КНУБА

SaaS-платформа для керування мікропроєктами у малих командах

Здобувач: Луцишин Дмитро Сергійович

Керівник: к.т.н., доцент Гуменний Дмитро Олександрович

Кафедра: Кібербезпеки та комп’ютерної інженерії

КИЇВ - 2025

Загальна характеристика

магістерської роботи

Актуальність

Інтенсивне зростання малих IT-команд та стартапів виявляє критичну прогалину: **існуючі Enterprise-системи є надто громіздкими**, а "легкі" трекери **ігнорують психоемоційний стан виконавців**, що призводить до професійного вигорання.

Проблема дослідження

Малі команди стикаються з дилемою: обирати між надмірною складністю інструментів або відсутністю життєво важливої аналітики навантаження. Це створює неефективність та стрес.

Мета роботи

Розробка інноваційної SaaS-платформи, яка поєднає **гнучкість Kanban-методології** з інструментами запобігання вигоранню, підвищуючи ефективність та добробут команди.

Ключові аспекти дослідження

Задачі

- Аналіз існуючих систем та їх недоліків.
- Обґрунтування архітектури та стеку технологій.
- Проєктування схеми БД з Row-Level Security.
- Розробка алгоритмів Fractional Indexing та Well-being аналітики.
- Реалізація та тестування програмного продукту.

Методи

- Системний аналіз для формування вимог.
- Об'єктно-орієнтоване проєктування архітектури.
- Математичне моделювання для алгоритмів.
- Емпіричні методи для тестування продуктивності.

Наукова новизна

Удосконалення методу керування чергою завдань завдяки поєднанню **алгоритму Fractional Indexing** з моделлю оцінки навантаження користувача (Well-being).

Робота складається з:

- Вступу
- 3 Розділів
- Висновків
- Додатків
- Рецензії

Проблематика керування мікропроєктами та Human Factor

Криза існуючих інструментів

Enterprise-рішення (Jira): Надлишковий функціонал, високий поріг входження, сповільнення процесів у малих командах.

Легкі трекери (Trello, Todoist): Відсутність глибинної аналітики та інструментів для довгострокового планування.

Ігнорування психоемоційного стану

Сучасні системи зосереджені лише на статусі задачі (To Do → Done), повністю **ігноруючи стан виконавця**.

Актуальність

Зростання випадків професійного вигорання (burnout) в IT-сфері вимагає інтеграції метрик Well-being безпосередньо у робочий процес, що є критично важливим для стабільності та продуктивності команди.

Джерела проблематики:

** Tkachenko O. M. Comparative Analysis of Architectural Solutions for Building SaaS Platforms // Bulletin of NTUU "KPI". Informatics, Operation and Computer Science. — 2022. — No. 75. — P. 45–52.*

** Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). — 7th ed. — Newtown Square : PMI, 2021. — 370 p.*

Порівняльний аналіз існуючих рішень

	Jira	Notion	Trello / asana	
UX/Швидкість	Низька	Висока	Середня	Висока
Гнучкість	Висока (складна)	Середня	Дуже висока (неструктурована)	Оптимальна
Вартість	Висока	Середня/Низька	Середня	Доступна
Well-being метрики	Відсутні	Відсутні	Відсутні	Інтегровані
RLS (безпека)	Комплексна	Слабка	Слабка	Інтегрована

Рішення пропонує **гібридний підхід**, що поєднує інтуїтивність Kanban-дошки з інтегрованим Well-being моніторингом та надійною Row-Level Security, заповнюючи критичні прогалини ринку.

Вибір технологій та архітектурні рішення

Backend: Golang

Обраний для забезпечення **високої конкурентності** (goroutines) та ефективного використання пам'яті, що є критичним для Real-time оновлень.

Frontend: React + TypeScript

Гарантує **надійність коду** через типізацію та **високу швидкодію** інтерфейсу завдяки використанню Virtual DOM для Single Page Application (SPA).

