**Full Stack Development with MERN**

**Project Documentation format**

**1. Introduction**

* **Project Title:** Flight Booking APP
* **Team Members:**

K .Chakri - Frontend

Karthik – Backend

Subhramanyam-API Testing

Triveni – UI

**2. Project Overview**

* **Purpose:**

This Flight Booking APP is the ultimate digital platform designed to revolutionize the way you book flight tickets. With this app your flight travel experience will be elevated to new heights of convenience and efficiency. Our user-friendly web app empowers travelers to effortlessly discover, explore, and reserve flight tickets based on their unique preferences. Whether you're a frequent commuter or an occasional traveler, finding the perfect flight journey has never been easier.

This successful flight booking app combines a user-friendly interface, efficient search and booking capabilities, personalized features, robust security measures, reliable performance, and continuous improvement based on user feedback.

* **Features:**

User registration and authentication.

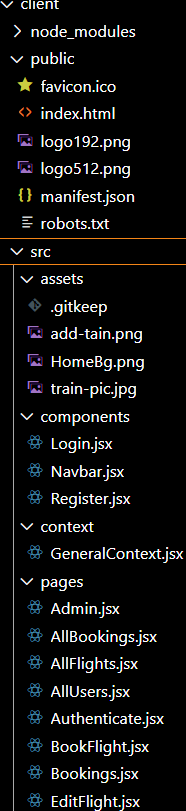
Flight search based on destination, date, and time.

Booking management.

Payment integration.

Admin panel for managing flights and bookings.

* **3. Architecture**
* **Frontend:**



The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Flight Search, and Booking.

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 ):**

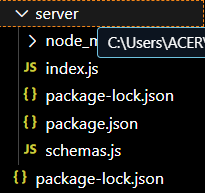
|  |  |
| --- | --- |
|  | * Flight Searching code: With the given inputs, we need to fetch the available flights. With each flight, we add a button to book the flight, which redirects to the flight-Booking page. |
| **3.** | **Fetching user bookings:**   * In the bookings page, along with displaying the past bookings, we will also |

* 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., 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.
2. **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.

**Backend:** Outline the backend architecture using Node.js and Express.js.



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.
2. **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.
3. **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.
4. **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.
5. **User Authentication:**
   * Create routes and middleware for user registration, login, and logout.
   * Set up authentication middleware to protect routes that require user authentication.
6. **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.
7. **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.
8. **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.

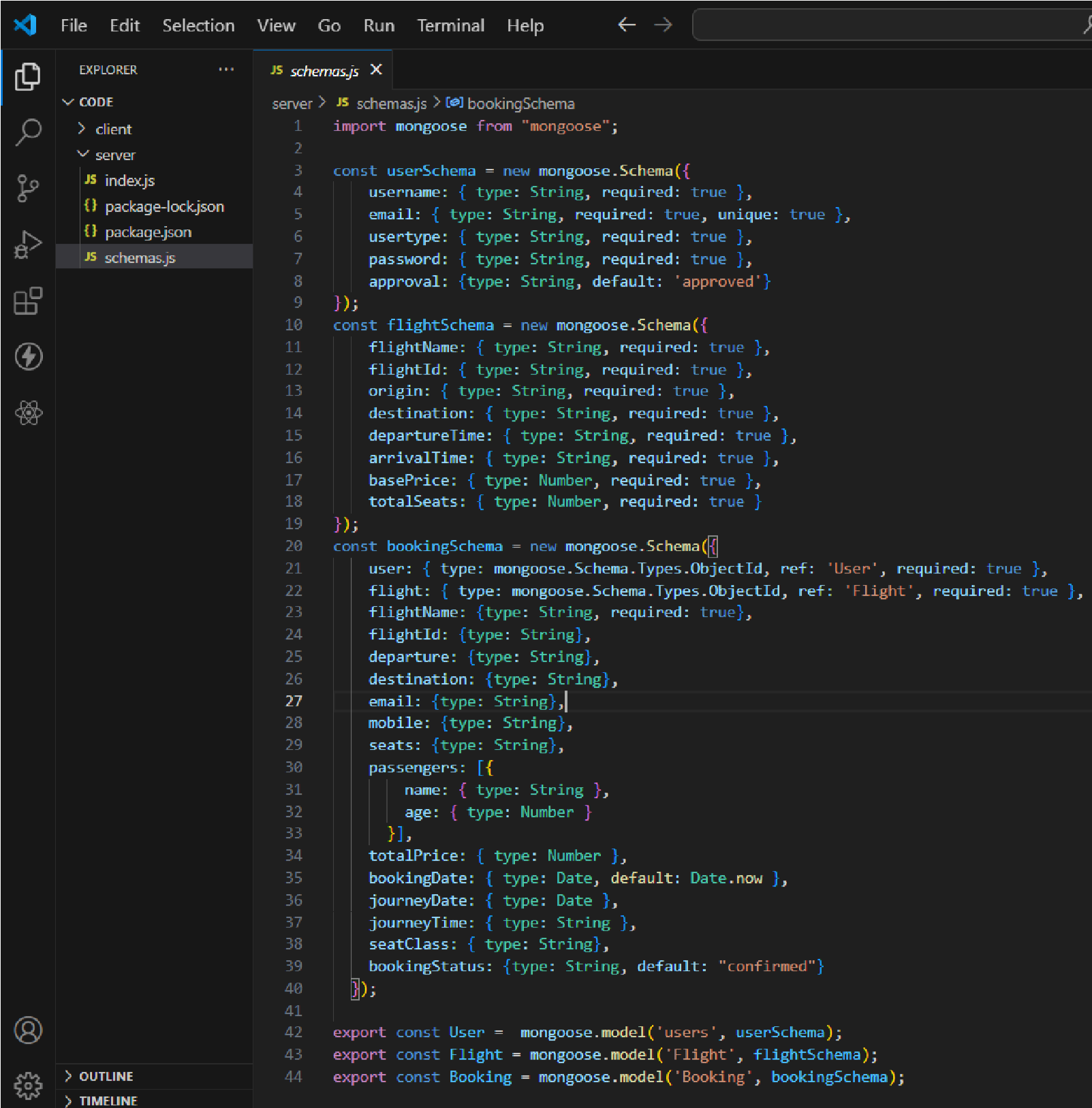
* The backend is represented by the "Backend" section, consisting of API endpoints for Users, Flights, Admin and Bookings. It also includes Admin Authentication and an Admin Dashboard.

