**Flight Ticket Booking Development with MERN**

**Project Documentation**

**1. Introduction**

* **Project Title:** Flight Ticket Booking Web App
* **Team Members:** Logeshwaran V(Team leader),Mahesh K(Team Member),Hariharasudhan K(Team Member), Danian J(Team Member).

**2. Project Overview**

* **Purpose:**

The Flight Ticket Booking System, built using the MERN stack (MongoDB, Express.js, React, Node.js), aims to provide a seamless platform for users to search, compare, and book flights. It ensures a user-friendly interface, secure payment processing, and personalized flight recommendations. The system is designed to handle scalability, offer real-time updates, and simplify the booking process with efficient search and filtering options, creating a modern and reliable solution for flight ticket reservations.

* **Features:**

**Flight Search and Filtering**

Search flights based on destination, departure date, return date, and passenger count.

Filter options for airlines, price range, travel class, layovers, and flight duration.

**Flight Details**

View detailed flight information, including departure and arrival times, stops, and baggage allowance.

**Secure Booking and Payments**

Easy booking process with secure online payment options.

Support for multiple payment methods (credit/debit cards, wallets, etc.).

**User Account Management**

Register and log in to manage bookings and preferences.

View booking history and download tickets.

**Real-Time Notifications**

Receive updates on booking confirmations, flight status, and cancellations.

**Responsive Design**

Optimized for use on desktops, tablets, and mobile devices.

**Admin Panel**

Manage flight schedules, availability, and pricing.

Monitor user activity and generate reports.

**Personalized Recommendations**

Suggest flights based on user search history and preferences.

**Cancellation and Refund Management**

Simplified process for canceling bookings and processing refunds.

**Multilingual and Multi-Currency Support**

Cater to users from different regions with multiple language and currency options.

**3. Architecture**

**1. Front-End (Client Layer)**

* Technology: React.js
* Responsibilities:
  + Provides an interactive and responsive user interface.
  + Handles user interactions such as searching flights, viewing details, booking, and payments.
  + Communicates with the backend API for data retrieval and updates.
  + Implements state management using tools like React Context API or Redux.

**2. Back-End (Application Layer)**

* **Technology**: Node.js with Express.js
* **Responsibilities**:
  + Exposes RESTful APIs for communication between the front-end and the database.
  + Implements business logic, such as search filters, booking workflows, and payment validation.
  + Handles user authentication and authorization using JSON Web Tokens (JWT).
  + Manages flight scheduling, pricing algorithms, and cancellation policies.
* **Database:**
* Technology: MongoDB
* Responsibilities:
  + Stores and manages application data, including:
    - Flight schedules and details.
    - User information and booking history.
    - Payment records and transaction logs.
  + Implements indexing for fast search and retrieval of flight data.

**4. Setup Instructions**

**Prerequisites**

1. Install [Node.js](https://nodejs.org/) (LTS version recommended).
2. Install [MongoDB](https://www.mongodb.com/try/download/community) and ensure it is running locally or have access to a cloud database (e.g., MongoDB Atlas).
3. Install [Git](https://git-scm.com/).
4. Ensure you have a modern web browser (e.g., Google Chrome).
5. (Optional) Install a package manager like [Yarn](https://yarnpkg.com/) if preferred over npm.
   1. **Clone the Repository**

bash

Copy code

git clone <repository-url>

cd flight-ticket-booking-system

* 1. **Install Backend Dependencies**

bash

Copy code

cd server

npm install

* 1. Install Frontend Dependencies

bash

Copy code

cd client

npm install

4. **Configure Environment Variables**

**Backend**:

PORT=5000

MONGO\_URI=<your-mongodb-connection-string>

JWT\_SECRET=<your-secret-key>

PAYMENT\_API\_KEY=<payment-gateway-api-key> (if applicable)

Frontend:

* + In the frontend folder, create a .env file with:

REACT\_APP\_API\_URL=http://localhost:5000

**5 . Access the App**

**5. Access the Application**

* Open your web browser and navigate to:  
  http://localhost:3000 (Front-end)  
  http://localhost:5000 (Back-end API)

**5. Folder Structure**

**1. Client (React Frontend)**

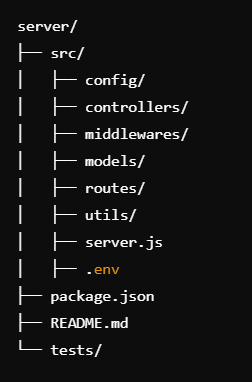
The React frontend is organized as follows:



* The components/ folder contains reusable UI elements like buttons, forms, or headers.
* The pages/ folder includes specific pages, such as HomePage.js, BookingPage.js, and Paymentgate.js.
* API calls and utilities are abstracted into the utils/ folder.
* The context/ folder manages global states, such as the Flight booking website or authentication state.