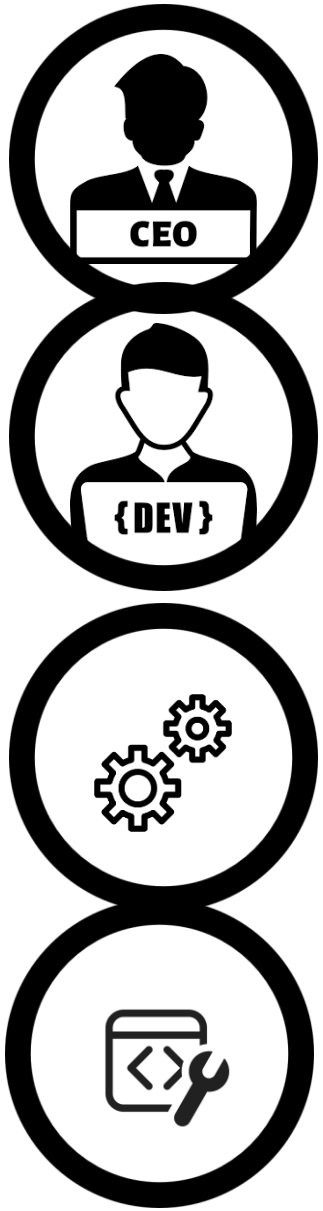
База даних: PostgreSQL + RLS

Використання реляційної моделі забезпечує **цілісність даних**. Row-Level Security (RLS) є ключовим архітектурним рішенням для **ізоляції даних** різних команд (Multitenancy) на рівні БД, що підвищує безпеку.

Алгоритмічна база: Fractional Indexing

Цей алгоритм забезпечує **оптимізоване сортування завдань** зі складністю **O(1)** замість O(n), значно покращуючи продуктивність.

Аналіз вимог та сценаріїв використання



Актор: Адміністратор/Менеджер

Створення дощок, запрошення учасників, моніторинг аналітики вигорання команди.

Актор: Користувач (Розробник)

Виконання операцій CRUD із завданнями, зміна статусів, автоматичний трекінг активності.

