INTRODUCTION

* **Project title:** Flight booking app
* **Team Members:** 1. Sneha V

2. Eswari A

3. Yamini T

4. Poornima KL

PROJECT OVERVIEW

**Purpose:**

The purpose of a flight booking app using the MERN stack (MongoDB, Express.js, React.js, Node.js) is to create a seamless platform for users to search, book, and manage flight reservations. The goals typically include providing a user-friendly interface, efficient data management, real-time updates , and secure transactions. Below are the key aspects of the project:

**Purpose of the Flight Booking App**

* User -Centric Design: To offer an intuitive and engaging user experience for booking flights.
* Streamlined Booking Process: To simplify the process of searching for flights, selecting seats, and completing bookings.
* Real-Time Data Access: To provide users with up-to-date information on flight availability, pricing, and schedules.
* Secure Transactions: To ensure that user data and payment information are handled securely through robust authentication and encryption methods.

**Goals of the Project**

**Feature-Rich Functionality:**

* Implement user registration and authentication using Passport.js or JWT.
* Allow users to search for flights based on various criteria (e.g., destination, date).
* Enable seat selection and ticket cancellation features.
* Integrate payment processing through a secure gateway.

**Backend Development:**

* Create RESTful API endpoints for managing flights, users, and bookings.
* Utilize MongoDB for efficient data storage and retrieval.

**Frontend Development:**

* Build a responsive and dynamic user interface using React.js.
* Ensure smooth interaction with the backend through API calls using Axios or Fetch.

**Testing and Deployment:**

* Conduct thorough testing to ensure functionality and reliability.
* Deploy the application on platforms like Heroku or AWS for public access.

**Continuous Improvement:**

* Gather user feedback to enhance features and user experience over time.
* Stay updated with the latest technologies and best practices in web development.

**Features:**

When developing a flight booking app using the MERN stack, several key features and functionalities should be considered:

**User Authentication:**

Implement secure user registration and login processes to manage user accounts and sessions.

**Flight Search and Filtering:**

Allow users to search for flights based on various criteria such as departure and arrival cities, dates, and airlines.

**Booking Management:**

Enable users to book flights, view their bookings, and manage cancellations or changes to their reservations.

**Payment Integration:**

Incorporate a secure payment gateway for processing transactions, ensuring compliance with PCI standards.

**Dynamic Seat Selection:**

Provide a user-friendly interface for selecting seats, including options for different classes and preferences.

**Profile Management:**

Allow users to view and edit their profiles, including personal information and booking history.

**Admin Panel:**

Create an admin interface for managing flights, users, and bookings, including the ability to add, update, or remove flight listings.

**QR Code Verification:**

Implement QR code generation for tickets to streamline the check-in process and enhance security.

**Responsive Design:**

Ensure the application is mobile-friendly, providing a seamless experience across various devices.

**AI Chatbot Integration:**

Include an AI-powered chatbot to assist users with inquiries and provide support throughout the booking process.

**Real-time Notifications:**

Send users updates regarding flight status, booking confirmations, and reminders through email or push notifications.

**GDS API Integration:**

Integrate with Global Distribution Systems (GDS) for real-time access to flight data and booking capabilities.

**Testing and Debugging:**

Implement thorough testing for both frontend and backend components to ensure reliability and performance.

**Deployment:**

Plan for deployment on cloud platforms, ensuring scalability and availability for users.

* By incorporating these features, a flight booking app built with the MERN stack can provide a comprehensive and user-friendly experience for travelers.

**ARCHITECTURE:**

**Backend Development**

1. **Database Configuration:**

* Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
* Create a database and define the necessary collections for flights, users, bookings, and other relevant data.

1. **Create Express.js Server:**

* Set up an Express.js server to handle HTTP requests and serve API endpoints.
* Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

1. **Define API Routes:**

* Create separate route files for different API functionalities such as flights, users, bookings, and authentication.
* Define the necessary routes for listing flights, handling user registration and login managing bookings, etc.
* Implement route handlers using Express.js to handle requests and interact with the database.

1. **Implement Data Models:**

* Define Mongoose schemas for the different data entities like flights, users, and bookings.
* Create corresponding Mongoose models to interact with the MongoDB database. Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

1. **User Authentication:**

* Create routes and middleware for user registration, login, and logout.
* Set up authentication middleware to protect routes that require user authentication.

1. **Handle new Flights and Bookings:**

* Create routes and controllers to handle new flight listings, including fetching flight data from the database and sending it as a response.
* Implement booking functionality by creating routes and controllers to handle booking requests, including validation and database updates.

