

FLIGHT BOOKING APP

1.Introduction:

Project Title: Flight Booking App

Team Members:

S.NO	NAME	ROLE	RESPONSIBILITIES
1	Manikavalli K	Full-Stack Developer	Responsible for overall development, Including front-end, back-end, server-side logic, and database design
2	Kowsalya P	Frontend Developer	Responsible for UI/UX design using React.js, Optimize Frontend performance for seamless user experience
3	Krishna Raj E	Database Administrator	Responsible for MongoDB, Implement Data backup and recovery plans to ensure security.
4	Lekha Shree S	Backend Developer	Responsible for Express.js setup and API development

2.Project Overview:

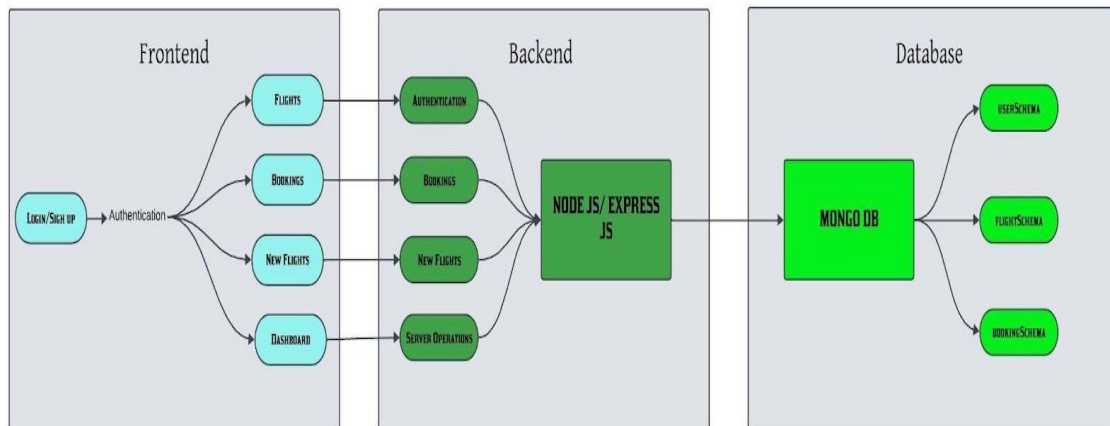
Purpose:

A flight booking app built with the MERN stack (MongoDB, Express.js, React, Node.js) is to provide users with an easy-to-use platform for searching, booking, and managing flights. the app aims to simplify the flight booking process, improve user engagement, and provide a scalable solution for handling high volumes of users and transactions. The app aims to streamline the flight booking process, improve user experience, and provide a reliable backend for managing the application's data and user transactions.

Features:

- **User Authentication**
 - ✓ Secure user registration and login with **JWT**.
 - ✓ Role-based access control (e.g., admin and user).
- **Flight Search and Filtering**
 - ✓ Search flights by origin, destination, travel dates, and passenger count.
 - ✓ Apply filters like price range, airlines, departure/arrival time, and layovers.
- **Flight Booking**
 - ✓ Select available flights and seats.
 - ✓ Calculate ticket price based on class, seats, and passengers. Save booking details for confirmation.
- **Payment Integration**
 - ✓ Enable secure payment processing using third-party gateways (e.g., **Stripe, Razorpay**).
 - ✓ Generate payment receipts and transaction logs.
- **Secure Data Management**
 - ✓ Protect sensitive information like user details and payment data using **encryption**.

3.Architecture:



Frontend:

- Design and implement responsive and user-friendly interfaces using HTML, CSS, JavaScript, and frameworks (e.g., React, Angular).
- Create interactive features like flight search, booking forms, and payment pages.
- Ensure cross-browser compatibility and a consistent user experience.
- Optimize front-end performance and maintain accessibility standard.

Backend:

- Build and maintain the server-side logic, APIs, and business processes using Node.js, Django, or similar frameworks.
- Implement secure user authentication and session management.
- Handle core functionalities like flight data retrieval, ticket booking, and payment processing.
- Ensure the back-end is scalable, secure, and integrates seamlessly with the database.

3. Database:

MongoDB is chosen for its flexibility and scalability, making it ideal for managing diverse and evolving data structures in a Flight Booking App.

It is used to define the application's schemas and models, with key collections such as:

- **Users:** Stores user information (e. g: name, email, password, role)
- **Flights:** Stores flight details, such as airline, source, destination, departure Time, arrival Time, price.
- **Bookings:** Tracks user reservations, including user Id, flight Id, seats Booked, total Price, booking Status.
- **Payments:** Manages transaction data, including payment Id, booking Id, amount, payment Status.

4. Setup Instructions:

Prerequisites:

1. Node.js and npm:

- Install Node.js to enable server-side JavaScript execution.
- Use npm (Node Package Manager) to manage dependencies.

2. MongoDB:

- Set up a running MongoDB instance, either locally or using a cloud solution like MongoDB Atlas.

3. Express.js:

- Install Express.js, a Node.js framework for routing, middleware, and API development.

Installation:

1. Clone the Repository: `git clone https://github.com/harsha-varadhan-reddy-07/Flight-Booking-App-MERN.git`

2. Install Dependencies:

- Navigate to the root folder and run:

```
npm install
```

3. Start the Development Server:

- Run the following command to start the server:

```
npm run dev  
npm run start
```

4. Configure Environment Variables:

- Create a .env file in the root directory and define:

```
env  
MONGO_URI=mongodb://localhost:27017/flight_booking_app # or  
MongoDB Atlas URI  
PORT=5000 # or your desired port  
JWT_SECRET=your_jwt_secret # for secure authentication
```

5. Start the Backend Server:

- Run the following command in the backend directory:

```
npm start
```

6. Start the Frontend Server:

- Move to the frontend directory and run:

```
npm start
```

5. Folder Structure:

Client:

- **src/components/**: Reusable UI components like Nav Bar, Footer, Flight Card, and Booking Form.
- **src/pages/**: Main pages such as Home, Search Flights, Booking, and User Profile.