**2. Server:**

The Node.js backend is structured as follows:



**6. Running the Application**

**Start MongoDB**:

Start MongoDB locally or ensure your MongoDB Atlas instance is running.

**Start Backend**:

From the backend folder

npm start

**Start Frontend**:

From the frontend folder

npm start

**7. API Documentation**

**1. User Authentication**

**1.1 Register a User**

**Method**: POST  
**Endpoint**: /api/auth/register  
**Description**: Registers a new user.

**Request Body**:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "securepassword"

}

**Response**:

json

Copy code

{

"success": true,

"message": "User registered successfully."

}

**POST /api/auth/login**

* **Description**: Logs in an existing user and provides a JWT token.
* **Request Method**: POST
* **Request Body**:

json

Copy code

{

"email": "john.doe@example.com",

"password": "password123"

}

* + email: (string) The user's email.
  + password: (string) The user's password.
* **Response**:

**200 OK**:

json

Copy code

{

"token": "your\_jwt\_token\_here"

}

**400 Bad Request**: Invalid credentials.

json

Copy code

{

"error": "Invalid email or password."

}

**1.2 Login a User**

**Method**: POST  
**Endpoint**: /api/auth/login  
**Description**: Authenticates a user and returns a JWT token.

**Request Body**:

{

"email": "john.doe@example.com",

"password": "securepassword"

}

**Response**:

{

"success": true,

"token": "eyJhbGciOiJIUzI1NiIsInR5..."

}

**2. Flights**

**2.1 Search Flights**

**Method**: GET  
**Endpoint**: /api/flights/search  
**Description**: Searches for available flights based on filters.

**Query Parameters**:

* from: Departure location (e.g., "JFK")
* to: Destination location (e.g., "LAX")
* departureDate: Date of departure (e.g., "2024-11-20")
* returnDate (optional): Return date (e.g., "2024-11-25")
* passengers: Number of passengers (e.g., 2)

**Example Request**:

http

Copy code

GET /api/flights/search?from=JFK&to=LAX&departureDate=2024-11-20&passengers=2

**Response**:

json

Copy code

[

{

"id": "12345",

"airline": "Airline Name",

"from": "JFK",

"to": "LAX",

"departureTime": "2024-11-20T08:00:00Z",

"arrivalTime": "2024-11-20T12:00:00Z",

"price": 250.00,

"seatsAvailable": 10

}

]

**GET /api/auth/me**

* **Description**: Retrieves the current logged-in user's profile using the provided JWT token.
* **Request Method**: GET
* **Headers**:
  + Authorization: Bearer <JWT\_TOKEN>
* **Response**:

**200 OK**:

json

Copy code

{

"id": "123456",

"name": "John Doe",

"email": "john.doe@example.com"

}

**401 Unauthorized**: Token is missing or invalid.

json

Copy code

{

"error": "No token provided. Authorization denied."

}

**2.2 Get Flight Details**

**Method**: GET  
**Endpoint**: /api/flights/:id  
**Description**: Retrieves detailed information about a specific flight.

**Path Parameter**:

* id: Flight ID

**Response**:

json

Copy code

{

"id": "12345",

"airline": "Airline Name",

"from": "JFK",

"to": "LAX",

"departureTime": "2024-11-20T08:00:00Z",

"arrivalTime": "2024-11-20T12:00:00Z",

"price": 250.00,

"seatsAvailable": 10,

"baggageAllowance": "20kg",

"flightDuration": "4 hours"

}

**3. Booking**

**3.1 Book a Flight**

**Method**: POST  
**Endpoint**: /api/bookings  
**Description**: Creates a booking for a flight.

**Request Body**:

json

Copy code

{

"userId": "67890",

"flightId": "12345",

"passengers": [

{

"name": "John Doe",

"age": 30,

"passportNumber": "A12345678"

}

]

}

**Response**:

json

Copy code

{

"success": true,

"bookingId": "98765",

"message": "Flight booked successfully."

}

**3.2 Get User Bookings**

**Method**: GET  
**Endpoint**: /api/bookings/user/:userId  
**Description**: Retrieves all bookings made by a specific user.

**Path Parameter**:

* userId: User ID

**Response**:

json

Copy code

[

{

"bookingId": "98765",

"flightId": "12345",

"status": "Confirmed",

"passengers": [

{

"name": "John Doe",

"age": 30

}

],

"totalPrice": 250.00,

"bookingDate": "2024-11-15"

}

]

**4. Admin**

**4.1 Add a Flight**

**Method**: POST  
**Endpoint**: /api/admin/flights  
**Description**: Adds a new flight to the system (admin only).

**Request Body**:

json

Copy code

{

"airline": "Airline Name",

"from": "JFK",

"to": "LAX",

"departureTime": "2024-11-20T08:00:00Z",

"arrivalTime": "2024-11-20T12:00:00Z",

"price": 250.00,

"seatsAvailable": 50,

"baggageAllowance": "20kg"

}

**Response**:

json

Copy code

{

"success": true,

"message": "Flight added successfully."