1. **Admin Functionality:**

* Implement routes and controllers specific to admin functionalities such as adding flights, managing user bookings, etc.
* Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

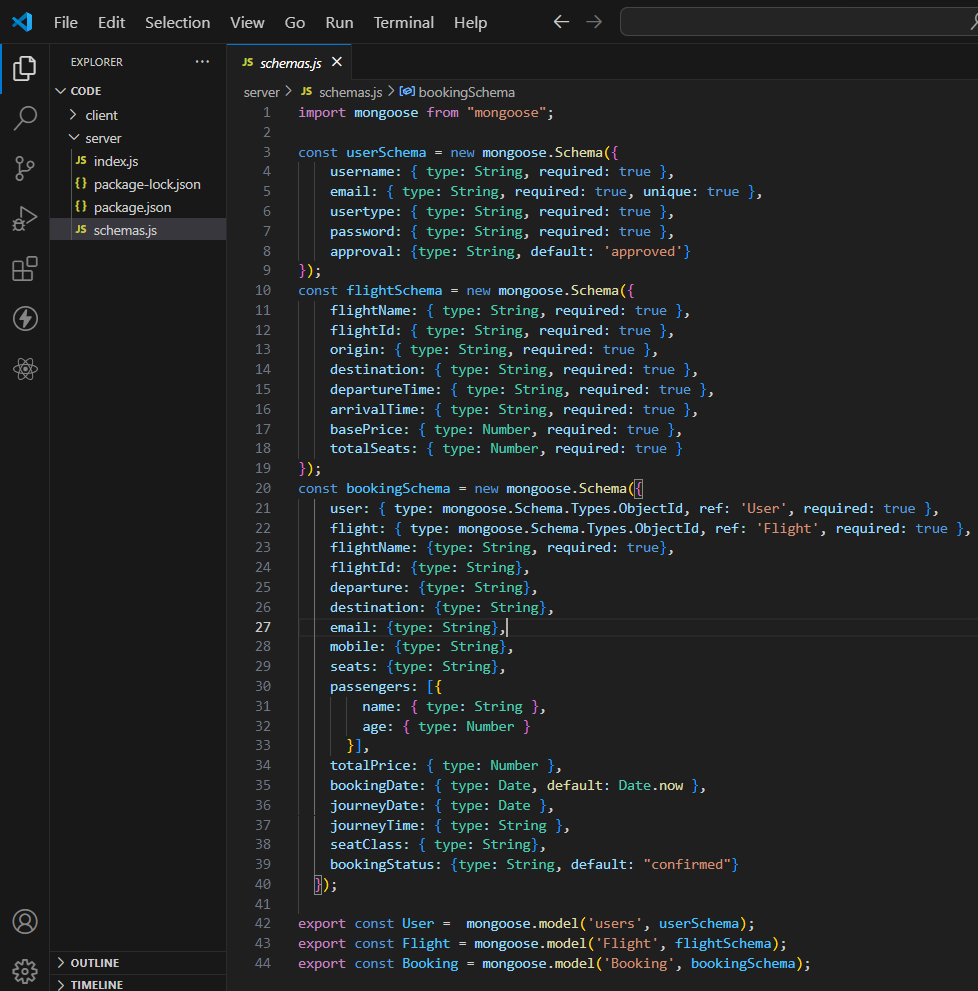
1. **Error Handling:**

* Implement error handling middleware to catch and handle any errors that occur during the API requests.
* Return appropriate error responses with relevant error messages and HTTP status codes.

**Database development**

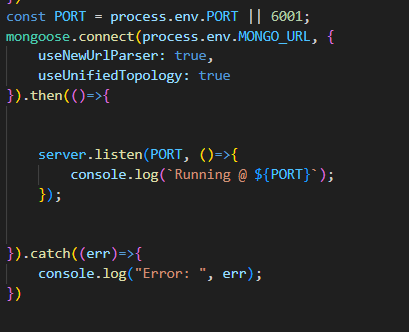
* **Configure schema**

Firstly, configure the Schemas for MongoDB database, to store the data in such a pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.

****

* **Connect database to backend**

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.



### Frontend development

1. **Login/Register**

* Create a Component which contains a form for taking the username and password.
* If the given inputs matches the data of user or admin or flight operator then navigate it to their respective home page

1. **Flight Booking (User):**

* In the frontend, we implemented all the booking code in a modal. Initially, we need to implement flight searching feature with inputs of Departure city, Destination, etc.,
* Flight Searching code: With the given inputs, we need to fetch the available flights. With each flight, we add a button to book the flight, which redirects to the flight-Booking page.

1. **Fetching user bookings:**

* In the bookings page, along with displaying the past bookings, we will also provide an option to cancel that booking.

1. **Add new flight(Admin):**

* Now, in the admin dashboard, we provide functionality to add new flights.
* We create a html form with required inputs for the new flight and then send an httprequest to the server to add it to the database.

1. **Update Flight:**

* Here, in the admin dashboard, we will update the flight details in case if we want to make any edits to it
* Along with this, implement additional features to view all flights, bookings, and users in the admin dashboard.

