ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: « Розробка інформаційної системи управління ІТ-проєктами»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем КОЛОМІЄЦЬ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Артем КОЛОМІЄЦЬ

Керівник: Юрій ЗАДОНЦЕВ
канд. техн. наук

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коломійцю Артему Григоровичу

1. Тема кваліфікаційної роботи: « Розробка інформаційної системи управління IT-проєктами»
керівник кваліфікаційної роботи канд. техн. наук Юрій ЗАДОНЦЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи та засоби управління IT-проєктами, опис принципів Kanban-методології та рольового контролю доступу, технічна документація з описом сучасних вебтехнологій для розробки інформаційних систем, зокрема Next.js, TypeScript, Prisma ORM, SQLite, Auth.js, Tailwind CSS, shadcn/ui, @dnd-kit та Recharts.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз предметної галузі управління IT-проєктами, а також дослідження існуючих програмних рішень для організації командної роботи, планування задач і моніторингу виконання проєктів.

2. Визначення функціональних і нефункціональних вимог до інформаційної системи управління IT-проєктами з підтримкою Kanban-дошки, рольового контролю доступу, управління проєктами, задачами, користувачами та запрошеннями.
3. Проєктування архітектури інформаційної системи управління IT-проєктами, побудова діаграм варіантів використання, діаграми діяльності, структури клієнтської та серверної частин, а також ER-діаграми бази даних.
4. Програмна реалізація інформаційної системи управління IT-проєктами з використанням сучасних вебтехнологій: Next.js, TypeScript, Prisma ORM, SQLite, Auth.js, Tailwind CSS, shadcn/ui, @dnd-kit та Recharts.
5. Тестування розробленої інформаційної системи за основними сценаріями: реєстрація та автентифікація користувачів, рольова модель доступу, управління проєктами, механізм запрошень, робота з задачами та Kanban-дошкою.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Формування вимог до інформаційної системи управління IT-проєктами	08.03-19.03.2026	
4	Проєктування архітектури системи, Kanban-дошки та бази даних	20.03-29.03.2026	
5	Програмна реалізація застосунку	30.03-12.04.2026	
6	Тестування застосунку	13.04- 25.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	26.04-05.05.2026	

8	Розробка демонстраційних матеріалів	06.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	
10	Подання кваліфікаційної роботи	23.05-29.05.2026	

Здобувач вищої освіти _____ Артем Коломієць
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 0 табл., 25 рис., 25 джерел.

Мета роботи – покращення процесів планування, організації та контролю виконання ІТ-проєктів за рахунок розробки інформаційної системи з підтримкою Kanban-методології, рольового контролю доступу та інструментів командної роботи.

Об'єкт дослідження – процес управління ІТ-проєктами та задачами в командному середовищі розробки програмного забезпечення.

Предмет дослідження – методи та засоби розробки інформаційної системи управління ІТ-проєктами з використанням сучасних вебтехнологій, Kanban-дошки та рольової моделі доступу.

Короткий зміст роботи: В роботі проаналізовано підходи та програмні засоби управління ІТ-проєктами в контексті організації командної роботи, планування задач і контролю їх виконання. Проаналізовано інструментальні засоби для управління ІТ-проєктами: Jira, Trello, Asana. Розроблено архітектуру інформаційної системи та програмно реалізовані ключові функціональні можливості, зокрема: реєстрація й автентифікація користувачів, управління проєктами, задачами та учасниками, запрошення користувачів, Kanban-дошка з drag-and-drop, пошук і інформаційна панель зі статистикою. Проведено функціональне тестування системи. В роботі використано Next.js, TypeScript, Prisma ORM, SQLite, Auth.js, Tailwind CSS, shadcn/ui, @dnd-kit та Recharts.

Сферою використання застосунку є організація командної роботи невеликих ІТ-команд, навчальних груп і студентських проєктів, де необхідно забезпечити планування задач, контроль їх виконання, розподіл ролей між учасниками та наочне відстеження прогресу проєкту.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, УПРАВЛІННЯ ІТ-ПРОЄКТАМИ, KANBAN, RBAC, NEXT.JS, TYPESCRIPT, PRISMA, SQLITE, AUTH.JS, DRAG-AND-DROP, WEB APPLICATION.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface, інтерфейс програмування застосунків.
Auth.js - бібліотека автентифікації для Next.js, також відома як NextAuth.js v5.
CUID - Collision-resistant Unique Identifier, унікальний ідентифікатор, стійкий до колізій.
CDN - Content Delivery Network, мережа доставки контенту.
CSR - Client-Side Rendering, клієнтський рендеринг.
CSS - Cascading Style Sheets, каскадні таблиці стилів.
CRUD - Create, Read, Update, Delete, основні операції з даними.
БД - база даних.
ER-діаграма - Entity-Relationship diagram, діаграма «сутність-зв'язок».
HTML - HyperText Markup Language, мова розмітки гіпертексту.
HTTP - HyperText Transfer Protocol, протокол передачі гіпертексту.
HTTPS - HyperText Transfer Protocol Secure, захищений протокол передачі гіпертексту.
IT - інформаційні технології.
JS - JavaScript, мова програмування для вебзастосунків.
JSON - JavaScript Object Notation, формат обміну даними.
JWT - JSON Web Token, токен для автентифікації та авторизації.
ORM - Object-Relational Mapping, об'єктно-реляційне відображення.
RBAC - Role-Based Access Control, рольова модель управління доступом.
REST - Representational State Transfer, архітектурний стиль вебсервісів.
SQL -Structured Query Language, мова структурованих запитів.
SSG - Static Site Generation, статична генерація сторінок.
SSR - Server-Side Rendering, серверний рендеринг.
SSL - Secure Sockets Layer, протокол захищеного з'єднання.
SVG - Scalable Vector Graphics, масштабована векторна графіка.
TS - TypeScript, типізована мова програмування на основі JavaScript.
UI - User Interface, інтерфейс користувача.
URL - Uniform Resource Locator, уніфікований локатор ресурсів.
UX - User Experience, досвід користувача.

ЗМІСТ

ВСТУП

Актуальність: управління ІТ-проєктами є важливою складовою сучасної розробки програмного забезпечення, оскільки саме від ефективної організації командної роботи, планування задач і контролю їх виконання залежить успішність реалізації проєкту. Зі зростанням складності програмних продуктів та поширенням командної роботи постає потреба в зручних інформаційних системах, які дозволяють структурувати робочий процес, розподіляти ролі між учасниками та відстежувати стан виконання задач. Використання вебзастосунків у цьому процесі дозволяє організувати управління проєктами в єдиному цифровому середовищі, забезпечити доступність даних, візуалізацію прогресу та оперативну взаємодію між учасниками команди.

Інформаційна система управління ІТ-проєктами виступає як універсальний інструмент для підтримки процесів планування, координації та контролю виконання задач. Вона об'єднує в собі створення проєктів, управління задачами, Kanban-дошку, рольову модель доступу, механізм запрошень користувачів, пошук та інформаційну панель зі статистикою. Цифрові інструменти дозволяють користувачам створювати робочі простори, організовувати команду, призначати виконавців, контролювати статус задач і наочно відстежувати загальний прогрес роботи. Завдяки використанню Kanban-методології, рольового контролю доступу та сучасних вебтехнологій процес управління ІТ-проєктами стає більш структурованим, прозорим і зручним для невеликих команд та навчальних проєктів.

Об'єктом дослідження є процес управління ІТ-проєктами та задачами в командному середовищі розробки програмного забезпечення.

Предметом дослідження є програмні засоби, що забезпечують підтримку процесів планування, організації, контролю виконання ІТ-проєктів та управління доступом користувачів із використанням сучасних вебтехнологій.

Метою роботи є покращення процесів планування, організації та контролю виконання ІТ-проєктів за рахунок розробки інформаційної системи з підтримкою Kanban-методології, рольового контролю доступу та інструментів командної роботи.

Методи дослідження: першим етапом дослідження стало ознайомлення з особливостями управління IT-проєктами, принципами Kanban-методології та підходами до рольового контролю доступу. Це дозволило сформулювати функціональні та нефункціональні вимоги до майбутньої інформаційної системи. Далі було проведено аналіз існуючих програмних рішень для управління проєктами, зокрема Jira, Trello та Asana, що дало змогу визначити їхні переваги, недоліки та обґрунтувати потребу у створенні власного рішення. Для реалізації системи було обрано фреймворк Next.js, мову програмування TypeScript, бібліотеку React для побудови інтерфейсу, Prisma ORM для роботи з базою даних SQLite, Auth.js для автентифікації користувачів, Tailwind CSS та shadcn/ui для стилізації інтерфейсу, @dnd-kit для реалізації drag-and-drop функціональності Kanban-дошки та Recharts для візуалізації статистичних даних. Архітектуру програмного забезпечення спроєктовано з використанням діаграм, що відображають структуру системи, маршрути, логіку обробки запитів та модель бази даних. Реалізацію функціоналу здійснено поетапно з подальшим ручним функціональним тестуванням основних сценаріїв роботи системи.

Наукова новизна роботи полягає у розробці інформаційної системи, що поєднує в межах одного вебзастосунку Kanban-дошку з drag-and-drop взаємодією, рольовий контроль доступу, механізм запрошень користувачів за токеном, автоматичне визначення пріоритету задач на основі дедлайну та інформаційну панель для відстеження стану проєктів. Особливістю системи є використання ізольованих робочих просторів у вигляді фірм, що дозволяє кожному користувачу створювати власне середовище для управління проєктами та запрошувати до нього учасників. Перевірка прав доступу реалізована на рівні серверної логіки, що підвищує надійність і безпеку роботи системи.

Практична значущість результатів полягає у можливості використання створеної інформаційної системи невеликими IT-командами, навчальними групами та студентськими проєктами для організації командної роботи, планування задач, контролю їх виконання та розподілу ролей між учасниками. Система реалізована як вебзастосунок, що може бути запущений у локальному середовищі розробки та використовує SQLite без необхідності налаштування окремого сервера бази даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі управління ІТ-проєктами

Управління ІТ-проєктами є однією з ключових дисциплін сучасної індустрії програмного забезпечення. Зі зростанням масштабів та складності програмних систем організації дедалі частіше стикаються з потребою у структурованому підході до планування, виконання та контролю проєктів. Ефективне управління ІТ-проєктами дозволяє скоротити витрати, дотриматись термінів і підвищити якість кінцевого продукту [1].

Управління проєктами (англ. project management) - це сукупність методів, інструментів і практик, спрямованих на досягнення визначених цілей у встановлені терміни та в рамках виділеного бюджету. У контексті ІТ-галузі ця дисципліна набуває особливої актуальності, оскільки програмні проєкти характеризуються високим ступенем невизначеності, частими змінами вимог та необхідністю координації роботи розподілених команд [2].

Сучасний підхід до управління ІТ-проєктами ґрунтується переважно на гнучких (agile) методологіях, таких як Scrum та Kanban. На відміну від традиційного каскадного підходу (Waterfall), agile-методології передбачають ітеративну розробку, тісну взаємодію з замовником та швидку реакцію на зміни вимог. Kanban, зокрема, фокусується на візуалізації робочого процесу, обмеженні незавершеної роботи та безперервному вдосконаленні потоку виконання задач [3].

У Kanban-підході робочий процес подається у вигляді дошки, на якій задачі розміщуються відповідно до поточного етапу виконання. Такий спосіб організації дає змогу швидко оцінити, які задачі ще не розпочато, які перебувають у роботі, а які вже завершено. У межах розроблюваної системи цей принцип реалізовано через три основні стани задач: To Do, In Progress та Done. Переміщення задач між цими станами відображає зміну їхнього статусу та допомагає команді контролювати загальний хід виконання проєкту [4].

Важливою складовою управління IT-проєктами є розмежування ролей учасників. У межах проєктної команди виокремлюють такі основні ролі:

– адміністратор фірми (ADMIN) - має повний доступ до всіх ресурсів фірми, може керувати користувачами, проєктами та задачами незалежно від свого членства в проєкті;

– менеджер (MANAGER) - відповідає за планування та організацію роботи в рамках конкретного проєкту, визначає задачі та призначає виконавців;

– учасник (MEMBER) - безпосередньо виконує задачі, має обмежений доступ до проєктів, у яких є учасником, і може працювати із задачами відповідно до наданих прав у межах проєкту.

Чітке визначення прав доступу для кожної ролі є необхідною умовою забезпечення безпеки та цілісності даних у системі управління проєктами. Концепція рольового управління доступом (Role-Based Access Control, RBAC) широко застосовується у сучасних інформаційних системах для обмеження дій користувачів відповідно до їхніх функціональних обов'язків [5].

Управління задачами є центральним процесом у будь-якій системі управління проєктами. Задача (task) - це одиниця роботи, яка має визначену назву, опис, статус виконання, пріоритет та, за необхідності, дедлайн і відповідального виконавця. Статус задачі відображає її поточний стан у процесі виконання, а пріоритет дозволяє команді зосередитися на найважливіших завданнях у першу чергу [2].

Пріоритизація задач є одним із ключових інструментів ефективного управління часом та ресурсами команди. Автоматичне визначення пріоритету на основі терміну виконання (дедлайну) дозволяє знизити ризик прострочення критичних задач. Зокрема, задачі з дедлайном, що наближається, автоматично отримують підвищений пріоритет, що сигналізує команді про необхідність їхнього першочергового виконання [3].

Інформаційна система управління IT-проєктами - це програмний засіб, що автоматизує процеси планування, координації та контролю виконання проєктів. Вона забезпечує єдиний простір для взаємодії учасників команди, зберігання та

відображення актуального стану проєкту і задач, а також надає аналітичні інструменти для оцінки прогресу роботи [1].

Основні функціональні вимоги до такої системи охоплюють: реєстрацію та автентифікацію користувачів; управління проєктами та членством у них; створення, редагування і видалення задач; переміщення задач між статусами за допомогою drag-and-drop на Kanban-дошці; пошук по проєктах і задачах; відображення зведеної статистики на інформаційній панелі (dashboard); а також диференціацію прав доступу відповідно до ролі користувача.

Таким чином, управління IT-проєктами є комплексною дисципліною, що вимагає застосування спеціалізованих програмних засобів. Розробка інформаційної системи, орієнтованої на підтримку Kanban-методології з чіткою моделлю рольового доступу, є актуальним та практично значущим завданням, що відповідає сучасним потребам IT-команд.

1.2 Аналіз існуючих програмних рішень

На сучасному ринку програмного забезпечення існує широкий спектр інструментів для управління IT-проєктами. Для визначення функціональних вимог до розроблюваної системи та виявлення її конкурентних переваг необхідно провести аналіз найбільш поширених рішень у цій сфері. До розгляду обрано три найбільш відомі системи: Jira, Trello та Asana [6].

1.2.1 Jira

Jira - це потужна система управління проєктами, розроблена компанією Atlassian та широко використовувана в IT-командах по всьому світу. Вона орієнтована переважно на команди, що працюють за agile-методологіями Scrum і Kanban, та пропонує розширені інструменти для планування спринтів, відстеження помилок і управління беклогом [7].

Основні функції Jira включають:

– гнучке налаштування робочих процесів (workflows) із визначенням власних статусів і переходів між ними;

– управління беклогом та плануванням спринтів у Scrum-проєктах;

– Kanban-дошки з підтримкою перетягування задач між стовпцями;

– розгалужена система звітності та аналітики (burn-down charts, velocity charts);

– інтеграція з репозиторіями коду (GitHub, GitLab, Bitbucket);

– гнучке управління правами доступу на рівні проєкту.

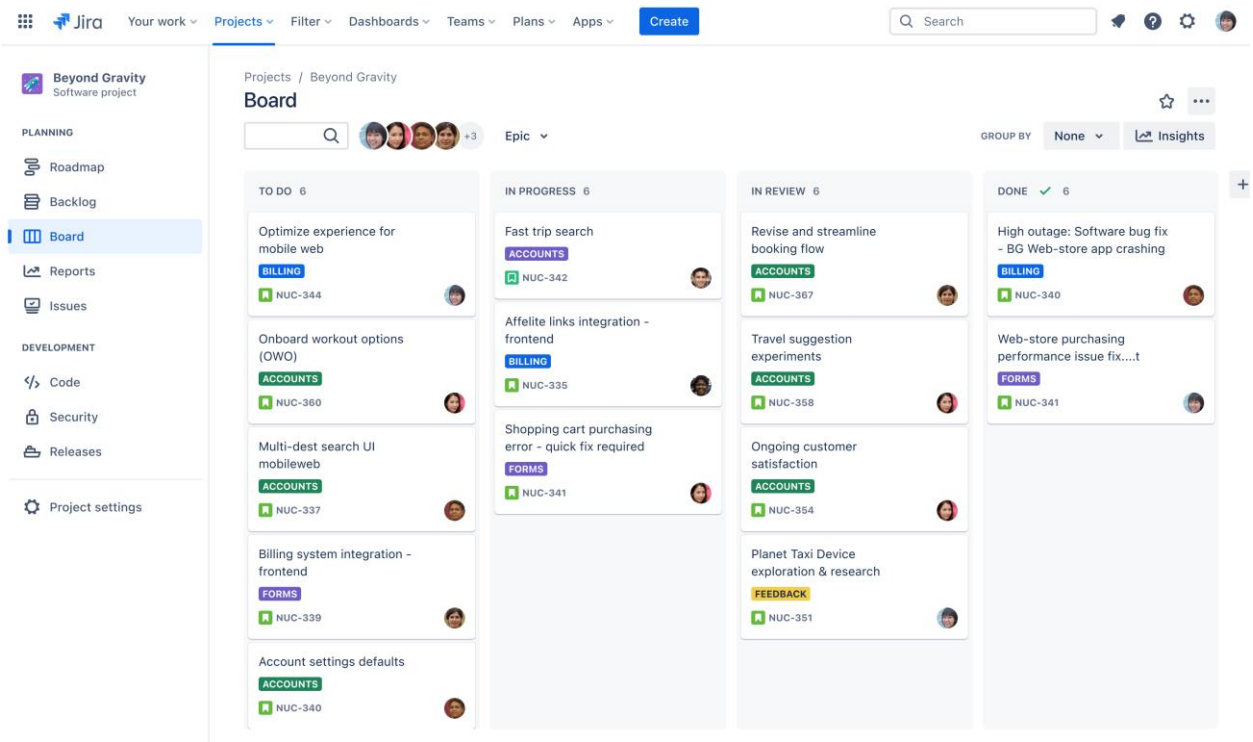


Рис. 1.1. Приклад Kanban-дошки у системі Jira

Переваги Jira полягають у широких можливостях налаштування, потужній аналітиці та великій екосистемі інтеграцій. Проте система має суттєві недоліки: складний і перевантажений інтерфейс, значна крива навчання для нових користувачів, а також висока вартість для комерційного використання. Для

невеликих команд або навчальних проєктів функціональність Jira є надлишковою [7].

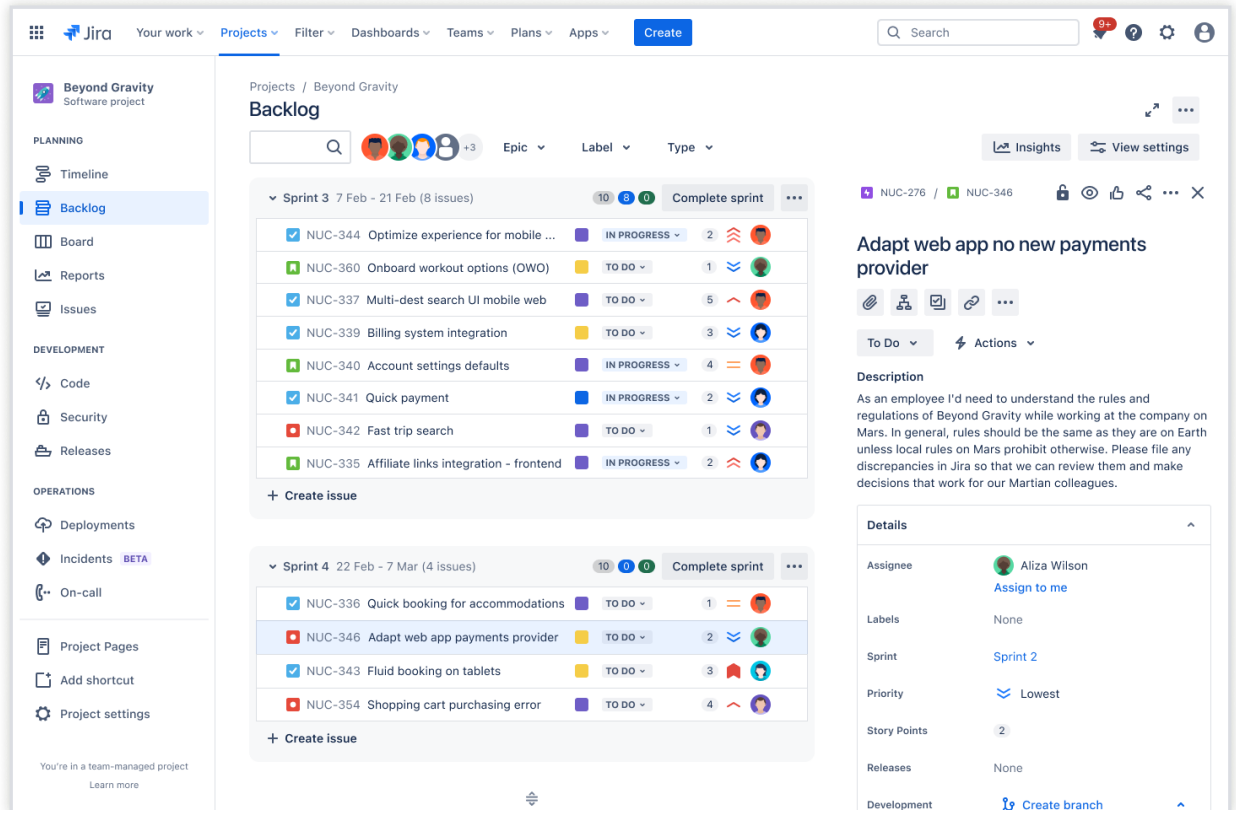


Рис. 1.2. Сторінка управління беклогом у системі Jira

1.2.2 Trello

Trello - це хмарний інструмент управління задачами від компанії Atlassian, побудований на основі Kanban-методології. Система вирізняється простотою та інтуїтивністю інтерфейсу, що робить її популярним вибором серед невеликих команд і для особистої організації задач [8].