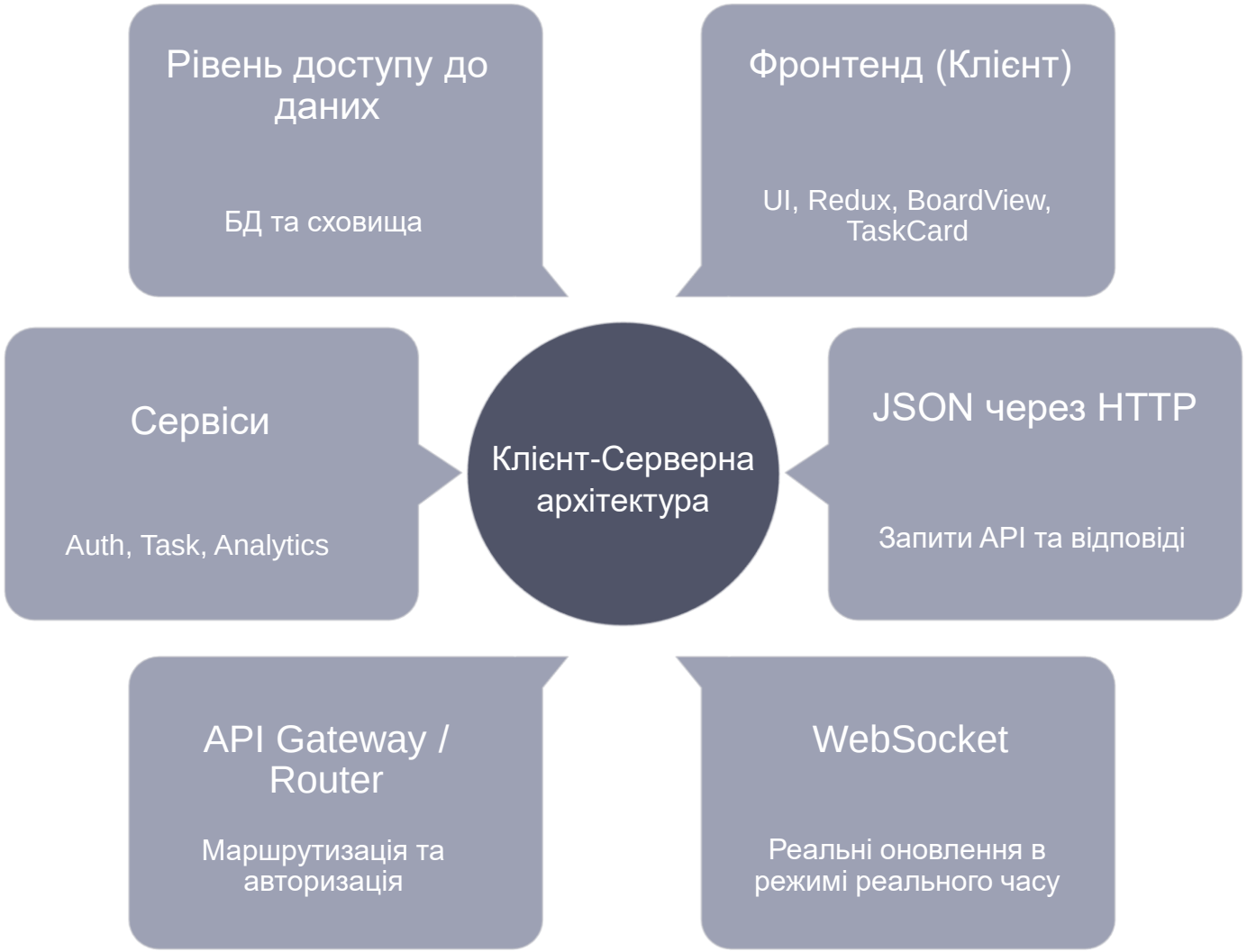
Ключові функціональні вимоги

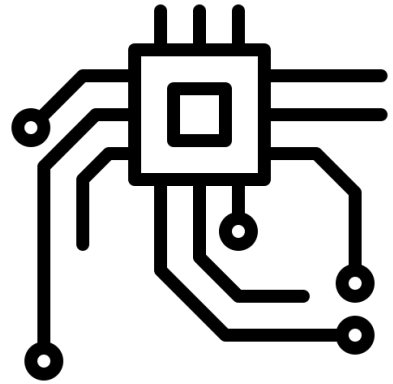
- Підтримка Drag-and-Drop інтерфейсу.
- Система "Тенантів" для ізоляції даних команд.
- Збір метрик для розрахунку Well-being індексу.

Нефункціональні вимоги

Швидкість відгуку інтерфейсу < 100мс (Optimistic UI) та безпека даних (JWT).

Компонентна архітектура системи





Frontend (Клієнт)

Відповідає за відображення компонентів (BoardView, TaskCard) та логічні модулі (Redux/State Management), забезпечуючи інтерактивність та швидкість інтерфейсу.

Backend (Сервер)

Включає API Gateway/Router для обробки запитів, Services Layer для бізнес-логіки (AuthService, TaskService, AnalyticsService) та Data Access Layer для роботи з БД.

Інфологічна модель та схема бази даних

Тип БД: Реляційна (PostgreSQL)

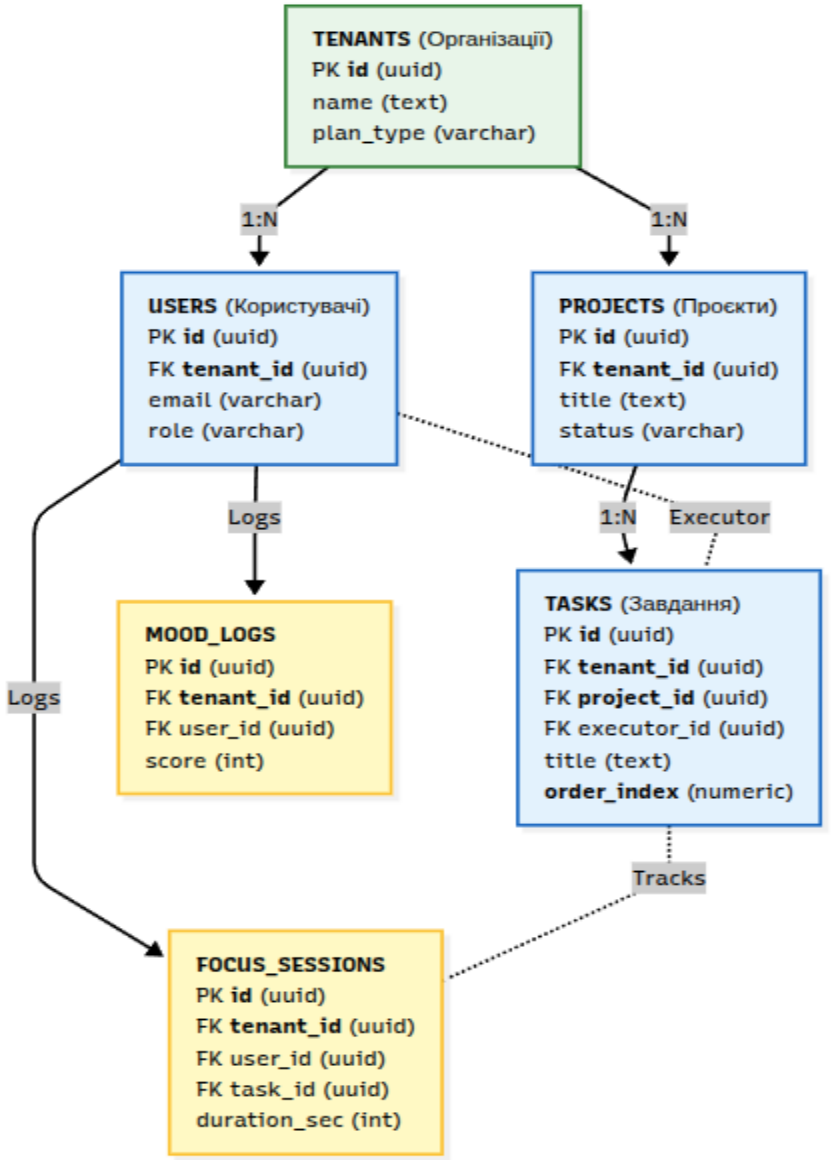
Обрана для забезпечення **цілісності даних (ACID)** та надійності складних взаємозв'язків між сутностями.

