ПОРІВНЯЛЬНИЙ АНАЛІЗ ТЕХНОЛОПЇ HOTWIRE ТА ТРАДИЦІЙНОГО REST API + REACT SPA ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

Хоменко Максим Олександрович, ІПЗ-4

ст.в. Захоженко Павло Олегович

МЕТА І ЗАВДАННЯ

Мета

Порівняти на практиці два підходи до побудови веб-застосунків:

React SPA + REST API vs Hotwire + Rails 7

Чому ці технології?

- React - найпопулярніший фронтенд-бібліотека, типовий вибір для SPA
- Hotwire - новий стек від Basecamp для побудови динаміки з меншою кількістю JavaScript
  - Набирає популярність завдяки інтеграції в Rails 7
  - Зростає інтерес до SSR і спрощеної архітектури у бекенд-командах

ТЕОРЕТИЧНИЙ КОНТЕКСТ

Ціль: Пояснити основи архітектур (SSR, SPA, гібриди), щоб закласти фундамент для розуміння дослідження.

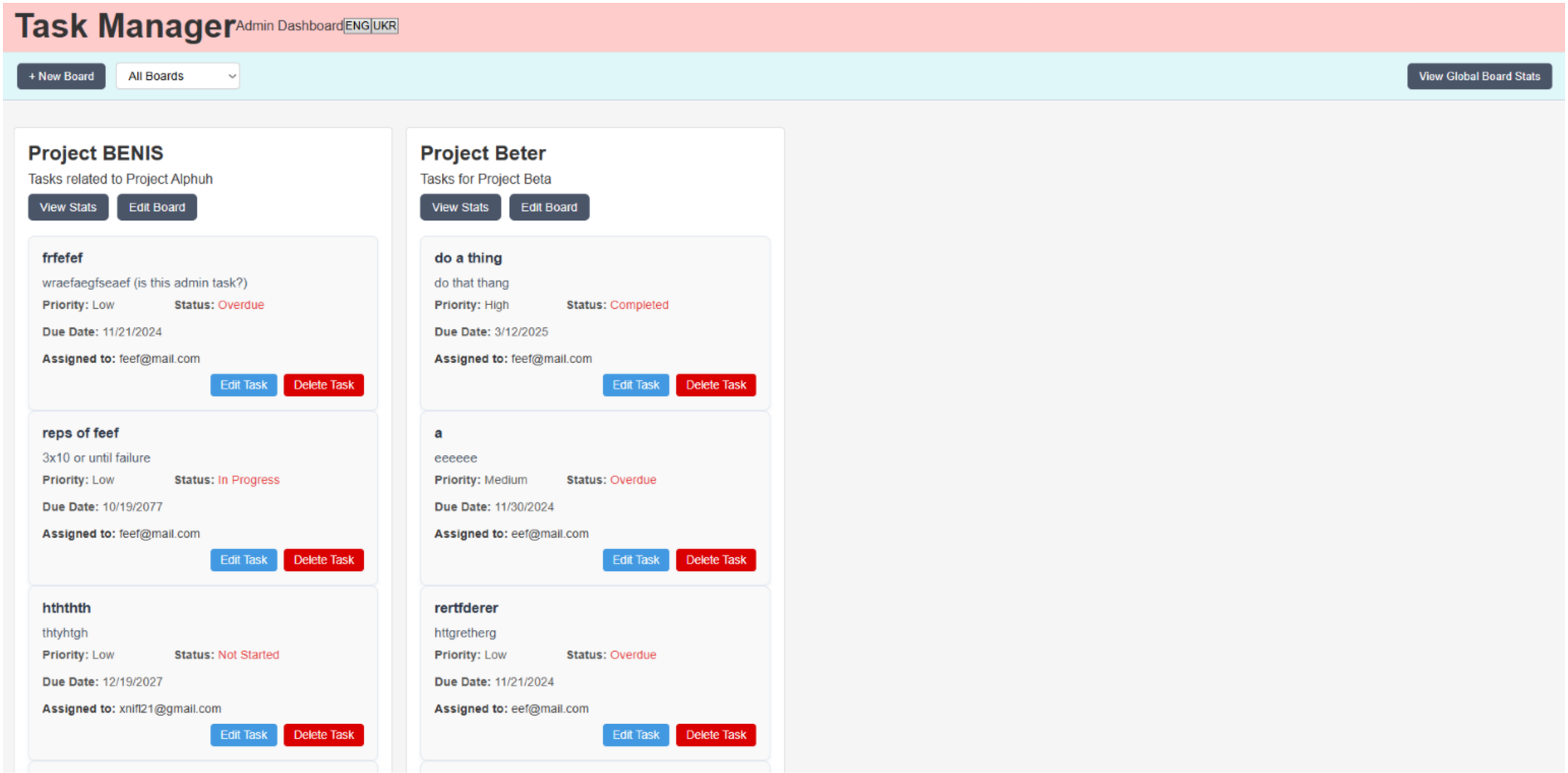
- **SSR (Server-Side Rendering):**
  - Рендер HTML на сервері, кожен запит повертає фрагмент HTML.
  - Краще для SEO, швидший перший рендер.
  - Менш гнучкий для складної інтерактивності.
- **SPA (Single Page Application):**
  - Клієнтський рендеринг через JavaScript (CSR).
  - Плавна навігація, гнучкість UI, але проблеми з SEO і FCP.
- **Гібридні рішення:**
  - Поєднують SSR + SPA, використовуються у Next.js, Nuxt, тощо.
  - Баланс між швидкістю, SEO та UX.
- **Hotwire як альтернатива SPA:**
  - Не потребує важкого JS-фронтенду.
  - Пропонує сучасний підхід до технології SSR