**Database:** Detail the database schema and interactions with MongoDB.

The Database section represents the database that stores collections for Users, Flights, and Flight Bookings.

* **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.



**4. Setup Instructions**

* **Prerequisites:** List software dependencies (e.g., Node.js, MongoDB).

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>

**To Connect the Database with Node JS go through the below provided link:**

* <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb>

**To run the existing Flight Booking App project downloaded from github:**

Follow below steps:

**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**

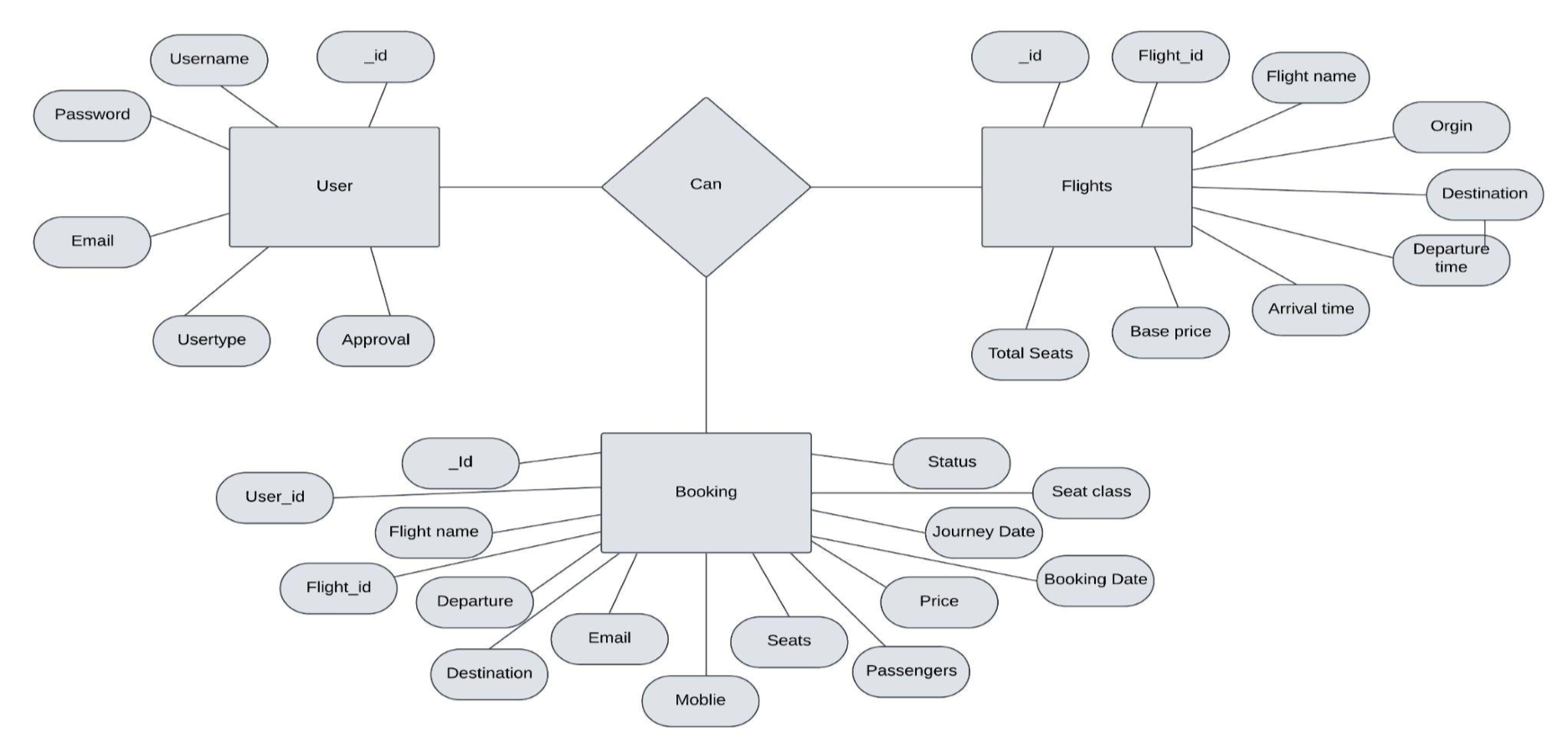
* The e-commerce app will be accessible at <http://localhost:3000> by default. You can change the port configuration in the .env file if needed.