Ключові сутності

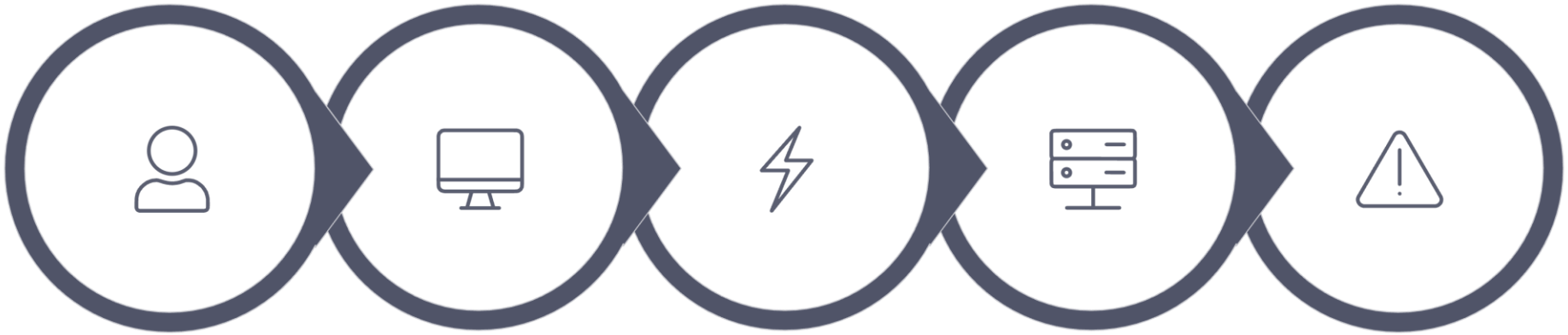
Users, Workspaces (Тенанти), а також ієрархічна структура **Boards -> Columns -> Tasks** для організації проєктів.

Особливості проєктування

Використання `order_index` **(float/double)** у таблиці Tasks для реалізації **алгоритму Fractional Indexing**, що дозволяє ефективно вставляти завдання без перерахунку черги. Реалізовано зв'язки One-to-Many з каскадним видаленням.



Діаграма послідовності переміщення задачі



01

Optimistic UI Update

Інтерфейс миттєво відображає переміщення задачі, забезпечуючи бездоганний користувацький досвід и без затримок.

02

Async Request

Фонове відправлення запиту на сервер дозволяє інтерфейсу залишатися чуйним, не блокуючи взаємодію користувача.

03

Server Validation & Update

Сервер перевіряє права доступу та коректність індексів, оновлюючи базу даних.

04

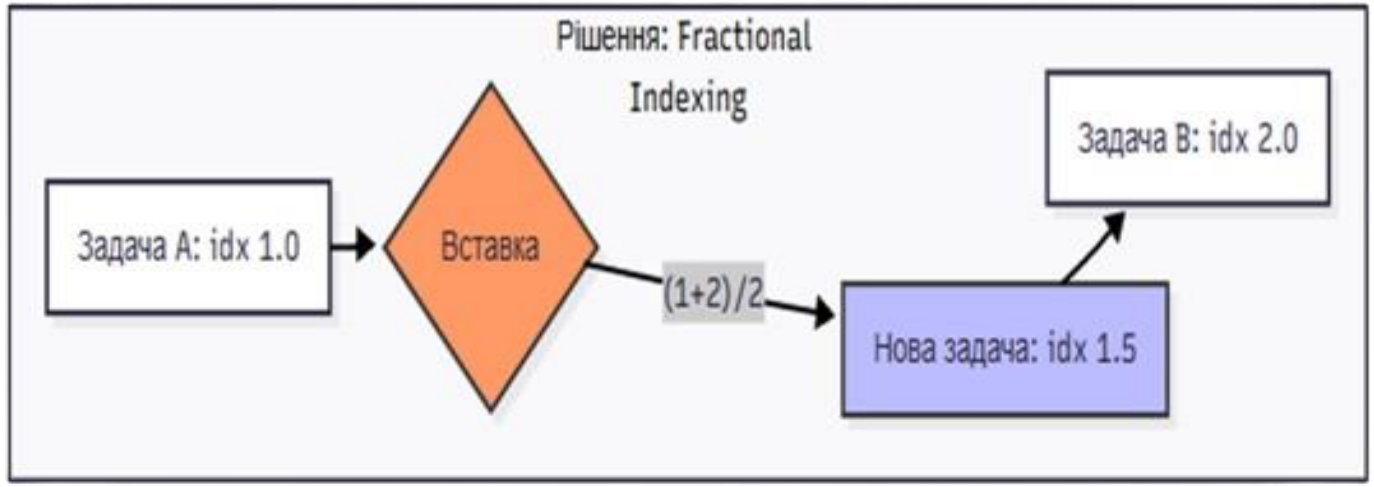
Error Handling (Rollback)