**SETUP INSTRUCTIONS**

**PRE-REQUISITES**

To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

**Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

* Download: <https://nodejs.org/en/download/>
* Installation instructions: <https://nodejs.org/en/download/package-manager/>

**MongoDB**: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.

* Download: <https://www.mongodb.com/try/download/community>
* Installation instructions: <https://docs.mongodb.com/manual/installation/>

**Express.js**: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing,middleware, and API development.

* Installation: Open your command prompt or terminal and run the following command: npm install express

**React.js**: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

**HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling,and JavaScript for client-side interactivity is essential.

**Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

**Front-end Framework:** Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

**Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

* Git: Download and installation instructions can be found at: <https://gitscm.com/downloads>

**Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

* Visual Studio Code: Download from <https://code.visualstudio.com/download>
* Sublime Text: Download from <https://www.sublimetext.com/download>
* WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

**INSTALLATION**

**Clone the repository:**

* Open your terminal or command prompt.
* Navigate to the directory where you want to store the e-commerce app.
* Execute the following command to clone the repository:

**Git clone:** <https://github.com/harsha-vardhan-reddy-07/Flight-Booking-App-MERN>

**Install Dependencies:**

* Navigate into the cloned repository directory:

**cd Flight-Booking-App-MERN**

* Install the required dependencies by running the following command:
* **npm install**

**Start the Development Server:**

* To start the development server, execute the following command:

**npm run dev or npm run start**

* The e-commerce app will be accessible at [http://localhost:3000](http://localhost:3000/) by default. You can change the port configuration in the .env file if needed.

**FOLDER STRUCTURE**

**CLIENT:**

In a MERN stack flight booking app, the React frontend typically follows a structured folder organization to maintain clarity and scalability. Here’s a general overview of the folder structure:

**Public/:** Contains static files like the index.html, images, and favicon.

**src/ :** Contains the main source code for the React application. This directory is further divided into several subdirectories:

**components/:**

Houses reusable UI components that can be shared across different parts of the application, such as buttons, forms, and modals.

**pages/:**

Each page of the application has its own folder within this directory. For example, you might have HomePage, BookingPage, and ConfirmationPage, each containing components and logic specific to that page.

**services/:**

Contains files responsible for handling API communication and data fetching. This could include functions to interact with the backend for booking flights, retrieving flight data, and managing user authentication.

**utils/:**

If there are utility functions or helper files that are used across multiple components, they can be placed in this directory. This might include formatting dates, validating inputs, or managing local storage.

**App.js:**

Serves as the main entry point for the React application, setting up routing and global component initialization.

**index.js:**

Responsible for rendering the React application and mounting it into the DOM.

**assets/** (optional):

If your application requires additional static assets like images or CSS files, consider creating an assets directory within the client directory.

* This structured approach helps in maintaining a clean and organized codebase, making it easier to manage and scale the application as it grows.

**SERVER**

In a MERN stack flight booking app, the Node.js backend typically follows a structured folder organization to maintain clarity, scalability, and separation of concerns. Here's a general overview of the folder structure you might use for the backend:

**Explanation of Each Directory**

**/config:**

* This directory contains configuration files for the application.
* db.js manages the database connection (e.g., MongoDB), and keys.js holds sensitive information like API keys and environment variables.

**/controllers:**

* Controllers contain the business logic for handling requests and responses.
* Each controller file corresponds to a specific resource (e.g., flights, users, bookings) and contains functions for CRUD operations.

/models:

* This directory contains Mongoose models that define the schema for the data stored in MongoDB.
* Each model corresponds to a collection in the database (e.g., Flight, User , Booking).

**/routes:**

* Routes define the API endpoints and map them to the appropriate controller functions.
* Each route file corresponds to a specific resource and defines the endpoints for that resource.

**/middleware:**

* Middleware functions are used to handle requests before they reach the route handlers.
* This can include authentication checks, logging, and error handling.

**/utils:**

* Utility functions that can be reused across the application are stored here.
* This may include functions for sending emails, validating input data, or formatting responses.

**/tests:**

* This directory contains test files for the application.
* Each file can correspond to a specific feature or functionality and is used to ensure the application behaves as expected.

**server.js:**

* This is the main entry point for the Node.js application.
* It initializes the server, connects to the database, and sets up middleware and routes.

**package.json:**

* This file lists the project dependencies and scripts to run the application (e.g., start, test).

**Conclusion**

* This structured approach helps in maintaining a clean and organized codebase, making it easier to manage and scale the application as it grows. Each folder serves a specific purpose, promoting separation of concerns and enhancing code maintainability.

**RUNNING THR APPLICATION**

**FRONTEND:**

To run the frontend of a MERN stack flight booking application, you typically need to follow a few steps to ensure that everything is set up correctly. Here's a general guide to help you get started:

**Prerequisites**

**Node.js and npm:** Make sure you have Node.js and npm (Node Package Manager) installed on your machine. You can download it from Node.js official website.

**MongoDB:** Ensure that you have a MongoDB instance running (either locally or on a cloud service like MongoDB Atlas).

**Backend API:** Ensure that your backend server (Node.js/Express) is running. This usually involves starting your server in a separate terminal.

