CSCC01

Group 200-OK

Software Design Diagram

Elham Badri
Amey Damle
Magen Chen
Monte Ho
Kendrick Joo
Zusheng Lu
Kevin Tian

SDD

System Interaction with the Environment

Operating System

The Playbook app can be developed to run on any operating system that supports Node.js and MongoDB, such as Windows, macOS, or Linux.

Programming Language

The backend of Playbook is developed using the MERN stack, which consists of MongoDB (NoSQL database), Express.js (web application framework), React.js (frontend library), and Node.js (JavaScript runtime). The frontend is primarily built with React and Redux.

Database

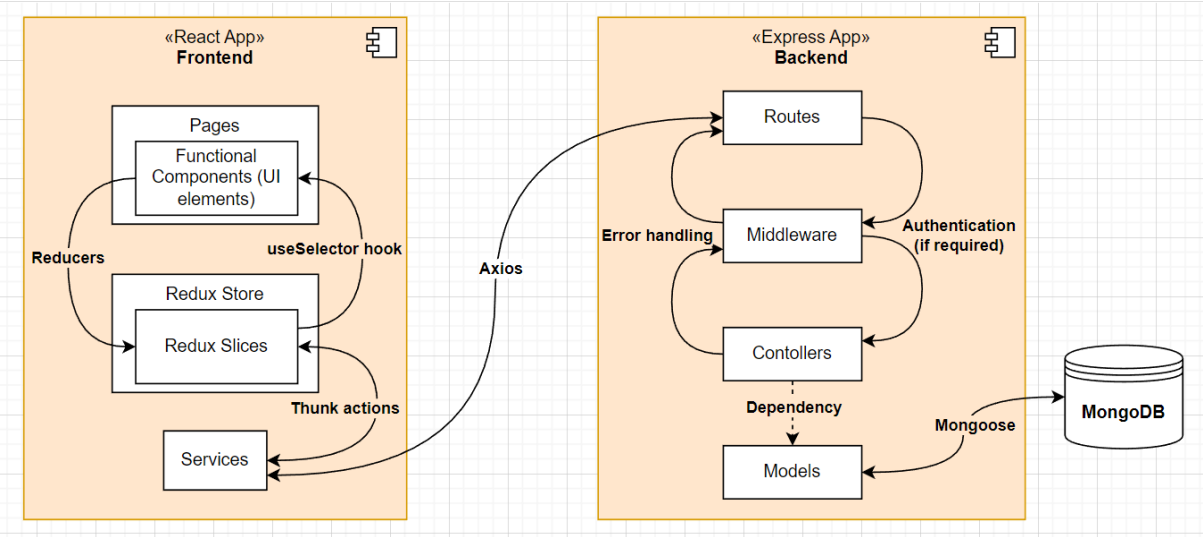
Playbook utilizes MongoDB as its database system to store user, post, LFGPost, and profile data.

Network Configuration

The app requires an internet connection to interact with the Playbook server and for users to access and communicate with each other.

System Architecture

The Playbook app follows a client-server architecture with a layered structure. The components and their relationships can be described as follows:



Frontend

Functional Components

These components represent the UI elements of the app, such as buttons, forms, and navigation bars. They are reusable and independent.

Pages

Pages are composed of functional components and represent different screens or views of the app, like the home page, user profile page, or post creation page.

Redux Store

The app's global state is managed using Redux, which includes reducers, actions, and selectors. Redux provides a predictable state container for managing data and

application state.

Redux Slices

Slices are Redux reducers combined with actions and selectors, focusing on a specific domain or functionality, such as user authentication, post management, or profile editing.

Services

Services handle API calls to the backend server using libraries like Axios. They encapsulate logic related to data fetching and sending.

Backend

Controllers

Controllers handle the request-response flow, processing incoming requests from the frontend, and sending responses. They interact with the models and services to perform CRUD operations.

Routes

Routes define the API endpoints and their associated controller functions. They establish the communication between the frontend and backend.

Models

Models represent the data structure and schema of the app entities, such as User, Post, LFGPost, and Profile. They interact with the database to perform operations like data retrieval, insertion, updating, and deletion.

Middleware

Middleware functions handle error handling and authentication processes. They intercept incoming requests and perform necessary checks before passing control to the appropriate controller.

The components mentioned above interact with each other following the flow: Frontend Components -> Services -> Controllers -> Models -> Database (MongoDB).

Error and Exception Handling

Invalid User Input

The frontend can validate user input before sending requests to the backend. Additionally, the backend can perform input validation and respond with appropriate error messages if validation fails.

Network or External System Failure

The system can handle network or external system failures by implementing proper error handling mechanisms, such as retrying failed requests, logging errors for debugging, and providing informative error messages to the user interface.

The software responds to errors and exceptions by sending appropriate HTTP status codes (e.g., 400 for bad requests, 500 for server errors) and returning error messages or JSON objects with error details when necessary.