- **src/services/:** Contains helper functions for making API calls (e.g., `api.js`) or separate files for managing user, flight, and booking requests.
- **Environment Variables:** Store client-specific environment variables in `.env` (e.g., `REACT_APP_API_URL` for backend communication).
- **package.json:** Lists dependencies and scripts for building and running the React app.

Server:

- **controllers/:** Business logic for each route (e.g., `flightController.js`, `UserController.js`, `bookingController.js`).
- **models/:** Defines Mongoose schemas for collections like `User.js`, `Flight.js`, and `Booking.js`.
- **routes/:** Defines API endpoints using Express (e.g., `flightRoutes.js`, `userRoutes.js`, `bookingRoutes.js`).
- **middleware/:** Custom middleware for authentication, error handling, or validation (e.g., `authMiddleware.js`).
- **package.json:** Lists backend dependencies and scripts for starting or running the server.

6. Running the Application:

1.Frontend: To start the frontend server, run the following command in the client folder:

```
npm start
```

2.Backend: To start the backend server, run the following command in the server folder:

```
npm run
```

7.API Documentation:

1.POST /users/register: Register a new user.

➤ Request Body:

```
{  
  "name": "John Doe",  
  "email": "john.doe@example.com",  
  "password": "securepassword"  
}
```

➤ Response:

```
{  
  "message": "User registered successfully",  
  "user": {  
    "id": "userId",  
    "name": "John Doe",  
    "email": "john.doe@example.com"  
  }  
}
```

2.POST /users/login: Authenticate user and generate a JWT token.

➤ Request Body:

```
{  
  "email": "john.doe@example.com",  
  "password": "securepassword"  
}
```

➤ Response:

```
{  
  "token": "jwt_token",  
  "user": {  
    "id": "userId",  
    "name": "John Doe",  
    "email": "john.doe@example.com"  
  }  
}
```

3.GET /flights/search: Search for available flights based on criteria

➤ Response:

```
[
  {
    "flightId": "123",
    "airline": "Airline Name",
    "source": "NYC",
    "destination": "SFO",
    "departureTime": "10:00 AM",
    "arrivalTime": "1:00 PM",
    "price": 200,
    "availableSeats": 50
  }
]
```

4.GET /flights/:id:Retrieve details of a specific flight.

➤ Response:

```
{
  "flightId": "123",
  "airline": "Airline Name",
  "source": "NYC",
  "destination": "SFO",
  "departureTime": "10:00 AM",
  "arrivalTime": "1:00 PM",
  "price": 200,
  "availableSeats": 50,
  "class": "Economy"
}
```

5.POST /bookings: Book a flight for a user.

➤ Request Body:

```
{
  "userId": "userId",
  "flightId": "flightId",
  "seats": 2,
```



```
"class": "Economy"  
}
```

➤ Response:

```
{  
  "message": "Booking successful",  
  "bookingId": "booking123",  
  "totalPrice": 400  
}
```

6.GET /bookings/user/:userId: Retrieve all bookings made by a user.

➤ Response:

```
[  
  {  
    "bookingId": "booking123",  
    "flightId": "123",  
    "airline": "Airline Name",  
    "source": "NYC",  
    "destination": "SFO",  
    "departureTime": "10:00 AM",  
    "seats": 2,  
    "totalPrice": 400,  
    "status": "Confirmed"  
  }  
]
```

7.POST /payments: Initiate a payment for a booking.

➤ Request Body:

```
{  
  "bookingId": "booking123",  
  "amount": 400,  
  "paymentMethod": "Credit Card"  
}
```

➤ Response:

```
{
```

```
"message": "Payment successful",  
"transactionId": "txn123",  
"paymentStatus": "Success"  
}
```

8.Authentication AND Authorization:

Authentication:

Authentication for the Flight Booking App involves using JWT (JSON Web Tokens) to securely manage user registration, login, and access to protected features. Users can register by providing their details, and upon successful login, a token is generated and sent to the client. Protected routes, such as booking flights, are secured using middleware that verifies the validity of the JWT token, ensuring only authenticated users can access sensitive functionalities.

- Verifies the identity of the user is (login username and password).

Authorization:

Authorization in the Flight Booking App ensures that users have the appropriate permissions to access specific features or resources. Once a user is authenticated (via JWT), the app checks their role or permissions before granting access to protected routes or actions. Authorization is typically handled through middleware that verifies the user's role or permissions, ensuring they are authorized to perform certain tasks or access specific data within the app.

- Determines what the authenticated user can do (e.g., whether they can view orders, book flights, or manage the system as an admin)

9.User Interface:

User Interface (UI) for the Flight Booking App is the visual part of the application that users interact with. It is designed to provide a seamless and intuitive experience for booking flights, managing personal information, and accessing booking details.

Key elements of the UI might include:

1. Login and Registration Pages:

These allow users to securely log in or create an account. They typically consist of fields for entering a username, password, and sometimes additional information like email or phone number.

2. Homepage:

The main dashboard that might display available flights, search filters, and user options to book a flight or view existing bookings.

3. Flight Search and Booking Form:

A form or search bar that allows users to select their origin, destination, travel dates, and other preferences to find available flights.

4. Booking Summary Page:

A page displaying flight details such as flight time, price, seat options, and passenger information, where users can confirm their booking.