**Steps to Run the Frontend**

**Navigate to the Client Directory:** Open your terminal (command prompt, PowerShell, etc.) and navigate to the client directory of your application. This is usually done using the cd command. For example

**Install Dependencies:** If you haven't already installed the necessary dependencies for the frontend, run:

This command will read the package.json file in the client directory and install all the required packages.

**Start the Development Server:** After the dependencies are installed, you can start the development server with:

*npm start*

This command typically runs the application in development mode and opens it in your default web browser. The application will usually be accessible at http://localhost:3000 (or another port if specified in your configuration).

**Troubleshooting**

**Port Issues:** If you encounter a port conflict, you may need to specify a different port. You can do this by setting the PORT environment variable:

*PORT=3001 npm start*

**Backend Connection:** Ensure that your frontend is correctly configured to connect to your backend API. This often involves setting the correct API URL in your environment variables or configuration files.

**Build Issues:** If you run into issues while building, check your console for error messages, and ensure all dependencies are compatible with your version of Node.js.

**Conclusion**

After following these steps, your flight booking application's frontend should be up and running. You can now interact with the app, and it should communicate with the backend to handle flight bookings. If you encounter any specific errors, feel free to ask for further assistance!

**BACKEND:**

To run the backend of a MERN stack flight booking application, you'll typically follow a series of steps similar to those for the frontend. Here’s a guide to help you get started with running the backend server:

**Prerequisites**

**Node.js and npm:** Ensure you have Node.js and npm installed on your machine. You can download it from Node.js official website.

**MongoDB:** Make sure you have a MongoDB instance running. This could be a local installation or a cloud-based service like MongoDB Atlas.

**Frontend:** If your application has a frontend, ensure that it is set up as needed, but you can run the backend independently.

**Steps to Run the Backend**

**Navigate to the Server Directory:** Open your terminal and navigate to the server directory of your application. Use the cd command to change directories. For example:

*cd path/to/your/server*

Install Dependencies: If you haven’t already installed the necessary dependencies for the backend, run:

*npm install*

This command will read the package.json file in the server directory and install all required packages.

**Set Environment Variables:** Before starting the server, ensure that any necessary environment variables are set up. This often includes database connection strings and other configuration settings. You can create a .env file in the server directory with variables like:

*MONGODB\_URI=mongodb://localhost:27017/your\_database\_name*

*PORT=5000*

**Start the Server:** After installing dependencies and setting up environment variables, you can start the backend server with:

*npm start*

This command typically runs the application in development mode. The server will usually be accessible at http://localhost:5000 (or another port if specified in your configuration).

**Additional Commands**

**Using Nodemon:** If your project is set up to use Nodemon (a tool that automatically restarts the server when file changes are detected), you can run:

*npm run dev*

Ensure that your package.json has a script for dev that uses Nodemon.

**Troubleshooting**

**Port Issues:** If you encounter a port conflict, make sure no other applications are using the same port. You can change the port in your .env file or directly in the server code.

**Database Connection:** Ensure that your backend is correctly configured to connect to your MongoDB instance. Check the connection string and ensure that MongoDB is running.

**Error Messages:** If you run into issues, check the terminal for error messages that can help identify the problem.

**Conclusion**

By following these steps, your flight booking application's backend should be up and running, ready to handle requests from the frontend. If you encounter specific issues or error messages, feel free to ask for further assistance!

API DOCUMENTATION

To create a flight booking app using the MERN stack, you'll typically need to focus on several key components:

**Backend (Node.js & Express):** Set up RESTful APIs to handle user authentication, flight searches, bookings, and payment processing. Use MongoDB for data storage, including user profiles, flight details, and booking records.

**Frontend (React):** Develop a user-friendly interface that allows users to search for flights, view flight details, and manage their bookings. Utilize React Router for navigation and state management libraries like Redux or Context API for managing application state.

**API Endpoints:**

User Authentication:

* POST /api/auth/register: Register a new user.
* POST /api/auth/login: Authenticate a user and return a token.

**Flight Management:**

* GET /api/flights: Retrieve a list of available flights.
* GET /api/flights/:id: Get details of a specific flight.

**Booking Management:**

* POST /api/bookings: Create a new booking.
* GET /api/bookings/:userId: Retrieve bookings for a specific user.

**Database Schema:**

User Schema:

Fields: username, email, password, bookings

**Flight Schema:**

Fields: flightNumber, departure, arrival, price, availableSeats

**Booking Schema:**

Fields: userId, flightId, seatsBooked, totalPrice

**Payment Integration:** Implement a payment gateway (like Stripe or PayPal) to handle transactions securely.

**Deployment:** Use platforms like Heroku or Vercel for deploying the backend and frontend applications, ensuring that the APIs are accessible over the internet.

**Documentation:** Create comprehensive API documentation using tools like Swagger or Postman to help developers understand how to interact with your APIs.