}

**4.2 Delete a Flight**

**Method**: DELETE  
**Endpoint**: /api/admin/flights/:id  
**Description**: Deletes a flight from the system (admin only).

**Path Parameter**:

* id: Flight ID

**Response**:

json

Copy code

{

"success": true,

"message": "Flight deleted successfully."

}

**8. Authentication**

1. User Registration

New users can create an account by providing their name, email, and a secure password. The system ensures:

* Validation of user details.
* Prevention of duplicate email registrations.
* Secure storage of passwords using encryption techniques.

2. User Login

* Registered users can log in by providing their email and password. The system authenticates the user and, upon success, generates a **JWT token** that allows the user to access protected features.

3. Authentication Middleware

To secure backend endpoints, middleware is implemented to verify the JWT token included in the user’s request.

* If the token is valid, the user can proceed.
* If the token is invalid or missing, access is denied with an error message.

### **4. Token Management**

The JWT token includes user-specific data such as user ID and role. Tokens have a limited validity period to enhance security. Users must log in again when the token expires.

5. Password Security

Passwords are never stored in plain text. They are encrypted before storage to protect user data, even in the event of a breach. During login, the system compares the encrypted form of the entered password with the stored one.

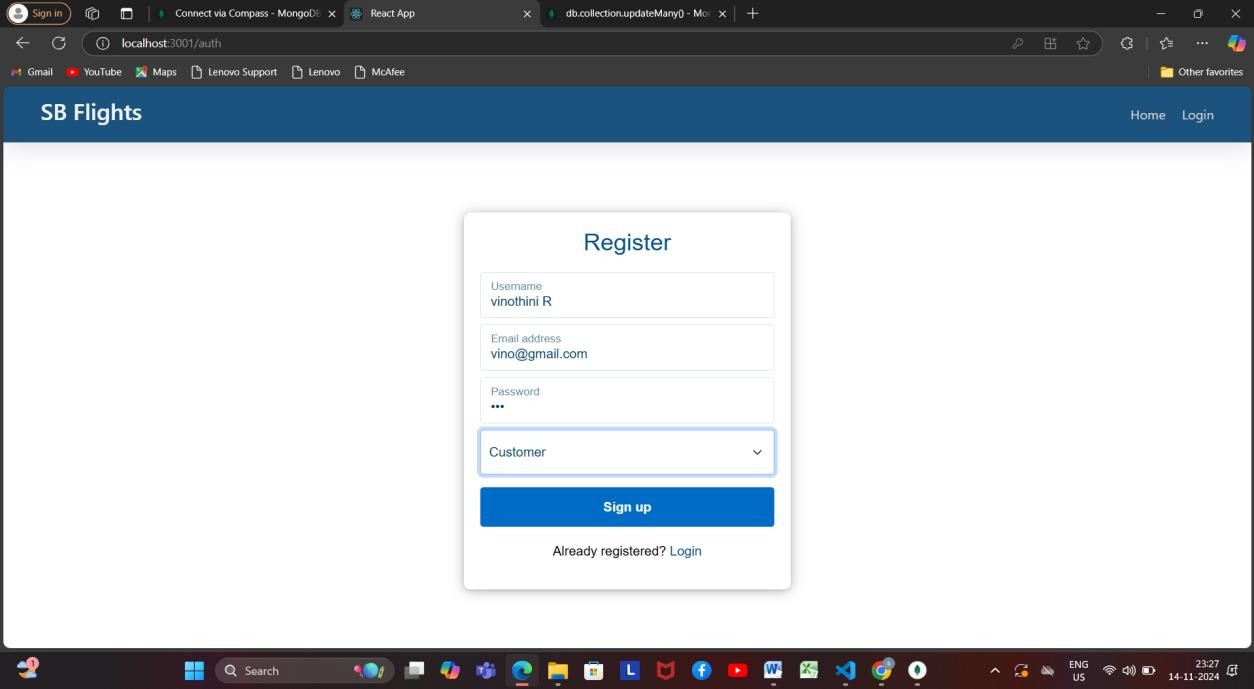
6. Protected Routes

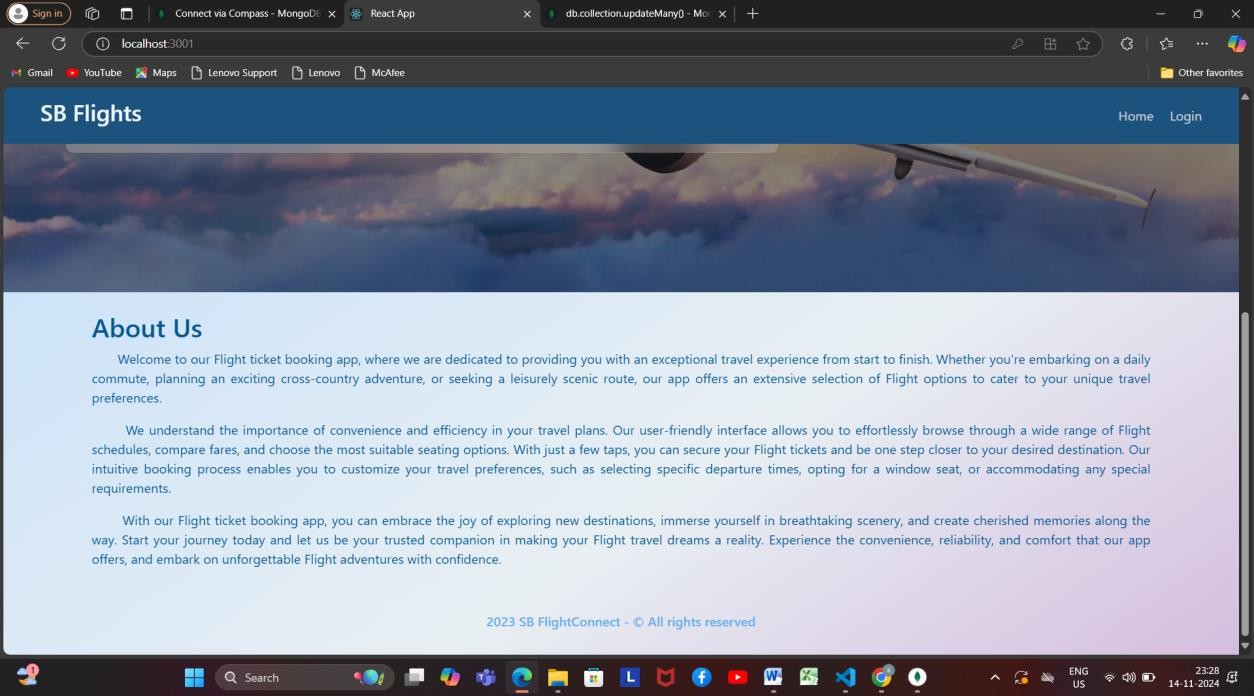
Certain functionalities, like viewing bookings or managing flights, require users to be authenticated. These routes check for the presence of a valid token before granting access.