РЕАЛІЗОВАНІ ФУНКЦІЇ

У обох додатках **реалізовано**:

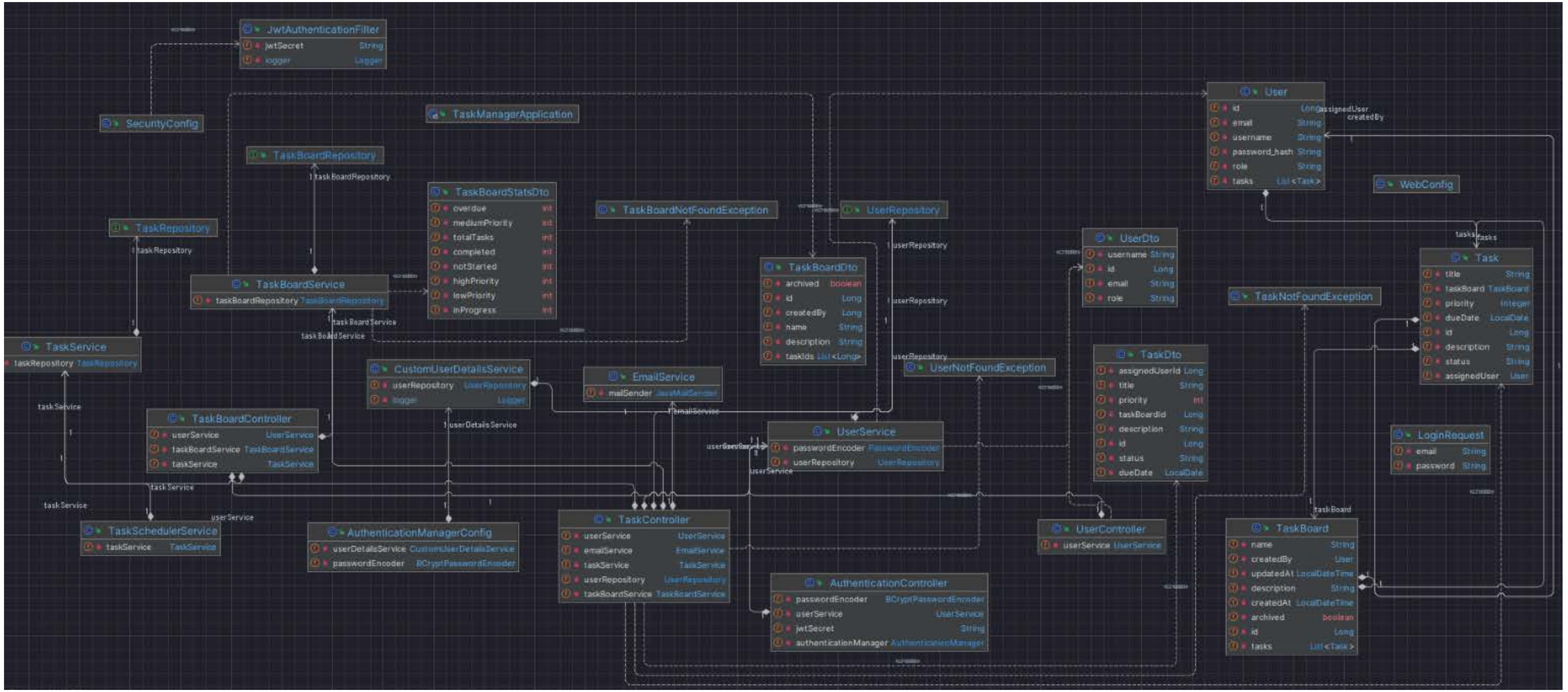
- Автентифікацію користувачів
- Розподіл відповідальностей (користувачі, адміністратори)
- Створення / редагування / видалення Kanban-дошок & завдань
- Пріоритети, статуси, дедлайни задач
- Інтернаціоналізацію
- Графіки для перегляду статистики (глобальної / конкретної дошки)
- Фонову задачу (автоматичне оновлення стану простроченого завдання)