* By following these guidelines, you can effectively build a flight booking application using the MERN stack, ensuring a smooth user experience and robust backend functionality.

**User interface**

To implement authentication in a flight booking app using the MERN stack, you can utilize libraries like Passport.js for user authentication and JSON Web Tokens (JWT) for secure token-based authentication. This involves setting up user registration and login routes, protecting sensitive routes with middleware to verify JWTs, and managing user sessions effectively. Here’s a structured approach to achieve this:

**1. User Registration and Login**

**Registration Route:** Create a route to handle user registration. This should include:

* Input validation (e.g., email format, password strength).
* Hashing passwords using bcrypt before storing them in the database.

**Login Route:** Create a route for user login that:

* Validates user credentials.
* Generates a JWT upon successful login.
* Sends the token back to the client for future requests.

**2. Using Passport.js for Authentication**

* **Setup Passport:** Integrate Passport.js into your application for handling authentication strategies.
* Use the Local Strategy for username and password authentication.
* Configure Passport to serialize and deserialize user instances.

**3. Token-Based Authentication**

* **JWT Generation:** Upon successful login, generate a JWT that includes user information (e.g., user ID).
* **Token Storage:** Store the JWT in the client (e.g., local storage or cookies) for subsequent requests.

**4. Protecting Routes**

* **Middleware for JWT Verification:** Create middleware to check for the presence of a JWT in incoming requests.
* Decode the token and verify its validity.
* Allow access to protected routes only if the token is valid.

**5. Logout Functionality**

* **Logout Route:** Implement a route to handle user logout.
* Invalidate the JWT on the client side (e.g., remove it from local storage).
* Optionally, implement server-side token blacklisting for added security.

**6. Additional Security Measures**

* **HTTPS:** Ensure your application uses HTTPS to protect data in transit.
* **Environment Variables:** Store sensitive information (e.g., JWT secret) in environment variables.
* **Rate Limiting:** Implement rate limiting on authentication routes to prevent brute-force attacks.
* By following these steps, you can create a secure authentication system for your flight booking application using the MERN stack. This will help ensure that user data is protected and that only authorized users can access certain features of the application.

**USER INTERFACE**

To create a user interface for a flight booking app using the MERN stack, focus on the following key components:

* **React for Frontend:** Utilize React to build a dynamic and responsive UI. Components can include search forms, flight listings, booking details, and user profiles.
* **Node.js for Backend:** Implement Node.js to handle server-side logic, manage API requests, and interact with the database. This will facilitate user authentication, flight data retrieval, and booking management.
* **MongoDB for Database:** Use MongoDB to store user information, flight details, and booking records. This NoSQL database allows for flexible data structures and efficient querying
* **User Authentication:** Integrate user authentication using libraries like Passport.js or JWT (JSON Web Tokens) to ensure secure access to user accounts and booking functionalities.
* **Flight Search Functionality:** Develop a search feature that allows users to filter flights based on criteria such as destination, date, and price. This can be achieved through API calls to fetch flight data.
* **Booking Process:** Create a seamless booking process that includes selecting flights, entering passenger details, and processing payments. Ensure that users receive confirmation notifications.
* **Responsive Design:** Implement responsive design principles using CSS frameworks like Tailwind CSS or Bootstrap to ensure the app is accessible on various devices, including mobile phones and tablets.
* **Error Handling:** Incorporate error handling mechanisms to manage issues such as failed API requests or invalid user inputs, providing users with clear feedback.
* **Testing and Deployment:** Conduct thorough testing of the application to identify and fix bugs. Once ready, deploy the app using platforms like Heroku or Vercel for public access.
* **User Experience Enhancements:** Consider adding features like user reviews, flight tracking, and personalized recommendations to enhance the overall user experience.

By focusing on these components, you can create a robust and user-friendly flight booking application using the MERN stack.