**Access the App:**

* Open your web browser and navigate to <http://localhost:3000>
* You should see the flight booking app's homepage, indicating that the installation and the setup was successful.

You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as needed.

# ER DIAGRAM :



The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

**USER:** Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.

**BOOKING:** Represents a specific flight booking made by a customer. A booking includes a particular flight details and passenger information. A customer can have multiple bookings.

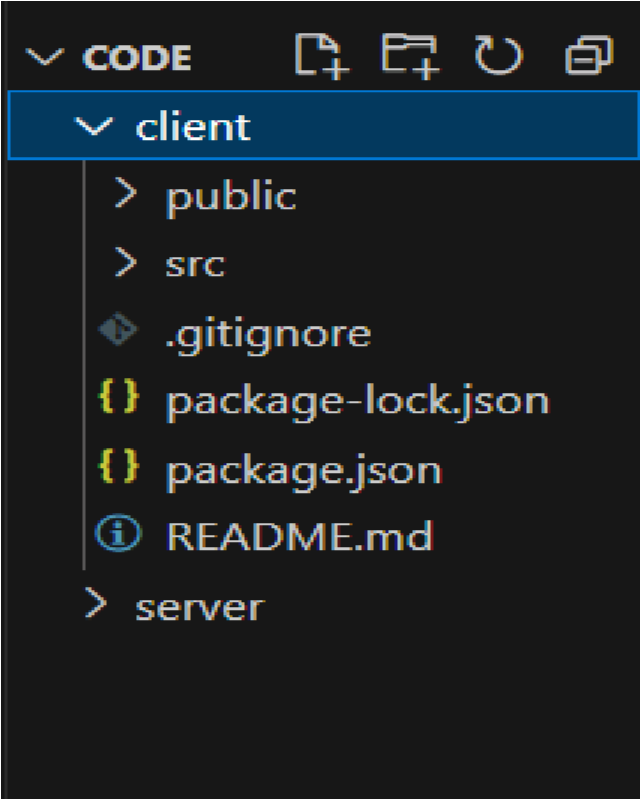
**FLIGHT**: Represents a flight that is available for booking. Here, the details of flight will be provided and the users can book them as much as the available seats.