Основні функції Trello включають:

– візуальні дошки (boards) з картками (cards), що представляють задачі;
– перетягування карток між стовпцями для зміни їхнього статусу;
– прикріплення файлів, коментарів і чеклістів до карток;
– встановлення дедлайнів і призначення відповідальних;
– розширення функціональності через Power-Ups (плагіни).

Рис. 1.3. Приклад дошки у системі Trello

Головною перевагою Trello є простота використання та швидке освоєння. Водночас система суттєво обмежена у можливостях звітності та аналітики, не підтримує розгалужену ієрархію задач і рольовий контроль доступу на рівні окремих задач. Відсутність вбудованої системи пріоритетів і обмежені можливості управління командами знижують її придатність для управління складними IT-проєктами [8].

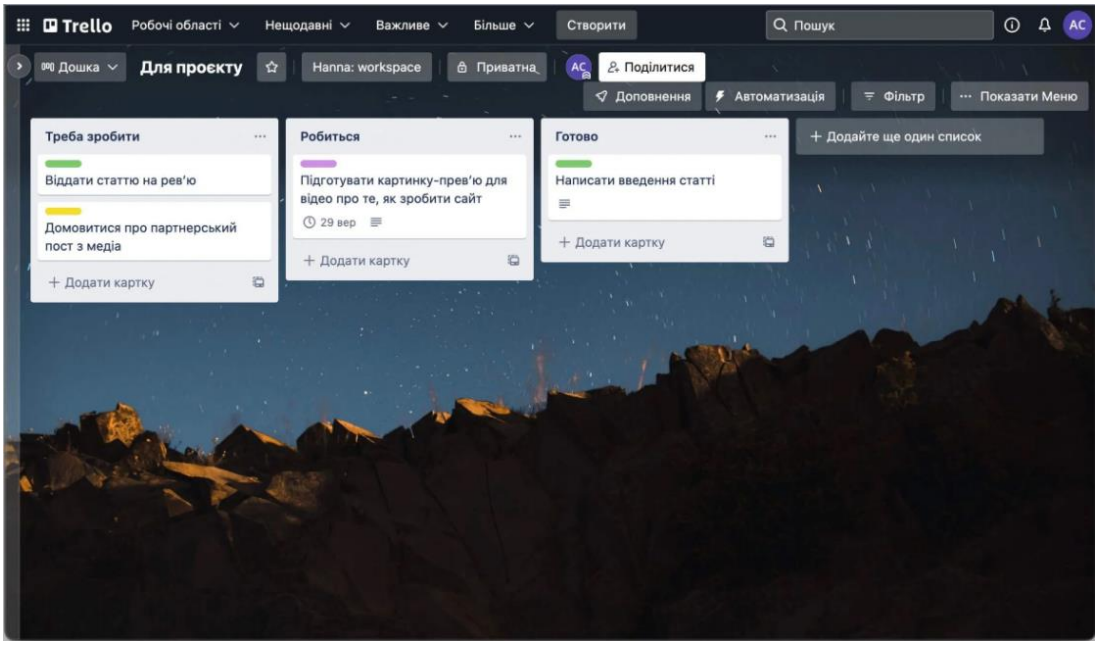


Рис. 1.4. Приклад картки задачі у системі Trello

1.2.3 Asana

Asana - це платформа управління роботою команди, розроблена однойменною компанією. Вона позиціонується як універсальний інструмент для управління проєктами будь-якого масштабу і пропонує кілька варіантів відображення задач: список, дошка, часова шкала та календар [9].

Основні функції Asana включають:

– управління задачами та підзадачами з можливістю встановлення залежностей;

– Kanban-дошки та часові шкали (timeline / Gantt);

– призначення відповідальних, встановлення пріоритетів і дедлайнів;

– автоматизація повторюваних процесів за допомогою правил;

– розгалужена система звітності та портфельного управління проєктами.

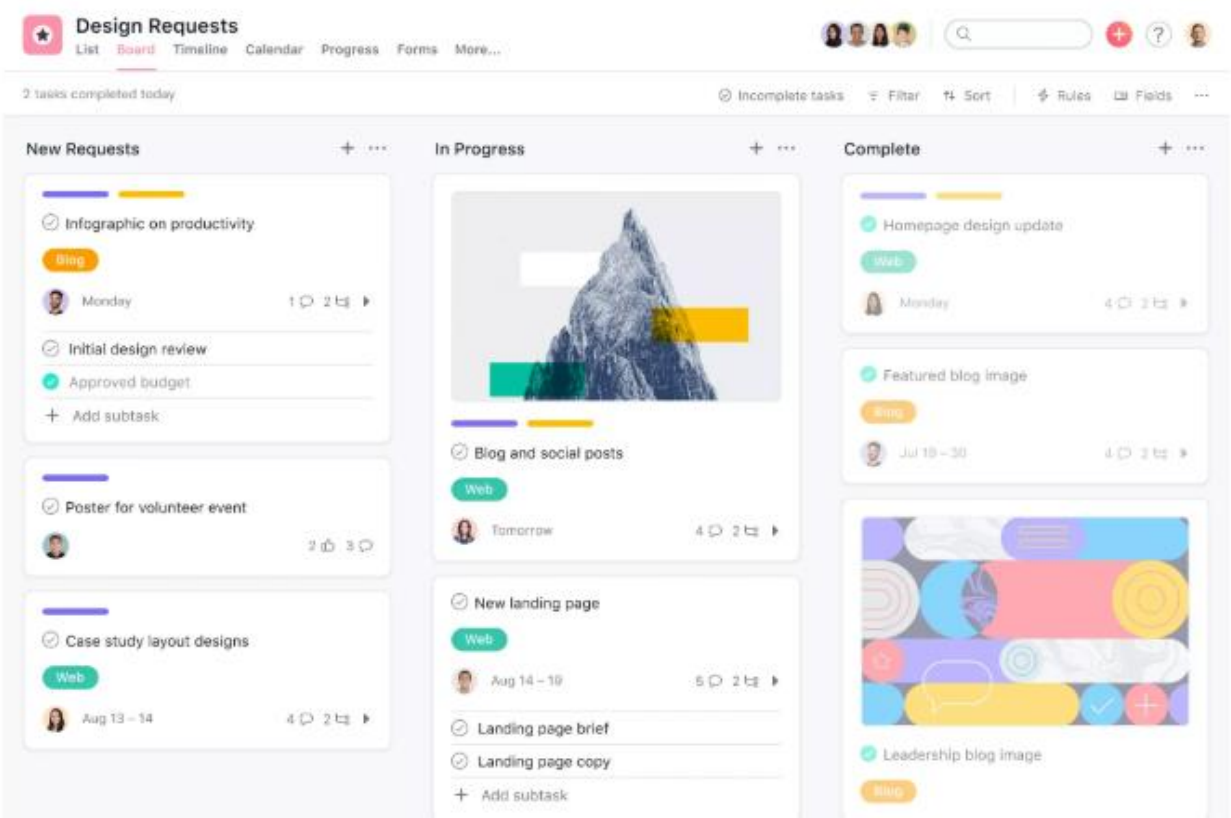


Рис. 1.5. Приклад Kanban-дошки у системі Asana

Asana відрізняється більш зручним інтерфейсом порівняно з Jira та ширшими можливостями порівняно з Trello. Проте повний функціонал доступний лише у платних тарифних планах, а безкоштовна версія має обмежену кількість учасників і проєктів. Крім того, відсутня глибока інтеграція з системами контролю версій коду, що є важливим для розробників [9].

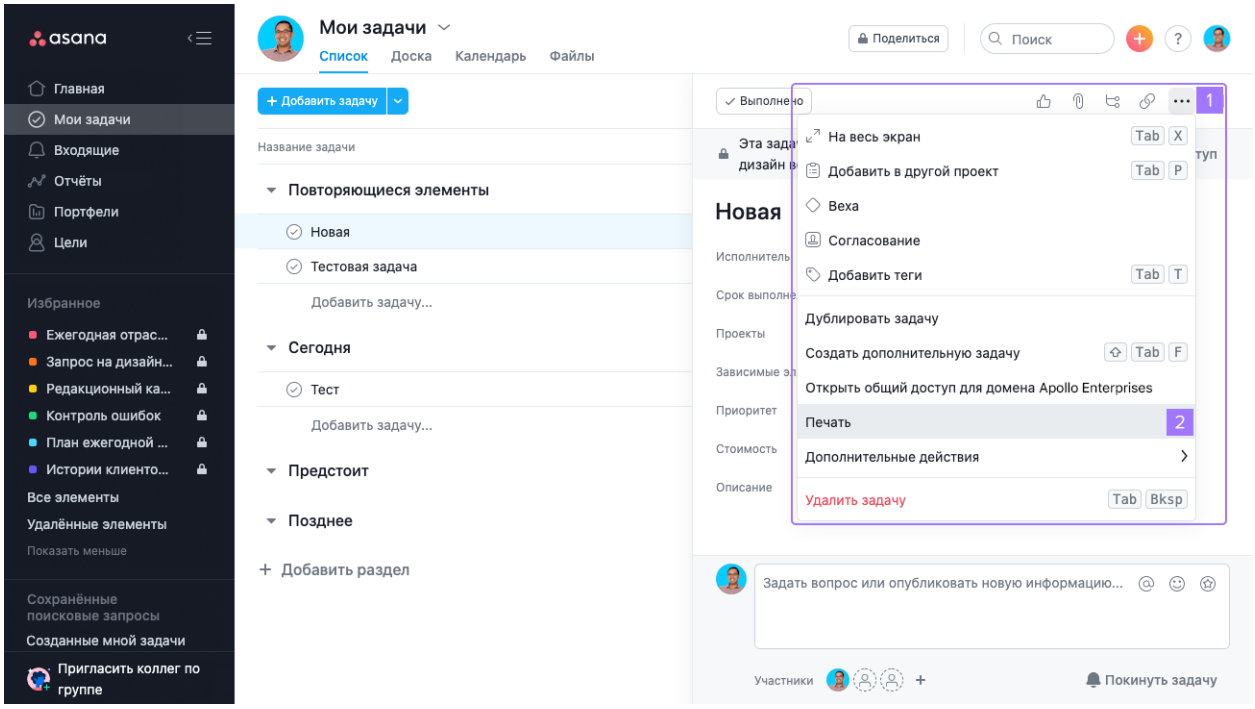


Рис. 1.6. Відображення задач у вигляді списку у системі Asana

1.2.4 Порівняльний аналіз існуючих рішень

За результатами аналізу розглянутих систем можна визначити їхні ключові характеристики та виявити загальні тенденції на ринку інструментів управління IT-проєктами. Jira є найбільш функціонально насиченою системою, однак складна у налаштуванні та дорога для комерційного використання. Trello - проста і зручна, але обмежена у можливостях аналітики та контролю доступу. Asana займає проміжну позицію, пропонуючи баланс між функціональністю та простотою, проте також є платною для повноцінного використання [6].

У процесі аналізу Jira, Trello та Asana встановлено, що ці платформи не завжди повністю відповідають потребам невеликих команд і навчальних проєктів. Частина функцій у них доступна лише в платних тарифах, окремі рішення мають

надмірну кількість налаштувань, а можливості адаптації під власні сценарії роботи часто є обмеженими.

З огляду на це доцільною є розробка власного вебзастосунку для управління IT-проєктами. Така система має бути орієнтована на просте створення проєктів і задач, роботу з Kanban-дошкою, розмежування прав користувачів та можливість запуску без складної інфраструктури.

1.3 Визначення вимог до інформаційної системи

На основі проведеного аналізу предметної галузі та існуючих програмних рішень сформовано перелік вимог до розроблюваної інформаційної системи управління IT-проєктами. Вимоги поділяються на функціональні та нефункціональні [10].

Функціональні вимоги визначають конкретні функції та можливості системи, які вона має реалізовувати для задоволення потреб користувачів.

Вимоги до модуля автентифікації та управління користувачами:

– система повинна забезпечувати реєстрацію нових користувачів із вказанням імені, електронної пошти та пароля без можливості самостійного вибору привілейованої ролі; система повинна забезпечувати автентифікацію користувачів за електронною поштою та паролем із використанням JWT-токенів;

– система повинна підтримувати три системні ролі: ADMIN, MANAGER та MEMBER, а також проєктні ролі PROJECT_MANAGER і MEMBER у межах членства конкретного проєкту; дані організуються в межах ізольованих робочих просторів - фірм (Firm), до яких належить кожен користувач;

– звичайна публічна реєстрація (без токена) повинна створювати нову робочу фірму (Firm) і надавати реєстранту роль ADMIN; запрошені учасники приєднуються до існуючої фірми через токен-механізм запрошень;

– система повинна перенаправляти неавторизованих користувачів на сторінку входу.

Вимоги до модуля управління проєктами:

– користувачі з роллю MANAGER або ADMIN повинні мати можливість створювати нові проєкти у межах своєї фірми;

– менеджер проєкту та адміністратор фірми повинні мати можливість редагувати та видаляти проєкти;

– система повинна підтримувати запрошення нових учасників до проєкту за електронною поштою, прийняття запрошення за токеном і видалення учасників із проєкту;

– користувач повинен бачити лише ті проєкти своєї фірми, учасником або менеджером яких він є (або всі проєкти фірми - для адміністратора).

Вимоги до модуля управління задачами:

– менеджер проєкту та адміністратор повинні мати можливість створювати, редагувати та видаляти задачі;

– учасник проєкту повинен мати можливість працювати із задачами в межах проєкту відповідно до наданої ролі, без права створення, видалення задач або керування складом команди;

– задача повинна мати атрибути: назва, опис, статус (TODO / IN_PROGRESS / DONE), пріоритет (LOW / MEDIUM / HIGH), дедлайн, виконавець;

– пріоритет задачі повинен визначатися автоматично на основі дедлайну;

– система повинна забезпечувати перетягування задач між стовпцями Kanban-дошки (drag-and-drop).

Вимоги до інформаційної панелі (Dashboard):

– система повинна відображати зведену статистику: кількість проєктів, задач, задач у процесі виконання та прострочених задач;

– система повинна відображати діаграму розподілу задач за статусами;

– система повинна відображати preview Kanban-дошки та список останніх задач.

Вимоги до модуля пошуку:

– система повинна забезпечувати пошук проєктів і задач за назвою або описом;

– результати пошуку повинні відображатися з урахуванням прав доступу поточного користувача.

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на зручність її використання та надійність.

Вимоги до безпеки: паролі користувачів повинні зберігатися у хешованому вигляді (bcryptjs); доступ до захищених маршрутів повинен контролюватися через middleware на рівні сервера; усі дії з проєктами та задачами повинні перевірятися на відповідність ролі користувача

Вимоги до продуктивності: система повинна забезпечувати швидке завантаження сторінок за рахунок серверного рендерингу (SSR) та оптимістичного оновлення UI при переміщенні задач на Kanban-дошці.

Вимоги до зручності використання: інтерфейс системи повинен бути адаптований для різних розмірів екранів; система повинна підтримувати темну та світлу теми оформлення; усі операції, що змінюють дані, повинні супроводжуватися сповіщеннями про успіх або помилку.

Вимоги до запуску та супроводу: система повинна мати зрозумілу структуру проєкту, файл прикладу змінних середовища, можливість локального запуску через npm-команди та використання SQLite без налаштування окремого сервера бази даних.

Сформовані вимоги є основою для проєктування архітектури системи та реалізації її функціональності у наступних розділах роботи.

1.4 Інформаційна модель предметної галузі

Інформаційна модель предметної галузі відображає ключові сутності системи управління IT-проєктами та зв'язки між ними. Вона слугує

концептуальною основою для проєктування бази даних і визначає склад даних, що зберігаються та обробляються системою [10]

У межах розроблюваної системи виокремлено сім основних сутностей: Фірма (Firm), Користувач (User), Членство у фірмі (FirmMember), Запрошення (Invitation), Проєкт (Project), Членство у проєкті (ProjectMember) та Задача (Task).

Сутність «Фірма» (Firm) є кореневою сутністю системи і представляє ізольований робочий простір організації. Вона містить: унікальний ідентифікатор, назву фірми, а також дати створення та оновлення. Під час реєстрації без запрошення система автоматично створює нову фірму, а реєстрант стає її ADMIN. Сутність «Членство у фірмі» (FirmMember) реалізує зв’язок між користувачами та фірмами (багато до багатьох) і містить: ідентифікатор, посилання на фірму та користувача, роль у фірмі (ADMIN, MANAGER або MEMBER) і дату приєднання. Пара (firmId, userId) є унікальною. Сутність «Користувач» (User) представляє зареєстрованого учасника системи і містить такі атрибути: унікальний ідентифікатор, ім’я, електронна пошта (унікальна), хешований пароль, системна роль, ідентифікатор поточної активної фірми (firmId), а також дату створення та оновлення запису. Реєстрація без токена автоматично створює нову фірму і присвоює роль ADMIN; реєстрація за токеном запрошення приєднує до існуючої фірми з роллю, визначеною в запрошенні.

Сутність «Запрошення» (Invitation) використовується для контрольованого додавання нових користувачів до фірми або до конкретного проєкту. Вона містить електронну пошту запрошеного користувача, роль запрошення (MANAGER або MEMBER), унікальний токен, дату завершення дії запрошення, дату прийняття, обов’язковий ідентифікатор фірми (firmId), ідентифікатор користувача, який створив запрошення, та необов’язковий ідентифікатор проєкту.

Сутність «Проєкт» (Project) представляє окремий IT-проєкт у системі і містить: унікальний ідентифікатор, назву, опис, ідентифікатор менеджера проєкту, ідентифікатор фірми (firmId), а також дати створення та оновлення. Кожен проєкт належить до конкретної фірми, пов'язаний з одним менеджером (зв'язок «один до багатьох» із сутністю User) та може мати довільну кількість учасників.

Сутність «Членство у проєкті» (ProjectMember) є проміжною таблицею, що реалізує зв’язок «багато до багатьох» між користувачами та проєктами. Вона містить унікальний ідентифікатор, ідентифікатор проєкту, ідентифікатор користувача, роль у межах проєкту (PROJECT_MANAGER або MEMBER) та дату приєднання до проєкту. Пара (projectId, userId) є унікальною, що унеможливлює дублювання членства.

Сутність «Задача» (Task) представляє одиницю роботи в рамках проєкту і містить: унікальний ідентифікатор, назву, опис, статус (TODO, IN_PROGRESS або DONE), пріоритет (LOW, MEDIUM або HIGH), дедлайн, позицію на Kanban-дошці, ідентифікатор проєкту та ідентифікатор виконавця (необов'язковий), а також дати створення та оновлення.

У межах даної системи інформаційна модель також визначає логіку взаємодії між основними об’єктами: користувачами, фірмами, проєктами, задачами та запрошеннями. Завдяки цьому можна простежити, як користувач потрапляє до системи, яким чином отримує роль, до яких проєктів має доступ і які дії може виконувати. Така модель є важливою для забезпечення цілісності даних, оскільки кожна задача належить конкретному проєкту, кожен проєкт пов’язаний із певною фірмою, а права користувачів визначаються через членство у фірмі та проєкті.

Особливістю побудованої інформаційної моделі є підтримка ізольованих робочих просторів у вигляді фірм. Це дозволяє розмежувати дані різних команд і забезпечити коректну роботу рольової моделі доступу. Наприклад, адміністратор має можливість керувати ресурсами своєї фірми, менеджер працює з проєктами, за які відповідає, а учасник має доступ лише до тих задач і проєктів, до яких його було додано. Таким чином, інформаційна модель не лише описує с

2 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Середовище розробки

Середовище розробки є фундаментальним інструментом, що безпосередньо впливає на продуктивність розробника та якість коду. Для реалізації розроблюваної системи було обрано редактор Visual Studio Code як основне робоче середовище завдяки його широким можливостям підтримки TypeScript та інтеграції з інструментами розробки [12].

2.1.1 Visual Studio Code

Visual Studio Code (VS Code) - це безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Він є одним із найпопулярніших інструментів серед веб-розробників завдяки широкій підтримці мов програмування, розширюваності через плагіни та вбудованій інтеграції з системами контролю версій [12].

Для розробки даного проєкту VS Code було обрано з таких причин:

– вбудована підтримка TypeScript із підсвічуванням синтаксису, автодоповненням та перевіркою типів у реальному часі;

– інтеграція з Git для управління версіями коду безпосередньо з редактора;

– підтримка розширень Prisma та ESLint для підвищення якості коду;

– вбудований термінал для виконання команд npm, prisma та next без перемикання між вікнами;

– безкоштовне використання та кросплатформеність (Windows, macOS, Linux).

