоноване втручання в робочий процес.

5. Розроблено інтерактивний користувацький інтерфейс. За допомогою бібліотеки @hello-pangea/dnd імплементовано складну механіку Drag-and-Drop, яка синхронізується з базою даних (json-server) через REST API. Використання

фреймворку Tailwind CSS та препроцесора SCSS дозволило створити адаптивну, сучасну та масштабовану дизайн-систему.

6. Забезпечено розгортання застосунку. Фінальний програмний продукт (клієнтська та серверна частини) успішно розгорнуто на хмарній платформі Render, що підтверджує його готовність до експлуатації в реальних умовах.

Розроблений програмний продукт має високу практичну цінність. Він може бути ефективно впроваджений у робочі процеси малих та середніх IT-команд, стартапів, а також використаний незалежними розробниками для систематизації проєктів. У перспективі функціонал системи може бути розширений шляхом додавання мікросервісної архітектури, розширеного логування дій користувачів та інтеграції з корпоративними месенджерами.

Версія застосунку успішно розгорнута на платформі Render (https://task-manager-frontend-okw6.onrender.com/#/login); сам проєкт розміщений на платформі Github: https://github.com/s-kostiantyn/task-manager-frontend/tree/main та https://github.com/s-kostiantyn/task-manager-backend

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

2. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. O'Reilly Media, 2020. 706 p. URL: https://forum.freemdict.com/uploads/short-url/lI79nDbeiEq7d3M6TUUfZGEP6zc.pdf

3. Flowbite — UI Component Library based on Tailwind CSS. URL: https://flowbite.com/docs/getting-started/introduction/

4. Freeman A. Pro React 16. Apress, 2019. 750 p. URL: https://edu.anarcho-copy.org/Programming%20Languages/Node/Pro%20React%2016.pdf

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

6. MDN Web Docs: Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

7. Sommerville I. Software Engineering / I. Sommerville. – 10th ed., Global ed. – Harlow : Pearson Education Limited, 2016. – 796

8. Документація TypeScript: Typed JavaScript at Any Scale. URL: https://www.typescriptlang.org/docs/

9. Документація пакету react-router-dom. URL: https://reactrouter.com/en/main

10. Документація фреймворку Tailwind CSS. URL: https://tailwindcss.com/docs

11. Офіційна документація React: The library for web and native user interfaces. URL: https://react.dev/

12. Офіційна документація бібліотеки @hello-pangea/dnd. URL: https://github.com/hello-pangea/dnd

13. Офіційна документація бібліотеки Zustand. URL: https://docs.pmnd.rs/zustand/getting-started/introduction

14. Офіційна документація платформи Render (Cloud Application Hosting). URL: https://render.com/docs

15. Специфікація та посібник з використання json-server. URL: https://github.com/typicode/json-server

16. Trello URL: https://trello.com/

17. Jira URL: https://www.atlassian.com/software/jira

18. Miro URL: https://miro.com/

Додатки

Додаток А

Лістинг коду TasksPage.tsx

task-manager-frontend/src/modules/TasksPage/TasksPage.tsx

```
import React, { useState, useEffect } from "react";
import { TasksBoardList } from "./components/TasksBoardList";
import { Filter } from "../shared/Filter";
import { TaskStatuses } from "../../types/TaskStatuses";
import { useProjectStore } from "../../stores/projectsStore";
import { GoCreate } from "../shared/GoCreate";

export const TasksPage: React.FC = () => {
  const [filterBy, setFilterBy] = useState<TaskStatuses | null>(null);
  const [sortBy, setSortBy] = useState(false);

  const statuses = Object.values(TaskStatuses);

  const { fetchProjects, loading, error, projects } = useProjectStore();

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

  return (
    <>
      <Filter
        filterBy={filterBy}
        statuses={statuses}
        sortBy={sortBy}
        setFilterBy={setFilterBy}
        setSortBy={setSortBy}
      />

      <div className="tasks__wrapper">
        {loading && <div className="message">loading...</div>}
        {!loading && error && <div className="message">{error}</div>}
        {!loading && !error && !projects.length && (
          <div className="message">
            <p className="dark:text-gray-200">You have no projects</p>
            <GoCreate />
          </div>
        )}
        {!loading && !error && projects.length !== 0 && (
          <TasksBoardList
            projects={projects}
            filterBy={filterBy}
            sortBy={sortBy}
```

```
          />
        )}
      </div>
    </>
  );
};
```