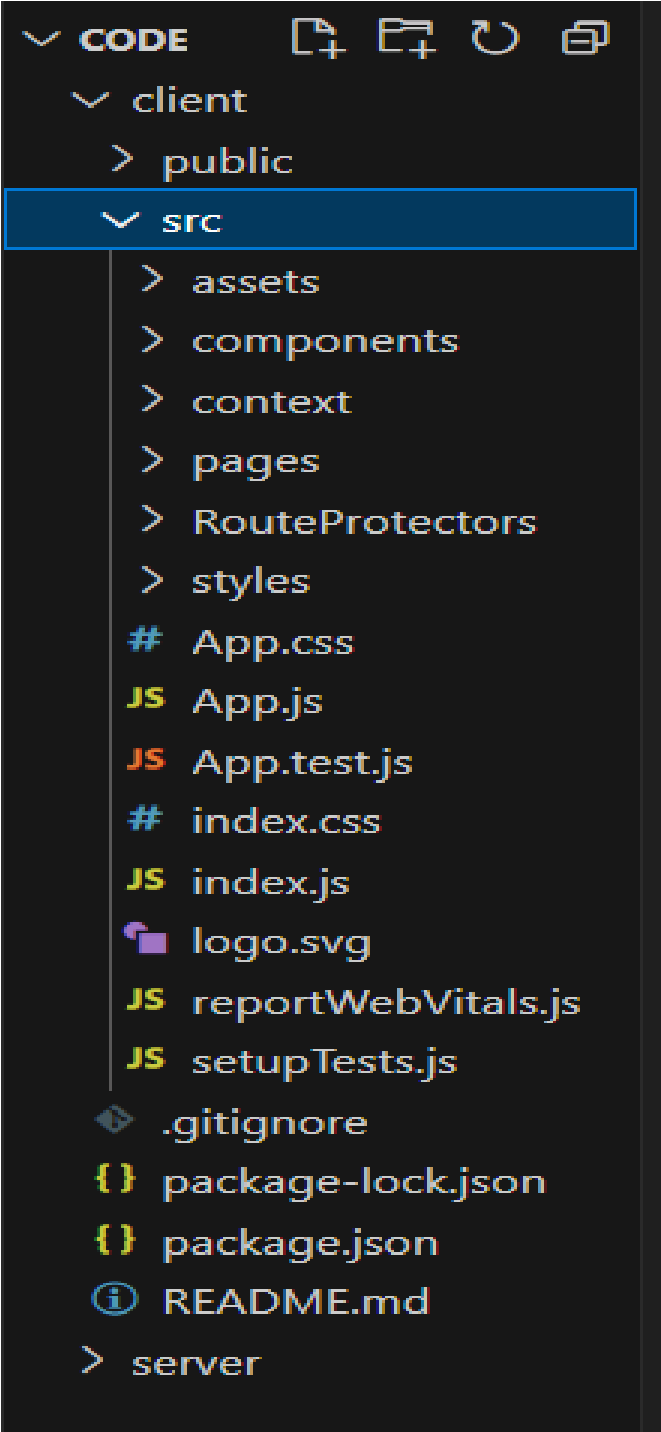
**ADMIN**: Admin is responsible for all the backend activities. Admin manages all the bookings, adds new flights,etc.,

**5. Folder Structure**

* **Client:**

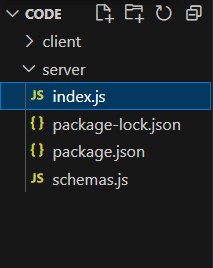
The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api’s.





* **Server:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with Api.



**6. Running the Application**

* Provide commands to start the frontend and backend servers locally.
  + **Frontend:** npm start in the client directory.
  + **Backend:** npm start in the server directory.

**7. API Documentation**

* POST /api/register – User registration
* POST /api/login – User login
* GET /api/flights – Fetch available flights
* POST /api/book – Book a flight
* GET /api/bookings – Fetch user bookings.

**8. Authentication**

Authentication is handled using JWT tokens.

Tokens are stored in HTTP-only cookies.

**Example of token verification middleware in Node.js:**

const jwt = require('jsonwebtoken');

const auth = (req, res, next) => {

const token = req.cookies.token;

if (!token) return res.status(401).json({ message: 'Unauthorized' });

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ message: 'Forbidden' });

req.user = user;

next();

});

};