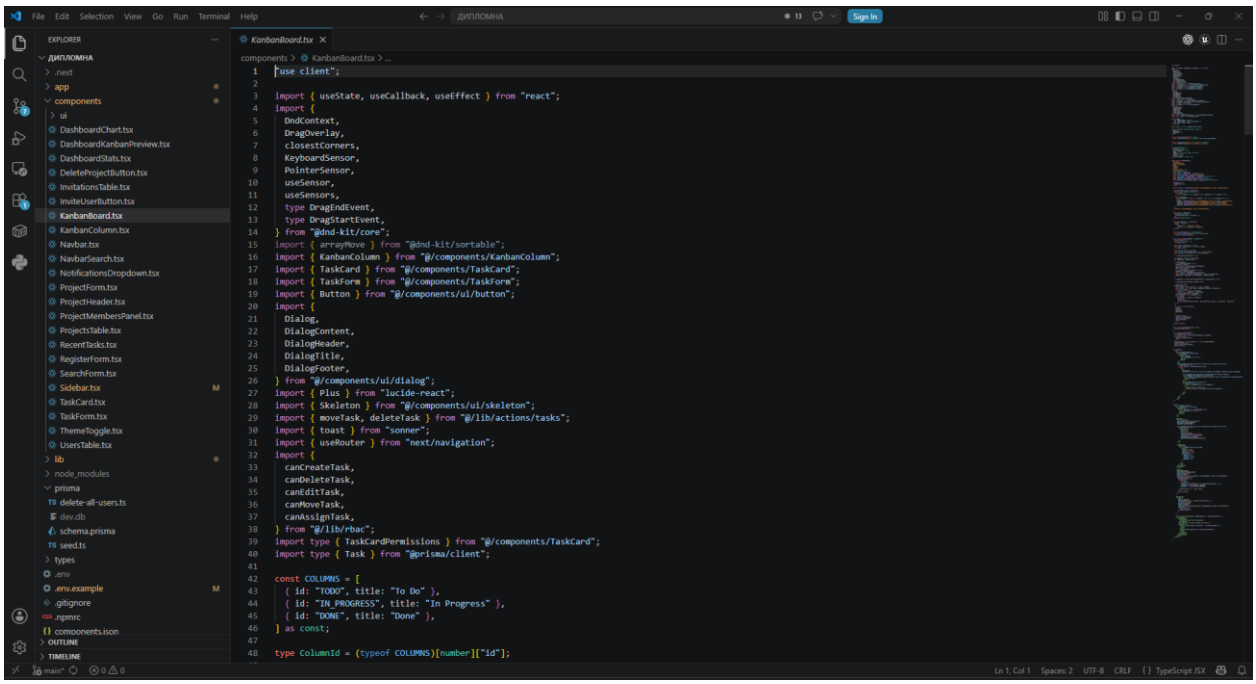


Рис. 2.1. Інтерфейс редактора Visual Studio Code

2.2 Мови програмування та технології

Для створення інформаційної системи управління IT-проєктами використано технології екосистеми JavaScript/TypeScript, оскільки вони дозволяють реалізувати клієнтську та серверну частини вебзастосунку в межах єдиної кодової бази. Основною мовою розробки обрано TypeScript, який дає змогу описувати структуру даних, ролі користувачів, статуси задач і параметри серверних функцій. Після компіляції TypeScript перетворюється на JavaScript, що забезпечує виконання застосунку в браузері та середовищі Node.js [13].

2.2.1 TypeScript

TypeScript - це мова програмування з відкритим вихідним кодом, розроблена компанією Microsoft як надмножина JavaScript. Головною особливістю TypeScript є статична типізація, що дозволяє виявляти помилки на етапі компіляції, а не під час виконання програми [13].

У розроблюваній системі TypeScript використовується як основна мова програмування на стороні як клієнта, так і сервера. Статична типізація забезпечує

надійність коду при роботі з моделями даних Prisma, параметрами Server Actions та пропсами React-компонентів. Зокрема, типи ролей користувача (ADMIN, MANAGER, MEMBER), статусів задач (TODO, IN_PROGRESS, DONE) та пріоритетів (LOW, MEDIUM, HIGH) визначені як рядкові літеральні типи, що унеможливлює передачу некоректних значень між компонентами системи.

2.2.2 JavaScript

JavaScript - це динамічна інтерпретована мова програмування, що є стандартом для розробки вебзастосунків і виконується як у браузері, так і на сервері в середовищі Node.js. TypeScript компілюється у JavaScript, тому останній використовується як цільова мова виконання [13].

Усі залежності проєкту (next, react, prisma, @dnd-kit, recharts тощо) реалізовані на JavaScript або TypeScript, що забезпечує повну сумісність та можливість використання найширшої екосистеми бібліотек npm. Пакетний менеджер npm використовувався для встановлення та управління залежностями проєкту протягом усього циклу розробки.

2.3 Веб-фреймворки та бібліотеки

Сучасна розробка вебзастосунків передбачає використання спеціалізованих фреймворків і бібліотек, що суттєво прискорюють реалізацію типових задач і забезпечують надійну архітектурну основу. Для розроблюваної системи обрано набір інструментів, що охоплює серверний фреймворк, бібліотеку інтерфейсу, систему стилізації, drag-and-drop взаємодію, аналітичні графіки та автентифікацію [14].

2.3.1 Next.js

Next.js - це фреймворк для розробки повностекових вебзастосунків на основі React. Він поєднує можливості серверного рендерингу (SSR), статичної генерації (SSG) та клієнтського рендерингу (CSR) в єдиній архітектурі App Router [14].

Next.js обрано як основний фреймворк проєкту з таких причин:

– App Router дозволяє визначати серверні та клієнтські компоненти в межах однієї кодової бази, що спрощує архітектуру застосунку;

– Server Actions забезпечують виконання серверної логіки (запити до БД, перевірка прав доступу) безпосередньо з компонентів без необхідності окремого REST API;

– вбудована підтримка middleware дозволяє захищати маршрути на рівні сервера до рендерингу сторінки;

– автоматична оптимізація зображень, шрифтів і статичних ресурсів підвищує продуктивність застосунку;

– зручний локальний запуск і збірка проєкту за допомогою npm-команд next dev та next build.

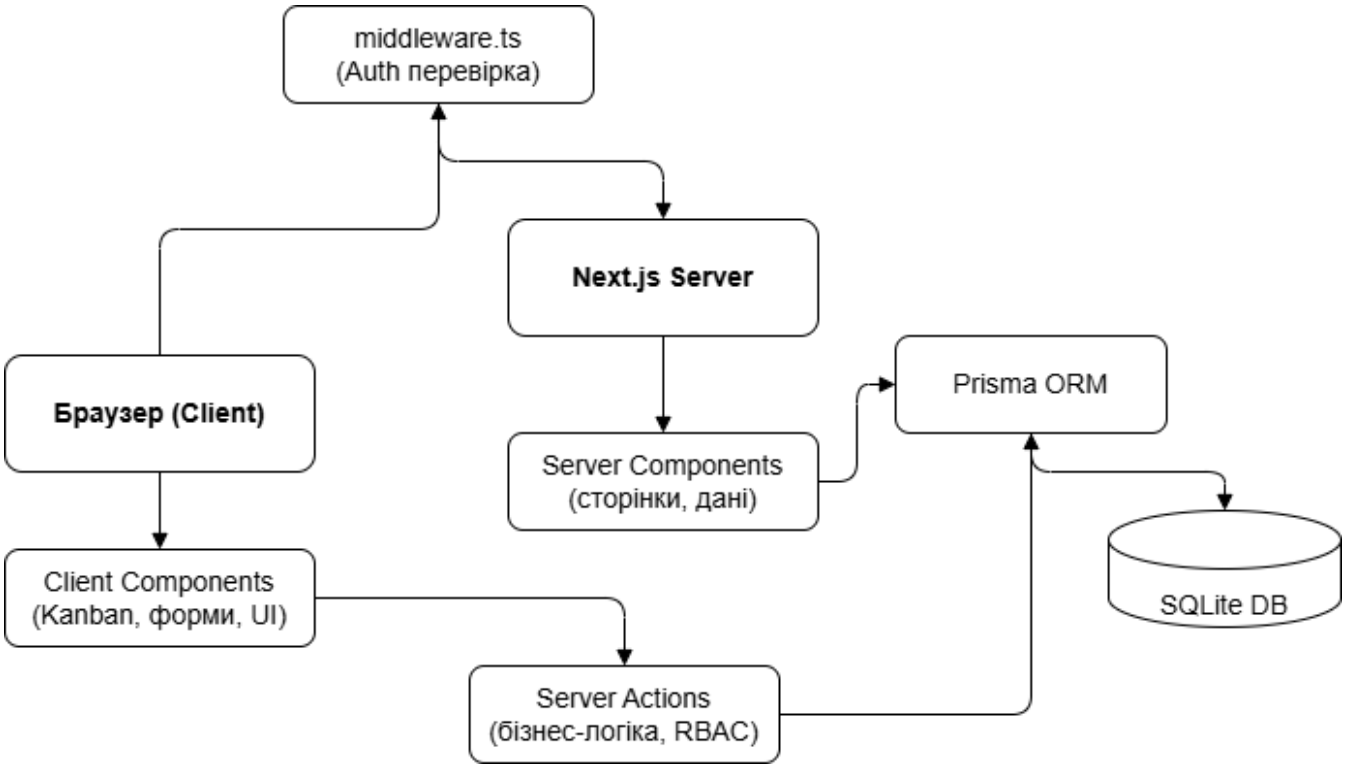


Рис. 2.2. Архітектура Next.js App Router

2.3.2 React

React - це JavaScript-бібліотека для побудови користувацьких інтерфейсів, розроблена компанією Meta. Вона базується на концепції компонентного підходу, де інтерфейс складається з незалежних, повторно використовуваних компонентів із власним станом і логікою [15].

У даному проєкті React використовується як основа для побудови клієнтської частини застосунку. Клієнтські компоненти (позначені директивою “use client”) застосовуються для реалізації інтерактивних елементів: Kanban-дошки з drag-and-drop, форм створення та редагування задач і проєктів, модальних діалогів, навігаційного пошуку та перемикача теми оформлення. Серверні компоненти використовуються для рендерингу сторінок із даними, отриманими безпосередньо з бази даних без додаткових HTTP-запитів.

2.3.3 Tailwind CSS та shadcn/ui

Tailwind CSS - це utility-first CSS-фреймворк, що надає набір готових класів для стилізації елементів інтерфейсу безпосередньо в розмітці. На відміну від традиційних CSS-фреймворків, Tailwind не нав'язує готових компонентів, а надає гнучкі примітиви для побудови власного дизайну [16].

У розробленій системі Tailwind CSS використано для оформлення інтерфейсу та налаштування зовнішнього вигляду компонентів. Для прискорення створення типових елементів інтерфейсу додатково застосовано shadcn/ui, що містить готові React-компоненти, зокрема кнопки, таблиці, діалогові вікна, випадаючі меню, вкладки та картки. Особливістю shadcn/ui є те, що компоненти додаються безпосередньо до кодової бази проєкту, тому їх можна змінювати відповідно до потреб системи та повністю контролювати їхню поведінку й оформлення.

2.3.4 @dnd-kit

@dnd-kit - це сучасна бібліотека для реалізації drag-and-drop взаємодії у React-застосунках. Вона забезпечує доступну, продуктивну та розширювану основу для переміщення елементів інтерфейсу за допомогою вказівника миші або

клавіатури [17]. Бібліотека складається з кількох субпакетів: @dnd-kit/core (основний контекст і сенсори), @dnd-kit/sortable (функції для сортованих списків: useSortable, SortableContext) та @dnd-kit/utilities (допоміжні утиліти CSS-трансформацій).

У розроблюваній системі @dnd-kit використовується для реалізації Kanban-дошки. Компонент DndContext обгортає дошку і відстежує події перетягування; SortableContext керує впорядкуванням задач у межах стовпця; DragOverlay відображає «тінь» задачі під час перетягування між стовпцями. При відпусканні задачі виконується оптимістичне оновлення інтерфейсу та асинхронний виклик Server Action для збереження нового статусу і позиції в базі даних.

2.3.5 Recharts

Recharts - це бібліотека для побудови діаграм і графіків у React-застосунках на основі SVG. Вона надає готові компоненти для різних типів візуалізації даних: лінійні графіки, стовпчасті діаграми, кругові діаграми тощо [18].

У розроблюваній системі Recharts використовується для відображення кругової діаграми (PieChart) розподілу задач за статусами на інформаційній панелі Dashboard. Діаграма наочно показує співвідношення задач зі статусами TO DO, IN_PROGRESS та DONE у всіх проєктах користувача, що дозволяє швидко оцінити загальний прогрес роботи.

2.3.6 NextAuth.js

NextAuth.js (Auth.js v5) - це бібліотека автентифікації для Next.js-застосунків, що підтримує різноманітні стратегії входу: OAuth-провайдери (Google, GitHub тощо), електронну пошту та облікові дані. Вона забезпечує управління сесіями, захист маршрутів та інтеграцію з базами даних [19].

У даному проєкті NextAuth.js використовується для реалізації автентифікації на основі облікових даних (email + пароль) із JWT-стратегією сесій. Бібліотека забезпечує: перевірку облікових даних через провайдер Credentials з хешуванням паролів bcryptjs; збереження ідентифікатора, імені та ролі користувача в JWT-

токені; доступ до даних сесії у серверних компонентах і Server Actions через функцію auth(); а також автоматичне перенаправлення неавторизованих запитів через middleware.

2.4 Система управління базами даних

Вибір системи управління базами даних та інструментів роботи з нею безпосередньо впливає на надійність, продуктивність і зручність розробки системи. Для розроблюваного застосунку обрано реляційну СУБД SQLite для середовища розробки у поєднанні з ORM-інструментом Prisma, що забезпечує типобезпечну роботу з даними [20].

2.4.1 SQLite

SQLite - це вбудована реляційна система управління базами даних, що зберігає всі дані в єдиному файлі на диску. Вона не потребує окремого серверного процесу та конфігурації, що робить її ідеальним вибором для розробки та тестування застосунків [20].

У розроблюваній системі SQLite використовується як база даних для локального середовища розробки та тестування. Файл бази даних (dev.db) зберігається локально у папці prisma/, що забезпечує швидкий запуск проєкту без необхідності налаштування окремого сервера БД. Такий вибір є доцільним для навчального проєкту, оскільки спрощує перевірку функціональності, демонстрацію роботи системи та перенесення проєкту між комп’ютерами.

2.4.2 Prisma ORM

Prisma - це сучасний ORM (Object-Relational Mapping) для Node.js і TypeScript, що значно спрощує роботу з базами даних. Він складається з трьох основних компонентів: Prisma Client (типобезпечний клієнт для запитів до БД), Prisma Migrate (система міграцій схеми) та Prisma Studio (графічний інтерфейс для перегляду та редагування даних) [21].

Prisma обрано як основний інструмент для роботи з базою даних з таких причин:

– автоматична генерація типів TypeScript на основі схеми даних забезпечує повну типобезпеку при роботі з БД і виключає помилки при зверненні до неіснуючих полів;

– декларативна схема (schema.prisma) є єдиним джерелом істини для структури бази даних і автоматично синхронізується з TypeScript-типами;

– зручний API для виконання складних запитів із підтримкою фільтрації, сортування, пагінації та вкладених включень (include) для отримання пов'язаних даних;

– підтримка транзакцій для атомарного виконання кількох операцій, наприклад при оновленні позицій задач на Kanban-дошці;

– система міграцій для відстеження та застосування змін схеми бази даних у командному середовищі.

2.5 Система контролю версій

Система контролю версій є обов'язковим інструментом у сучасній розробці програмного забезпечення, що дозволяє відстежувати зміни у коді, координувати роботу команди та зберігати повну історію розробки. Для даного проєкту використано Git як систему контролю версій та GitHub як платформу для хостингу репозиторію [22].

2.5.1 Git

Git - це розподілена система контролю версій, що дозволяє відстежувати зміни у вихідному коді, повертатись до попередніх версій та паралельно розробляти різні гілки функціоналу. Git є галузевим стандартом у сучасній розробці програмного забезпечення та підтримується переважною більшістю сучасних інструментів і платформ [22].

Git використовувався для збереження актуальної версії коду, відстеження основних змін у проєкті та резервного копіювання вихідних файлів. Вбудована інтеграція VS Code з Git забезпечувала зручне відображення змін безпосередньо у редакторі.

2.5.2 GitHub

GitHub є хмарним сервісом для розміщення Git-репозиторіїв, який забезпечує зручні інструменти для роботи з програмним кодом. Платформа дозволяє переглядати файли проєкту, історію комітів, гілки розробки, а також організовувати командну взаємодію під час створення програмного забезпечення. Використання GitHub дало змогу зберігати проєкт у централізованому репозиторії, відстежувати всі зміни, працювати з pull request, переглядати та обговорювати фрагменти коду, а за потреби - підключати автоматизовані процеси розгортання через CI/CD.

У межах цього проєкту GitHub застосовувався як основне середовище для збереження вихідного коду, обміну файлами, резервного копіювання та контролю версій. Крім того, платформа використовувалася для ведення супровідної документації, зокрема README-файлів. Завдяки цьому процес розробки був організований більш системно, прозоро та з можливістю відстеження всіх етапів роботи над застосунком.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Проєктування архітектури системи

Архітектуру розробленої системи побудовано як повностековий вебзастосунок, у якому клієнтська частина, серверна логіка та робота з даними організовані в межах одного проєкту. Використання Next.js App Router дозволило розмістити сторінки інтерфейсу, серверні компоненти та Server Actions в одній структурі застосунку. Завдяки цьому основна серверна логіка реалізована без створення окремого бекенд-додатку [14].

Логіку роботи системи умовно поділено на три основні рівні:

– рівень представлення (Presentation Layer) - React-компоненти, що формують користувацький інтерфейс і відображаються у браузері або на сервері залежно від типу компонента;

– рівень бізнес-логіки (Business Logic Layer) - Server Actions і серверні компоненти Next.js, що виконуються виключно на сервері, перевіряють права доступу та обробляють дані;

– рівень даних (Data Layer) - Prisma ORM у поєднанні з реляційною базою даних SQLite, що забезпечує зберігання та отримання всіх даних системи.

Робота між рівнями організована послідовно: користувач виконує дію в інтерфейсі, після чого клієнтський компонент звертається до відповідної Server Action. Серверна функція перевіряє активну сесію через auth(), визначає права користувача за правилами RBAC і лише після цього виконує запит до бази даних через Prisma Client. Після отримання результату інтерфейс оновлюється відповідно до змін. Такий підхід гарантує, що жодна операція зміни даних не може бути виконана в обхід серверної перевірки прав доступу.

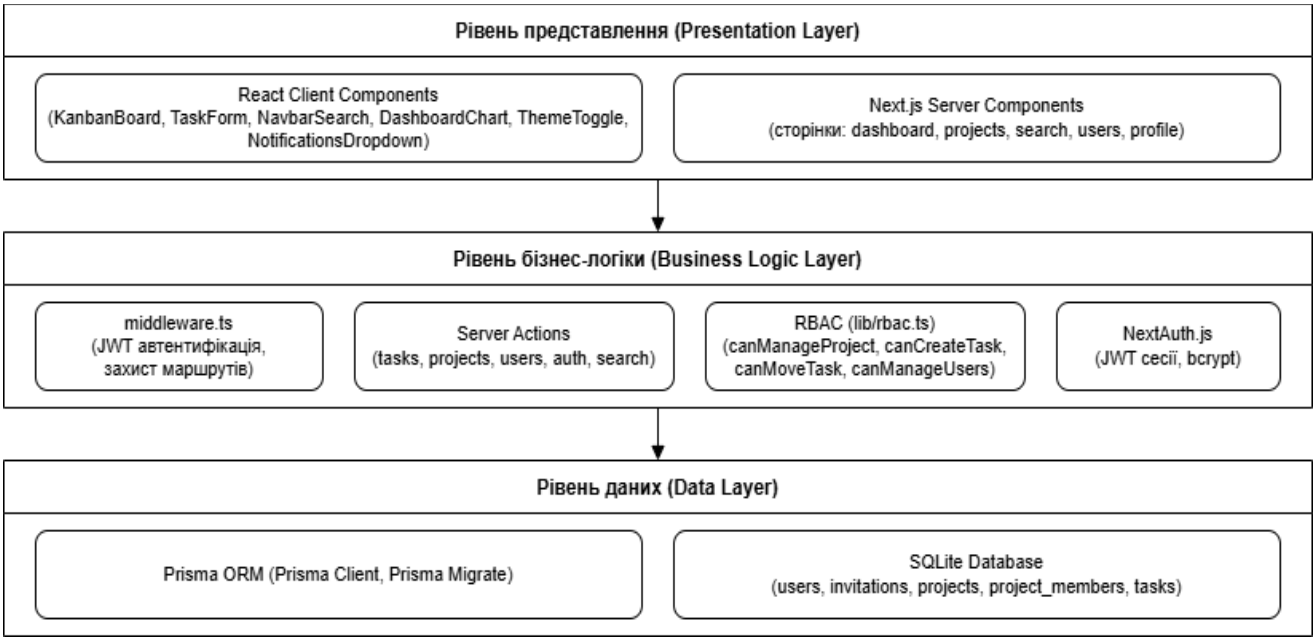


Рис. 3.1 Загальна архітектура інформаційної системи

3.1.1 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) відображає (див. рис. 3.2) функціональні можливості системи з точки зору кінцевих користувачів і визначає взаємодію між акторами та системою [24].

У системі виокремлено трьох акторів відповідно до визначеної моделі ролей:

– Адміністратор (Administrator) - має повний доступ до функцій керування користувачами, проєктами та задачами в межах своєї активної фірми;

– Менеджер (Manager) - управляє власними проєктами та задачами в них;

– Учасник (Member) - переглядає проєкти, учасником яких є, та виконує призначені йому задачі.

Основні варіанти використання системи охоплюють такі групи функцій:

– автентифікація (реєстрація, вхід, вихід із системи);

– управління проєктами (створення, редагування, видалення проєкту, додавання та видалення учасників);

– управління задачами (створення, редагування, видалення задачі, зміна статусу та пріоритету, призначення виконавця, переміщення на Kanban-дошці);

– перегляд інформаційної панелі; пошук проєктів і задач; управління профілем.

