CineNow - Design Document

TABLE OF CONTENTS

INTRODUCTION AND SYSTEM OVERVIEW

Introduction

CineNow is a real-time movie ticket booking application built with the MERN stack (MongoDB, Express, React, Node.js). It provides users with an efficient and seamless way to browse movies, select showtimes, and book tickets in real-time. The platform offers a user-friendly interface, real-time seat availability updates, and secure payment options. CineNow aims to revolutionize the movie ticket booking experience by minimizing delays and enhancing user convenience.

System Overview

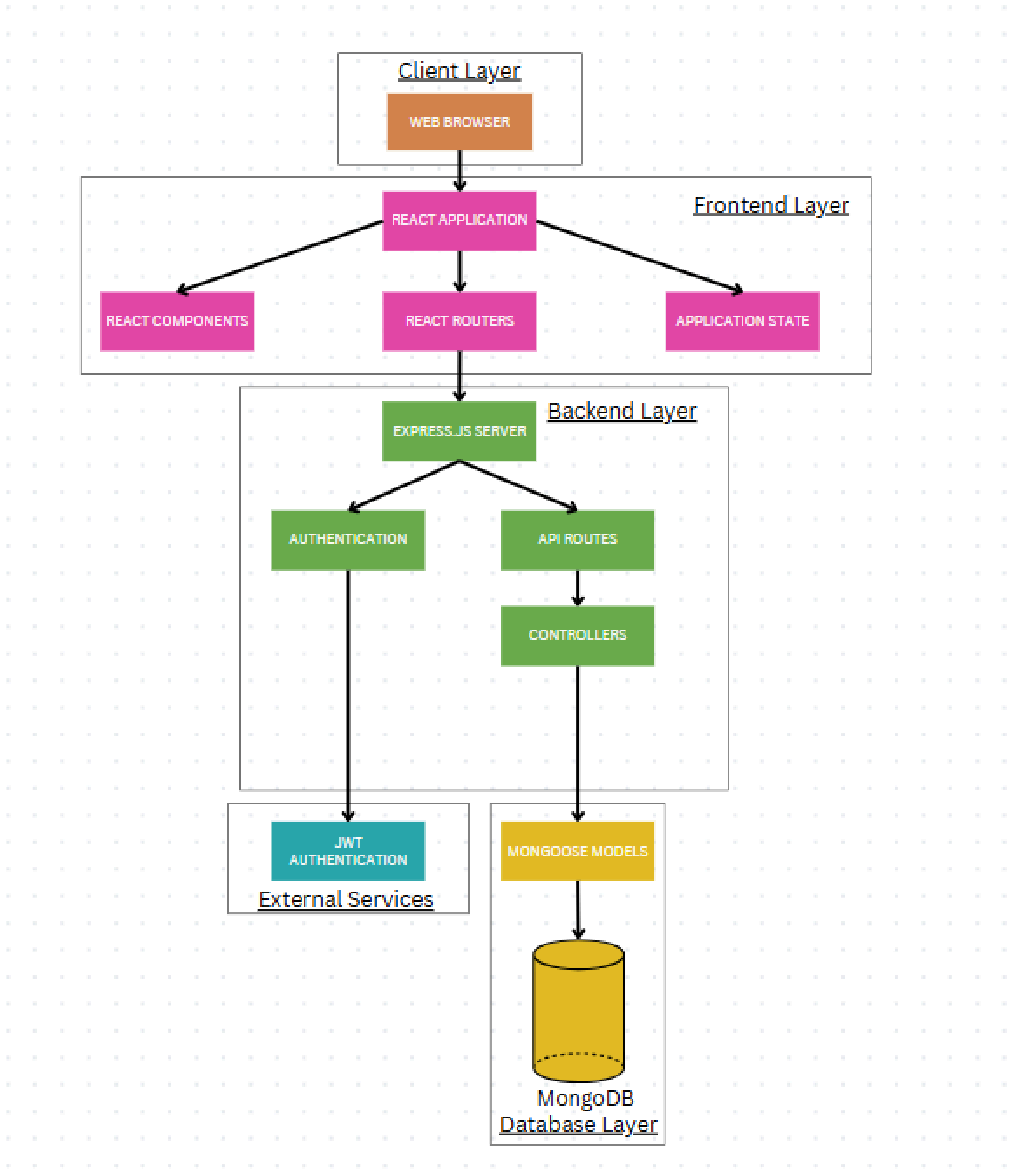
CineNow follows a client-server architecture:

- **Frontend (React.js):** Interactive and responsive interface for browsing movies, selecting showtimes, and booking tickets. Communicates with the backend via RESTful APIs.
- **Backend (Node.js with Express):** Handles API requests, business logic, real-time updates, and authentication using JWT. Interfaces with the MongoDB database for storing user and booking data.
- **Database (MongoDB):** Stores user profiles, movie details, showtimes, seat availability, and booking records.

ARCHITECTURE

Components Interaction and High-level Architecture:

- **Frontend**: React.js (Client-side)
- **Backend**: Node.js with Express (Server-side)
- **Database**: MongoDB (For persistent storage)
- **Authentication**: JWT (JSON Web Tokens for secure login)



DETAILED DESIGN

Frontend Design(React.js)

- **Components:**
  1. **Authentication**:
     - **Login**: Form for user login, retrieves JWT token upon success.
     - **Register**: Form for creating a new account.

  2. **Dashboard**:
     - Displays movie listings, user bookings, and account details.
  3. **Movie Details Page**
     - Showcases movie information, showtimes, and seat availability.
  4. **Booking System:**
     - Interactive seat selection and payment processing
     - Easy to search movies, and select seats.

