**Documentation**

**Project Overview**

This project is a simple full-stack application focused on the front-end. It implements a drag-and-drop grid of cards using React and mocks a back-end API using MSW (Mock Service Worker). The app displays, reorders, and adds items based on user interactions, persisting data locally for development purposes.

**How to Run the Application**

Prerequisites:

* Install Docker.
* Basic familiarity with npm and docker-compose.

Steps to Run:

* Build and start the application using Docker: docker-compose up --build
* Open your browser and navigate to: http://localhost:5173
* To stop the application: docker-compose down

**Features Implemented**

Frontend:

* Display a grid of draggable cards with thumbnails.
* Reorder cards via drag-and-drop functionality.
* Show an image overlay when a card is clicked.
* Automatically save the updated grid every 5 seconds, avoiding unnecessary saves if no changes are made.
* Show a spinner during saving and display time since the last save.

Mock API;

* Intercepts and simulates API calls using MSW.
* Provides endpoints for:

POST /api/data: Update the card ordering.

* Persists data in memory for session-based consistency.

Deployment;

* Uses Docker for easy setup and containerized development.
* Serves the app using Vite's development server.

How the Mock API Works;

* Library Used:

Mock Service Worker (MSW).

* Endpoints:

POST /api/data: Accepts JSON data to simulate reordering of cards.

* Startup:

The mock API starts automatically during development using the MSW worker.

It intercepts requests in the browser to simulate server behavior.

**Architectural Design**

Frontend Architecture

React (Functional Components):

* All components are functional and use React Hooks (e.g., useState, useEffect) for modern, clean, and reusable logic.
* Drag-and-drop functionality is implemented using the react-beautiful-dnd library for smooth reordering.

State Management:

* State for cards and their positions is managed locally within React.
* Data persistence is achieved via browser local storage, simulating a backend database.

API Interaction:

A simple custom hook in the frontend (hooks/useFetch.js) handles API interactions, decoupling API logic from components.

Mock API Design

The API is designed to simulate a simple CRUD backend for cards:

* Simplicity: Focuses only on the required endpoint (POST) for the assignment.
* Scalability: The architecture allows easy addition of more endpoints (e.g., PUT for updates, DELETE for removals) if needed in the future.

Why This Design?

Frontend-First Focus:

Since this is a frontend assignment, the architecture emphasizes simplicity and quick setup for development.

Future-Proof:

* Modular components and mock API allow easy scaling.
* Transitioning to a real backend would require minimal effort.

Performance and Usability:

* The app avoids unnecessary saves to the API, optimizing performance.
* Visual feedback (spinners) improves the user experience.

Lessons and Challenges

Drag-and-Drop Complexity:

* The implementation required careful handling of card reordering to avoid breaking the state.
* Used react-beautiful-dnd for its flexibility and ease of integration.

Mock API Integration:

Ensured the mock API (MSW) is only active for MSW endpoints, avoiding interference with other endpoints.

Dockerization:

* Simplified local development setup with a single Docker Compose configuration.
* Ensured the app runs consistently across different environments.

Future Improvements

Real Backend Integration:

* Replace MSW with an actual REST API (e.g., built with FastAPI or Django).
* Migrate mock data to a database like PostgreSQL for persistence.

Advanced Features:

* Add functionality to update or delete cards directly from the frontend.
* Improve drag-and-drop interactions for accessibility and responsiveness.