Адміністратор успадковує всі варіанти використання Менеджера та Учасника, доповнюючи їх можливістю керування користувачами, запрошеннями та всіма проєктами в межах своєї активної фірми. Менеджер може виконувати дії Учасника, а також управляти проєктами, менеджером яких він є. Учасник має обмежений доступ до проєктів, у яких є членом, і може працювати із задачами відповідно до своєї ролі у проєкті.

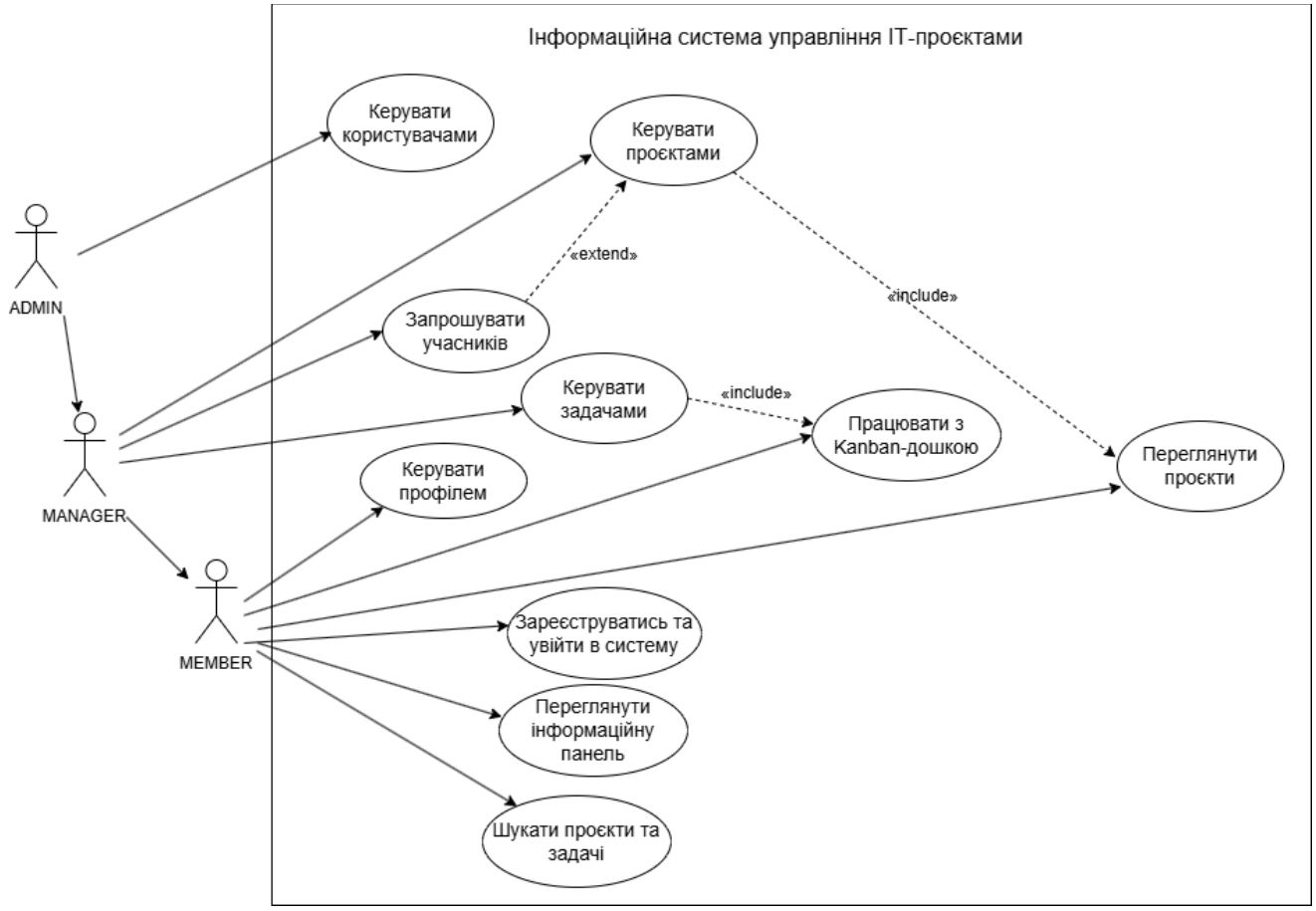


Рис. 3.2 Діаграма варіантів використання

3.1.2 Діаграма діяльності

Діаграма діяльності (Activity Diagram) описує послідовність дій у межах конкретного бізнес-процесу системи. Для ілюстрації ключового сценарію розроблено діаграму процесу роботи з задачами на Kanban-дошці [24].

Процес роботи з Kanban-дошкою включає такі кроки: користувач відкриває сторінку проєкту; система завантажує задачі з бази даних і відображає їх на дошці, розподіляючи за стовпцями відповідно до статусу; користувач розпочинає перетягування картки задачі; система перевіряє права доступу (canMoveTask); якщо перевірку пройдено — виконується оптимістичне оновлення інтерфейсу (картка переміщується у новий стовпець); паралельно викликається Server Action moveTask, що зберігає новий статус і позицію в базі даних; у разі помилки збереження інтерфейс повертається до попереднього стану із відображенням сповіщення про помилку.

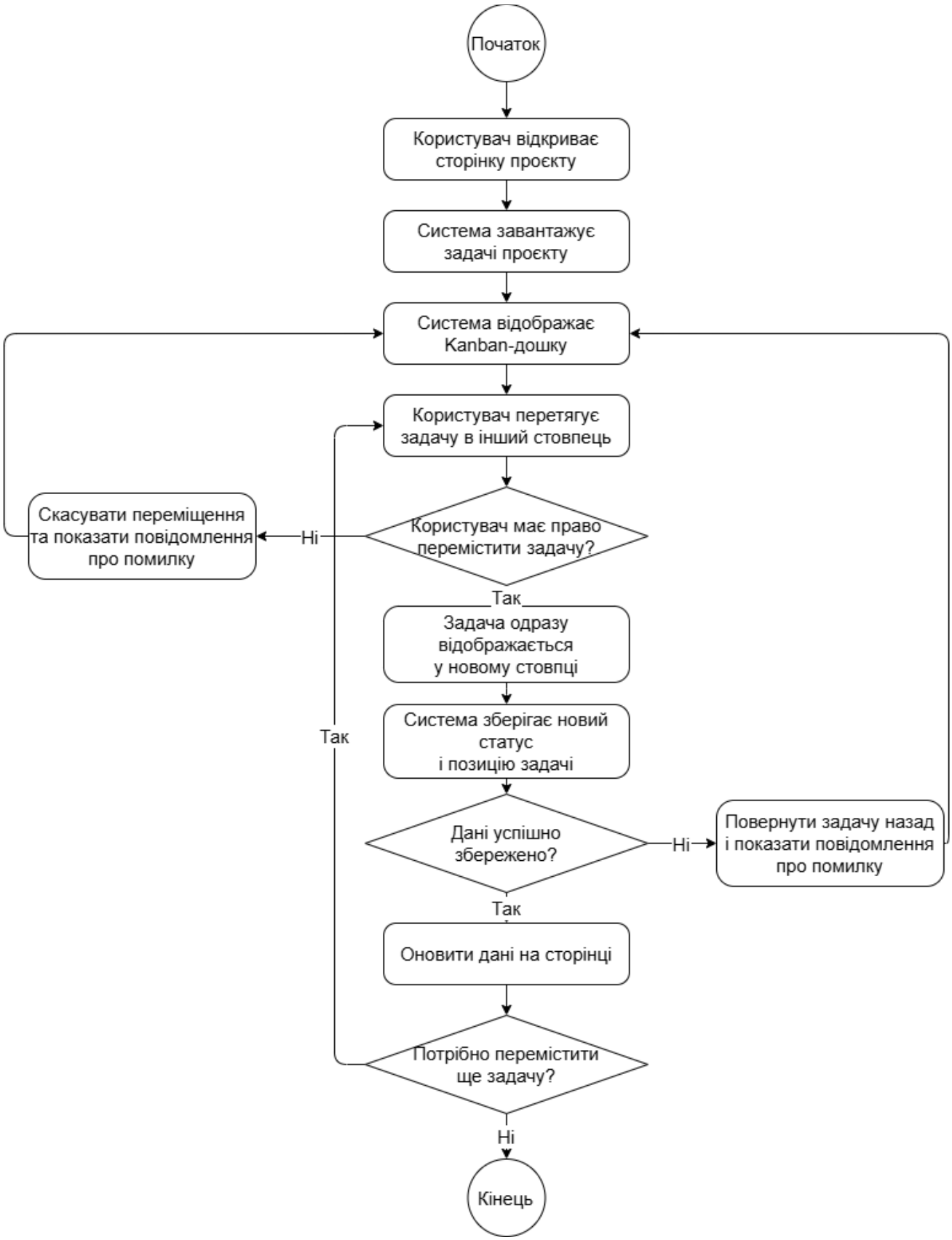


Рис. 3.3 Алгоритм зміни статусу задачі на kanban-дошці

3.2 Архітектура клієнтської частини

Клієнтська частина системи побудована на основі компонентної архітектури React у поєднанні з маршрутизацією Next.js App Router. Усі компоненти поділяються на два типи: серверні (Server Components) і клієнтські (Client Components), що визначає місце їхнього виконання та доступні можливості [15].

Серверні компоненти виконуються на сервері під час обробки запиту і мають прямий доступ до бази даних через Prisma. Вони використовуються для рендерингу сторінок із даними: списку проєктів, сторінки проєкту з Kanban-дошкою, інформаційної панелі та сторінки пошуку. Серверні компоненти не мають доступу до стану браузера (useState, useEffect) та обробників подій.

Клієнтські компоненти (позначені директивою “use client”) виконуються у браузері і використовуються для реалізації інтерактивної логіки. До клієнтських компонентів належать: KanbanBoard і KanbanColumn (drag-and-drop дошка), TaskForm і ProjectForm (форми з валідацією), NavbarSearch (пошук у реальному часі), NotificationsDropdown (випадаюче меню сповіщень), ThemeToggle (перемикач теми), DashboardChart (кругова діаграма Recharts).

Структура маршрутів застосунку (див. рис. 3.4) організована за допомогою групування (route groups) у папках з дужками:

– (auth) група маршрутів для неавторизованих користувачів: /login і /register;

– (dashboard) - захищена група маршрутів для авторизованих користувачів: /dashboard, /projects, /projects/[id], /users (лише для адміністратора), /search, /profile, /forbidden.

Компоненти спільного використання (Navbar, Sidebar, TaskCard, ProjectsTable тощо) винесено у окрему папку components/ і застосовуються на кількох сторінках. Такий підхід забезпечує повторне використання коду та єдність візуального стилю в усьому застосунку.

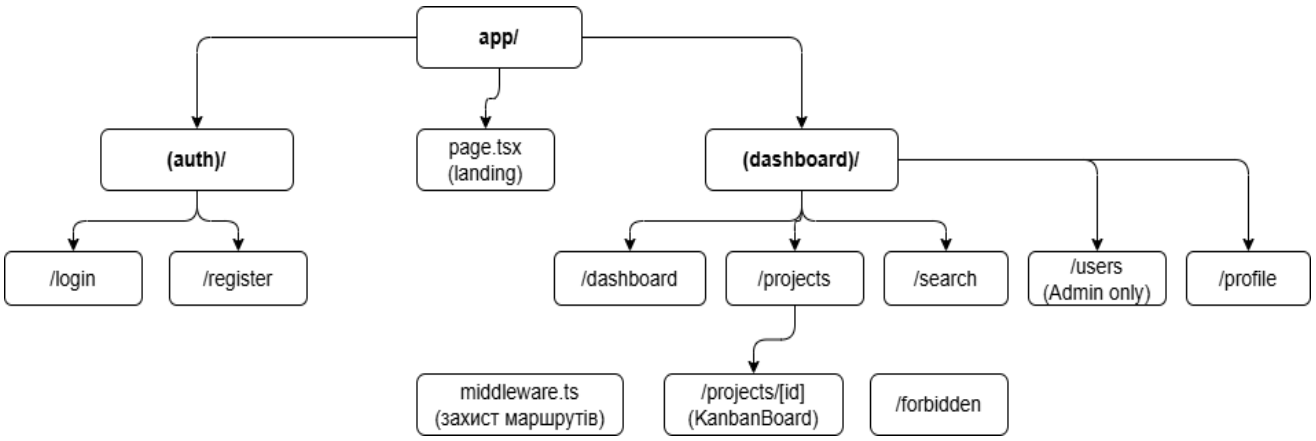


Рис. 3.4 Структура маршрутів клієнтської частини

3.3 Архітектура серверної частини

Серверна частина системи реалізована за допомогою Server Actions Next.js — функцій, що виконуються виключно на сервері і можуть викликатися безпосередньо з клієнтських компонентів без HTTP-запитів. Усі серверні функції організовані в директорії lib/actions/ та lib/ [14].

Серверна частина включає такі ключові модулі:

– lib/auth.ts - конфігурація NextAuth.js: визначення провайдера Credentials, JWT-колбеків для збереження id і ролі користувача в токені, налаштування сторінок входу та стратегії сесій;

– lib/rbac.ts - функції рольового контролю доступу: canManageProject, canManageTasksInProject, canEditTask, canMoveTask, canAccessProject та інші, що приймають роль і ідентифікатори та повертають булеве значення дозволу;

– lib/prisma.ts - єдиний екземпляр Prisma Client (singleton), що запобігає створенню зайвих підключень до БД у середовищі розробки з hot-reload;

– lib/actions/ - Server Actions для кожного модуля: дії автентифікації (lib/actions/auth.ts - реєстрація користувача, автоматичне створення фірми, прийняття запрошення та хешування пароля bcryptjs), дії управління фірмами (lib/actions/firms.ts - switchFirm для перемикання активної фірми), дії управління

користувачами і запрошеннями (lib/actions/users.ts - createInvitation, revokeInvitation, updateUserRole, deleteUser), дії проєктів (lib/actions/projects.ts - createProject, updateProject, deleteProject, removeProjectMember), дії задач (lib/actions/tasks.ts - createTask, updateTask, deleteTask, moveTask) та дії пошуку (lib/actions/search.ts - серверна фільтрація проєктів і задач за текстовим запитом через оператор contains Prisma).

Захист маршрутів реалізовано на двох рівнях. Перший рівень - middleware.ts, що перехоплює всі HTTP-запити до захищених маршрутів (dashboard, projects тощо) і перевіряє наявність активної сесії. Неавторизовані запити автоматично перенаправляються на сторінку /login. Другий рівень - перевірка прав доступу безпосередньо у Server Actions і серверних компонентах через функції RBAC, що унеможливлює несанкціоновані операції навіть при прямому виклику Server Action.

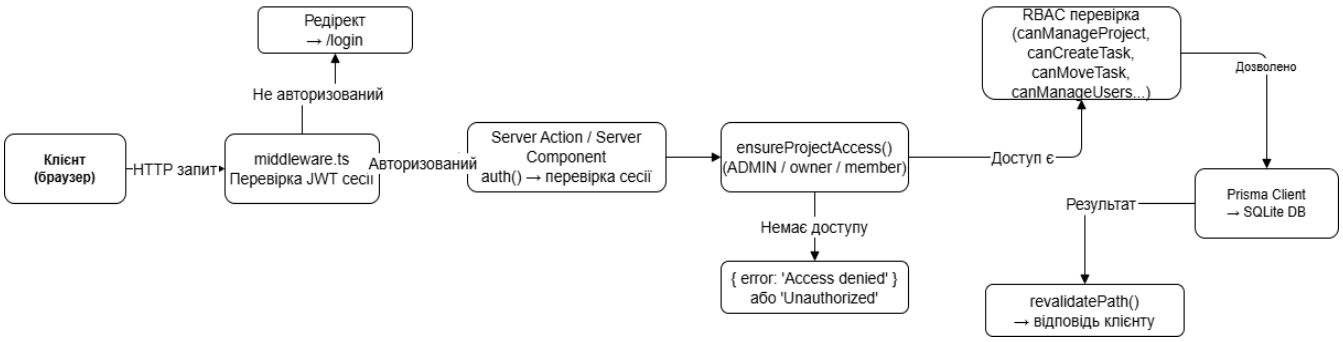


Рис. 3.5 Схема обробки запиту на серверній частині

3.4 Архітектура бази даних

База даних системи спроєктована на основі реляційної моделі та описана у вигляді декларативної схеми Prisma. До складу схеми входять таблиці firms, users, firm_members, invitations, projects, project_members і tasks. Вони пов’язані між собою через зовнішні ключі, а для окремих зв’язків додатково задано унікальні обмеження та правила каскадного видалення [21].

Таблиця firms є кореневою таблицею системи і зберігає дані про робочі простори (фірми). Вона містить ідентифікатор, назву та часові мітки. При видаленні фірми каскадно видаляються всі пов'язані користувачі, проєкти та запрошення. Таблиця firm_members реалізує зв'язок «багато до багатьох» між firms та users. Вона містить ідентифікатор, firmId, userId, роль у фірмі (ADMIN, MANAGER або MEMBER) та дату приєднання. Пара (firmId, userId) є унікальною. Таблиця users зберігає дані про зареєстрованих користувачів системи. Первинним ключем є рядковий ідентифікатор типу CUID (Collision-resistant Unique Identifier), що генерується автоматично. Поле email є унікальним і використовується для автентифікації. Поле role зберігає поточну активну роль користувача (ADMIN, MANAGER або MEMBER). Поле firmId посилається на поточну активну фірму користувача. Пароль зберігається виключно у хешованому вигляді (bcryptjs).

Таблиця invitations зберігає одноразові запрошення користувачів. Вона містить електронну пошту запрошеного користувача, роль запрошення, унікальний токен, дату завершення дії, дату прийняття запрошення, обов’язковий зв’язок із фірмою (firmId), посилання на користувача, який створив запрошення, та необов’язковий зв’язок із проєктом. Поле acceptedAt запобігає повторному використанню вже прийнятого запрошення.

Таблиця projects зберігає дані про проєкти. Кожен проєкт пов'язаний з фірмою через зовнішній ключ firmId (onDelete: Cascade) та з менеджером через зовнішній ключ managerId, що посилається на таблицю users. При видаленні менеджера всі його проєкти також видаляються (onDelete: Cascade). Поле description є необов'язковим.

Таблиця project_members є проміжною (junction table) для реалізації зв’язку «багато до багатьох» між users та projects. Пара (projectId, userId) визначена як унікальна, що гарантує відсутність дублікатів членства. Поле role визначає роль користувача в межах конкретного проєкту: PROJECT_MANAGER або MEMBER. При видаленні проєкту або користувача відповідні записи членства видаляються каскадно.

Таблиця tasks зберігає задачі у межах проєктів. Кожна задача обов'язково належить до певного проєкту (projectId, onDelete: Cascade) і може мати необов'язкового виконавця (assigneeId). При видаленні виконавця поле assigneeId

встановлюється у NULL (onDelete: SetNull), що зберігає задачу без призначеного виконавця. Поле position зберігає порядок задачі в межах стовпця Kanban-дошки для коректного відображення послідовності.

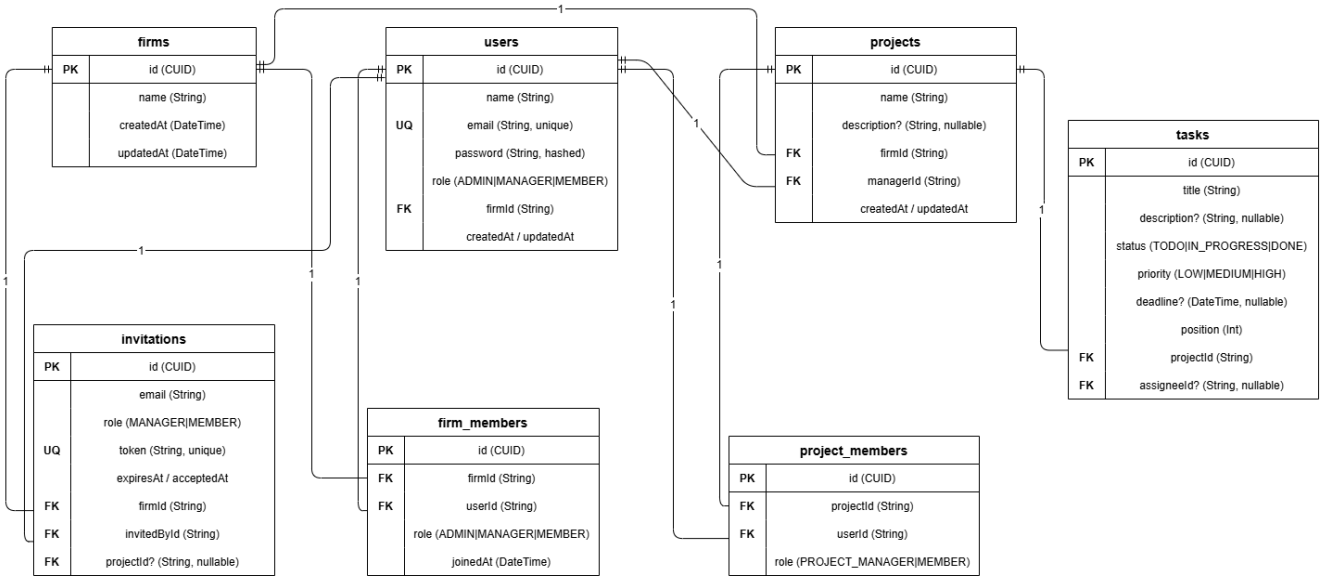


Рис. 3.6 ER-діаграма бази даних системи

На ER-діаграмі відображено зв’язки між основними таблицями бази даних системи. Центральними сутностями є users, firms і projects, оскільки саме через них формується структура доступу користувачів до робочих просторів і проєктів. Таблиці firm_members та project_members виконують роль проміжних сутностей, які дозволяють реалізувати зв’язки «багато до багатьох» між користувачами, фірмами та проєктами. Завдяки цьому один користувач може бути учасником кількох фірм або проєктів, а кожна фірма чи проєкт можуть містити декілька учасників із різними ролями.