5. User Profile:

A section where users can view and edit their personal details, booking history, and payment methods.

6.Navigation Elements:

Menus, buttons, and links that help users easily navigate between different sections of the app, such as the booking page, profile settings, and logout options.

10.Testing:

Testing for a Flight Booking App involves ensuring that both the frontend and backend perform as expected, including testing interactions with the database. Below is an outline of the testing strategy and tools that can be used:

- **Frontend testing:**

Jest, React Testing Library, and Cypress for simulating user interactions and verifying UI behavior.

- **Backend testing:**

Jest and Super test to validate API endpoints and business logic.

- **Database testing:**

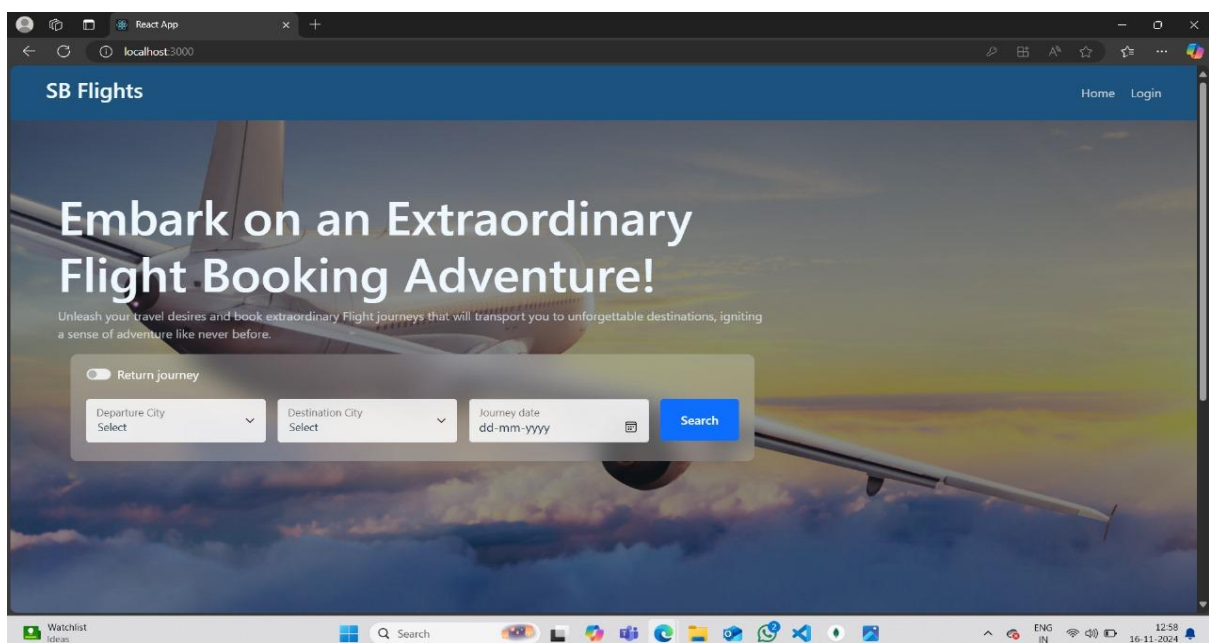
MongoDB In-Memory Server or Mock MongoDB to ensure that CRUD operations work correctly without affecting live data.

- **End-to-End (E2E) testing:**

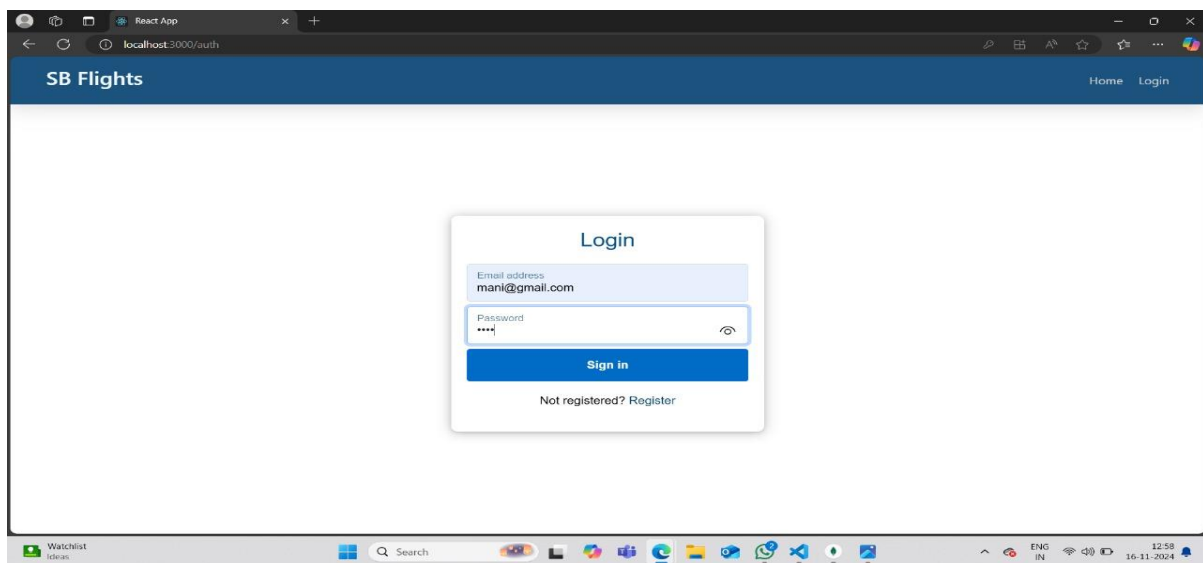
Cypress to simulate real user flows, ensuring that the app works as intended from a user's perspective.