У випадку помилки від сервера, інтерфейс автоматично "відкочує" задачу на попереднє місце, зберігаючи цілісність даних.

Оптимізація зміни порядку задач: Fractional Indexing

Переваги та обробка Edge Case

Складність операції переміщення знижено до O(1), оновлюється лише один рядок. Обробка переповнення точності Float здійснюється через періодичне перебалансування індексів.

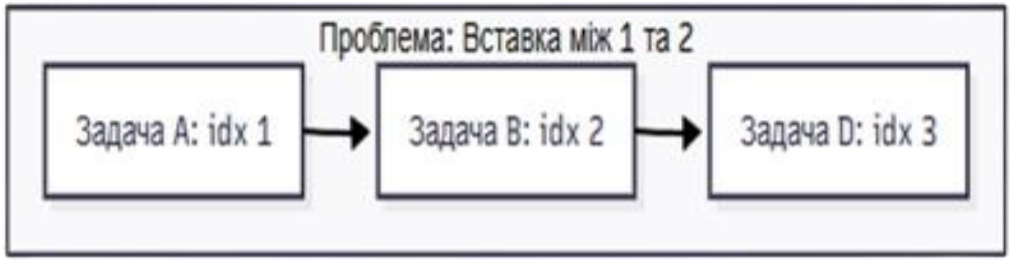


Проблема
класичного підходу

Традиційне індексування (1, 2, 3...) вимагає оновлення індексів усіх наступних задач у колонці (складність O(n)) при вставці нової задачі. Це створює значне навантаження на базу даних.

Рішення
Дробове індексування

Використання дробових значень (Float) для індексів. Наприклад, для вставки між Задачами А (індекс 1000) та Б (індекс 2000) нова задача отримує індекс 1500 (середнє значення).



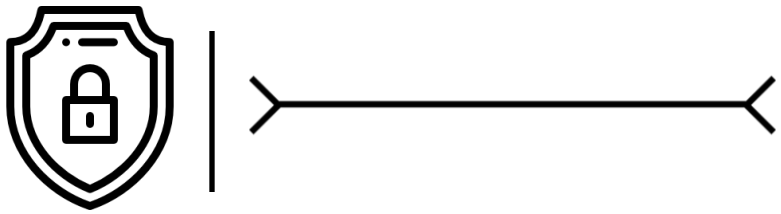
Захист даних та ізоляція тенантів (Multi-tenancy)

Принцип "Defense in Depth"

Безпека реалізується не лише на рівні API, але й глибоко інтегрується на рівні бази даних. Цей багаторівневий підхід забезпечує надійний захист даних від несанкціонованого доступу.