Окрему роль у моделі відіграють таблиці invitations і tasks. Таблиця invitations забезпечує контрольований механізм додавання користувачів до фірми або конкретного проєкту через унікальний токен запрошення. Таблиця tasks пов’язана з проєктом і зберігає дані про задачі, їхній статус, пріоритет, дедлайн, позицію на Kanban-дошці та можливого виконавця. Така структура бази даних забезпечує цілісність інформації, дозволяє коректно обмежувати доступ користувачів відповідно до їхніх ролей і підтримує основні функції системи управління IT-проєктами.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Реалізація серверної частини

Серверну логіку розробленої системи побудовано на основі механізму Server Actions у Next.js. Такі функції виконуються на сервері та використовуються для обробки основних дій користувача: автентифікації, керування користувачами, проєктами та задачами. Вони розміщені в директорії lib/actions/, що дозволяє логічно розділити серверні операції за окремими модулями. Перед внесенням змін до бази даних кожна дія послідовно перевіряє наявність активної сесії, доступ користувача до потрібного ресурсу, відповідність його ролі правилам RBAC, коректність переданих даних через lib/validations.ts, після чого виконує операцію в базі даних і оновлює кеш відповідних сторінок [14].

4.1.1 Реалізація автентифікації

Модуль автентифікації реалізовано у файлі lib/auth.ts на основі бібліотеки NextAuth.js v5. Для входу використовується провайдер Credentials, що приймає електронну пошту та пароль. У функції authorize виконується пошук користувача в базі даних за email та перевірка пароля через bcryptjs.compare. У разі успіху повертається об’єкт із полями id, name, email, role та firmId, які зберігаються у JWT-токені через колбек jwt і доступні у всіх Server Actions через session.user.

Захист маршрутів реалізовано у файлі middleware.ts. Middleware перехоплює всі HTTP-запити і перевіряє наявність активної сесії. Якщо користувач не авторизований і намагається відкрити захищений маршрут - він перенаправляється на /login. Якщо авторизований користувач відкриває сторінки /login або /register - він автоматично перенаправляється на /dashboard. Маршрутизатор налаштований через поле matcher так, щоб не перехоплювати статичні ресурси та API-маршрути.

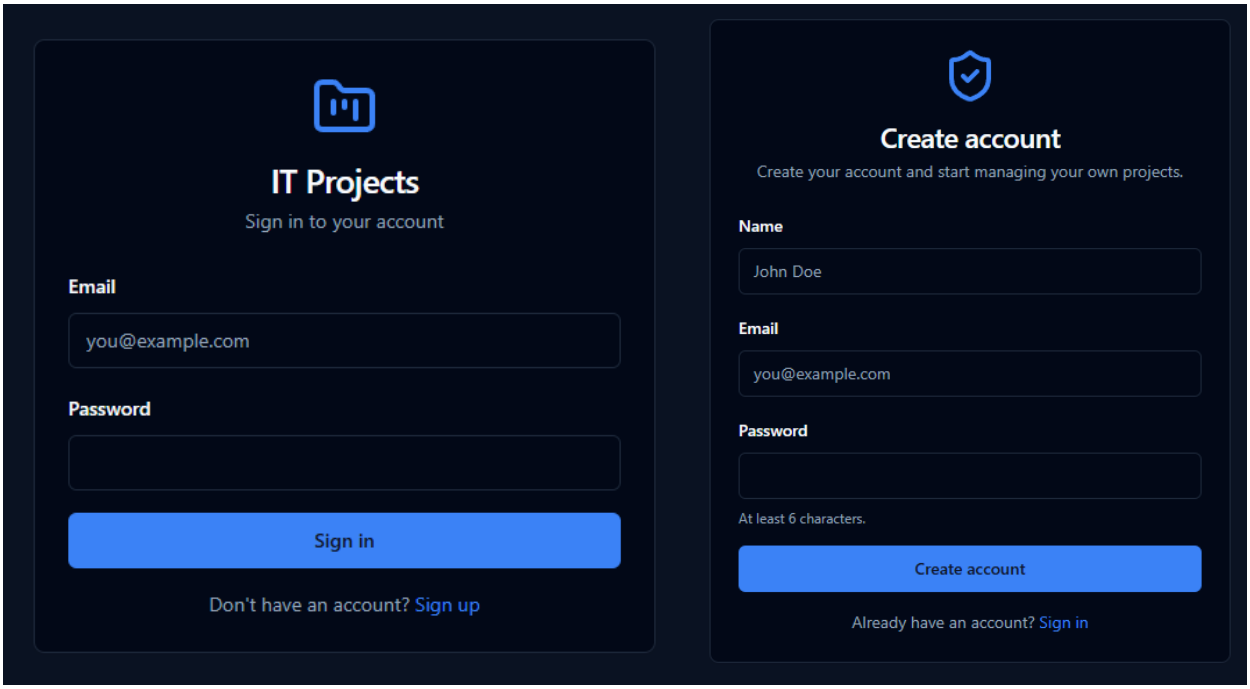


Рис. 4.1 Сторінки автентифікації системи

4.1.2 Реалізація Server Actions для задач

Серверні функції для роботи із задачами зосереджені у файлі lib/actions/tasks.ts і містять чотири основні функції: createTask, updateTask, deleteTask та moveTask.

Функція createTask приймає ідентифікатор проєкту та FormData з даними форми. Після перевірки сесії та доступу до проєкту функція перевіряє право на створення задач через canCreateTask. Пріоритет задачі визначається автоматично функцією getPriorityFromDeadline на основі дедлайну: якщо дедлайн відсутній або більше 7 днів - LOW, від 3 до 7 днів - MEDIUM, менше 3 днів - HIGH. Позиція нової задачі встановлюється як максимальна позиція серед задач зі статусом TODO плюс один, що розміщує її в кінці стовпця.

Функція moveTask реалізує переміщення задачі між стовпцями Kanban-дошки. Вона приймає ідентифікатор задачі, проєкту, новий статус і нову позицію. Після перевірки доступу функція canMoveTask визначає, чи має поточний користувач право переміщувати задачу: адміністратор та менеджер проєкту можуть переміщувати будь-які задачі в межах доступного проєкту, а учасник працює із задачами відповідно до своєї ролі та прав у межах проєкту. При

успішному переміщенні виконується оновлення статусу і позиції в базі даних та інвалідація кешу сторінок /projects/[id] і /dashboard через revalidatePath.

```
209  export async function moveTask(
210    taskId: string,
211    projectId: string,
212    newStatus: "TODO" | "IN_PROGRESS" | "DONE",
213    newPosition: number
214  ) {
215    const session = await auth();
216    if (!session?.user?.id || !session.user.firmId) return { error: "Unauthorized" };
217    if (!isTaskStatus(newStatus)) return { error: "Invalid task status" };
218    if (!Number.isInteger(newPosition) || newPosition < 0) {
219      return { error: "Invalid task position" };
220    }
221
222    const { error, project } = await ensureProjectAccess(
223      projectId,
224      session.user.id,
225      session.user.role as string,
226      session.user.firmId
227    );
228    if (error || !project) return { error: error ?? "Access denied" };
229
230    const task = await prisma.task.findUnique({ where: { id: taskId } });
231    if (!task) return { error: "Task not found" };
232    if (task.projectId !== projectId) return { error: "Task not found" };
233    if (task.projectId !== projectId) return { error: "Task not found" };
234    if (task.projectId !== projectId) return { error: "Task not found" };
235
236    const userProjectRole =
237      project.members.find((m) => m.userId === session.user.id)?.role ?? null;
238
239    if (!canMoveTask(session.user.role as string, project.managerId, session.user.id, task.assigneeId, userProjectRole)) {
240      return { error: "You can only move your own assigned tasks" };
241    }
242
243    await prisma.task.update({
244      where: { id: taskId },
245      data: { status: newStatus, position: newPosition },
246    });
247    revalidatePath(`/projects/${projectId}`);
248    revalidatePath("/dashboard");
249    return { success: true };
250  }
```

Рис. 4.2. Фрагмент реалізації Server Action moveTask

4.1.3 Реалізація Server Actions для проєктів

Серверні функції для роботи з проєктами містяться у файлі lib/actions/projects.ts і реалізують основні операції над проєктами: createProject, updateProject, deleteProject та removeProjectMember. Окремий модуль lib/actions/users.ts містить серверні функції управління користувачами та запрошеннями: createInvitation, revokeInvitation, updateUserRole і deleteUser. Функція реєстрації нового користувача виокремлена у lib/actions/auth.ts і реалізує два сценарії: реєстрація без токена - автоматично створює нову фірму (Firm) у транзакції, присвоює реєстранту роль ADMIN та записує його як члена фірми у

firm_members; реєстрація за токеном - приєднує користувача до існуючої фірми з роллю та проєктом, визначеними в запрошенні. В обох випадках виконуються хешування пароля через bcryptjs та відповідні записи у firm_members і project_members.

Функція createProject доступна користувачам із роллю ADMIN або MANAGER. Під час реєстрації без токена користувач отримує роль ADMIN своєї фірми і може одразу створювати проєкти. Запрошені учасники зі звичайним токеном отримують роль MANAGER або MEMBER. Новий проєкт прив'язується до firmId поточного користувача та створюється із ним як менеджером проєкту. Додавання нових учасників реалізовано через механізм запрошень: менеджер або адміністратор формує запрошення з електронною поштою, роллю і токеном; дозволяється запрошувати лише користувачів, що вже є членами тієї самої фірми. Видалення учасника з проєкту виконується функцією removeProjectMember; при цьому записи в project_members видаляються, а задачі, призначені цьому учаснику, залишаються без виконавця (assigneeId = NULL).

4.2 Реалізація клієнтської частини

КІнтерфейс користувача побудовано на основі компонентної структури React. Сторінки застосунку розміщено в директорії app/, а повторно використовувані елементи, такі як форми, таблиці, картки задач і навігаційні блоки, винесено до директорії components/ [15].

4.2.1 Реалізація Kanban-дошки

Основна робота із задачами в системі виконується через Kanban-дошку, реалізовану в компоненті KanbanBoard (components/KanbanBoard.tsx). Оскільки дошка потребує інтерактивної взаємодії в браузері, компонент працює як клієнтський і зберігає поточний список задач та активну картку перетягування у локальному стані.

Для реалізації drag-and-drop використовується бібліотека @dnd-kit. Компонент DndContext обгортає всю дошку і налаштований із двома сенсорами: PointerSensor (активується після переміщення на 8 пікселів для запобігання випадковому перетягуванню при кліку) та KeyboardSensor (для доступності). Алгоритм виявлення колізій closestCorners забезпечує точне визначення цільового стовпця або задачі при відпусканні.

Після завершення перетягування в обробнику handleDragEnd інтерфейс одразу відображає задачу в новому стовпці. Такий підхід дає користувачу швидкий візуальний результат, тоді як збереження змін у базі даних виконується окремим серверним викликом. Паралельно асинхронно викликається Server Action moveTask. У разі помилки збереження локальний стан відновлюється до початкового значення та відображається сповіщення через бібліотеку Sonner.

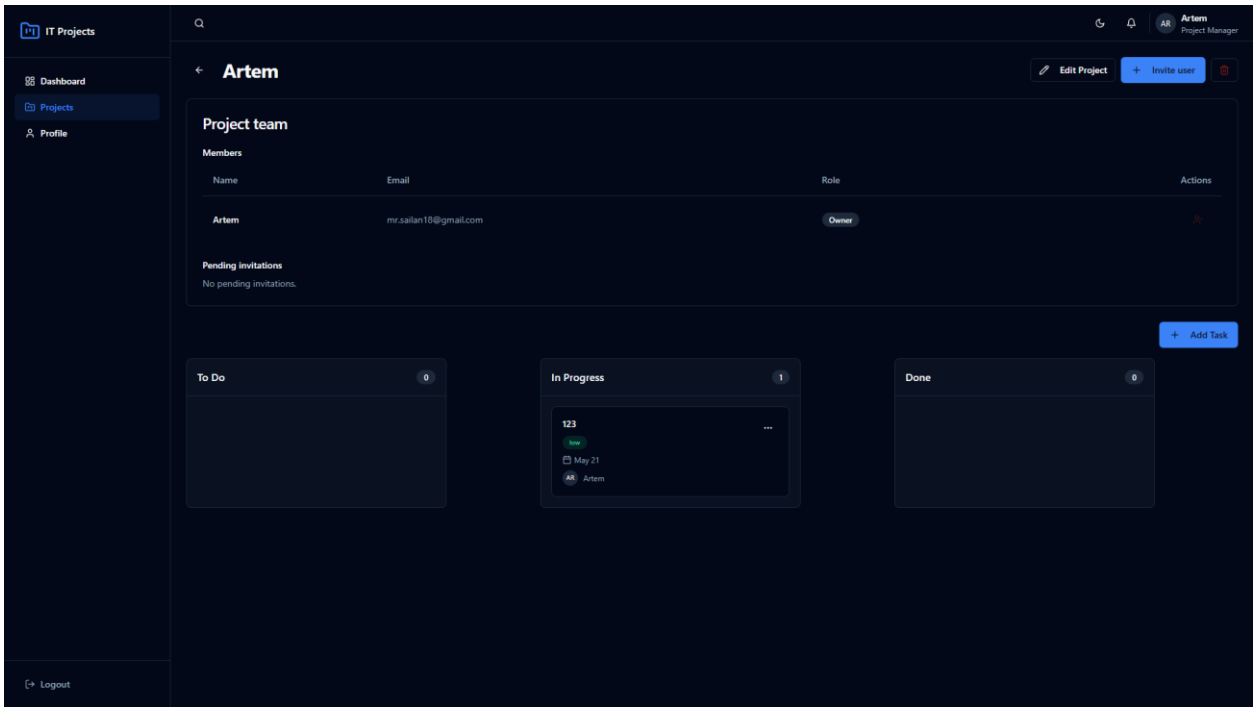


Рис. 4.3 Kanban-дошка системи управління IT-проєктами

Дошка складається з трьох фіксованих стовпців: To Do, In Progress та Done. Кожен стовпець реалізований як окремий компонент KanbanColumn, що використовує useDroppable з @dnd-kit для прийому задач, що перетягуються. Кожна задача відображається компонентом TaskCard, що містить назву, пріоритет

у вигляді кольорового бейджа, дедлайн, ім’я виконавця та кнопки редагування і видалення (відображаються відповідно до прав поточного користувача).

4.2.2 Реалізація інформаційної панелі

Інформаційна панель (Dashboard) реалізована у файлі app/(dashboard)/dashboard/page.tsx як серверний компонент. Під час відкриття сторінки викликається getDashboardData, яка вибирає з бази даних доступні користувачеві проєкти й пов’язані з ними задачі. На основі цих даних система формує показники для Dashboard: кількість проєктів, загальну кількість задач, задачі в роботі та прострочені задачі.

Статистика відображається компонентом DashboardStats у вигляді чотирьох карток із відповідними іконками (Folder, ListChecks, Clock, AlertTriangle). Картка прострочених задач виділяється червоним кольором за наявності прострочень. Кругова діаграма розподілу задач за статусами реалізована компонентом DashboardChart на основі PieChart із бібліотеки Recharts. Компонент DashboardKanbanPreview відображає попередній перегляд Kanban-дошки з агрегованими задачами всіх проєктів. Компонент RecentTasks відображає список нещодавно оновлених задач із зазначенням назви, статусу та пріоритету. У нижній частині інформаційної панелі розміщено стислий перелік активних проєктів з посиланнями на відповідні Kanban-дошки.

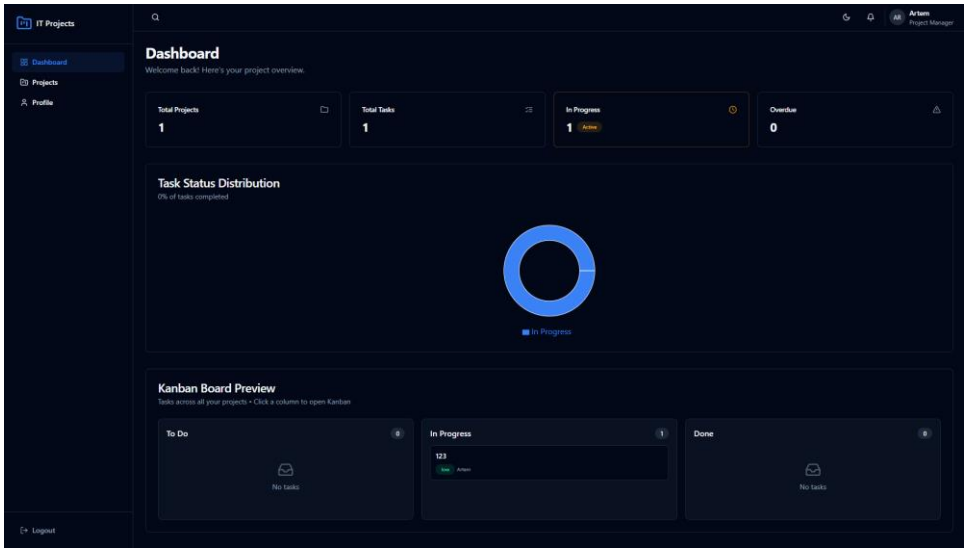


Рис. 4.4 Інформаційна панель системи (Dashboard)

4.2.3 Реалізація управління проєктами

Сторінка управління проєктами реалізована у файлі app/(dashboard)/projects/page.tsx як серверний компонент. Вона отримує список доступних проєктів із бази даних з урахуванням ролі поточного користувача: адміністратор бачить усі проєкти системи, менеджер - лише власні проєкти, учасник - проєкти, членом яких він є. Проєкти відображаються у вигляді таблиці з колонками: назва (з посиланням на Kanban-дошку), опис, менеджер, кількість задач, кількість учасників та кнопки дій.

Форма створення та редагування проєкту реалізована у компоненті ProjectsTable і відкривається у модальному діалозі shadcn/ui. Форма містить поля назви, опису та вибору учасників. Після відправки форми викликається відповідний Server Action (createProject або updateProject), що зберігає дані та інвалідує кеш сторінки.

Рис. 4.5. Сторінка управління проєктами

4.2.4 Реалізація пошуку

Функціонал пошуку реалізований на двох рівнях. Глобальний пошук у навігаційній панелі реалізований компонентом NavbarSearch і виконує пошук по

проєктах і задачах у реальному часі з debounce-затримкою для зменшення кількості запитів. Результати відображаються у випадаючому списку безпосередньо під полем пошуку.

Окрема сторінка розширеного пошуку /search реалізована як серверний компонент, що приймає пошуковий запит із параметрів URL і виконує фільтрацію проєктів і задач у базі даних через Prisma з використанням оператора contains. Результати групуються за типом (проєкти та задачі) і відображаються у вигляді карток із посиланнями на відповідні сторінки.

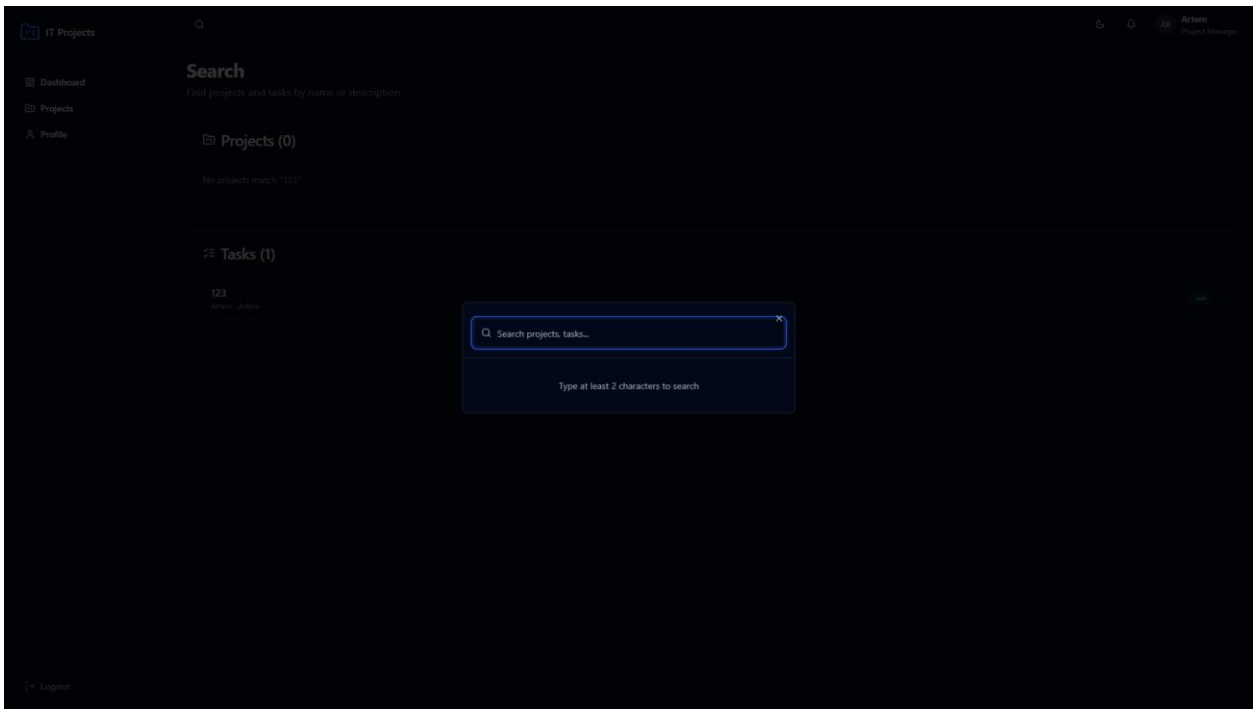


Рис. 4.6 Сторінка пошуку системи

4.2.5 Реалізація управління користувачами

Сторінка управління користувачами реалізована у файлі app/(dashboard)/users/page.tsx як серверний компонент і доступна виключно користувачам із роллю ADMIN. Спроба відкрити маршрут /users будь-яким іншим користувачем призводить до автоматичного перенаправлення на сторінку /forbidden. При завантаженні сторінки з бази даних отримується повний список зареєстрованих користувачів системи із зазначенням імені, електронної пошти, поточної ролі та дати реєстрації.