11.Screenshots:

Landing page:

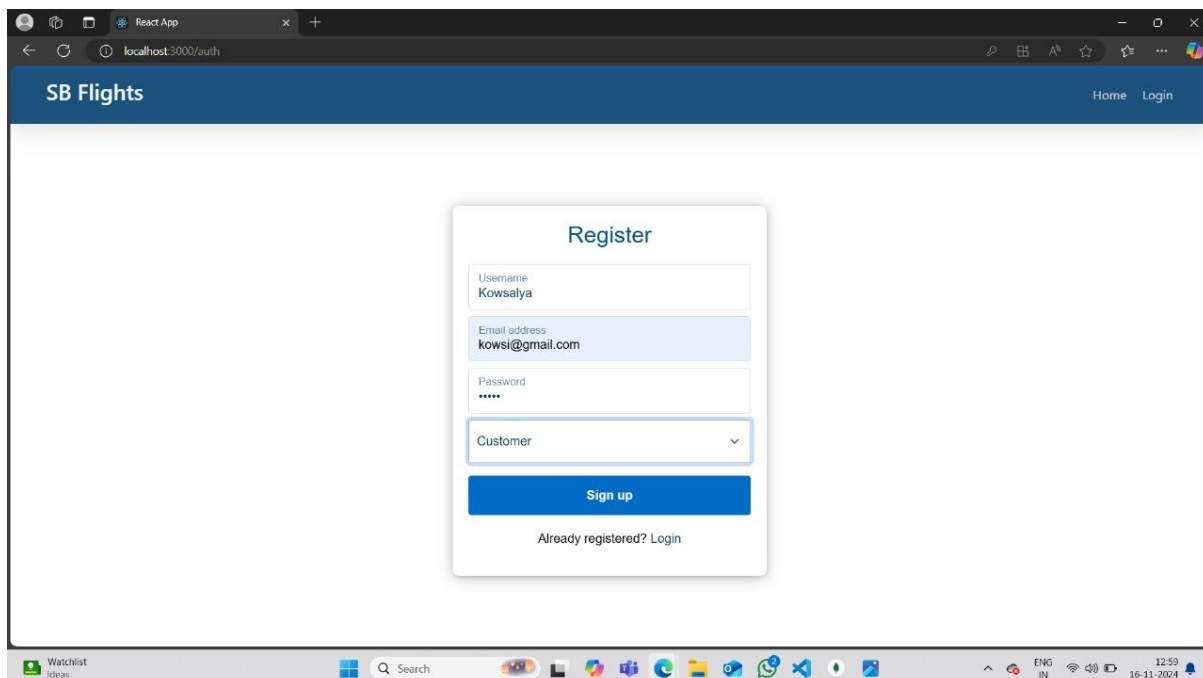


Login Page:



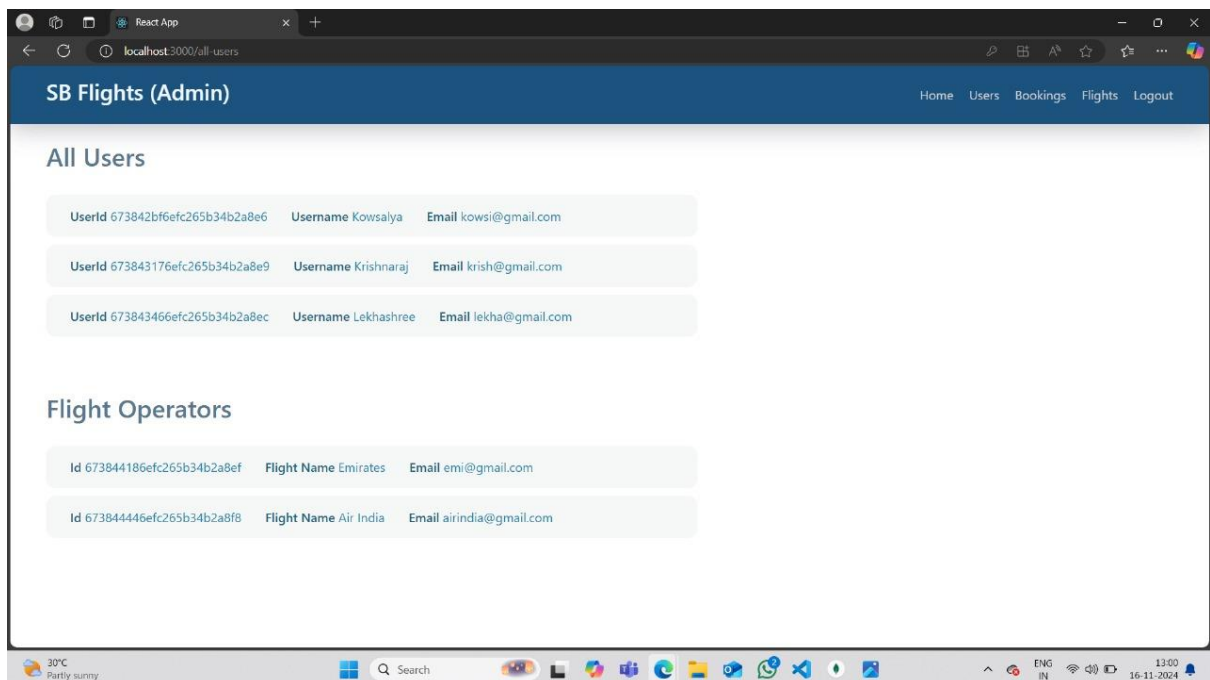
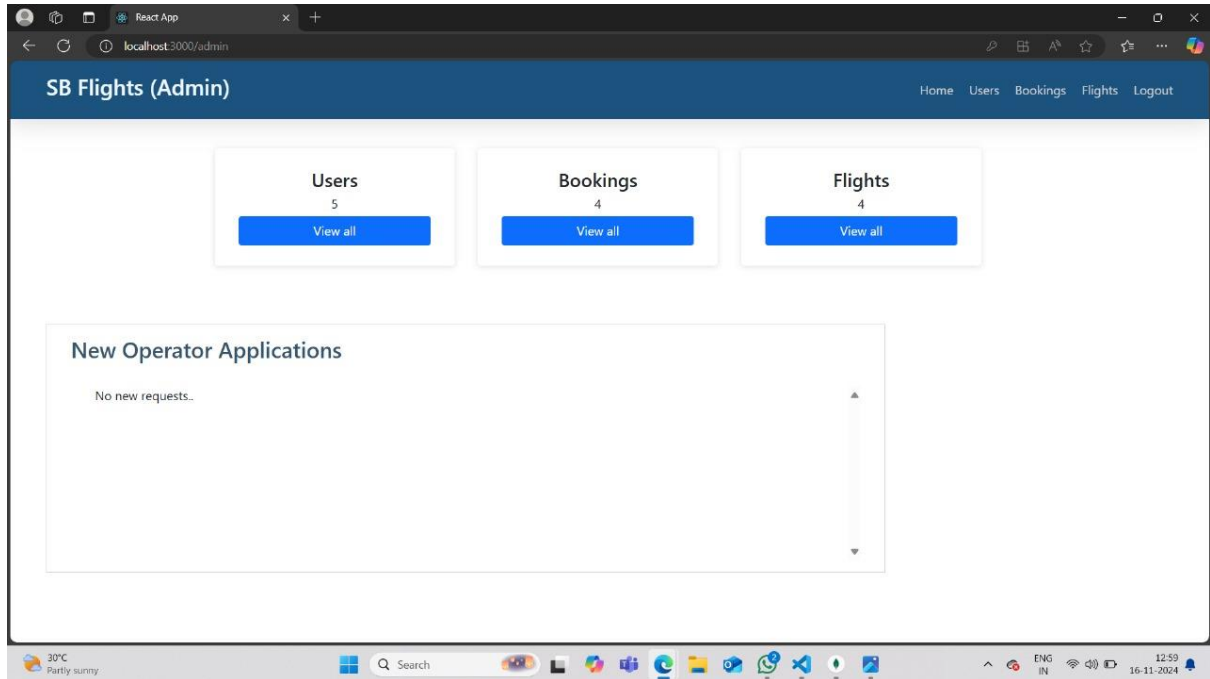
The screenshot shows a web browser window with the URL `localhost:3000/auth`. The page has a dark blue header with the text "SB Flights" on the left and "Home Login" on the right. The main content area is white and contains a centered "Login" form. The form has two input fields: "Email address" with the value "mani@gmail.com" and "Password" with masked characters "****". Below the fields is a blue "Sign in" button. At the bottom of the form, there is a link that says "Not registered? Register". The browser's taskbar at the bottom shows various application icons and the system clock indicating 12:58 on 16-11-2024.