**9. User Interface**

1. Home Page – Flight search form and featured flights.

2. Search Results – List of available flights matching the user's query.

3. Flight Details Page – Details of the selected flight.

4. Booking Page – Form to book seats and proceed to payment.

5. User Profile – Displays user's bookings and personal information**.**

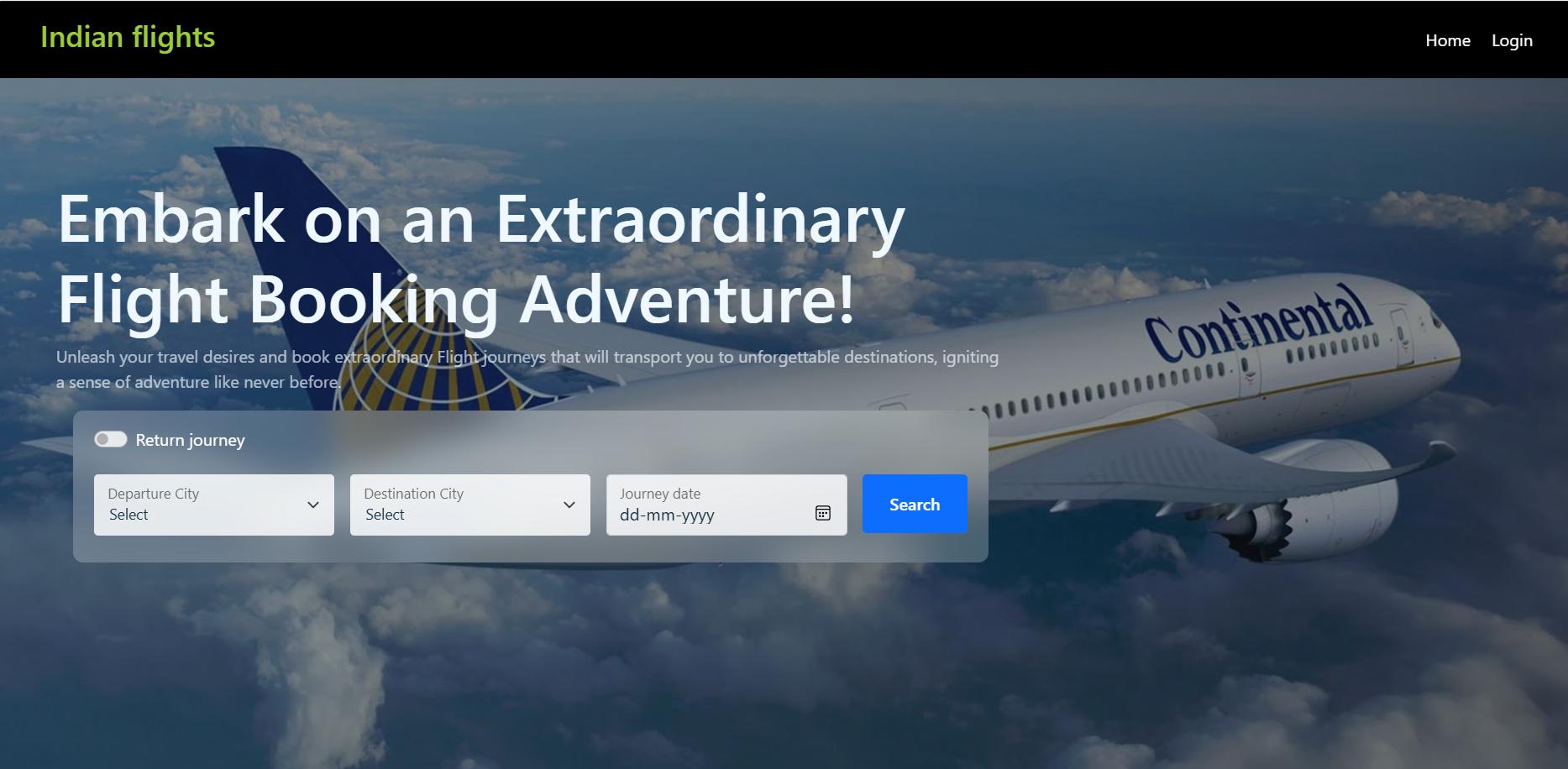
6. Admin Dashboard – Manage flights, bookings, and users.