РЕАЛІЗАЦІЯ ФРОНТЕНДУ (REACT SPA)



UI додатку React SPA

РЕАЛІЗАЦІЯ БЕКЕНДУ (SPRING BOOT API)

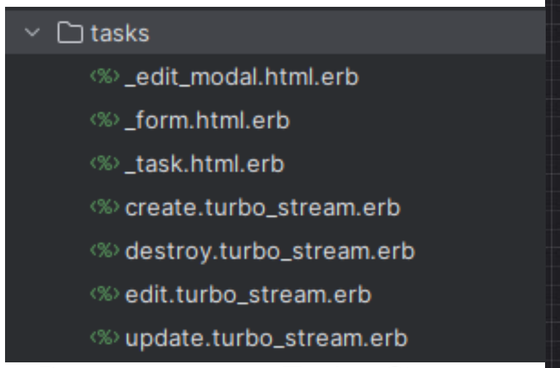


Backend-діаграма Spring Boot

РЕАЛІЗАЦІЯ БЕКЕНДУ (RAILS + HOTWIRE)

Головні компоненти:

- Використано Turbo Streams для live-оновлень (наприклад, після створення/редагування завдання).
- Фонові задачі - через Solid Queue, запуск по due_date для перевірки статусів.
- Devise забезпечує реєстрацію, скидання паролю, логін/лог-аут.
- Увесь рендеринг - на сервері, HTML формується в Rails-шаблонах.