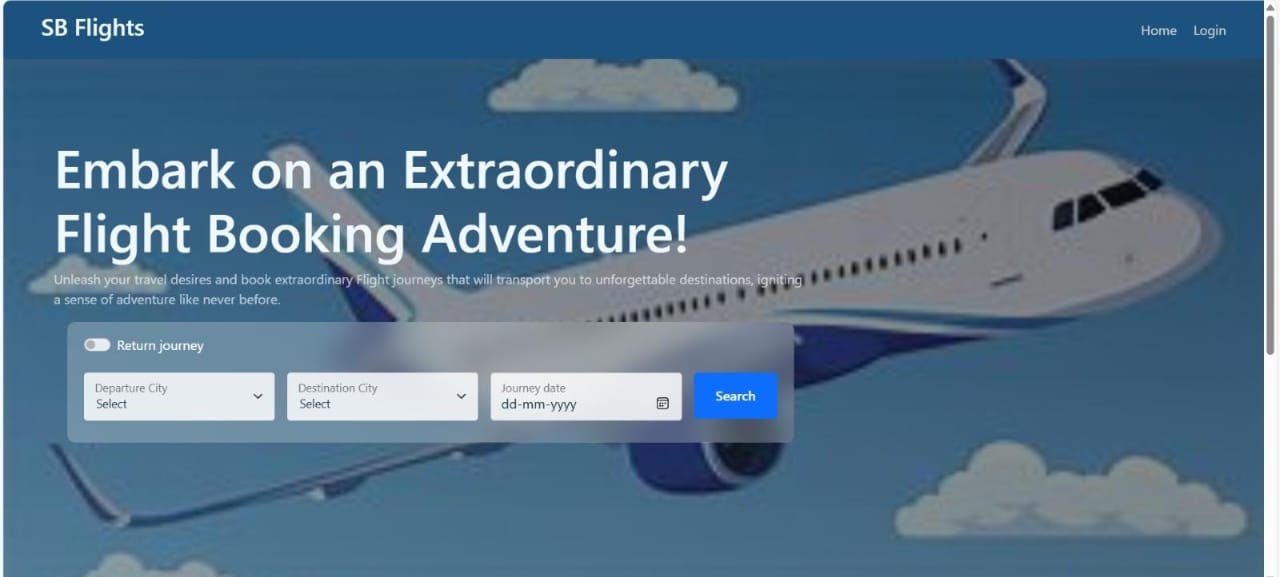
**TESTING**

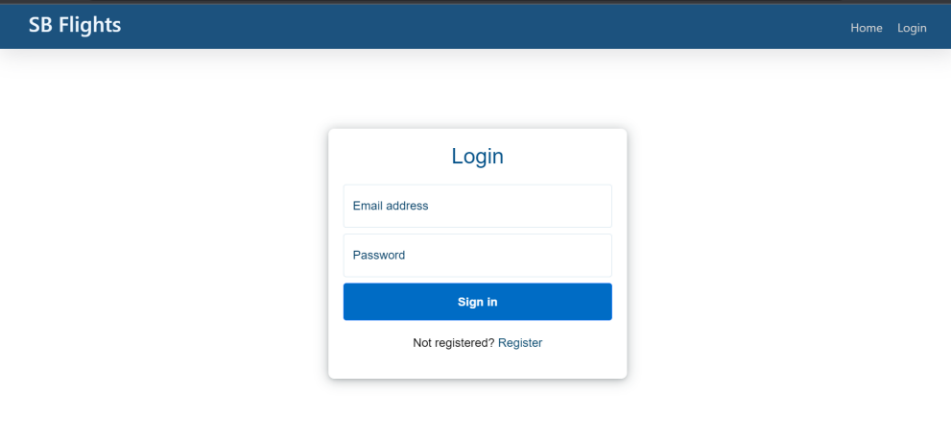
For a flight booking app built with the MERN stack, a comprehensive testing strategy typically includes several layers:

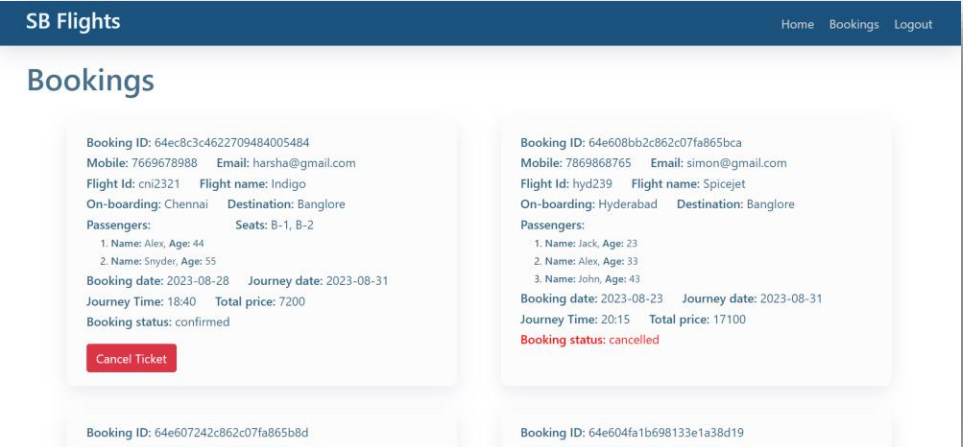
1. **Unit Testing:** Focuses on individual components or functions. Tools like Jest and Mocha are commonly used for testing React components and Node.js functions.
2. **Integration Testing :** Ensures that different modules or services work together as expected. Tools such as Supertest can be used to test API endpoints, while Enzyme or React Testing Library can be utilized for testing React components in conjunction with their child components.
3. **End-to-End Testing:** Simulates real user scenarios to validate the entire application flow. Cypress and Selenium are popular tools for conducting end-to-end tests, allowing for automated browser testing.
4. **Performance Testing:** Assesses the application's responsiveness and stability under load. Tools like JMeter or LoadRunner can be employed to simulate multiple users and measure performance metrics.
5. **User Acceptance Testing (UAT):** Involves real users testing the application to ensure it meets their requirements and expectations. This phase often includes feedback loops for further improvements.
6. **Error Handling and Logging:** Implementing global error handling in Express.js to catch and log errors effectively. Tools like Winston or Morgan can be used for logging purposes.
7. **Continuous Integration/Continuous Deployment (CI/CD):** Integrating testing into the CI/CD pipeline using tools like Jenkins or GitHub Actions to automate testing and deployment processes.
8. **Code Quality and Static Analysis:** Utilizing tools like ESLint and Prettier to maintain code quality and consistency throughout the development process.