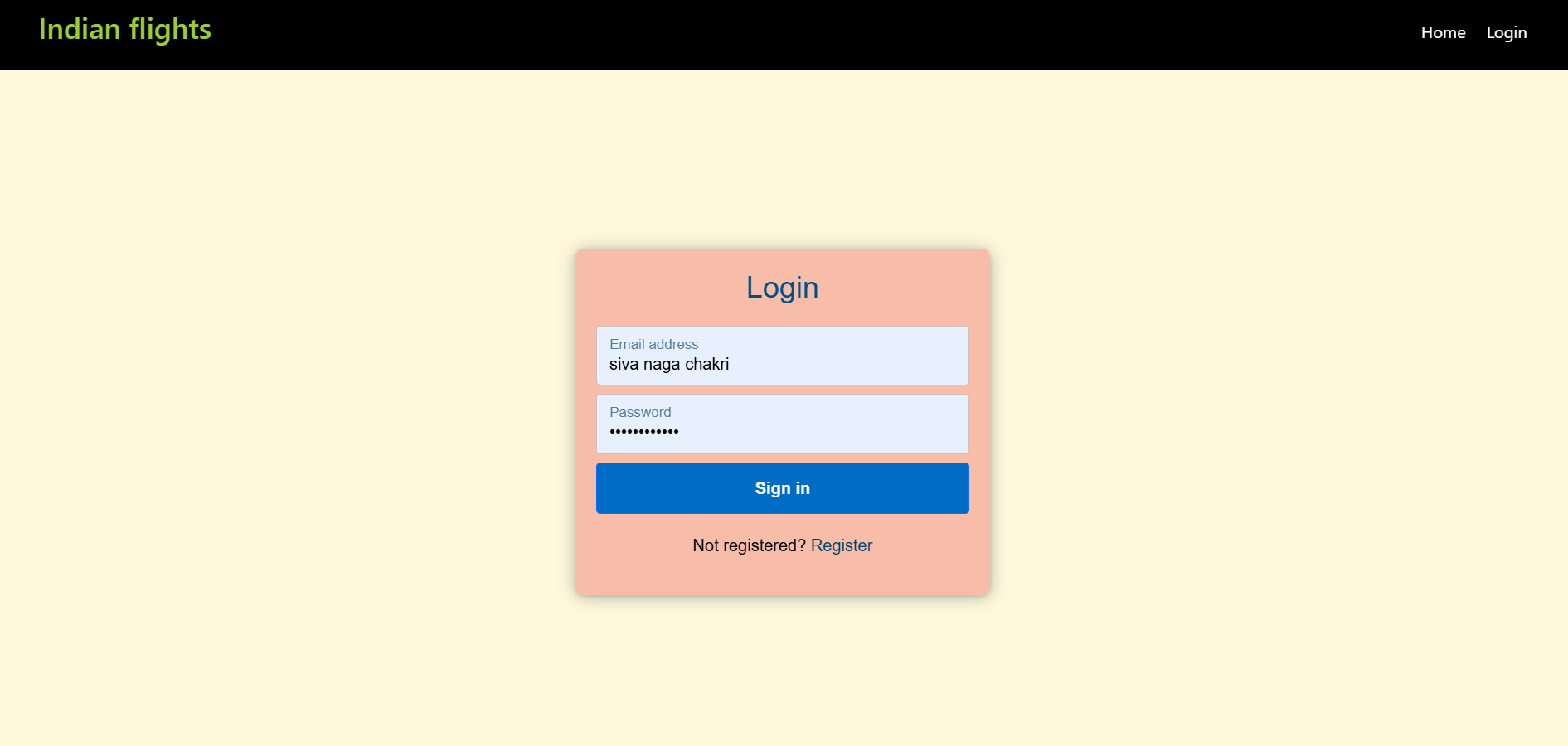
**10. Testing**

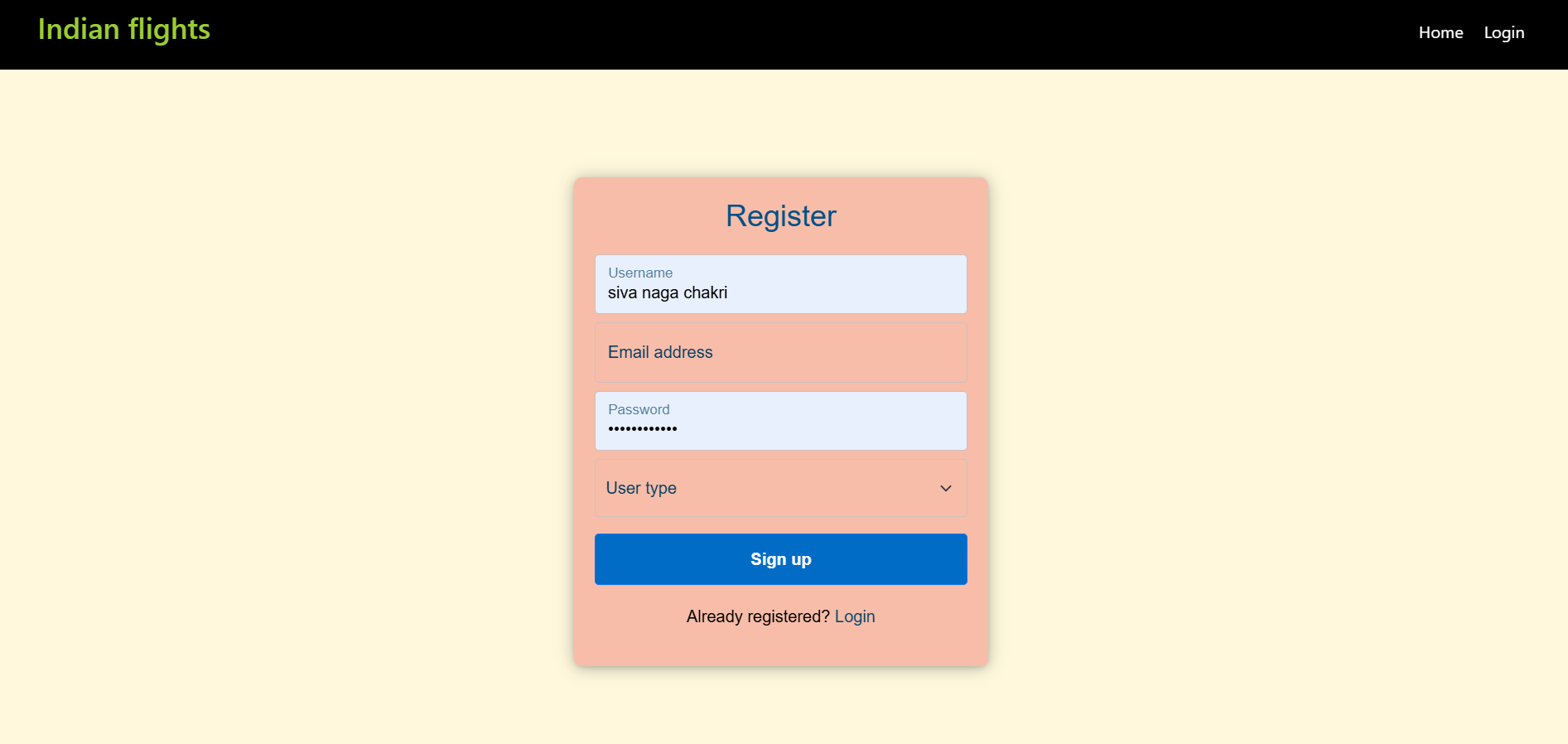
Backend: Using Postman and Thunderclient for API testing.