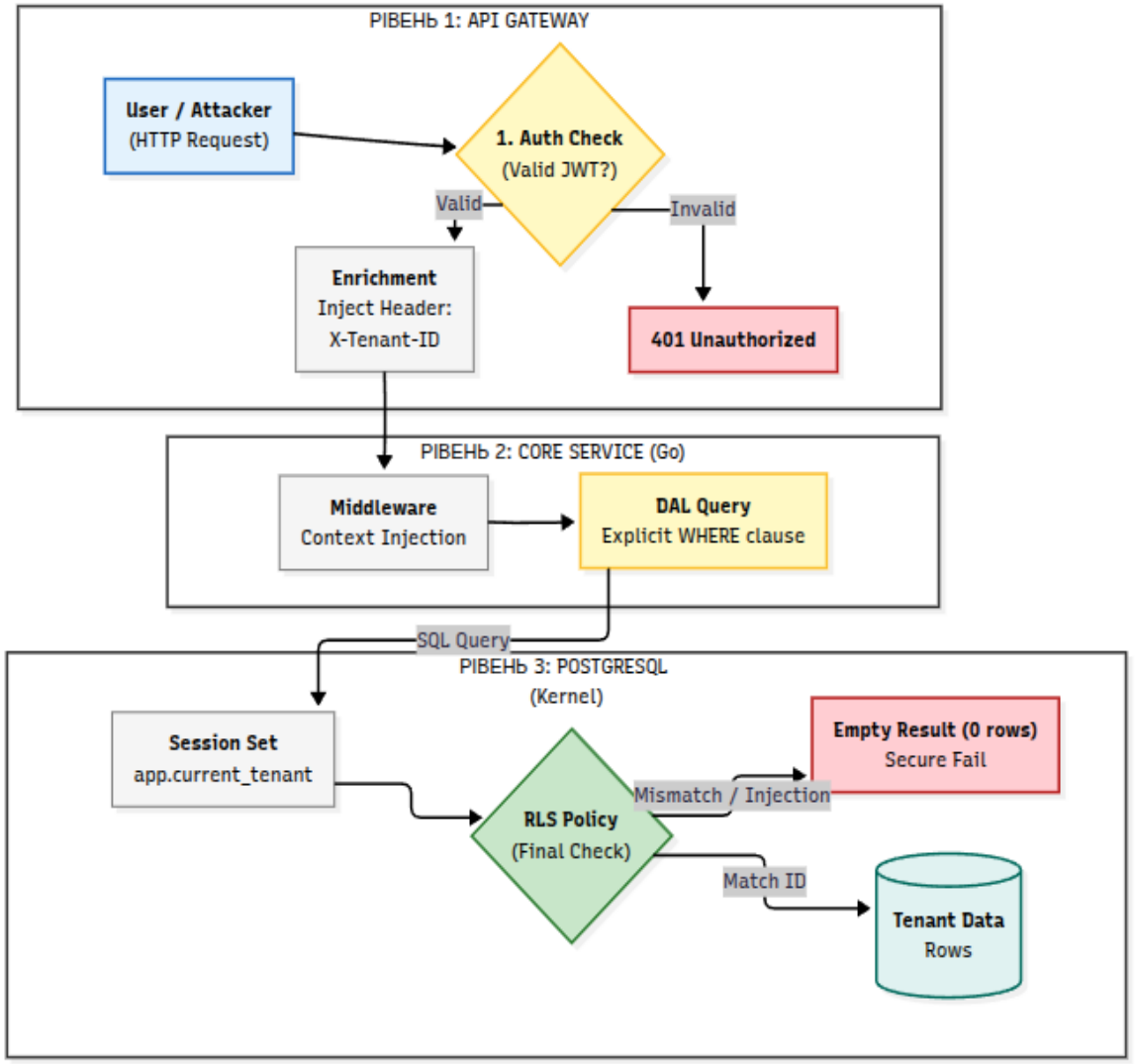
Механізм RLS (PostgreSQL)

Кожна таблиця містить обов'язкове поле `tenant_id`. Створені SQL-політики (Policies) автоматично фільтрують рядки, гарантуючи, що користувач бачить лише свої дані.



Приклад політики ізоляції

```
CREATE POLICY tenant_isolation ON tasks USING
(tenant_id = current_setting('app.current_tenant'));
```



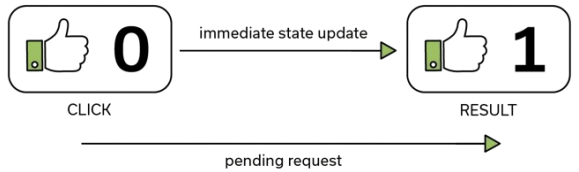
Реалізація Frontend: State Management та Optimistic UI

1

Технологічний Стек

Використання **React (Hooks)** для компонентної архітектури та **Redux Toolkit** для ефективного управління станом програми. Це забезпечує передбачуваність і легкість масштабування.

2



Логіка переміщення (DnD) та Optimistic UI

Миттєва зміна стану в Redux Store при перетягуванні елементів. Користувач бачить зміни одразу, що покращує взаємодію. Фоновий запит надсилається на сервер, не блокуючи інтерфейс.

3

Rollback Pattern

У разі помилки сервера, стан інтерфейсу автоматично повертається до попереднього значення. Цей механізм забезпечує надійність та бездоганний досвід користувача.

4

Взаємодія компонентів

Використання `DragDropContext` (бібліотека dnd-kit або react-beautiful-dnd) разом з кастомними хуками для ефективної обробки подій перетягування та скидання елементів.