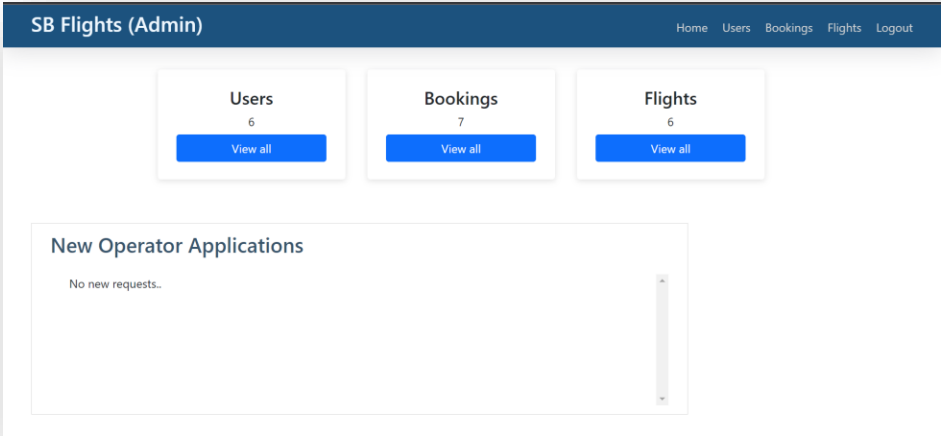
* By employing these strategies and tools, developers can ensure that the flight booking application is robust, reliable, and user-friendly.