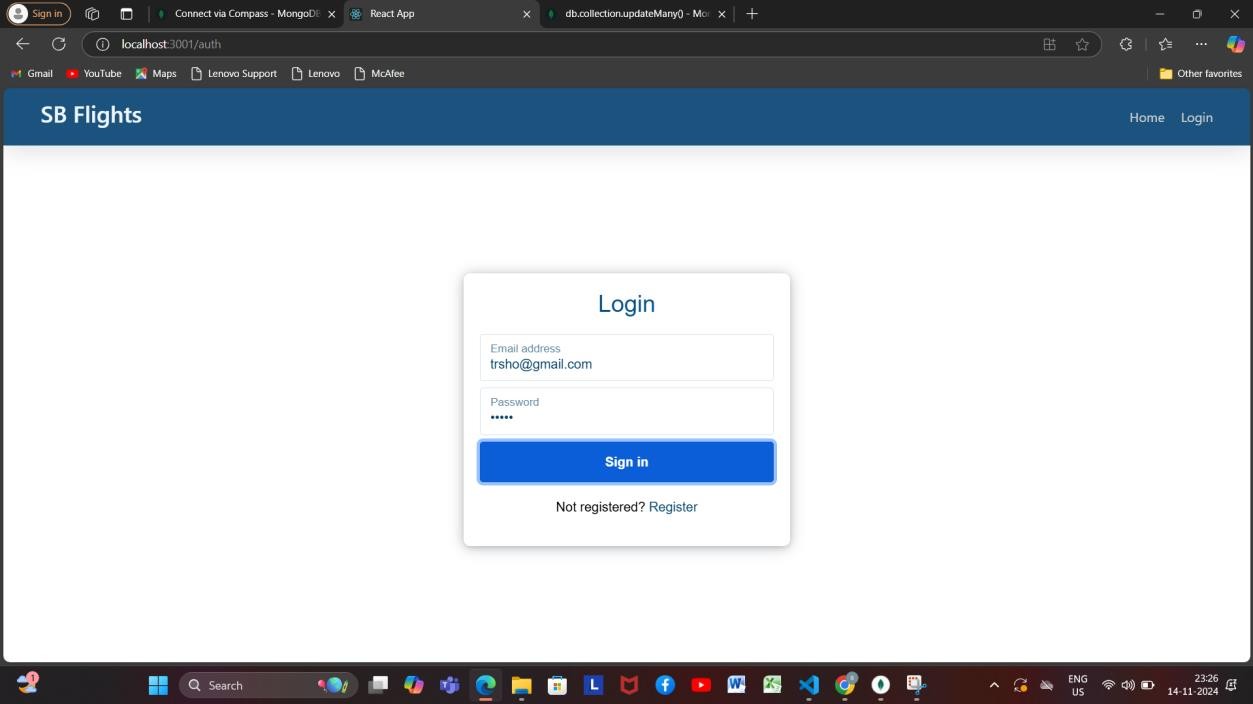
7. Security Practices

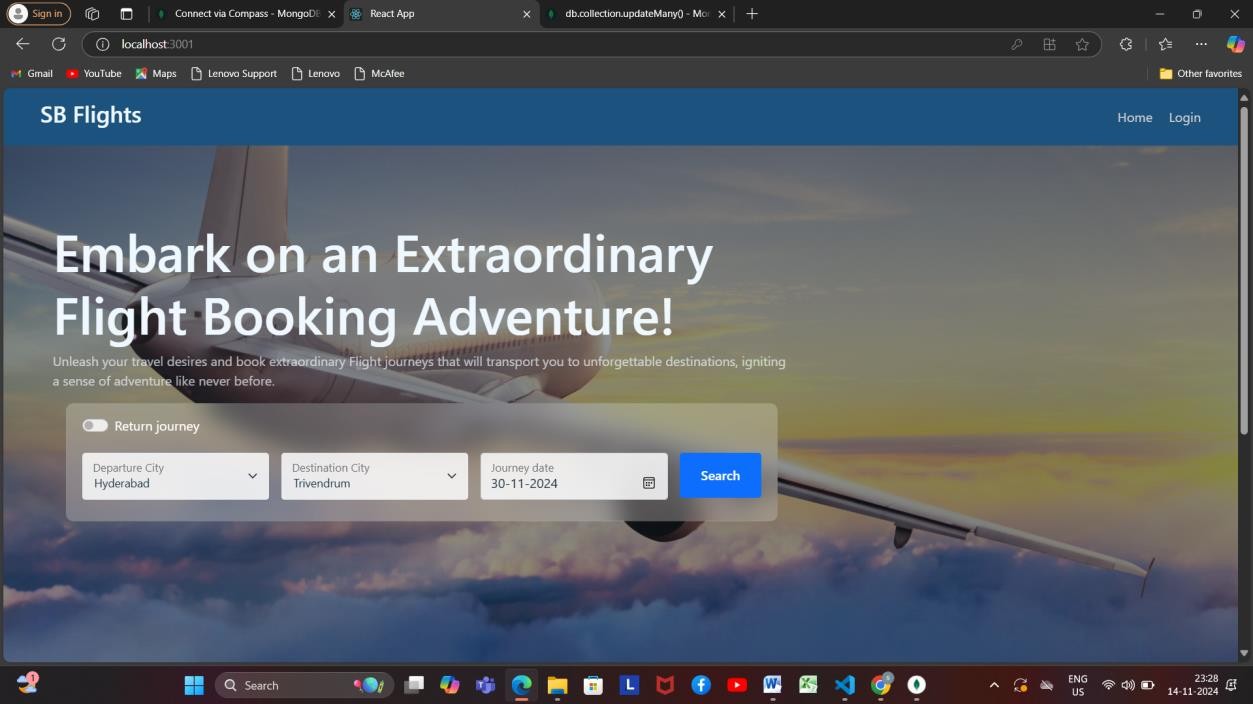
* Tokens are passed securely via headers in API requests.
* HTTPS is used to encrypt data transmission.
* Authentication errors provide generic responses to avoid exposing sensitive information.