- **State Management**:
  - **Redux** for managing global state, including user authentication, movie data, and seat availability.

- **UI/UX:**
  - **Styled-components** and responsive design for a seamless user experience across devices.

Backend Design(Node.js with Express)

- **API Endpoints:**

  1. **User:**
     - `POST /api/user/login`: Authenticate users
     - `POST /api/user/signup`: Register new users.
     - `GET /api/user`: Get all users.
     - `PUT /api/user/userid`: Update user details.
     - `DELETE /api/user/userid`: Delete Existing user.
  2. **Admin:**
     - `POST /api/admin/signup`: Grants admin access to user.
     - `POST /api/admin/login`: Authenticate admin login.
  3. **Movie:**
     - `POST /api/movie`: Add new movie.
     - `GET /api/movie`: Get all movies.
     - `GET /api/movie/movieid`: Get movie by id.
     - `GET /api/user/bookings/userid`: Get bookings of the user.
  4. **Bookings:**
     - `POST /api/booking`: Creates new booking.
     - `GET /api/booking/bookingid`: Get booking details.
     - `DELETE /api/booking/bookingid`: Delete movie by id.
     - `GET /api/user/booking`: Get all bookings .

- **Database Schema:**
    - **Users**: Store user profile information like username, email, password and all its booking.
    - **Movies**: Stores movie details, including title, desc, actors, release date, poster url, featured, bookings of the movie, and admin that added it.
    - **Bookings**: Store details of each booking like movie details, date of booking, seat number and user id.
    - **Admin:** Store admin details like email, password and all the movies added by the admin.

- **Technologies Used:**
    - **Node.js:** JavaScript runtime environment.
    - **Express.js:** Web application framework for building APIs.
    - **MongoDB:** NoSQL database for data storage.
    - **Mongoose:** ODM(Object Data Modeling) library for MongoDB.
    - **JWT:** JSON Web Tokens for authentication.
    - **Bcrypt:** Library for hashing passwords.

- **Project Structure:**
    - **app.js:** Entry point of the server application.
    - **controllers/:** Contains functions, including authentication, processing and etc.
    - **models/:** Mongoose schemas for users, movies, and bookings.
    - **routes/:** API endpoints.

- **MIDDLEWARE:**
    - JWT middleware validates tokens, attaches user data to requests, and handles invalid or expired tokens

DATABASE SCHEMA EXPLANATION

- **Users: (user.model.js)**
  - **name** (String, required): User’s full name, trimmed of whitespace.
  - **username** (String, required, unique): User’s username, converted to lowercase, and trimmed of whitespace.
  - **email** (String, required, unique): User’s email address, converted to lowercase, and trimmed of whitespace.
  - **password** (String, required): User’s password, with a minimum length of 6 characters and trimmed of whitespace.
  - **bookings** (Array of Object IDs, references **Booking**): Stores references to bookings made by the user.

- **Admin: (admin.model.js)**
  - **email** (String, unique): Admin’s email address.
  - **password** (String): Admin’s password, with a minimum length of 6 characters.
  - **addedMovies** (Array of Object IDs, references **Movie**): A list of movies added by the admin.

- **Bookings: (booking.model.js)**
  - **movie** (Object ID, required, references **Movie**): The ID of the booked movie.
  - **date** (Date, required): The date of the booking.
  - **seatNumber** (Number, required): The seat number for the booking.
  - **user** (Object ID, required, references **User**): The ID of the user who made the booking.

DATABASE SCHEMA EXPLANATION

- **Movies: (movie.model.js)**
  - **title** (String, required): The title of the movie.
  - **description** (String, required): A brief description of the movie.
  - **actors** (Array of Strings, required): A list of actors in the movie.
  - **releaseDate** (Date, required): The movie's release date.
  - **posterUrl** (String, required): The URL of the movie poster.
  - featured (String): Indicates whether the movie is featured.
  - **bookings** (Array of Object IDs, references Booking): A list of bookings associated with the movie.
  - **admin** (Object ID, required, references Admin): The admin who added the movie.

MIDDLEWARE

This middleware function verifies a JWT from the accessToken cookie or Authorization header, decodes the user data, attaches it to the req object, and handles errors if the token is invalid or expired

SECURITY CONSIDERATIONS

- Secure JWT-based authentication.
- Encrypted user passwords using bcrypt.
- HTTPS for secure data transmission

API DESIGN (RESTful)

- **/api/user :** Handles user authentication and profile management.
- **/api/movie :** Provides movie and showtime data.
- **/api/booking :** Manages ticket booking and retrieval.
- **/api/admin :** Handles admin authentication and login.

DATA FLOW AND ERROR HANDLING

- **Data Flow**:
    - User logs in and browses movies
    - User logs in and browses movies
    - Proceeds to payment and confirms booking.

- **Error Handling**:
    - Backend: Handles authentication errors, database errors, and invalid requests.
    - Frontend: Displays user-friendly error messages for various scenarios.

DEPLOYMENT

- **Frontend:**
  - Deployed on Vercel.

- **Backend:**
  - Deployed on AWS .

- **Database:**
  - Hosted on MongoDB Atlas.

FUTURE ENHANCEMENTS

- Integrate **real-time seat updates** using WebSocket.
- Add **payment gateways** for seamless transactions.
- Develop a **Progressive Web App (PWA)** for offline access.
- Incorporate **advanced analytics** for user insights**.**

CONCLUSION

CineNow aims to simplify and enhance the movie ticket booking process by leveraging the MERN stack and providing a user-centric design. Future enhancements will ensure it remains at the forefront of modern web applications.