Список користувачів відображається компонентом UsersTable у вигляді таблиці з можливістю зміни ролі кожного користувача (ADMIN / MANAGER / MEMBER) через випадаючий список і видалення облікового запису. Обидві операції викликають відповідні Server Actions із модуля lib/actions/users.ts, що перевіряють наявність прав адміністратора перед виконанням у базі даних.

Компонент InvitationsTable забезпечує перегляд та управління запрошеннями до проєктів у межах адміністративної панелі. Компонент InviteUserButton у карточці проєкту дозволяє менеджеру або адміністратору запрошувати нових учасників за електронною поштою. Компонент NotificationsDropdown, вбудований у навігаційну панель (Navbar), відображає актуальні сповіщення для поточного користувача у вигляді випадаючого меню з лічильником непрочитаних повідомлень. Компонент FirmSwitcher, розміщений у навігаційній панелі, дозволяє користувачеві, який є членом кількох фірм, перемикатися між ними: при виборі іншої фірми викликається Server Action switchFirm, що оновлює firmId у сесії та перезавантажує сторінку.

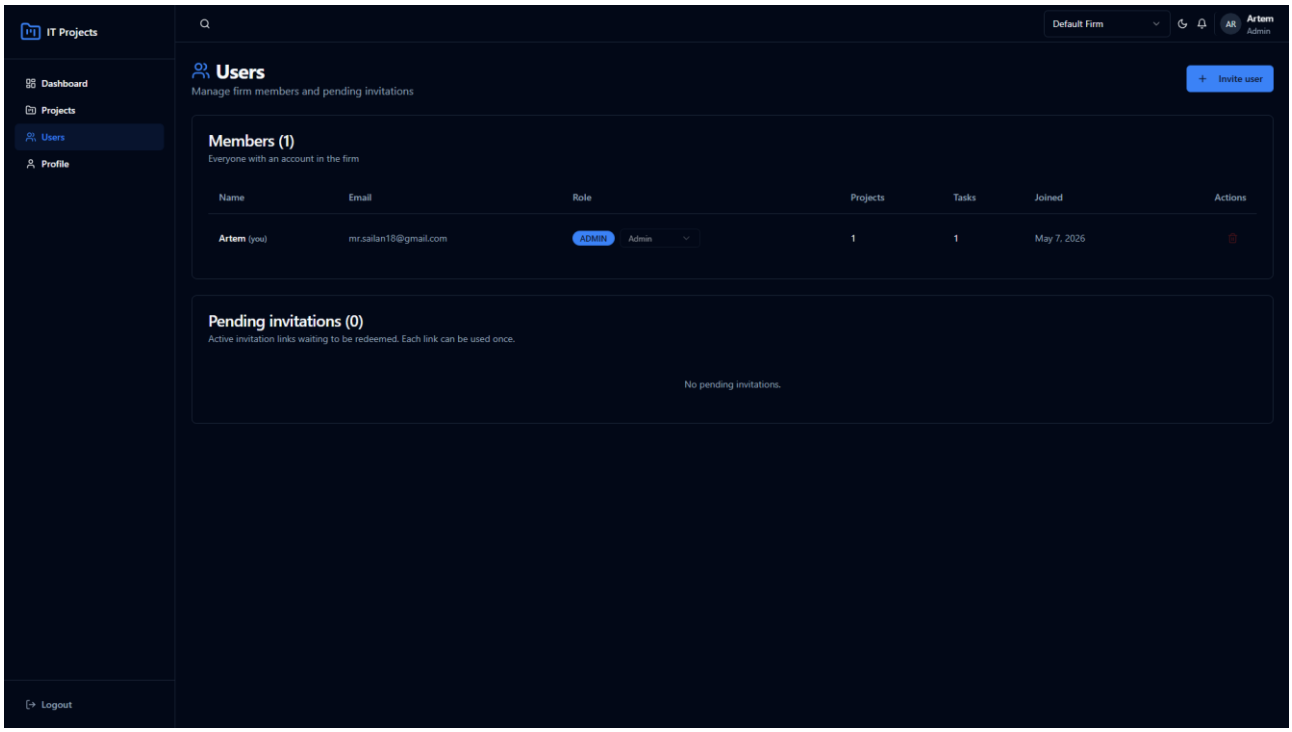


Рис. 4.7 Сторінка управління користувачами

4.3 Інтеграція компонентів системи

Інтеграція клієнтської та серверної частин системи забезпечується через механізм Server Actions Next.js, що дозволяє клієнтським компонентам викликати серверні функції безпосередньо, без явного HTTP-запиту. При цьому дані передаються у зашифрованому вигляді, а серверна функція виконується у захищеному середовищі [14].

Потік взаємодії між компонентами при виконанні типової операції (наприклад, переміщення задачі на Kanban-дошці) виглядає наступним чином: клієнтський компонент KanbanBoard отримує подію завершення перетягування від @dnd-kit; компонент виконує оптимістичне оновлення локального стану React; паралельно викликається Server Action moveTask із необхідними параметрами; сервер перевіряє сесію, права доступу та виконує оновлення бази даних; після успіху викликається revalidatePath для оновлення серверного кешу; клієнт отримує оновлені дані при наступному рендері.

Перемикання світлої та темної теми в системі виконано за допомогою next-themes і компонента ThemeToggle, розміщеного в навігаційній панелі. Вибрана тема зберігається у localStorage браузера і застосовується при кожному завантаженні сторінки без мерехтіння завдяки серверному рендерингу з атрибутом suppressHydrationWarning. Для відображення toast-сповіщень про результати операцій (успіх чи помилка) використовується бібліотека sonner, яка інтегрується через компонент Toaster у кореневому layout-файлі застосунку.

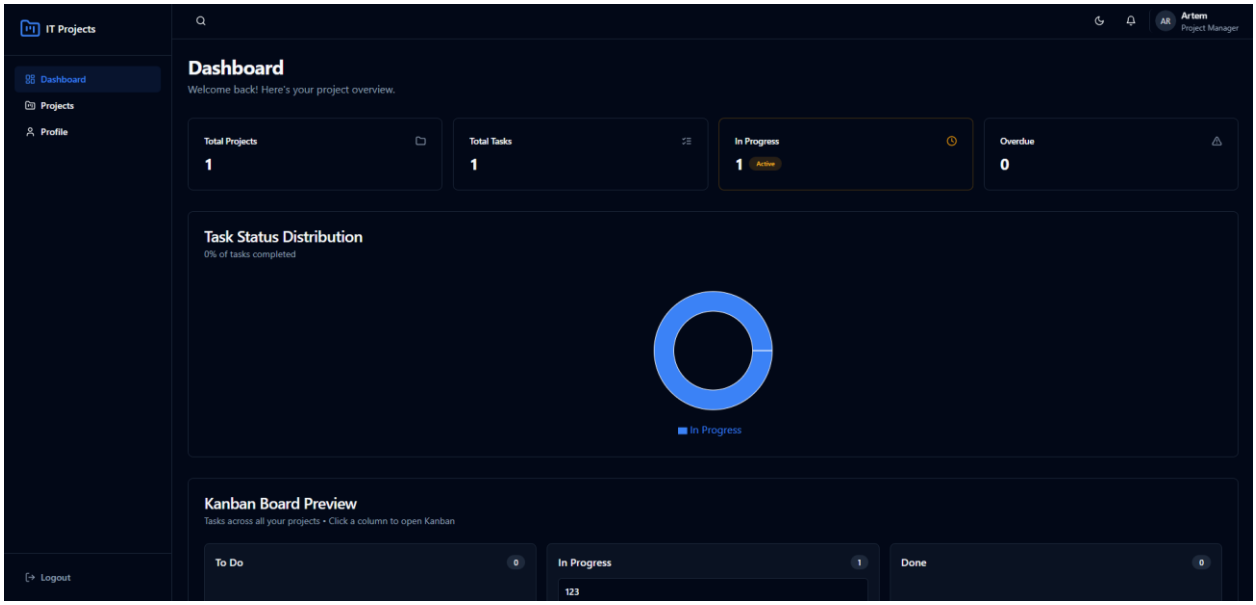


Рис. 4.8 Підтримка темної теми оформлення

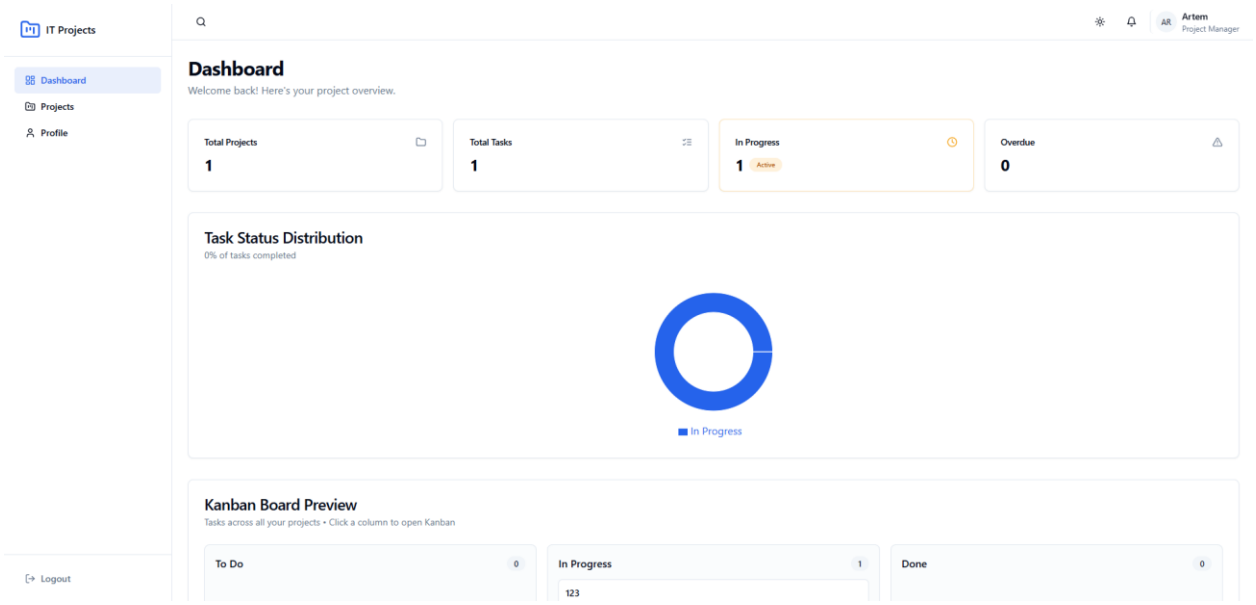


Рис. 4.9 Підтримка світлої теми оформлення

4.4 Тестування системи

Тестування інформаційної системи управління IT-проєктами проводилось методом ручного функціонального тестування. Метою тестування було перевірити коректність реалізації всіх заявлених функціональних вимог, а також перевірити систему захисту від несанкціонованого доступу [25].

4.4.1 Тестування модуля автентифікації

Тестування автентифікації охоплювало такі сценарії: реєстрація нового користувача без токена (очікуваний результат - автоматичне створення фірми, надання ролі ADMIN, перенаправлення на /login - результат пройдено); прийняття запрошення за валідним токеном (очікуваний результат - приєднання до фірми/проєкту з роллю, визначеною у запрошенні); спроба реєстрації з уже зареєстрованою електронною поштою (очікуваний результат - відображення повідомлення про помилку); вхід із коректними обліковими даними (успішна автентифікація); вхід із неправильним паролем (відображення помилки); спроба відкрити захищений маршрут без авторизації (перенаправлення на /login).

Усі тестові сценарії автентифікації пройдено успішно. Паролі коректно хешуються та перевіряються через bcryptjs. JWT-токен містить актуальні дані про роль користувача.

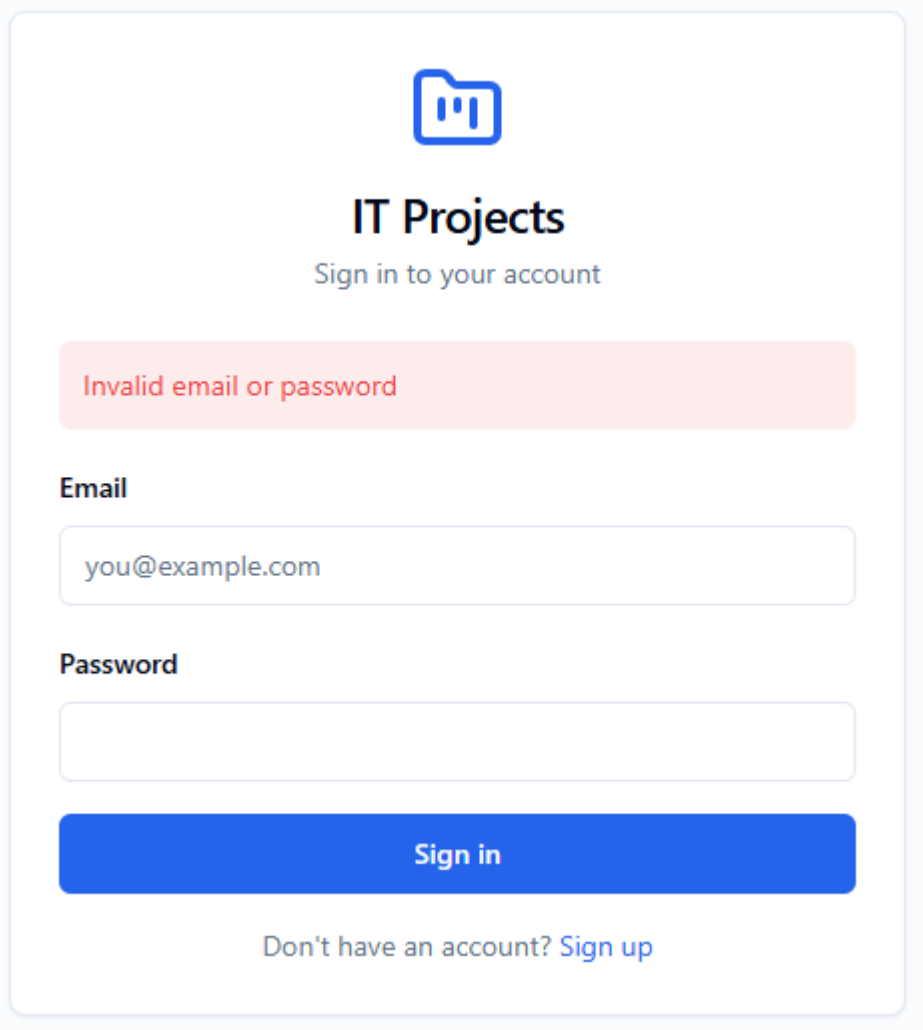


Рис. 4.10 Відображення помилки автентифікації

4.4.2 Тестування рольової моделі доступу

Тестування RBAC проводилось із трьома тестовими обліковими записами: Admin (роль ADMIN), Manager (роль MANAGER) та Member (роль MEMBER). Для кожного облікового запису перевірялась доступність функцій інтерфейсу та коректність виконання Server Actions.

Тестові сценарії для ролі ADMIN: доступ до всіх проєктів у межах активної фірми - пройдено; можливість редагувати та видаляти будь-який проєкт своєї фірми - пройдено; можливість переміщувати будь-яку задачу в межах доступних проєктів - пройдено.

Тестові сценарії для ролі MANAGER: видимість лише власних проєктів - пройдено; неможливість редагувати чужий проєкт (Server Action повертає помилку) - пройдено; можливість створювати та видаляти задачі у власному проєкті - пройдено.

Тестові сценарії для ролі MEMBER: обмежений доступ до проєктів, у яких користувач є учасником - пройдено; відсутність кнопок створення та видалення задач у недоступних сценаріях - пройдено; можливість працювати із задачами відповідно до прав у межах проєкту - пройдено; спроба викликати Server Action deleteTask напряму без необхідних прав - сервер повертає помилку «Only Admin or project Manager can delete tasks» - пройдено.

4.4.3 Тестування Kanban-дошки

Тестування Kanban-дошки охоплювало перевірку drag-and-drop функціональності, оптимістичного оновлення та коректності збереження даних. Перевірялись сценарії: переміщення задачі з To Do до In Progress - інтерфейс оновлюється миттєво, новий статус зберігається в БД; переміщення кількох задач підряд - порядок задач у стовпці зберігається коректно; оновлення сторінки після переміщення - задачі відображаються у правильних стовпцях.

Також перевірено сценарій помилки мережі: при симуляції відмови Server Action локальний стан коректно відновлюється до початкового, а користувач бачить сповіщення про помилку.

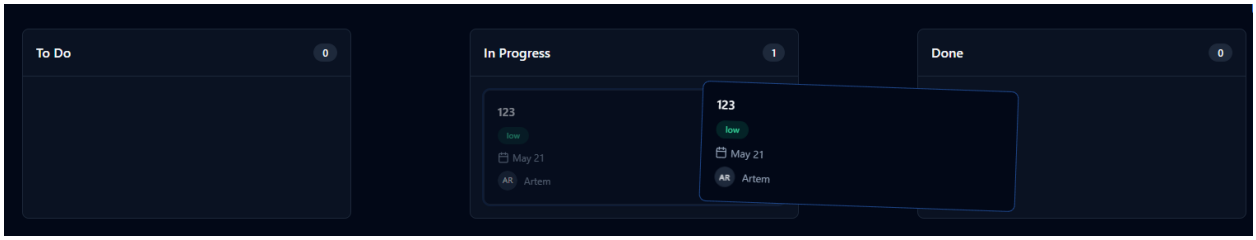


Рис. 4.11 Тестування drag-and-drop функціональності Kanban-дошки

Окремо перевірялась робота з кількома задачами підряд. У цьому випадку важливо було переконатися, що після декількох послідовних переміщень порядок задач у стовпцях не порушується, а дані залишаються узгодженими. Після оновлення сторінки задачі відображались у правильних стовпцях, що підтверджує успішне збереження змін у базі даних.

Додатково було протестовано сценарій виникнення помилки під час збереження змін. При симуляції відмови Server Action локальний стан Kanban-дошки коректно відновлювався до попереднього значення, а користувачу відображалося toast-сповіщення про помилку. Це дозволяє уникнути ситуації, коли в інтерфейсі показується один стан задачі, а в базі даних збережено інший.

Таким чином, тестування підтвердило коректну роботу Kanban-дошки, drag-and-drop взаємодії, оптимістичного оновлення інтерфейсу та механізму збереження змін через Server Actions. Усі перевірені тестові сценарії було пройдено успішно.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи створено вебзастосунок для організації роботи над IT-проєктами. Розроблена система дозволяє керувати проєктами, задачами й учасниками команди, розмежовувати права доступу між користувачами та відстежувати виконання робіт за допомогою Kanban-дошки.

У процесі виконання роботи отримано такі результати:

1. Проведено аналіз предметної галузі управління IT-проєктами. Розглянуто сучасні підходи до управління проєктами, зокрема agile-методології Scrum та Kanban. Виявлено ключові поняття предметної галузі: проєкт, задача, Kanban-дошка, рольова модель доступу. Визначено основні ролі учасників системи: адміністратор, менеджер проєкту та учасник.

2. Проведено дослідження популярних систем управління IT-проєктами, зокрема Jira, Trello та Asana. У процесі аналізу визначено, що ці інструменти не завжди є зручними для навчальних проєктів і невеликих команд, оскільки можуть мати складне налаштування, обмеження безкоштовних тарифів або недостатньо гнучку модель керування доступом. Отримані результати стали основою для формування функціональних і нефункціональних вимог до розроблюваної інформаційної системи.

3. Для реалізації системи використано стек вебтехнологій, який відповідає структурі розробленого застосунку: Next.js 15 забезпечує маршрутизацію, серверну логіку та Server Actions; TypeScript використовується для типізованої розробки; Prisma ORM і SQLite відповідають за роботу з даними; NextAuth.js забезпечує автентифікацію; Tailwind CSS і shadcn/ui застосовано для побудови інтерфейсу; @dnd-kit реалізує drag-and-drop взаємодію на Kanban-дошці, а Recharts використано для відображення статистики.

4. Спроєктовано архітектуру системи на основі трирівневої моделі (рівень представлення, бізнес-логіки та даних). Побудовано діаграми варіантів використання та діяльності, спроєктовано структуру маршрутів клієнтської

частини (включаючи захищений маршрут /users для адміністраторів), визначено склад серверних модулів (lib/validations.ts, lib/actions/auth.ts, lib/actions/firms.ts, lib/actions/users.ts, lib/actions/projects.ts, lib/actions/tasks.ts) та описано схему бази даних із сімома таблицями: Firm, User, FirmMember, Invitation, Project, ProjectMember та Task.