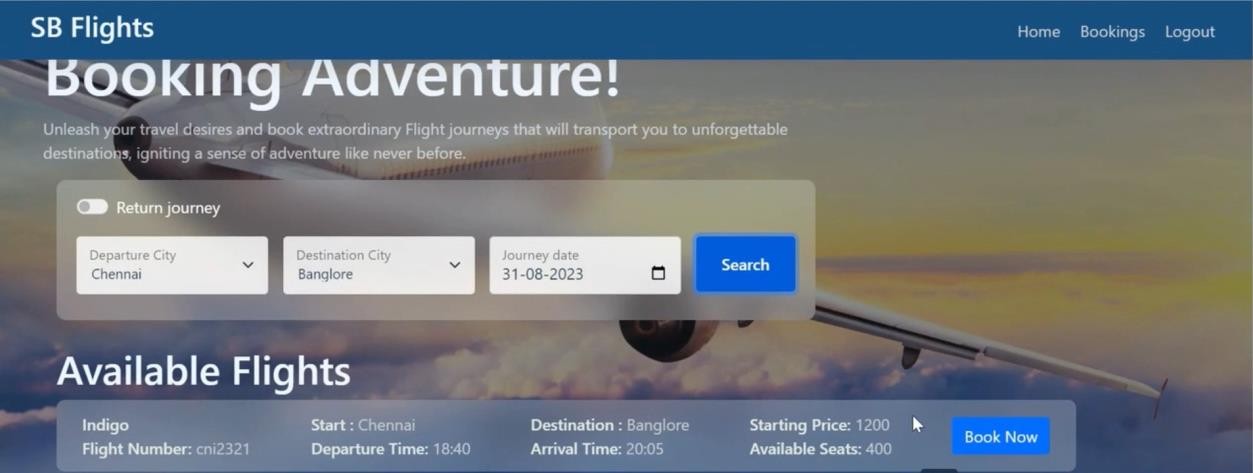
**9. User Interface**

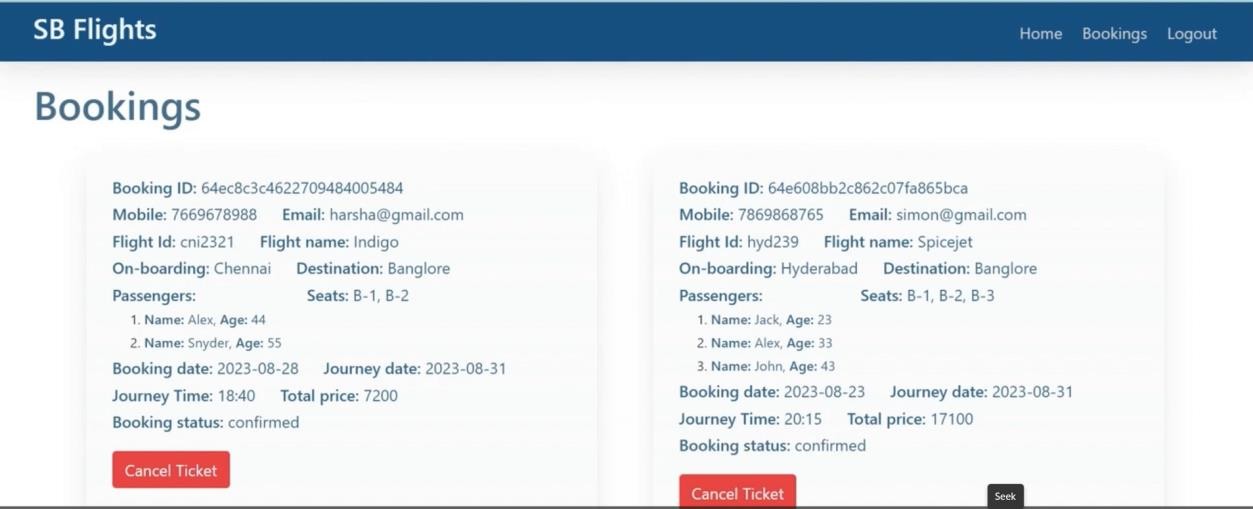