Використання Turbo Streams

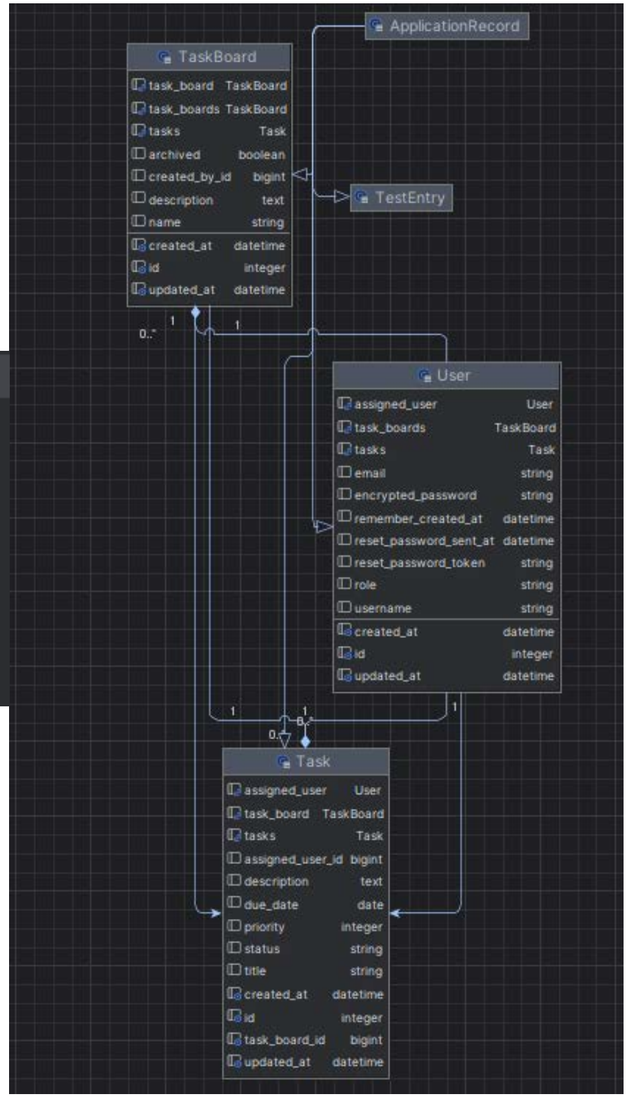
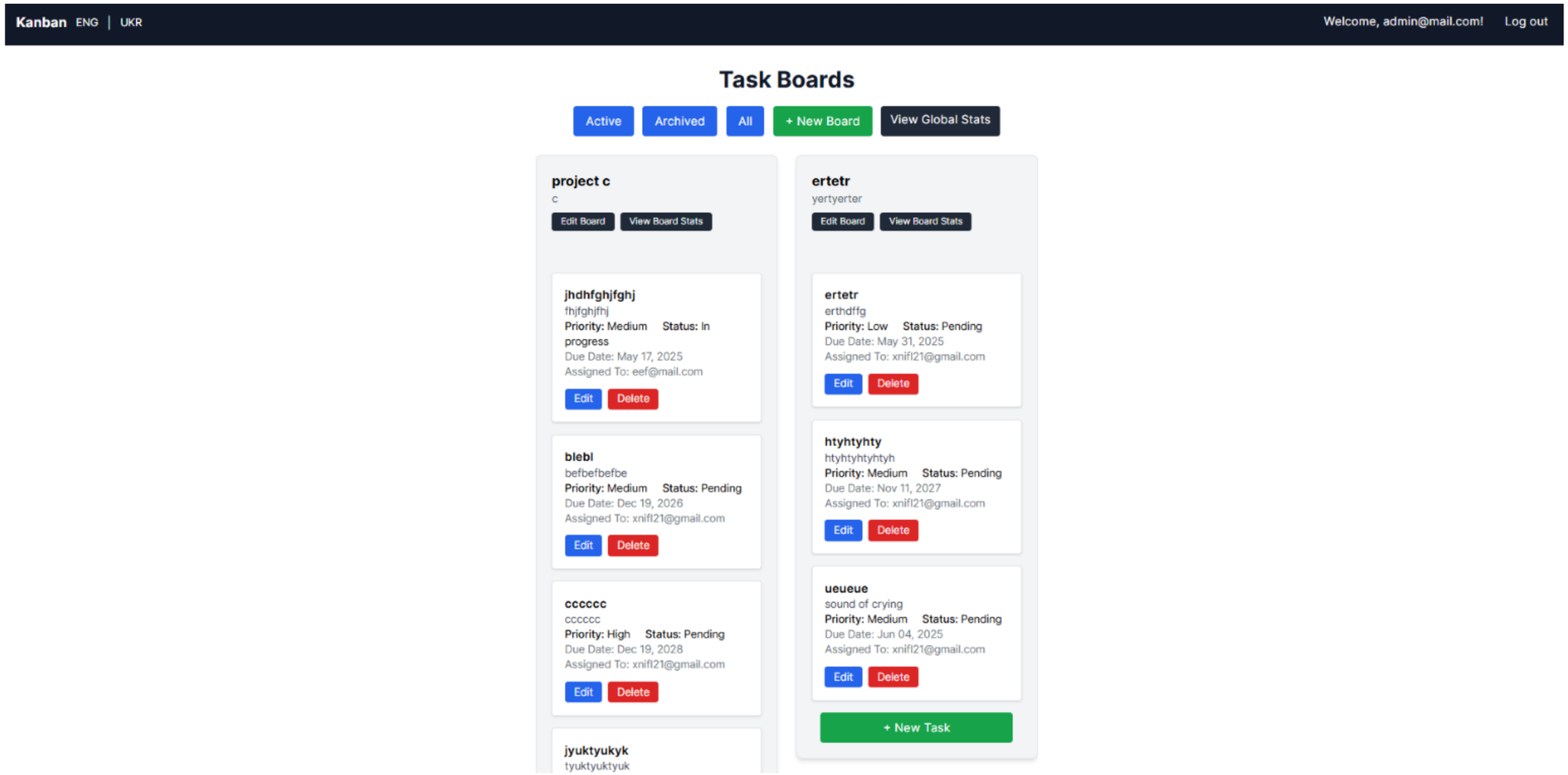


Схема моделей ActiveRecord

РЕАЛІЗАЦІЯ ФРОНТЕНДУ (HOTWIRE + RAILS 7)



UI додатку Hotwire + Rails 7

АРХІТЕКТУРНІ ВІДМІННОСТІ

Порівняльна таблиця - React + Spring Boot vs Hotwire + Rails 7

Аспект	React + Spring Boot	Hotwire + Rails 7
Рендеринг	CSR	SSR
Стек	JS-heavy	Ruby-first
Динаміка	useState, useEffect	Turbo Streams
Навігація	React Router	Turbo Drive
Email/фон	TaskScheduler	SolidQueue

МЕТРИКИ ПРОДУКТИВНОСТІ ТА API-ЛАТЕНТНІСТЬ (LIGHTHOUSE + DEVTOOLS)

Таблиця 4.1. Порівняння метрик React SPA / Hotwire.