5. Реалізовано інформаційну систему управління IT-проєктами, що включає: модуль автентифікації та авторизації з JWT-сесіями і захистом маршрутів через middleware; реєстрацію без токена з автоматичним створенням фірми і роллю ADMIN для засновника; механізм запрошень із токеном для приєднання до існуючих фірм; модуль управління проєктами з підтримкою CRUD-операцій та управлінням складом команди; Kanban-дошку з drag-and-drop переміщенням задач між стовпцями та оптимістичним оновленням інтерфейсу; адміністративну сторінку управління користувачами (/users) із можливістю зміни ролей і видалення облікових записів; компонент сповіщень NotificationsDropdown; інформаційну панель зі зведеною статистикою та круговою діаграмою розподілу задач; модуль пошуку по проєктах і задачах; компонент FirmSwitcher для перемикання між фірмами; підтримку темної та світлої теми оформлення.

6. Реалізовано модель рольового контролю доступу (RBAC) з трьома рівнями прав: адміністратор фірми має повний доступ до ресурсів у межах своєї активної фірми; менеджер управляє власними проєктами та задачами в них; учасник має обмежений доступ до проєктів, у яких є членом, і може працювати із задачами відповідно до своєї ролі у проєкті. Перевірка прав реалізована на рівні Server Actions, що унеможливлює обхід обмежень через клієнтський код.

7. Проведено функціональне тестування системи за основними сценаріями: автентифікація, рольова модель доступу, робота з проєктами, механізм запрошень та Kanban-дошка. Усі перевірені сценарії виконано успішно, що підтверджує коректність реалізації основних функціональних вимог системи.

Робота пройшла апробацію. За її результатами опубліковано наступні тези доповідей:

1. Коломієць А.Г., Задонцев Ю.В. Розробка інформаційної системи управління ІТ-проєктами. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ 2026. С. 75-77.
2. Коломієць А.Г., Задонцев Ю.В. Архітектура інформаційної системи управління ІТ-проєктами. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез *(подано до друку)*.

ПЕРЕЛІК ПОСИЛАНЬ

1. Larson E. W., Gray C. F. Project Management: The Managerial Process. 8th ed. New York: McGraw-Hill Education, 2021. 640 p.

2. Schwalbe K. Information Technology Project Management. 9th ed. Boston: Cengage Learning, 2019. 688 p.

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

4. Brechner E. Agile Project Management with Kanban. Redmond: Microsoft Press, 2015. 176 p.

5. Ferraiolo D., Kuhn D. R., Chandramouli R. Role-Based Access Control. 2nd ed. Norwood: Artech House, 2007. 328 p.

6. Portman H., Strikwerda J. Organizational Project Management: Linking Strategy and Projects. Oxon: Routledge, 2022. 256 p.

7. Jira Software. Atlassian. URL: https://www.atlassian.com/software/jira.

8. Trello. Atlassian. URL: https://trello.com.

9. Asana: Manage Your Team's Work, Projects, & Tasks Online. URL: https://asana.com .

10. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education Limited, 2016. 816 p.

11. Wieringa R. J. Design Science Methodology for Information Systems and Software Engineering. Berlin: Springer, 2014. 332 p.

12. Next.js 15 Documentation. Vercel Inc. URL: https://nextjs.org/.

13. TypeScript Documentation. Microsoft Corporation. URL: https://www.typescriptlang.org/docs.

14. Next.js App Router: Server Actions and Mutations. Vercel Inc. URL: https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations.

15. React Documentation. Meta Platforms, Inc. URL: https://react.dev.

16. Tailwind CSS Documentation. Tailwind Labs Inc. URL: https://tailwindcss.com/docs.

17. @dnd-kit: A modern drag and drop toolkit for React. URL: https://docs.dndkit.com.

18. Recharts: A Redefined Chart Library Built with React and D3. URL: https://recharts.org/en-US.

19. Auth.js (NextAuth.js v5) Documentation. URL: https://authjs.dev.

20. SQLite Documentation. SQLite Consortium. URL: https://www.sqlite.org/docs.html.

21. Prisma ORM Documentation. Prisma Data, Inc. URL: https://www.prisma.io/docs.

22. Pro Git. Chacon S., Straub B. 2nd ed. New York: Apress, 2014. 456 p. URL: https://git-scm.com/book/en/v2.

23. shadcn/ui: Beautifully designed components built with Radix UI and Tailwind CSS. URL: https://ui.shadcn.com.

24. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston: Addison-Wesley Professional, 2005. 496 p.

25. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken: John Wiley & Sons, 2011. 240 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка інформаційної системи управління ІТ-проєктами

Виконав студент 4 курсу групи ПД-42
Коломієць Артем Григорович
Керівник роботи доцент кафедри, канд. техн. наук
Задонцев Юрій Вікторович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення планування, організації та контролю виконання ІТ-проєктів за рахунок розробки інформаційної системи з підтримкою Kanban-методології.

Об'єкт дослідження: процеси управління ІТ-проєктами та задачами в командному середовищі розробки програмного забезпечення.

Предмет дослідження: технології, методи та засоби розробки інформаційної системи управління ІТ-проєктами з підтримкою Kanban-методології, рольового контролю доступу та інструментів командної роботи.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі управління IT-проєктами та визначити процеси планування , контролю і моніторингу задач.
2. Проаналізувати існуючі програмні рішення для планування задач і командної роботи та використати результати аналізу для формування вимог до власної системи.
3. Визначити функціональні та нефункціональні вимоги до інформаційної системи.
4. Спроєктувати архітектуру системи, структуру клієнтської та серверної частин і базу даних.
5. Реалізувати інформаційну систему з управління IT-проєктами із підтримкою Kanban-дошки, рольового доступу, управління проєктами, задачами та користувачами.
6. Провести функціональне тестування системи за основними сценаріями роботи.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Jira	Trello	Asana	IT Projects
Управління задачами	розширене, але складне в налаштуванні	просте, але обмежене	зручне, але частково платне	задачі з дедлайнами й автопріоритетом
Kanban-дошка	багато налаштувань	проста дошка	доступна як режим перегляду	drag-and-drop Kanban-дошка
Рольова модель доступу	складне налаштування	обмежені ролі	частково залежить від тарифу	чітка RBAC-модель
Простота використання	складна (потребує навчання, перевантажений інтерфейс)	проста (інтуїтивний drag-and-drop інтерфейс)	середня (баланс функціоналу і складності)	проста (мінімалістичний інтерфейс, швидкий доступ до функцій)
Системні вимоги	високі (велике навантаження на браузер, потребує стабільного інтернету)	низькі (працює навіть на слабких пристроях)	середні (залежить від кількості даних)	низькі (оптимізована робота, мінімальні вимоги до клієнта)
Використання ресурсів	високі (значне споживання RAM та CPU)	низькі (мінімальне навантаження)	середні (помірне використання ресурсів)	оптимізовані (ефективне використання пам'яті та швидке завантаження)

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Створення та управління проєктами.
2. Додавання, редагування та видалення задач.
3. Встановлення дедлайнів та пріоритетів.
4. Відображення задач у вигляді Kanban-дошки.
5. Система ролей (керівник, учасник).
6. Моніторинг стану виконання задач.

Нефункціональні вимоги

1. Зрозуміла навігація, доступ до основних функцій не більше ніж за 2 кліки.
2. Обробка запитів користувача до 1 секунди при стандартному навантаженні.
3. Автентифікація користувачів через логін/пароль, збереження паролів у зашифрованому вигляді з використанням алгоритму bcrypt, використання сесій для контролю доступу, захищена передача даних через протокол HTTPS.
4. Підтримка одночасної роботи не менше 100 користувачів без помітного сповільнення системи.

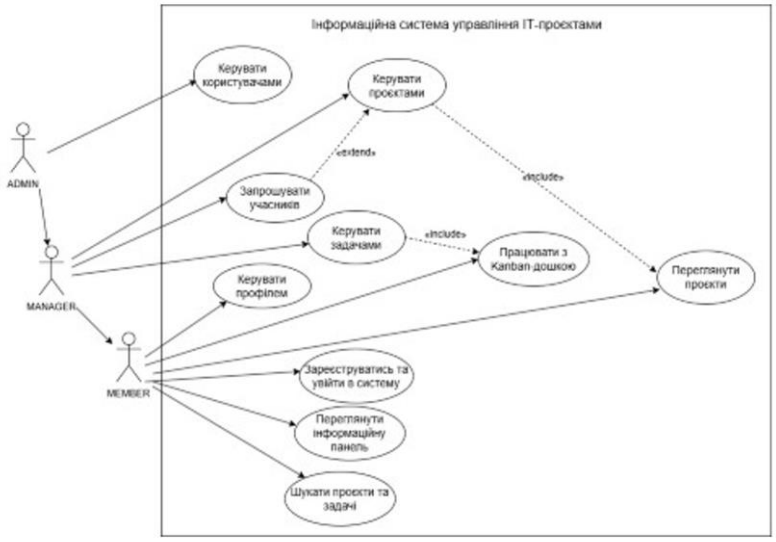
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

TypeScript

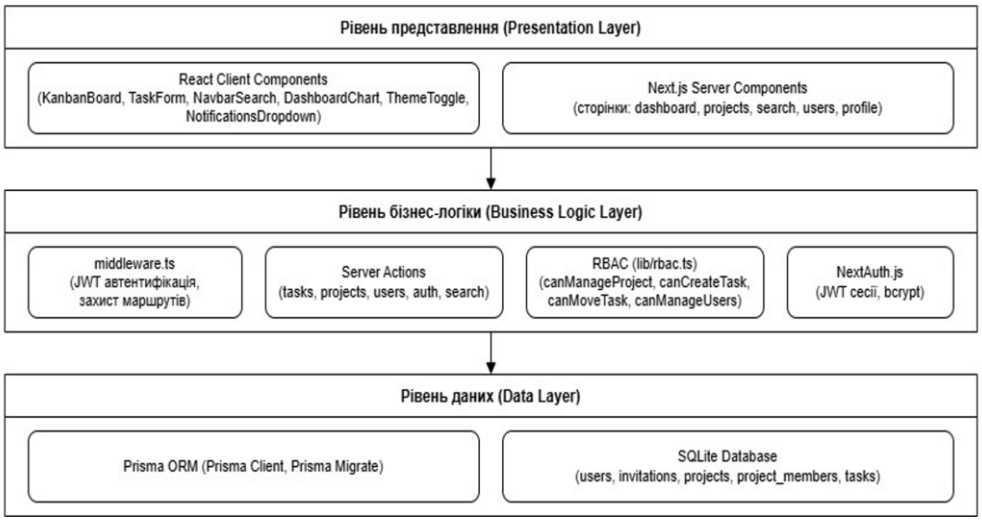
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



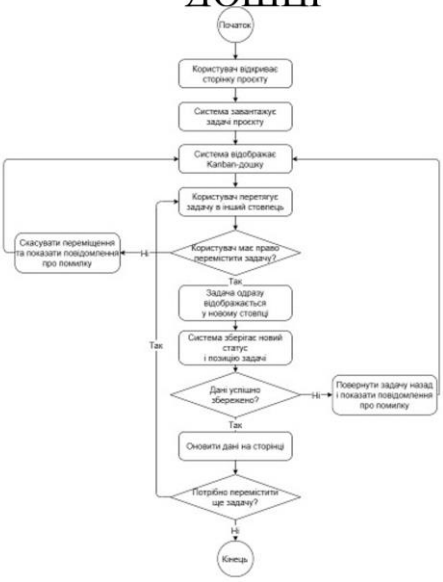
7

ЗАГАЛЬНА АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ



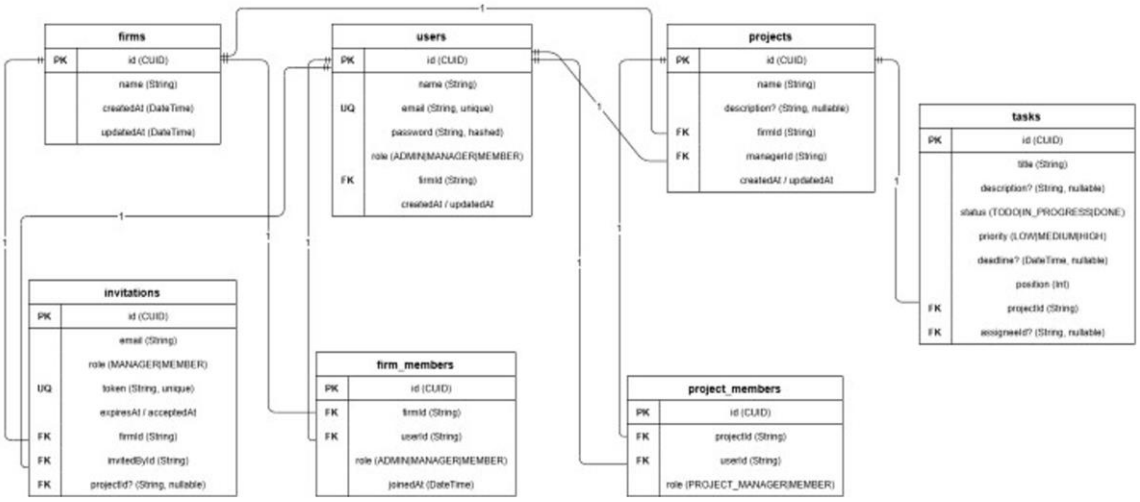
8

АЛГОРИТМ ЗМІНИ СТАТУСУ ЗАДАЧІ НА KANBAN - ДОШЦІ



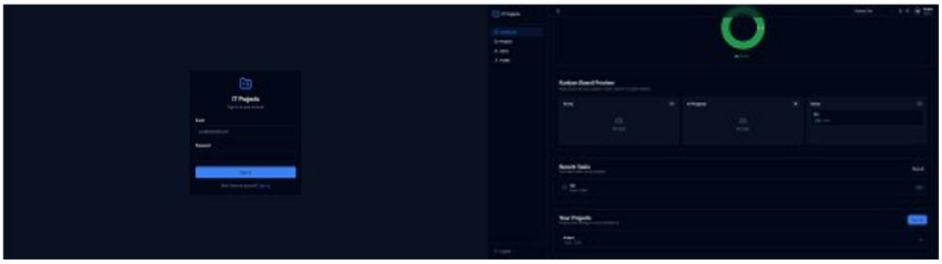
9

ER-ДІАГРАМА БАЗИ ДАНИХ СИСТЕМИ



10

ЕКРАННІ ФОРМИ



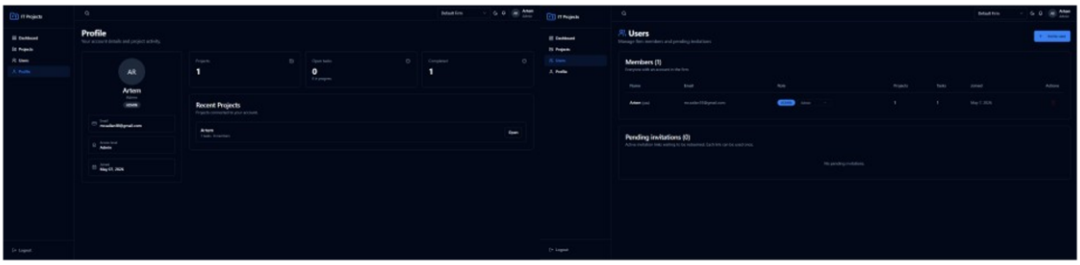
Скриншот 1.– сторінка авторизації

Скриншот 2. –сторінка канбан дошки

Скриншот3. - процес додавання завдання

Скриншот4. – сторінка зпроєктами

ЕКРАННІ ФОРМИ



Скриншот5. – сторінка профілю

Скриншот6. – сторінка управління учасниками

Скриншот7. – сторінка пошуку

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Коломієць А.Г., Задонцев Ю.В. Розробка інформаційної системи управління IT-проєктами. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ 2026. С. 75-77.
2. Коломієць А.Г., Задонцев Ю.В. Архітектура інформаційної системи управління IT-проєктами. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез *(подано до друку)*.

14

ВИСНОВКИ

1. Проведено аналіз предметної галузі, що дозволило визначити ключові процеси управління IT-проєктами та сформулювати вимоги до системи.
2. Проаналізовано існуючі рішення (Jira, Trello, Asana) та виявлено їхні основні недоліки: висока вартість, відсутність гнучкого рольового контролю та складність адаптації під потреби малих команд.
3. Сформовано функціональні та нефункціональні вимоги до інформаційної системи управління IT-проєктами.
4. Спроєктовано архітектуру інформаційної системи на основі Next.js App Router, React-компонентів, Server Actions, Prisma ORM та SQLite, що забезпечує логічне розділення інтерфейсу користувача, серверної бізнес-логіки та рівня роботи з даним.
5. Реалізовано веб-застосунок з Kanban-дошкою, рольовим контролем доступу, механізмом запрошень, Dashboard і пошуком.
6. Проведено функціональне тестування системи, яке підтвердило коректність основних сценаріїв роботи.

15

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Лістинг схеми бази даних Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Firm {
  id        String   @id @default(cuid())
  name      String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  users       User[]
  members     FirmMember[]
  projects    Project[]
  invitations Invitation[]

  @@map("firms")
}

model User {
  id        String   @id @default(cuid())
  name      String
  email     String   @unique
  password  String
  role      String   @default("MEMBER")
  firmId    String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  firm            Firm           @relation(fields: [firmId], references:
[id], onDelete: Cascade)
  firmMemberships FirmMember[]
  managedProjects Project[]      @relation("ProjectManager")
  memberProjects  ProjectMember[]
  assignedTasks   Task[]
  sentInvitations Invitation[]    @relation("InvitationInviter")

  @@index([firmId])
  @@map("users")
}

model FirmMember {
  id      String   @id @default(cuid())
  firmId  String
  userId  String
  role    String   @default("MEMBER")
  joinedAt DateTime @default(now())

  firm Firm @relation(fields: [firmId], references: [id],
onDelete: Cascade)
  user User @relation(fields: [userId], references: [id],
onDelete: Cascade)

  @@unique([firmId, userId])
  @@index([userId])
  @@map("firm_members")
}

model Invitation {
  id        String   @id @default(cuid())
  email     String
  role      String
  token     String   @unique
  expiresAt DateTime
  acceptedAt DateTime?
  invitedBy String
  firmId    String
  projectId String?
  firm      Firm     @relation(fields: [firmId], references: [id],
onDelete: Cascade)
  inviter   User     @relation("InvitationInviter", fields:
[invitedBy], references: [id], onDelete: Cascade)
  project   Project? @relation(fields: [projectId], references:
[id], onDelete: Cascade)
  createdAt DateTime @default(now())

  @@index([email])
  @@index([firmId])
  @@index([projectId])
  @@map("invitations")
}

model Project {
  id          String   @id @default(cuid())
  name        String
  description String?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  managerId   String
  firmId      String

  firm       Firm           @relation(fields: [firmId], references:
[id], onDelete: Cascade)
  manager    User           @relation("ProjectManager", fields:
[managerId], references: [id], onDelete: Cascade)
  members    ProjectMember[]
  tasks      Task[]
  invitations Invitation[]

  @@index([firmId])
  @@map("projects")
}

model ProjectMember {
  id        String   @id @default(cuid())
  projectId String
  userId    String
  role      String   @default("MEMBER")
```

```
  joinedAt  DateTime @default(now())

  project Project @relation(fields: [projectId], references: [id],
onDelete: Cascade)
  user    User    @relation(fields: [userId], references: [id],
onDelete: Cascade)

  @@unique([projectId, userId])
  @@map("project_members")
}

model Task {
  id          String    @id @default(cuid())
  title       String
  description String?
  status      String    @default("TODO")
  priority    String    @default("MEDIUM")
  deadline    DateTime?
  position    Int       @default(0)
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
  projectId   String
  assigneeId  String?

  project  Project @relation(fields: [projectId], references: [id],
onDelete: Cascade)
  assignee User?   @relation(fields: [assigneeId], references:
[id], onDelete: SetNull)

  @@map("tasks")
}
```

Лістинг модуля аутентифікації

```
import NextAuth from "next-auth";
import Credentials from "next-auth/providers/credentials";

export const { handlers, signIn, signOut, auth, unstable_update:
updateSession } = NextAuth({
  providers: [
    Credentials({
      name: "credentials",
      credentials: {
        email: { label: "Email", type: "email" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) return
null;
        const { prisma } = await import("./prisma");
        const bcrypt = await import("bcryptjs");
        const user = await prisma.user.findUnique({
          where: { email: credentials.email as string },
        });
        if (!user) return null;
        const valid = await bcrypt.compare(
          credentials.password as string,
          user.password
        );
        if (!valid) return null;
        return {
          id: user.id,
          name: user.name,
          email: user.email,
          role: user.role,
          firmId: user.firmId,
        };
      },
    }),
  ],
  callbacks: {
    async jwt({ token, user, trigger, session }) {
      if (user) {
        token.id = user.id;
        token.role = user.role;
        token.firmId = user.firmId;
      }
      if (trigger === "update" && session?.user) {
        if (session.user.role) token.role = session.user.role;
        if (session.user.firmId) token.firmId = session.user.firmId;
      }
      return token;
    },
    async session({ session, token }) {
      if (session.user) {
        session.user.id = token.id as string;
        session.user.role = token.role as string;
        session.user.firmId = token.firmId as string;
      }
      return session;
    },
  },
  pages: {
    signIn: "/login",
  },
  session: {
    strategy: "jwt",
  },
});
```

Лістинг посередника маршрутів

```
import { auth } from "@/lib/auth";

export default auth((req) => {
  const isLoggedIn = !!req.auth;
  const isLoginRoute =
req.nextUrl.pathname.startsWith("/login");
  const isRegisterRoute =
req.nextUrl.pathname.startsWith("/register");
  const isAuthRoute = isLoginRoute || isRegisterRoute;
  if (isLoginRoute && isLoggedIn) {
    return Response.redirect(new URL("/dashboard", req.url));
  }
  if (!isAuthRoute && !isLoggedIn && req.nextUrl.pathname
!== "/") {
    return Response.redirect(new URL("/login", req.url));
  }
  return undefined;
});
```