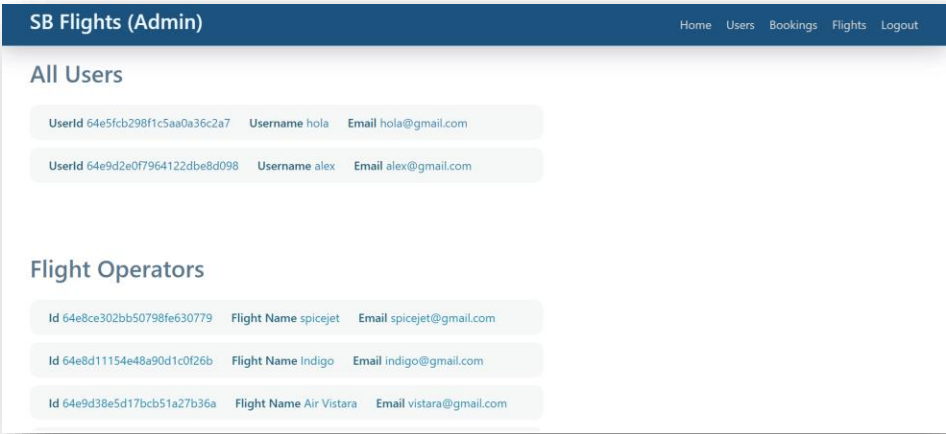
**SCREENSHORTS OR DEMO**

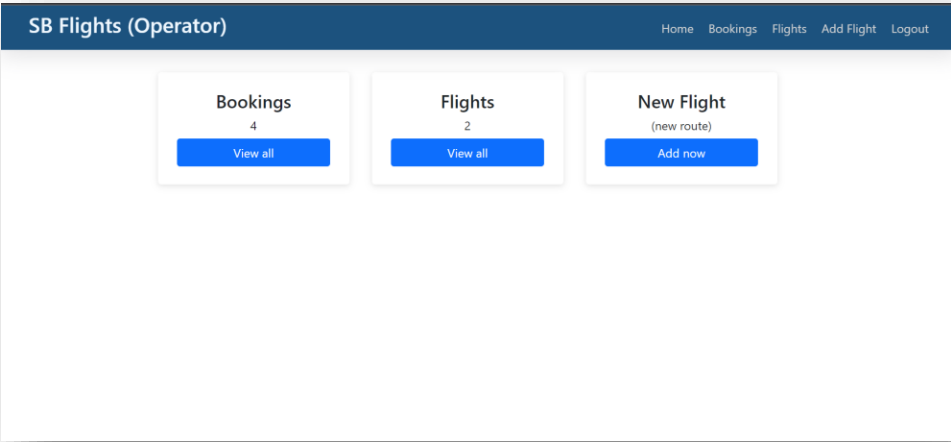


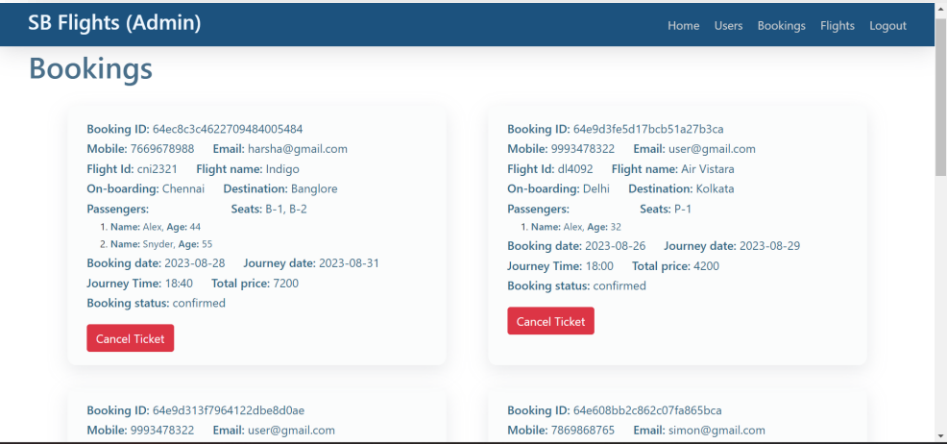
****

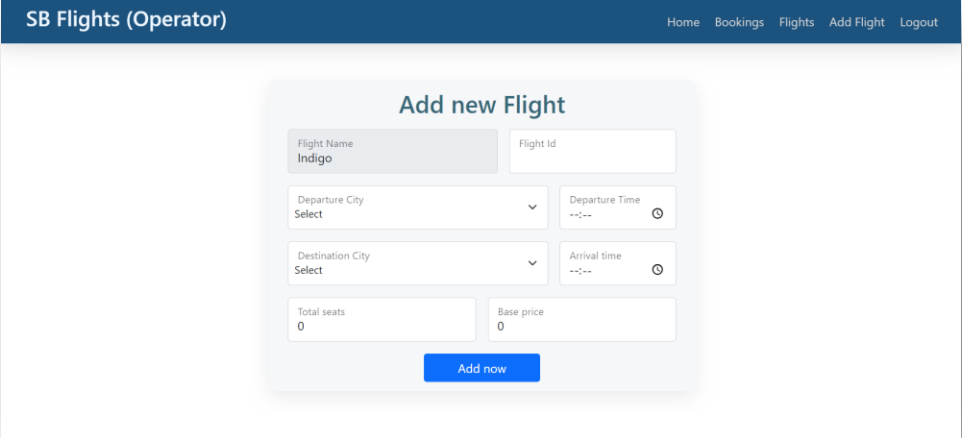
****

****

****

****

****

****

**FUTURE ENHANCEMENTS**

Here are some potential future features and improvements for a flight booking app built with the MERN stack:

1. **Enhanced User Interface:** Improve the design for a more intuitive user experience, including better navigation and responsive layouts for mobile devices.
2. **Real-time Flight Updates:** Integrate APIs to provide users with real-time updates on flight statuses, delays, and cancellations.
3. **Dynamic Seat Selection:** Implement a more interactive seat selection feature that allows users to view available seats in real-time and select their preferred seats.
4. **Personalized Recommendations:** Use machine learning algorithms to analyze user preferences and provide personalized flight and destination recommendations.
5. **Multi-Currency Support:** Allow users to view prices and make payments in multiple currencies to cater to international travelers.
6. **Social Features:** Introduce social sharing options where users can share their travel plans with friends and family, and possibly collaborate on itineraries.
7. **Itinerary Planning:** Enable users to create and manage their travel itineraries, including flight details, accommodation, and activities.
8. **User Reviews and Ratings:** Allow users to leave reviews and ratings for airlines and destinations, helping future travelers make informed decisions.
9. **Loyalty Programs:** Integrate loyalty programs that reward users for frequent bookings, offering discounts or points redeemable for future travel.
10. **Chatbot Integration:** Implement a chatbot for customer support to assist users with common queries and booking issues in real-time.
11. **Augmented Reality Features:** Explore the use of augmented reality to enhance the travel experience, such as virtual tours of destinations or airport navigation.
12. **Accessibility Features:** Ensure the app is accessible to all users, including those with disabilities, by incorporating features like voice commands and screen reader compatibility.
13. **Advanced Search Filters:** Provide users with advanced search options, allowing them to filter flights based on various criteria such as layover duration, airline preference, and price range.
14. **Payment Gateway Integration:** Expand payment options by integrating multiple payment gateways, including digital wallets and buy-now-pay-later services.
15. **Data Analytics Dashboard:** Create an admin dashboard for tracking user behavior, booking trends, and other analytics to inform business decisions and improve user experience.