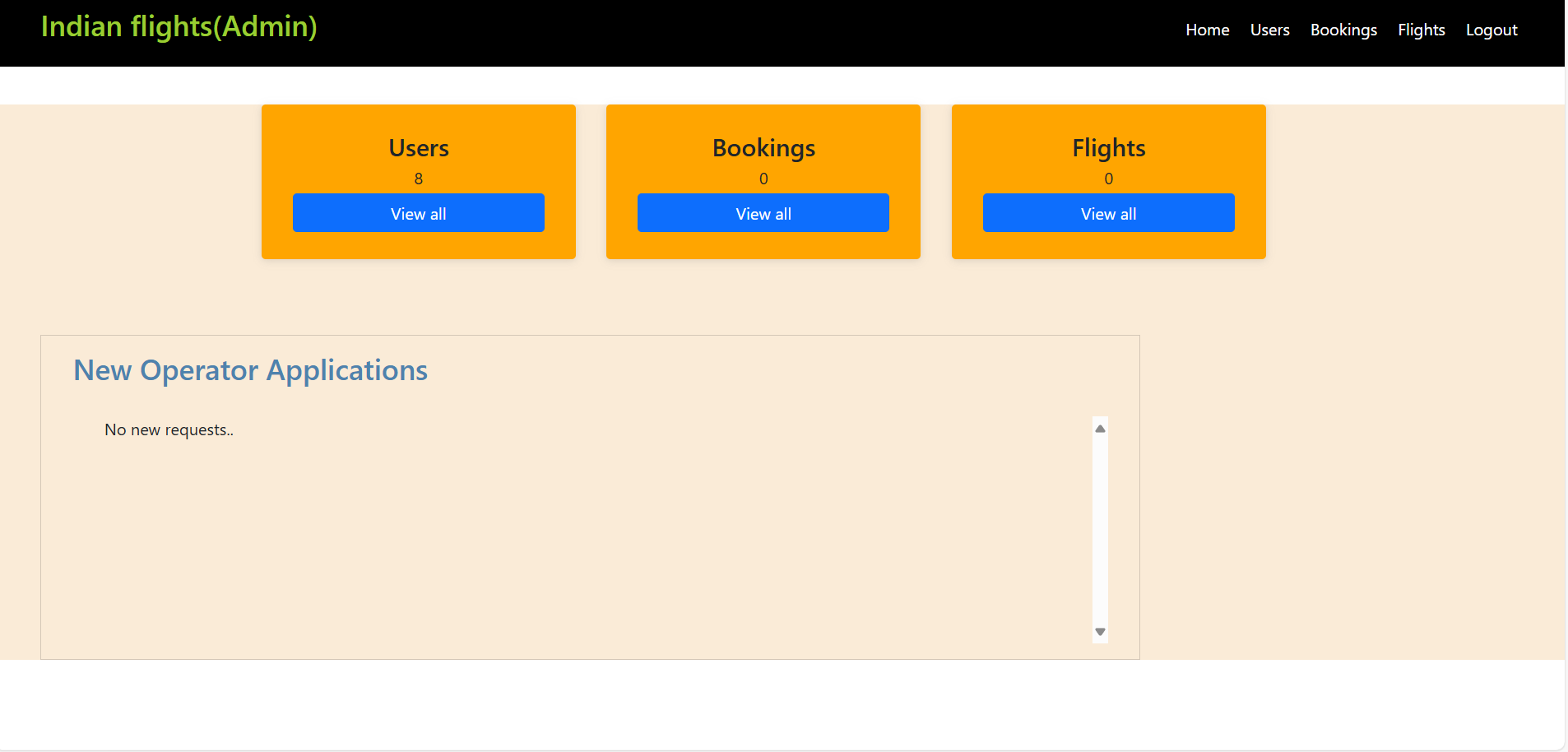
Frontend: Using React Testing Library.