```
export const config = {
  matcher:
["/((?!api|_next/static|_next/image|favicon.ico|assets).*)"],
};
```

Лістинг модуля розмежування доступу (RBAC)

```
export type Role = "ADMIN" | "MANAGER" | "MEMBER";
export type ProjectRole = "PROJECT_MANAGER" |
"MEMBER";

export const ROLES: Role[] = ["ADMIN", "MANAGER",
"MEMBER"];
export const PROJECT_ROLES: ProjectRole[] =
["PROJECT_MANAGER", "MEMBER"];

export function isRole(value: unknown): value is Role {
  return value === "ADMIN" || value === "MANAGER" ||
value === "MEMBER";
}

export function isInviteRole(value: unknown): value is
Exclude<Role, "ADMIN"> {
  return value === "MANAGER" || value === "MEMBER";
}

export function isProjectRole(value: unknown): value is
ProjectRole {
  return value === "PROJECT_MANAGER" || value ===
"MEMBER";
}

function hasProjectManagerPower(projectRole?: string | null):
boolean {
  return projectRole === "PROJECT_MANAGER";
}

export function isAdmin(role: string): role is "ADMIN" {
  return role === "ADMIN";
}

export function canManageUsers(role: string): boolean {
  return role === "ADMIN";
}

export function canManageProject(
  role: string,
  projectManagerId: string,
  userId: string,
  projectRole?: string | null
): boolean {
  return (
    role === "ADMIN" ||
    ((role === "ADMIN" || role === "MANAGER") &&
projectManagerId === userId) ||
    hasProjectManagerPower(projectRole)
  );
}

export function canManageTasksInProject(
  role: string,
  projectManagerId: string,
  userId: string,
  projectRole?: string | null
): boolean {
  return canManageProject(role, projectManagerId, userId,
projectRole);
}

export function canCreateTask(
  role: string,
  projectManagerId: string,
  userId: string,
  projectRole?: string | null
): boolean {
  return canManageTasksInProject(role, projectManagerId,
userId, projectRole);
}

export function canDeleteTask(
  role: string,
  projectManagerId: string,
  userId: string,
  projectRole?: string | null
): boolean {
  return canManageTasksInProject(role, projectManagerId,
userId, projectRole);
}

export function canAssignTask(
  role: string,
  projectManagerId: string,
  userId: string,
  projectRole?: string | null
): boolean {
  return canManageTasksInProject(role, projectManagerId,
userId, projectRole);
}

export function canEditTask(
  role: string,
  projectManagerId: string,
  userId: string,
  taskAssigneeId: string | null,
  projectRole?: string | null
): boolean {
  if (!userId) return false;
  if (canManageTasksInProject(role, projectManagerId, userId,
projectRole)) return true;
  if (projectRole === "MEMBER") return true;
  return role === "MEMBER" && !!taskAssigneeId &&
taskAssigneeId === userId;
}

export function canMoveTask(
  role: string,
  projectManagerId: string,
  userId: string,
  taskAssigneeId: string | null,
  projectRole?: string | null
): boolean {
  return canEditTask(role, projectManagerId, userId,
taskAssigneeId, projectRole);
}
```

```
export function canAccessProject(
  role: string,
  projectManagerId: string,
  userId: string,
  isMember: boolean
): boolean {
  return role === "ADMIN" || projectManagerId === userId ||
isMember;
}
```

Лістинг серверних дій для роботи з задачами

```
"use server";

import { auth } from "@/lib/auth";
import { prisma } from "@/lib/prisma";
import { revalidatePath } from "next/cache";
import { getPriorityFromDeadline } from "@/lib/validations";
import {
  canCreateTask,
  canDeleteTask,
  canEditTask,
  canMoveTask,
  canAssignTask,
} from "@/lib/rbac";

const TASK_STATUSES = ["TODO", "IN_PROGRESS",
"DONE"] as const;

function isTaskStatus(status: string): status is (typeof
TASK_STATUSES)[number] {
  return TASK_STATUSES.includes(status as (typeof
TASK_STATUSES)[number]);
}

async function ensureProjectAccess(
  projectId: string,
  userId: string,
  role: string,
  firmId: string
) {
  const project = await prisma.project.findUnique({
    where: { id: projectId },
    include: { members: true },
  });
  if (!project) return { error: "Project not found" as const,
project: null };
  if (project.firmId !== firmId) {
    return { error: "Project not found" as const, project: null };
  }

  const isManager = project.managerId === userId;
  const membership = project.members.find((m) => m.userId
=== userId);
  const isMember = !!membership;
  const hasAccess = role === "ADMIN" || isManager ||
isMember;
  if (!hasAccess) return { error: "Access denied" as const,
project: null };

  return { error: null, project };
}

export async function createTask(projectId: string, formData:
FormData) {
  const session = await auth();
  if (!session?.user?.id || !session.user.firmId) return { error:
"Unauthorized" };

  const { error, project } = await ensureProjectAccess(
    projectId,
    session.user.id,
    session.user.role as string,
    session.user.firmId
  );
  if (error || !project) return { error: error ?? "Access denied" };

  const userProjectRole =
    project.members.find((m) => m.userId ===
session.user.id)?.role ?? null;

  if (!canCreateTask(session.user.role as string,
project.managerId, session.user.id, userProjectRole)) {
    return { error: "Only Admin or project Manager can create
tasks" };
  }

  const title = formData.get("title") as string;
  const description = (formData.get("description") as string) ||
"";
  const deadlineStr = formData.get("deadline") as string | null;
  const assigneeIdRaw = formData.get("assigneeId") as string;
  const assigneeId = assigneeIdRaw && assigneeIdRaw.trim()
? assigneeIdRaw : null;

  if (!title?.trim()) return { error: "Task title is required" };

  const deadline = deadlineStr ? new Date(deadlineStr) : null;
  const priority = getPriorityFromDeadline(deadline);

  const maxPos = await prisma.task.aggregate({
    where: { projectId, status: "TODO" },
    _max: { position: true },
  });
  const position = (maxPos._max.position ?? -1) + 1;

  const task = await prisma.task.create({
    data: {
      title: title.trim(),
      description: description.trim(),
      deadline,
      priority,
      projectId,
      assigneeId: assigneeId || null,
      status: "TODO",
      position,
    },
    include: {
      assignee: { select: { id: true, name: true } },
    },
  });
  revalidatePath(`/projects/${projectId}`);
  revalidatePath("/dashboard");
  return { success: true, task };
}

export async function updateTask(
  taskId: string,
```

```
  projectId: string,
  formData: FormData
) {
  const session = await auth();
  if (!session?.user?.id || !session.user.firmId) return { error:
"Unauthorized" };

  const { error, project } = await ensureProjectAccess(
    projectId,
    session.user.id,
    session.user.role as string,
    session.user.firmId
  );
  if (error || !project) return { error: error ?? "Access denied" };

  const task = await prisma.task.findUnique({ where: { id:
taskId } });
  if (!task) return { error: "Task not found" };

  const canEdit = canEditTask(
    session.user.role as string,
    project.managerId,
    session.user.id,
    task.assigneeId,
    project.members.find((m) => m.userId ===
session.user.id)?.role ?? null
  );
  if (!canEdit) return { error: "You can only edit your own
assigned tasks" };

  const title = formData.get("title") as string | null;
  const description = formData.get("description") as string |
null;
  const deadlineStr = formData.get("deadline") as string | null;
  const assigneeValue = formData.get("assigneeId");
  const assigneeId =
    typeof assigneeValue === "string" && assigneeValue.trim()
      ? assigneeValue
      : null;
  const status = formData.get("status") as string | null;

  const canChangeAssignee = canAssignTask(
    session.user.role as string,
    project.managerId,
    session.user.id,
    project.members.find((m) => m.userId ===
session.user.id)?.role ?? null
  );
  const canChangeDetails = canEdit;

  const updates: {
    title?: string;
    description?: string;
    deadline?: Date | null;
    assigneeId?: string | null;
    status?: "TODO" | "IN_PROGRESS" | "DONE";
    priority?: "LOW" | "MEDIUM" | "HIGH";
  } = {};

  if (title !== null && canChangeDetails) updates.title =
title.trim();
  if (description !== null && canChangeDetails)
updates.description = description.trim();
  if (deadlineStr !== null && canChangeDetails) {
    updates.deadline = deadlineStr ? new Date(deadlineStr) :
null;
    updates.priority = getPriorityFromDeadline(updates.deadline
?? task.deadline);
  }
  if (assigneeValue !== null && canChangeAssignee)
updates.assigneeId = assigneeId;
  if (status && canEdit) {
    if (!isTaskStatus(status)) return { error: "Invalid task status"
};
    updates.status = status;
  }

  await prisma.task.update({
    where: { id: taskId },
    data: updates,
  });
  revalidatePath(`/projects/${projectId}`);
  revalidatePath("/dashboard");
  return { success: true };
}

export async function deleteTask(taskId: string, projectId:
string) {
  const session = await auth();
  if (!session?.user?.id || !session.user.firmId) return { error:
"Unauthorized" };

  const { error, project } = await ensureProjectAccess(
    projectId,
    session.user.id,
    session.user.role as string,
    session.user.firmId
  );
  if (error || !project) return { error: error ?? "Access denied" };

  const userProjectRole =
    project.members.find((m) => m.userId ===
session.user.id)?.role ?? null;

  if (!canDeleteTask(session.user.role as string,
project.managerId, session.user.id, userProjectRole)) {
    return { error: "Only Admin or project Manager can delete
tasks" };
  }

  const task = await prisma.task.findUnique({ where: { id:
taskId } });
  if (!task) return { error: "Task not found" };

  await prisma.task.delete({ where: { id: taskId } });
  revalidatePath(`/projects/${projectId}`);
  revalidatePath("/dashboard");
  return { success: true };
}

export async function moveTask(
  taskId: string,
  projectId: string,
  newStatus: "TODO" | "IN_PROGRESS" | "DONE",
  newPosition: number
) {
  const session = await auth();
  if (!session?.user?.id || !session.user.firmId) return { error:
"Unauthorized" };
```

```
  if (!isTaskStatus(newStatus)) return { error: "Invalid task
status" };
  if (!Number.isInteger(newPosition) || newPosition < 0) {
    return { error: "Invalid task position" };
  }

  const { error, project } = await ensureProjectAccess(
    projectId,
    session.user.id,
    session.user.role as string,
    session.user.firmId
  );
  if (error || !project) return { error: error ?? "Access denied" };

  const task = await prisma.task.findUnique({ where: { id:
taskId } });
  if (!task) return { error: "Task not found" };
  if (task.projectId !== projectId) return { error: "Task not
found" };
  if (task.projectId !== projectId) return { error: "Task not
found" };
  if (task.projectId !== projectId) return { error: "Task not
found" };

  const userProjectRole =
    project.members.find((m) => m.userId ===
session.user.id)?.role ?? null;

  if (!canMoveTask(session.user.role as string,
project.managerId, session.user.id, task.assigneeId,
userProjectRole)) {
    return { error: "You can only move your own assigned tasks"
};
  }

  await prisma.task.update({
    where: { id: taskId },
    data: { status: newStatus, position: newPosition },
  });
  revalidatePath(`/projects/${projectId}`);
  revalidatePath("/dashboard");
  return { success: true };
}
```

Лістинг компонента Kanban-дошки

```
"use client";

import { useState, useCallback, useEffect } from "react";
import {
  DndContext,
  DragOverlay,
  closestCorners,
  KeyboardSensor,
  PointerSensor,
  useSensor,
  useSensors,
  type DragEndEvent,
  type DragStartEvent,
} from "@dnd-kit/core";
import { arrayMove } from "@dnd-kit/sortable";
import { KanbanColumn } from
"@/components/KanbanColumn";
import { TaskCard } from "@/components/TaskCard";
import { TaskForm } from "@/components/TaskForm";
import { Button } from "@/components/ui/button";
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogFooter,
} from "@/components/ui/dialog";
import { Plus } from "lucide-react";
import { Skeleton } from "@/components/ui/skeleton";
import { moveTask, deleteTask } from "@/lib/actions/tasks";
import { toast } from "sonner";
import { useRouter } from "next/navigation";
import {
  canCreateTask,
  canDeleteTask,
  canEditTask,
  canMoveTask,
  canAssignTask,
} from "@/lib/rbac";
import type { TaskCardPermissions } from
"@/components/TaskCard";
import type { Task } from "@prisma/client";

const COLUMNS = [
  { id: "TODO", title: "To Do" },
  { id: "IN_PROGRESS", title: "In Progress" },
  { id: "DONE", title: "Done" },
] as const;

type ColumnId = (typeof COLUMNS)[number]["id"];

const statusOrder: Record<ColumnId, number> = {
  TODO: 0,
  IN_PROGRESS: 1,
  DONE: 2,
};

function getStatusOrder(status: string) {
  return statusOrder[status as ColumnId] ??
Number.MAX_SAFE_INTEGER;
}

function isColumnId(status: string): status is ColumnId {
  return COLUMNS.some((column) => column.id === status);
}

type KanbanBoardProps = {
  projectId: string;
  projectManagerId: string;
  tasks: Task[];
  members: { id: string; name: string }[];
  userRole: string;
  userId: string;
  userProjectRole?: string | null;
};
```

```
export function KanbanBoard({
  projectId,
  projectManagerId,
  tasks: initialTasks,
  members,
  userRole,
  userId,
  userProjectRole = null,
}: KanbanBoardProps) {
  const router = useRouter();
  const [mounted, setMounted] = useState(false);
  const [tasks, setTasks] = useState(initialTasks);
  const [activeTask, setActiveTask] = useState<Task |
null>(null);
  const [addTaskOpen, setAddTaskOpen] = useState(false);
  const [editTask, setEditTask] = useState<Task | null>(null);
  const [deleteTaskId, setDeleteTaskId] = useState<string |
null>(null);

  useEffect(() => {
    setMounted(true);
  }, []);

  const canCreate = canCreateTask(userRole,
projectManagerId, userId, userProjectRole);

  const getTaskPermissions = useCallback(
    (task: Task): TaskCardPermissions => {
      if (!userId) {
        return { canEdit: false, canDelete: false, canAssign: false,
canDrag: false };
      }
      const assigneeId =
        task.assigneeId ?? (task as { assignee?: { id: string }
}).assignee?.id ?? null;
      return {
        canEdit: canEditTask(userRole, projectManagerId, userId,
assigneeId, userProjectRole),
        canDelete: canDeleteTask(userRole, projectManagerId,
userId, userProjectRole),
        canAssign: canAssignTask(userRole, projectManagerId,
userId, userProjectRole),
        canDrag: canMoveTask(userRole, projectManagerId,
userId, assigneeId, userProjectRole),
      };
    },
    [userRole, projectManagerId, userId, userProjectRole]
  );

  const sensors = useSensors(
    useSensor(PointerSensor, {
      activationConstraint: { distance: 8 },
    }),
    useSensor(KeyboardSensor)
  );

  const getTasksByStatus = useCallback(
    (status: ColumnId) =>
      tasks
        .filter((t) => t.status === status)
        .sort((a, b) => a.position - b.position),
    [tasks]
  );

  function handleDragStart(e: DragStartEvent) {
    const task = tasks.find((t) => t.id === e.active.id);
    if (task) setActiveTask(task);
  }

  async function handleDragEnd(e: DragEndEvent) {
    setActiveTask(null);
    const { active, over } = e;
    if (!over) return;

    const taskId = active.id as string;
    const task = tasks.find((t) => t.id === taskId);
    if (!task) return;

    const overId = over.id as string;
    const overIsColumn = COLUMNS.some((c) => c.id ===
overId);
    const overTask = tasks.find((t) => t.id === overId);

    if (!isColumnId(task.status)) return;

    let newStatus: ColumnId = task.status;
    let newPosition = task.position;

    if (overIsColumn) {
      if (!isColumnId(overId)) return;
      newStatus = overId;
      const colTasks = getTasksByStatus(newStatus);
      newPosition = colTasks.length;
    } else if (overTask) {
      if (!isColumnId(overTask.status)) return;
      newStatus = overTask.status;
      const colTasks = getTasksByStatus(newStatus);
      const overIndex = colTasks.findIndex((t) => t.id ===
overTask.id);
      newPosition = overIndex >= 0 ? overIndex :
colTasks.length;
    }

    if (newStatus === task.status && newPosition ===
task.position) return;

    if (!getTaskPermissions(task).canDrag) return;

    setTasks((prev) => {
      const without = prev.filter((t) => t.id !== taskId);
      const updated = { ...task, status: newStatus, position:
newPosition };
      const col = without
        .filter((t) => t.status === newStatus)
        .sort((a, b) => a.position - b.position);
      col.splice(newPosition, 0, updated);
      col.forEach((t, i) => (t.position = i));
      return without
        .filter((t) => t.status !== newStatus)
        .concat(col)
        .sort((a, b) => {
          return getStatusOrder(a.status) - getStatusOrder(b.status)
|| a.position - b.position;
        });
    });

    const result = await moveTask(
      taskId,
      projectId,
      newStatus,
```

```
    newPosition
  );

  if (result?.error) {
    toast.error(result.error);
    setTasks(initialTasks);
    router.refresh();
    return;
  }
  router.refresh();
 }

 async function handleDelete(task: Task) {
  setDeleteTaskId(task.id);
 }

 async function confirmDelete() {
  if (!deleteTaskId) return;
  const result = await deleteTask(deleteTaskId, projectId);
  if (result?.error) {
    toast.error(result.error);
    return;
  }
  setTasks((prev) => prev.filter((t) => t.id !== deleteTaskId));
  setDeleteTaskId(null);
  toast.success("Task deleted");
  router.refresh();
 }

 if (!mounted) {
  return (
    <div className="space-y-4">
      <div className="flex justify-end">
        {canCreate && (
          <Button disabled>
            <Plus className="h-4 w-4 mr-2" />
            Add Task
          </Button>
        )}
      </div>
      <div className="grid gap-4 grid-cols-1 md:grid-cols-2
lg:grid-cols-3 w-full">
        {COLUMNS.map((col) => {
          const colTasks = getTasksByStatus(col.id);
          return (
            <div
              key={col.id}
              className="flex flex-col flex-1 min-w-[320px] max-
w-[400px] rounded-lg border bg-muted/30"
            >
              <div className="flex items-center justify-between
p-4 border-b shrink-0">
                <h3 className="font-semibold">{col.title}</h3>
                <span className="rounded-md bg-muted px-2 py-
0.5 text-xs font-medium text-muted-foreground">
                  {colTasks.length}
                </span>
              </div>
              <div className="flex-1 p-4 space-y-2">
                {colTasks.length === 0 ? (
                  <Skeleton className="h-24 w-full rounded-lg" />
                ) : (
                  colTasks.slice(0, 3).map((t) => (
                    <Skeleton key={t.id} className="h-24 w-full
rounded-lg" />
                  ))
                )}
              </div>
            </div>
          );
        })}
      </div>
    </div>
  );
 }

 return (
   <div className="space-y-4">
     <div className="flex justify-end">
       {canCreate && (
         <Button onClick={() => setAddTaskOpen(true)}>
           <Plus className="h-4 w-4 mr-2" />
           Add Task
         </Button>
       )}
     </div>

     <DndContext
       sensors={sensors}
       collisionDetection={closestCorners}
       onDragStart={handleDragStart}
       onDragEnd={handleDragEnd}
     >
       <div className="grid gap-4 grid-cols-1 md:grid-cols-2
lg:grid-cols-3 w-full">
         {COLUMNS.map((col) => (
           <KanbanColumn
             key={col.id}
             id={col.id}
             title={col.title}
             tasks={getTasksByStatus(col.id)}
             onEditTask={(t) => setEditTask(t)}
             onDeleteTask={handleDelete}
             getTaskPermissions={getTaskPermissions}
           />
         ))}
       </div>

       <DragOverlay>
         {activeTask ? (
           <div className="rotate-2 scale-105">
             <TaskCard
               task={activeTask}
               onEdit={() => {}}
               onDelete={() => {}}
               permissions={{
                 canEdit: false,
                 canDelete: false,
                 canAssign: false,
                 canDrag: false,
               }}
             />
           </div>
         ) : null}
       </DragOverlay>
     </DndContext>

     <TaskForm
       open={addTaskOpen}
       onOpenChange={setAddTaskOpen}
```

```
        projectId={projectId}
        members={members}
        canAssign={canAssignTask(userRole, projectManagerId,
userId, userProjectRole)}
        onSuccess={(newTask) => {
          setAddTaskOpen(false);
          if (newTask) {
            const task = {
              ...newTask,
              deadline: newTask.deadline ? new
Date(newTask.deadline) : null,
              createdAt: new Date(newTask.createdAt),
              updatedAt: new Date(newTask.updatedAt),
            };
            setTasks((prev) => [...prev, task]);
          }
          router.refresh();
        }}
      />

      {editTask && (
        <TaskForm
          open={!!editTask}
          onOpenChange={(open) => !open &&
setEditTask(null)}
          projectId={projectId}
          editTask={editTask}
          members={members}
          canAssign={canAssignTask(userRole,
projectManagerId, userId, userProjectRole)}
          onSuccess={() => {
            router.refresh();
            setEditTask(null);
          }}
        />
      )}

      <Dialog open={!!deleteTaskId} onOpenChange={() =>
setDeleteTaskId(null)}>
        <DialogContent>
          <DialogHeader>
            <DialogTitle>Delete Task</DialogTitle>
          </DialogHeader>
          <p>Are you sure you want to delete this task?</p>
          <DialogFooter>
            <Button variant="outline" onClick={() =>
setDeleteTaskId(null)}>
              Cancel
            </Button>
            <Button variant="destructive"
onClick={confirmDelete}>
              Delete
            </Button>
          </DialogFooter>
        </DialogContent>
      </Dialog>
    </div>
  );
}
```