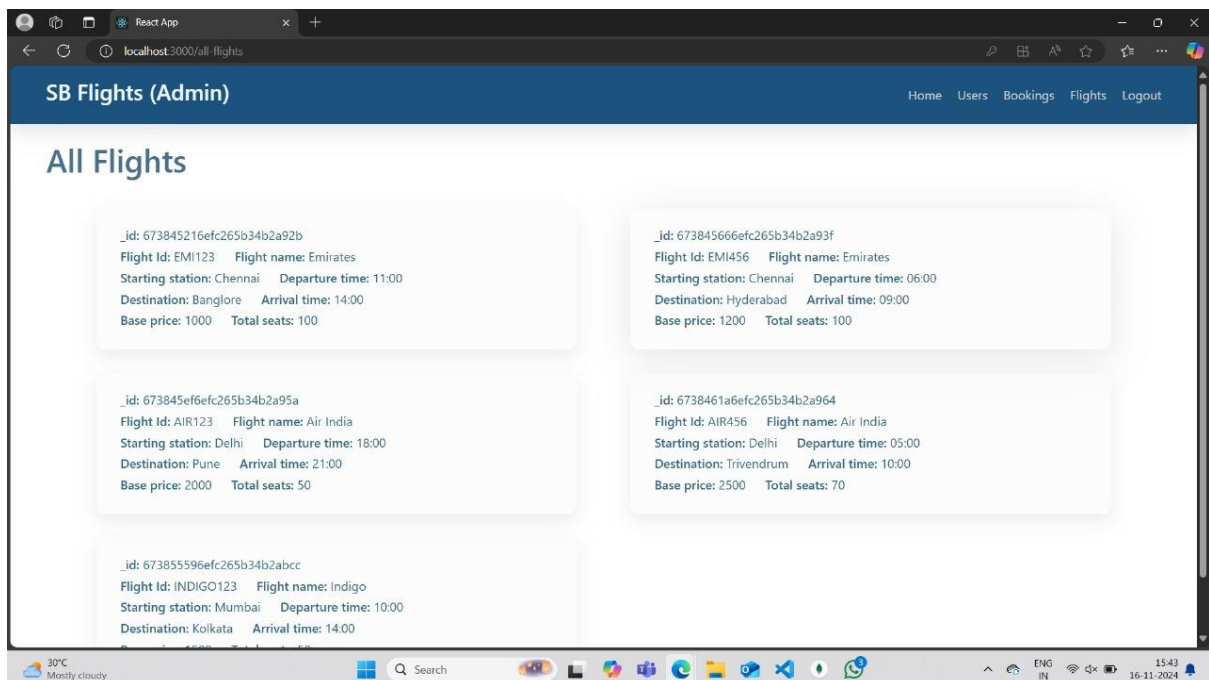
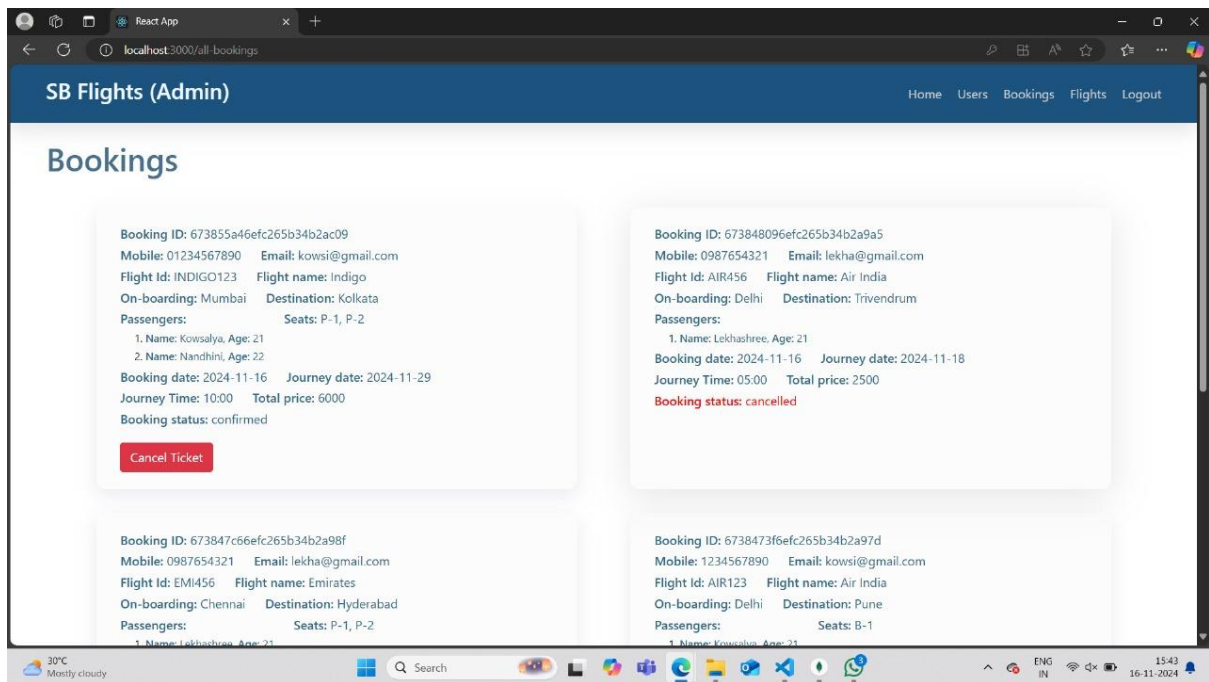
Authentication:



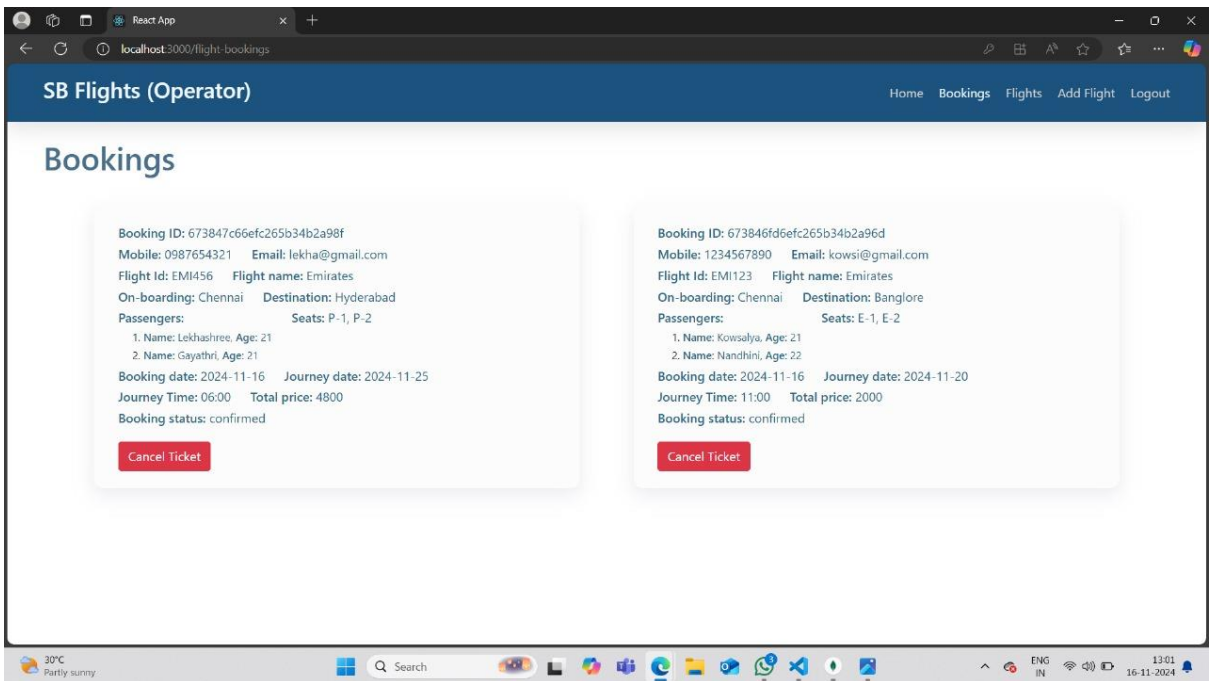
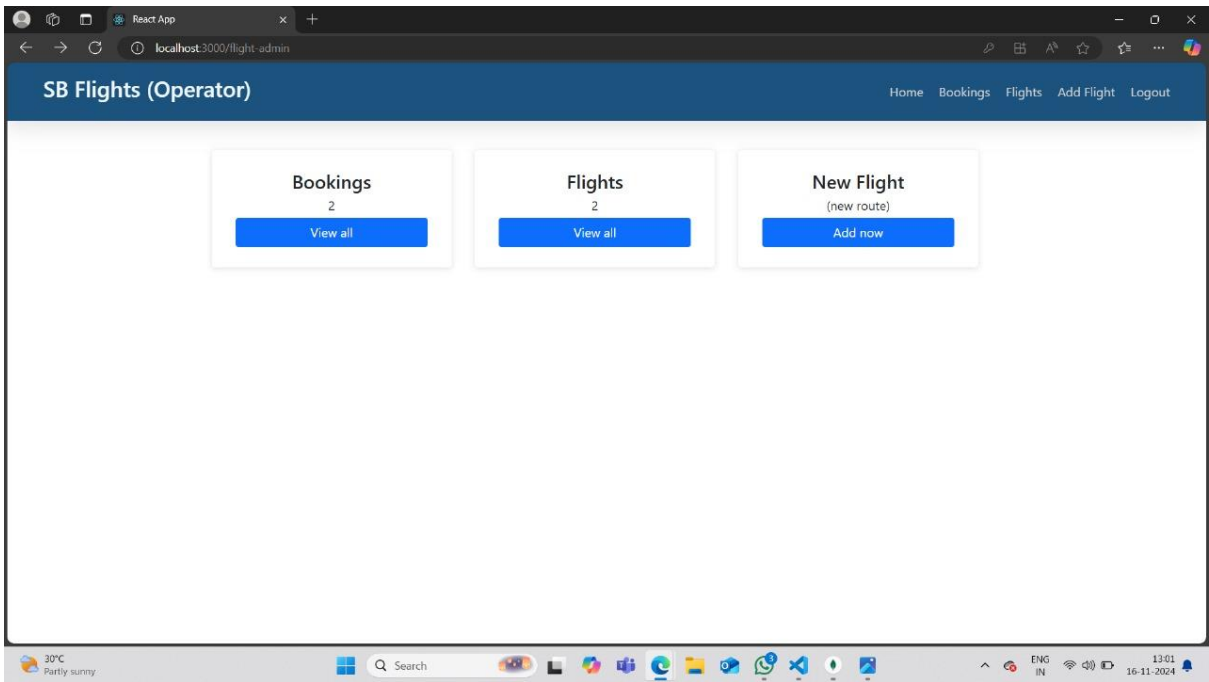
The screenshot shows the same web browser window, but the URL is still `localhost:3000/auth` and the page content has changed to a "Register" form. The form includes four input fields: "Username" with the value "Kowsalya", "Email address" with the value "kowsi@gmail.com", "Password" with masked characters "*****", and a dropdown menu for "Customer" currently showing "Customer". A blue "Sign up" button is positioned below these fields. At the bottom of the form, there is a link that says "Already registered? Login". The browser's taskbar remains the same, showing the system clock at 12:59 on 16-11-2024.

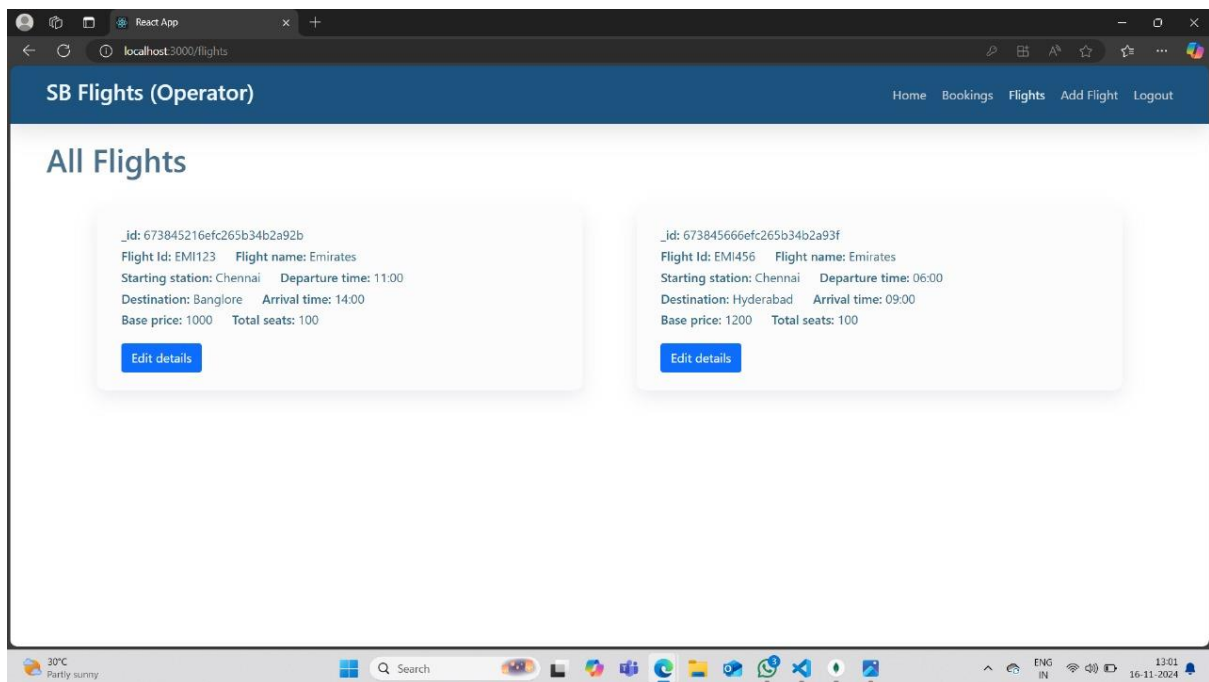
Admin dashboard:



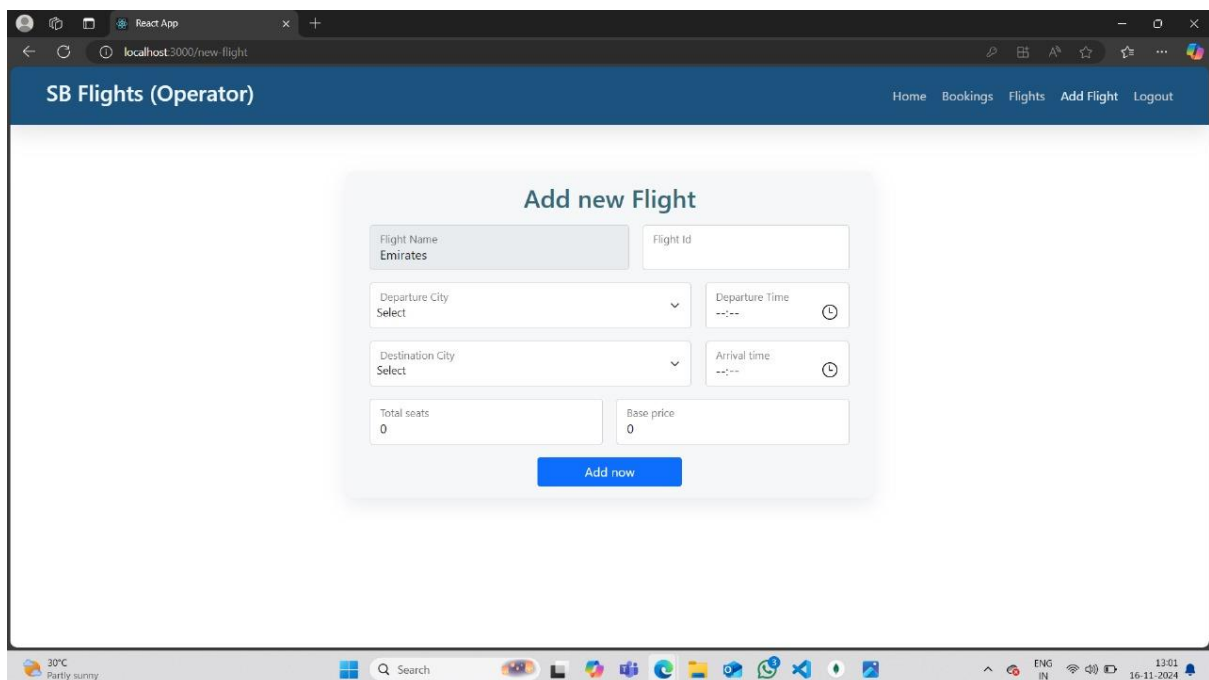


Flight Operator Dashboard:





New Flight Booking:



12. Known Issues:

While the Flight Booking App is generally functional, there are a few known issues that both developers and users should be aware of:

1. Performance Issues with Large Datasets

- As the number of users, menu items, bookings, and orders increases, MongoDB queries can become slower. This is particularly noticeable if indexes are not properly configured, leading to longer response times and a slower overall user experience

2. Security Vulnerabilities

- The application might be vulnerable to attacks such as NoSQL injection, weak password storage, and insecure API endpoints.

3. Payment Gateway Integration Issues

- There might be issues with the payment gateway integration, such as payment failures, discrepancies in payment statuses, or problems updating the order status after a successful payment.

4. High Disk Usage

- The application might consume excessive disk space, especially if large collections of booking data are not properly managed. This can lead to performance issues or increased operational costs due to inefficient storage management.

5. Issues with Order Tracking and Status Updates

- Users may face difficulties in tracking their orders. Order status updates might be delayed or fail to appear properly in the app.

6. Push Notification Problems

- Users might not receive push notifications in a timely manner, or they may be overwhelmed by too many notifications.

13.Future Enhancements:

The Flight Booking App is functional, several future enhancements can be implemented to improve performance, security, user experience, and scalability. These enhancements can make the app more competitive, user-friendly, and efficient. Below are some potential improvements:

1. Performance Optimization:

- **Implement Caching:** Use caching mechanisms (e.g., Redis) to store frequently accessed data, like flight availability or pricing, to reduce database load and improve response times.

2. Enhanced User Experience:

- **Personalized Recommendations:** Implement machine learning algorithms to provide users with personalized flight recommendations based on their booking history, preferences, and search patterns.

3. Improved Security:

- **Two-Factor Authentication (2FA):** Add two-factor authentication (2FA) to enhance user login security. This could be done through email, SMS, or authentication apps like Google Authenticator.

4. Payment Integration Enhancements:

- **Multiple Payment Gateways:** Integrate multiple payment providers (e.g., PayPal, Stripe, Google Pay) to give users more payment options and ensure reliability in case of gateway failures

5. Mobile App Development:

- **Native Mobile App:** Develop native mobile applications for iOS and Android to offer a better user experience, especially for users on the go. The mobile app can leverage device-specific features, such as push notifications, biometrics, and offline capabilities.