Додаток Б

Лістинг коду ProjectsPage.tsx

task-manager-frontend/src/modules/ProjectsPage/ProjectsPage.tsx

```
import React, { useState, useEffect } from "react";
import { ProjectList } from "./components/ProjectsList";
import { Project } from "../../types/Project";
import { Filter } from "../shared/Filter";
import { Statuses } from "../../types/Statuses";
import { getPreparedDatas } from "../../services/getPreparedDatas";
import { useProjectStore } from "../../stores/projectsStore";
import { GoCreate } from "../shared/GoCreate";

export const ProjectsPage: React.FC = () => {
  const { projects, fetchProjects, loading, error } = useProjectStore();

  const [filterBy, setFilterBy] = useState<Statuses | null>(null);
  const [sortBy, setSortBy] = useState(false);

  const statuses = Object.values(Statuses);

  const preparedProjects = getPreparedDatas<Project[], Statuses | null>(
    projects,
    {
      filterBy,
      sortBy,
    }
  );

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

  return (
    <section className="">
      <header className="flex gap-4 items-start flex-col justify-between md:flex-row
md:items-center">
        <div className="">
          <Filter
            filterBy={filterBy}
            statuses={statuses}
```

```
          setFilterBy={setFilterBy}
          sortBy={sortBy}
          setSortBy={setSortBy}
        />
       </div>
       <p className="text-gray-600 dark:text-gray-300 whitespace-nowrap">
        Projects: {projects.length}
       </p>
      </header>

      {loading && <div className="message">loading...</div>}
      {!loading && error && <div className="message">{error}</div>}
      {!loading && !error && !projects.length && (
       <div className="message">
        <p className="dark:text-gray-200">You have no projects</p>
        <GoCreate />
       </div>
      )}
      {!loading && !error && projects.length !== 0 && (
       <ProjectList projects={preparedProjects} />
      )}
     </section>
  );
};
```