Контейнеризація та процес розгортання (Deployment)

1

Docker та Docker Compose

Усі сервіси, включаючи Backend, Frontend, Database та Migrations, контейнеризовано за допомогою Docker. `docker-compose` використовується для зручної локальної розробки та оркестрації.

2

Керування конфігурацією

Критичні дані, такі як паролі до БД та API ключі, керуються за допомогою змінних середовища (.env файли), що забезпечує безпеку та гнучкість конфігурації.

3

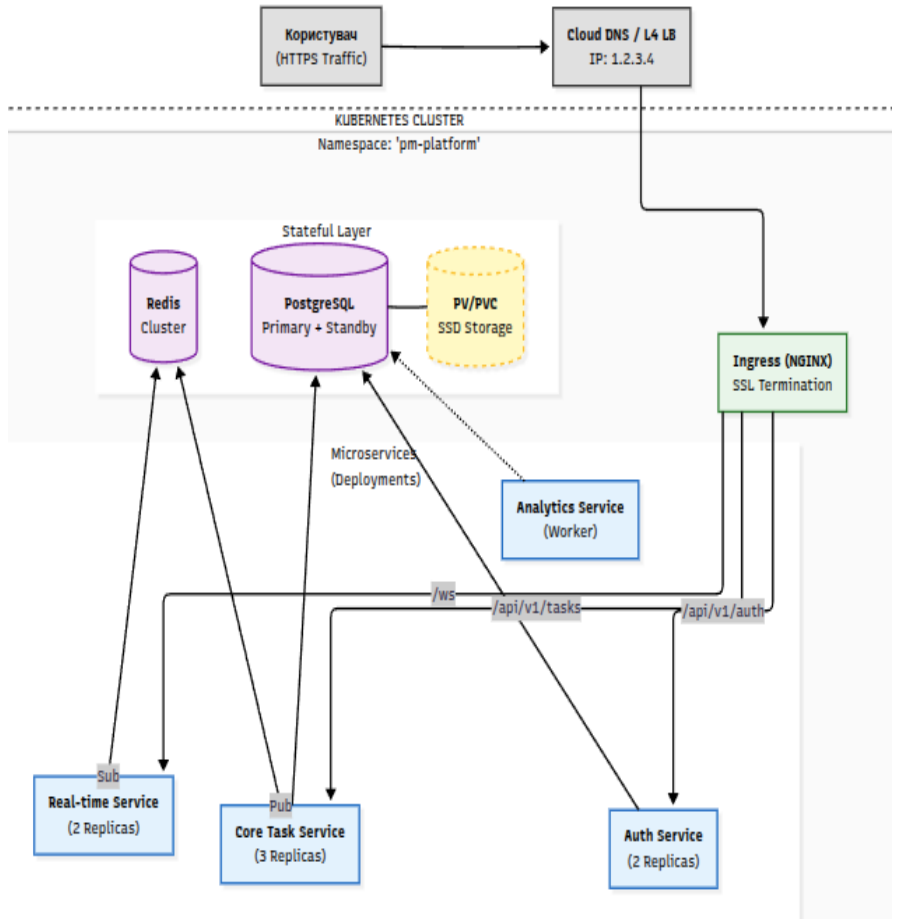
CI/CD Pipeline (GitHub Actions)

Автоматизований конвеєр розгортання: Stage 1 включає Linting та Unit Tests для перевірки коду. Stage 2 відповідає за збірку Docker-образів, що гарантує якість та послідовність.

