Інструкції щодо запуску

Frontend (React + Vite):

Встановіть залежності за допомогою npm install.
Запустіть сервер розробки за допомогою npm run dev.

Backend (Node):

Встановіть залежності за допомогою npm install.
Запустіть сервер розробки за допомогою npm run dev.
Або запустіть сервер за допомогою Docker контейнера за вказаними інструкціями.
Backend (Node) з Docker контейнером:

Встановіть Docker.
Виконайте команди docker pull ka4alkin/movies і docker run --name movies -p 8000:8050 -e APP_PORT=8050 ka4alkin/movies для запуску сервера у Docker контейнері.

Insomnia JSON файл:
В папці сервера (server/test) знаходиться файл insomnia.json з підготовленими API-ендпоінтами, який можна імпортувати у Insomnia для зручного тестування.

Опис тестового завдання

Frontend (React + Vite)

Frontend реалізований з використанням React та Vite, що забезпечує досить швидку розробку, покращену продуктивність, та збирання проекту.

- Використовується підхід на основі компонентів, де кожен компонент відповідає за певну частину інтерфейсу.
- Реалізовано роутінг за допомогою react-router-v6.
- Redux для керування станом додатку та зв'язку між компонентами.
- Для зручного взаємодії з сервером використовується Axios, який дозволяє виконувати HTTP-запити до серверу.
- Застосовані стилізаційні засоби для забезпечення гарного вигляду інтерфейсу за допомогою Tailwind CSS.

Backend (Node)

Backend реалізований з використанням Node.js

• Модульність досягнута за допомогою розбиття коду на логічні компоненти (контролери, моделі, маршрути) згідно шаблону MVC.
• Використовується Express.js для створення сервера та обробки маршрутів.
• Використовуються Sequelize ORM для спрощення роботи з базою даних та забезпечення атомарних операцій.

Атомарність операцій та використання транзакцій.

Для забезпечення атомарності операцій під час імпорту фільмів та акторів зі списку використовуються транзакції від Sequelize.
• Під час імпорту фільмів та акторів зі списку виконується перевірка наявності фільмів та акторів у базі даних.
• Транзакція починається перед виконанням операцій вставки даних до бази даних.
• Якщо будь-яка з операцій вставки невдала, транзакція скасовується (ROLLBACK), забезпечуючи атомарність операцій.
• Якщо всі операції вставки успішні, транзакція фіксується (COMMIT).

Використання проміжної таблиці для зв'язку фільмів та акторів

Для зв'язку фільмів та акторів використовується проміжна таблиця, яка дозволяє зберігати відношення "багато-до-багатьох" між ними.

• В проміжній таблиці зберігаються зовнішні ключі фільмів та акторів.
• Це дозволяє виконувати в майбутньому складні запити, щоб отримати інформацію про фільми та акторів, пов'язаних між собою.

Покращення
• Можливість спростити код та виділити загальні функції для зменшення дублювання.
• Оптимізація фронтенду: видалити лишні запити до серверу, наприклад коли можна працювати зі Redux store.
• Використання Promise.all та інші покращення для оптимізації асинхронних запитів.
• Додати Swagger для документації.

Висновок
Ця архітектура розроблена з метою забезпечити модульність, розширюваність та зручність розгортання. Вона розбиває додаток на логічні компоненти, дозволяє взаємодіяти з даними та здійснювати відображення через веб-інтерфейс. Також вона надає інструкції щодо запуску на різних середовищах, включаючи Docker контейнеризацію для забезпечення стандартизованого розгортання.