Додаток В

Лістинг коду PeoplesPage.tsx

task-manager-frontend/src/modules/PeoplesPage/PeoplesPage.tsx

```
import React, { useEffect, useState, useCallback } from "react";
import debounce from "lodash.debounce";
import { Outlet, Link } from "react-router-dom";
import { TablePeoples } from "./components/TablePeoples";
import { usePeopleStore } from "../../stores/peoplesStore";
import { People } from "../../types/People";

function getSortedUsers(
  users: People[],
  { appliedQuery }: { appliedQuery: string }
) {
  let sortedUser = [...users];
  const normilizeQuery = appliedQuery.toLowerCase().trim();

  if (appliedQuery) {
    sortedUser = sortedUser.filter((u) => {
      const name = u.firstName.toLowerCase().trim();
      const surname = u.lastName.toLowerCase().trim();
      const login = u.login.toLowerCase().trim();
```

```
        const role = u.role.toLowerCase().trim();

        return (
          name.includes(normilizeQuery) ||
          surname.includes(normilizeQuery) ||
          login.includes(normilizeQuery) ||
          role.includes(normilizeQuery)
        );
      });
    }

    return sortedUser;
  }

  export const PeoplesPage: React.FC = () => {
    const { loading, error, users, fetchUsers } = usePeopleStore();
    const [query, setQuery] = useState("");
    const [appliedQuery, setAppliedQuery] = useState("");
    const [isLoading, setIsLoading] = useState(false);

    const applyQuery = useCallback(
      debounce((v: string) => {
        setAppliedQuery(v);
        setIsLoading(false);
      }, 1000),
      []
    );

    const sortedUsers = getSortedUsers(users, { appliedQuery });

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

    const handleQueryChange = (e: React.ChangeEvent<HTMLInputElement>) => {
      setQuery(e.target.value);
      setIsLoading(true);
      applyQuery(e.target.value);
    };

    return (
      <section>
        <div className="bg-white dark:bg-gray-800 relative shadow-md sm:rounded-lg overflow-hidden">
          <div className="flex flex-col md:flex-row items-center justify-between space-y-3 md:space-y-0 md:space-x-4 p-4">
            <div className="w-full md:w-1/2">
              <form className="flex items-center">
                <label htmlFor="simple-search" className="sr-only">
                  Search
                </label>
                <div className="relative w-full">
```

```
                <div className="absolute inset-y-0 left-0 flex items-center pl-3 pointer-events-none">
                  <svg
                    aria-hidden="true"
                    className="w-5 h-5 text-gray-500 dark:text-gray-400"
                    fill="currentColor"
                    viewBox="0 0 20 20"
                    xmlns="http://www.w3.org/2000/svg"
                  >
                    <path
                      fillRule="evenodd"
                      d="M8 4a4 4 0 100 8 4 4 0 000-8zM2 8a6 6 0 1110.89 3.476l4.817 4.817a1 1 0 01-1.414 1.414l-4.816-4.816A6 6 0 012 8z"
                      clipRule="evenodd"
                    />
                  </svg>
                </div>
                <input
                  type="text"
                  id="simple-search"
                  className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg focus:ring-primary-500 focus:border-primary-500 block w-full pl-10 p-2 dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white dark:focus:ring-primary-500 dark:focus:border-primary-500"
                  placeholder="Search"
                  value={query}
                  onChange={handleQueryChange}
                  required
                />
              </div>
            </form>
          </div>

          <Link
            to="create"
            type="button"
            className="flex items-center justify-center text-white bg-primary-700 hover:bg-primary-800 focus:ring-4 focus:ring-primary-300 font-medium rounded-lg text-sm px-4 py-2 dark:bg-primary-600 dark:hover:bg-primary-700 focus:outline-none dark:focus:ring-primary-800"
          >
            <svg
              className="h-3.5 w-3.5 mr-2"
              fill="currentColor"
              viewBox="0 0 20 20"
              xmlns="http://www.w3.org/2000/svg"
              aria-hidden="true"
            >
              <path
                clipRule="evenodd"
                fillRule="evenodd"
                d="M10 3a1 1 0 011 1v5h5a1 1 0 110 2h-5v5a1 1 0 11-2 0v-5H4a1 1 0 110-2h5V4a1 1 0 011-1z"
```

```
                />
              </svg>
              Add User
            </Link>
          </div>
          <div className="relative overflow-x-auto">
            {loading && (
              <div className="px-4 py-3 font-medium text-gray-400">
                loading...
              </div>
            )}
            {!loading && error && (
              <div className="px-4 py-3 font-medium text-gray-400">{error}</div>
            )}
            {!loading && !error && !sortedUsers.length && (
              <div className="px-4 py-3 font-medium text-gray-400">
                Have no peoples
              </div>
            )}
            {!loading && !error && sortedUsers.length !== 0 && (
              <>
                <TablePeoples peoples={sortedUsers} />

                {isLoading && (
                  <div className="loading dark:bg-gray-800 opacity-50">
                    <div
                      role="status"
                      className="absolute -translate-x-1/2 -translate-y-1/2 top-2/4 left-1/2"
                    >
                      <svg
                        aria-hidden="true"
                        className="w-8 h-8 text-gray-200 animate-spin dark:text-gray-600 fill-blue-
600"
                        viewBox="0 0 100 101"
                        fill="none"
                        xmlns="http://www.w3.org/2000/svg"
                      >
                        <path
                          d="M100 50.5908C100 78.2051 77.6142 100.591 50 100.591C22.3858
100.591 0 78.2051 0 50.5908C0 22.9766 22.3858 0.59082 50 0.59082C77.6142 0.59082 100
22.9766 100 50.5908ZM9.08144 50.5908C9.08144 73.1895 27.4013 91.5094 50 91.5094C72.5987
91.5094 90.9186 73.1895 90.9186 50.5908C90.9186 27.9921 72.5987 9.67226 50 9.67226C27.4013
9.67226 9.08144 27.9921 9.08144 50.5908Z"
                          fill="currentColor"
                        />
                        <path
                          d="M93.9676 39.0409C96.393 38.4038 97.8624 35.9116 97.0079
33.5539C95.2932 28.8227 92.871 24.3692 89.8167 20.348C85.8452 15.1192 80.8826 10.7238
75.2124 7.41289C69.5422 4.10194 63.2754 1.94025 56.7698 1.05124C51.7666 0.367541 46.6976
0.446843 41.7345 1.27873C39.2613 1.69328 37.813 4.19778 38.4501 6.62326C39.0873 9.04874
41.5694 10.4717 44.0505 10.1071C47.8511 9.54855 51.7191 9.52689 55.5402 10.0491C60.8642
10.7766 65.9928 12.5457 70.6331 15.2552C75.2735 17.9648 79.3347 21.5619 82.5849
```

```
25.841C84.9175 28.9121 86.7997 32.2913 88.1811 35.8758C89.083 38.2158 91.5421 39.6781
93.9676 39.0409Z"
                        fill="currentFill"
                      />
                    </svg>
                    <span className="sr-only">Loading...</span>
                  </div>
                </div>
              )}
            </>
          )}
        </div>
      </div>

      <Outlet />
    </section>
  );
};
```