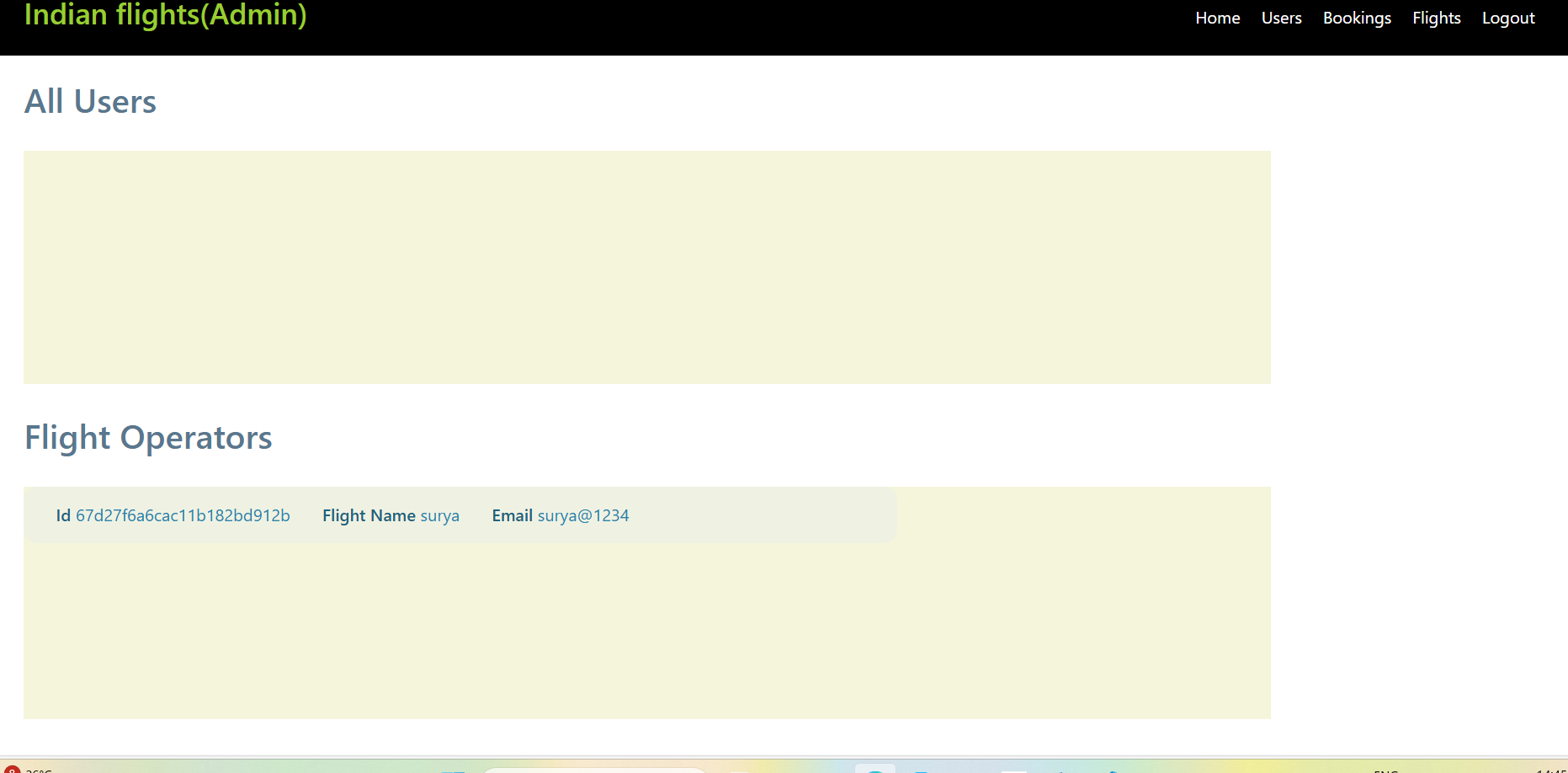
**11. Screenshots**

****

****

****

****

****

****

****

**12. Known Issues**

* Payment gateway integration is under development.
* Occasional lag when loading search results with large datasets.

**13. Future Enhancements**

* Integration with multiple airline APIs for real-time data.
* Dynamic pricing based on demand and season.
* Advanced filtering (e.g., direct flights, flexible dates).
* User reviews and ratings for flights.