Метрика	React SPA (Spring Boot)	Hotwire (Rails 7)
First Contentful Paint	0.20 с (мін: 0.2, макс: 0.2)	0.66 с (мін: 0.5, макс: 0.9)
Largest Contentful Paint	1.00 с (мін: 1.0, макс: 1.0)	0.66 с (мін: 0.5, макс: 0.9)
Speed Index	0.40 с (мін: 0.4, макс: 0.4)	0.69 с (мін: 0.5, макс: 1.0)

Таблиця 4.2. Порівняльна таблиця API-латентності Spring Boot / Rails 7.

Метрика	React + Spring Boot	Hotwire + Rails 7
DNS Lookup (середнє)	0.000023 с	0.000025 с
Connect Time (середнє)	0.000157 с	0.000214 с
Start Transfer (середнє)	0.005757 с	0.008019 с
Total Time (середнє)	0.005944 с	0.008075 с

СИМУЛЯЦІЯ НАВАНТАЖЕННЯ (WRK)

Таблиця 4.3. Результати симуляції навантаження Hotwire (Rails 7).

Threads	Connections	Req/sec	Avg Latency	Max Latency	Transfer/sec	Timeouts
2	10	541.55	19.7ms	50.4ms	704.44 KB	10
4	10	530.84	16.1ms	65.3ms	690.51 KB	8
8	10	553.99	15.4ms	48.3ms	720.63 KB	8
2	50	436.99	230.9ms	1.21s	569.23 KB	0
4	50	470.09	215.9ms	917.8ms	612.34 KB	48
8	50	472.23	215.5ms	928.6ms	615.13 KB	48
2	100	468.51	466.9ms	2.00s	610.29 KB	256
4	100	471.56	479.7ms	2.00s	614.26 KB	216
8	100	468.39	493.5ms	2.00s	610.13 KB	173

Ендпоінт - /task_boards.json

Таблиця 4.4. Результати симуляції навантаження React + Spring Boot.

Threads	Connections	Req/sec	Avg Latency	Max Latency	Transfer/sec	Timeouts
2	10	472.91	22.0ms	72ms	263.79 KB	10
4	10	470.24	17.8ms	95ms	262.34 KB	8
8	10	472.11	17.6ms	52ms	263.36 KB	8
2	50	471.92	110ms	359ms	263.26 KB	50
4	50	452.72	105ms	240ms	252.50 KB	-
8	50	474.83	105ms	172ms	264.85 KB	48
2	100	470.97	220ms	403ms	262.68 KB	100
4	100	471.38	219ms	418ms	262.93 KB	100
8	100	474.28	209ms	433ms	264.53 KB	96

Ендпоінт - /api/task_boards
(JWT додавався до запиту)

UX / DX

- React:
  - + швидка навігація
  - + компоненти
  - - складна інтеграція
- Hotwire:
  - + швидкий старт
  - - важче дебажити Turbo Streams
- DX: Rails швидше в циклі "редагування → перевірка", але React має більше інструментів

ОБМЕЖЕННЯ/НЕДОЛІКИ ДОСЛІДЖЕННЯ

Технічні

- Відмова від Stimulusy Hotwire зменшила складність клієнтської логіки.
- Замість Spring Boot слід було використати Rails - для кращого архітектурного паритету

Методологічні

- Відсутність тестування при деплої (Heroku, Vercel).
- Вузький фокус: лише продуктивність + UX/DX, без доступності або безпеки.

ВИСНОВКИ

React:

- Потужний стек для складних SPA з розширюваною архітектурою.
- Підходить для команд, які готові до більш комплексної структури коду.

Hotwire:

- Простота, мінімум залежностей, швидкий старт.
- Ідеальний для MVP, внутрішніх інструментів.

Рекомендація:

- Обирати архітектуру залежно від розміру команди, складності інтерфейсу та часу на розробку.
- **Hotwire не конкуренція для React, а інша парадигма.**

ДЯКУЮ ЗА УВАГУ!

Готовий відповісти на питання.