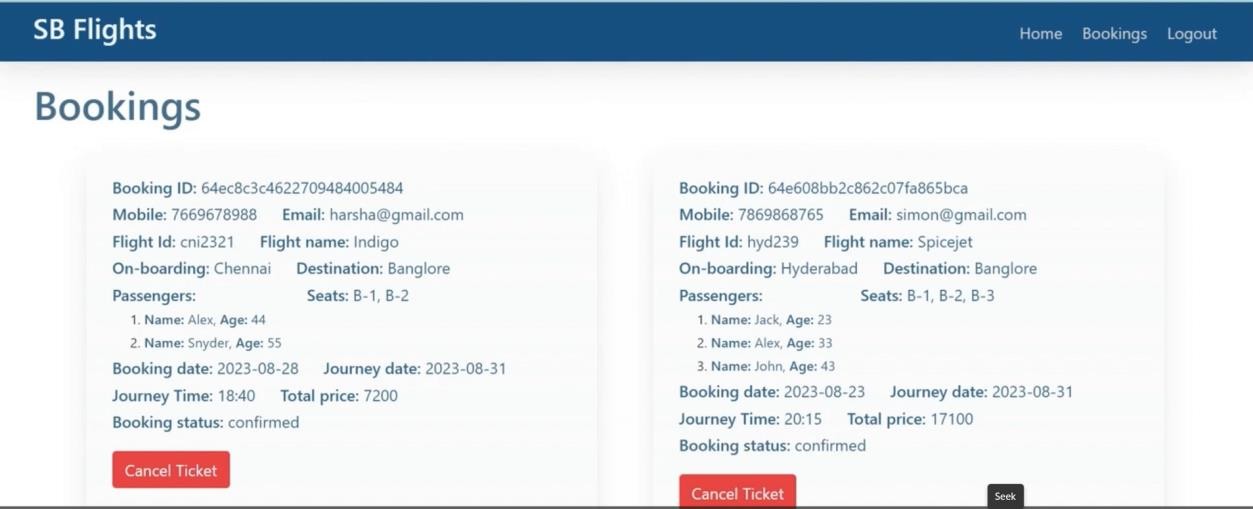


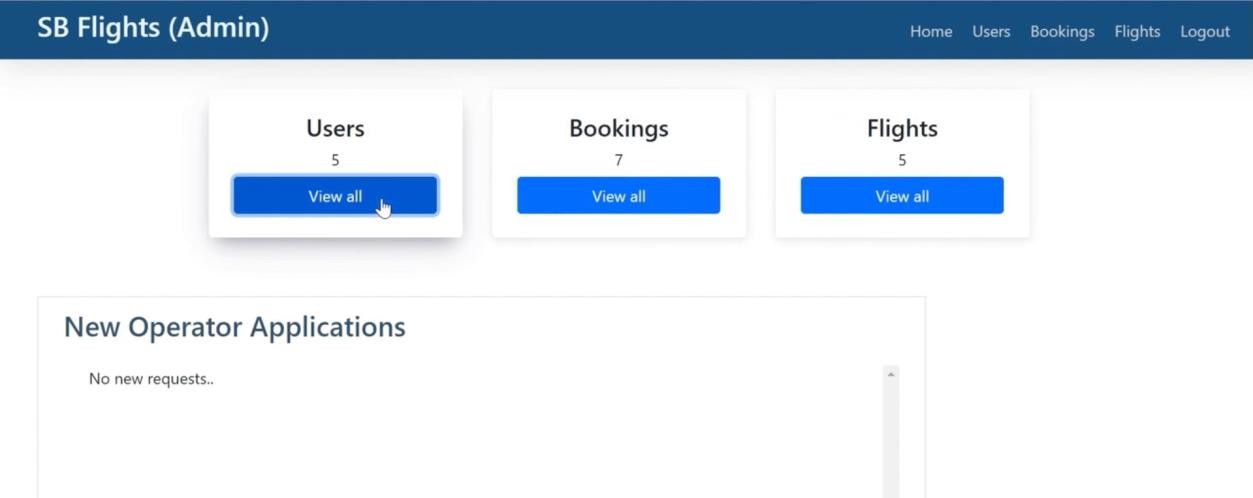


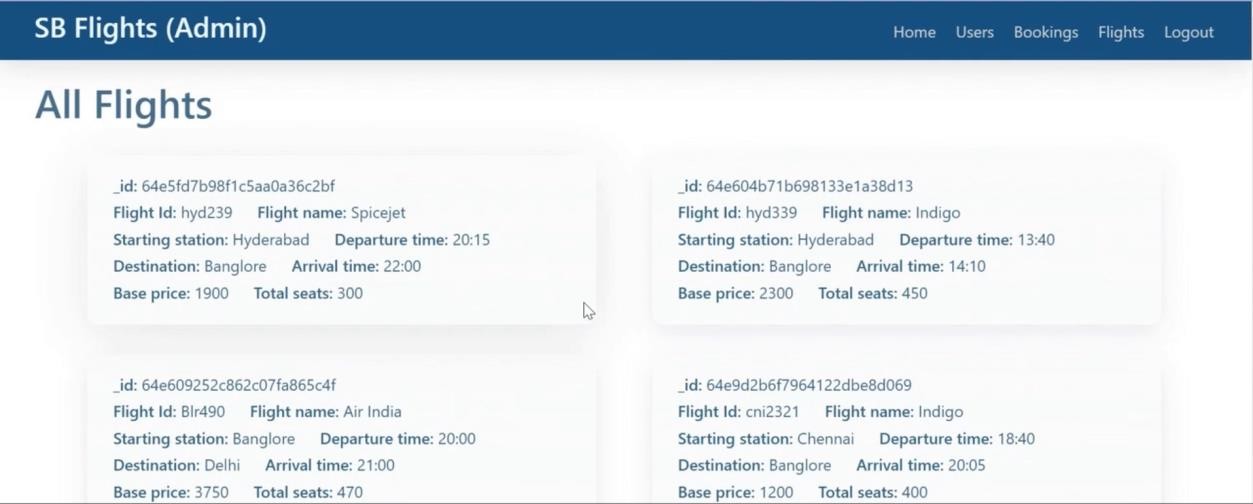