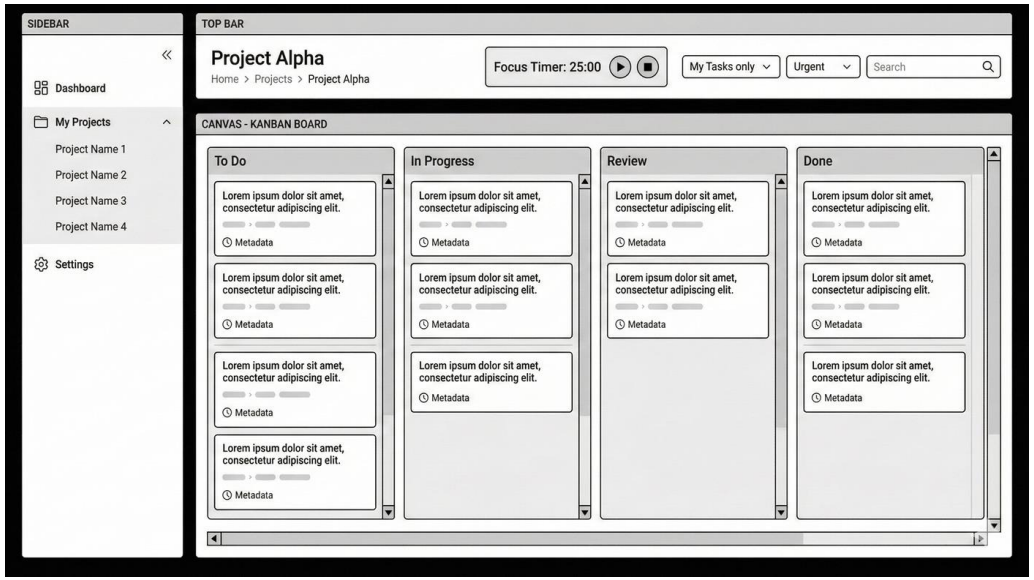
4

Готовність до хмарного розгортання

Система повністю готова до розгортання в будь-якому хмарному середовищі, забезпечуючи масштабованість та високу доступність.

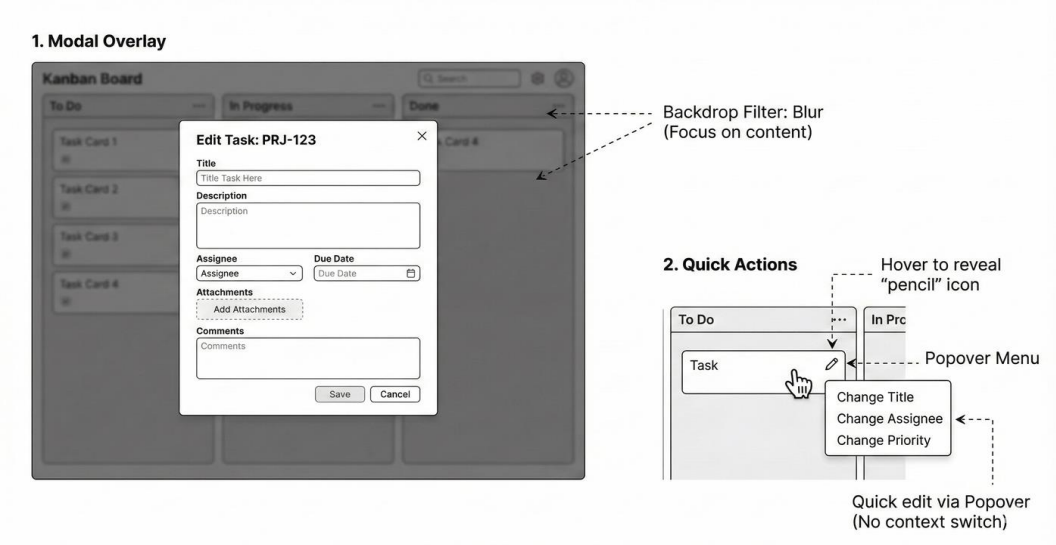


Тестування системи та результати розробки



Скріншот: ***Дошка задач***

Інтуїтивно зрозумілий інтерфейс для керування задачами з функціоналом перетягування (Drag-and-Drop).



Скріншот: ***Вікно редагування задачі***

Візуалізація картки редагування задачі

Види тестування

Unit-тести (Go): Покриття бізнес-логіки понад 70%, що гарантує стабільність основних функцій.

Integration Tests: Ретельна перевірка API ендпоінтів для забезпечення коректної взаємодії між сервісами.

Результати навантаження

Час відгуку API (Latency) становить менше 50 мс для більшості основних операцій, що свідчить про високу продуктивність системи.

Висновки та результати

Основні результати роботи

Розроблено повнофункціональну **SaaS-платформу (MVP)** для управління проєктами з інтегрованою системою оцінки Well-being. Реалізовано архітектуру безпеки даних **Defense in Depth (RLS + JWT)**, оптимізований алгоритм роботи зі списками (**Fractional Indexing**) та реактивний інтерфейс з **Optimistic UI**.

Розробка прототипів та тестування

Крім розробки прототипів дизайну, та функціональних компонентів, було здійснено аналіз споживання ресурсів. Мікросервіси на Go продемонстрували надзвичайно високу ефективність використання пам’яті.

Відгук рецензента

Робота рекомендована до захисту з оцінкою «**Добре**». Відзначено актуальність обраного стеку технологій, глибину опрацювання питань безпеки даних (Multi-tenancy) та практичну цінність модуля аналітики навантаження.

Рекомендації щодо впровадження

Рекомендовано для використання у малих IT-командах, стартапах та студіях розробки як основний інструмент операційного менеджменту, підвищуючи ефективність та добробут співробітників.

Дякую за увагу!