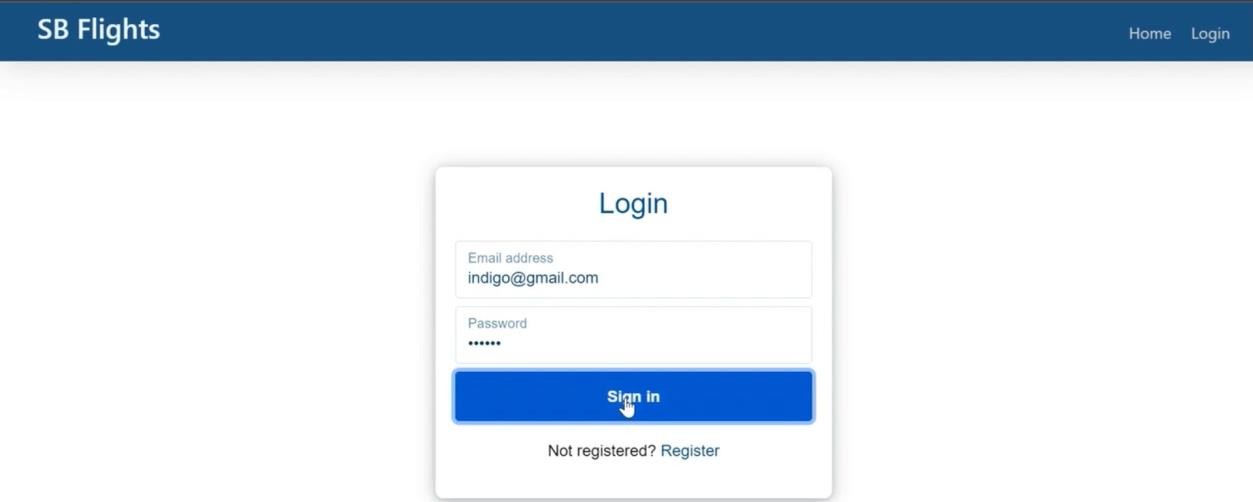


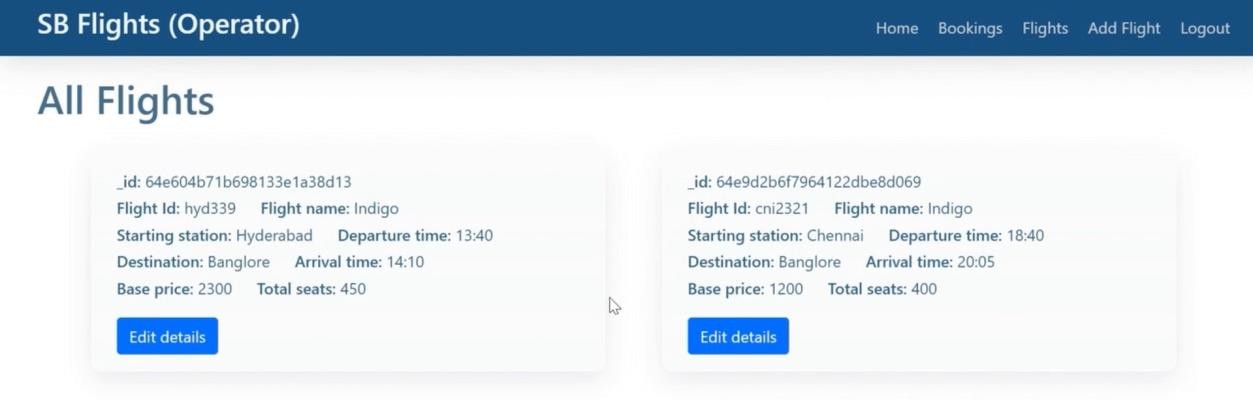


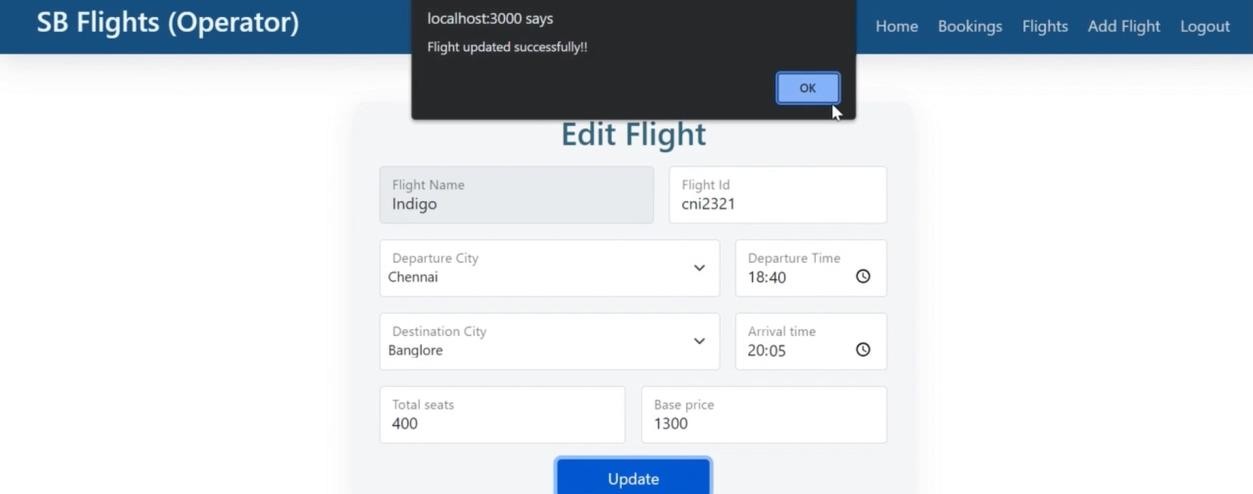


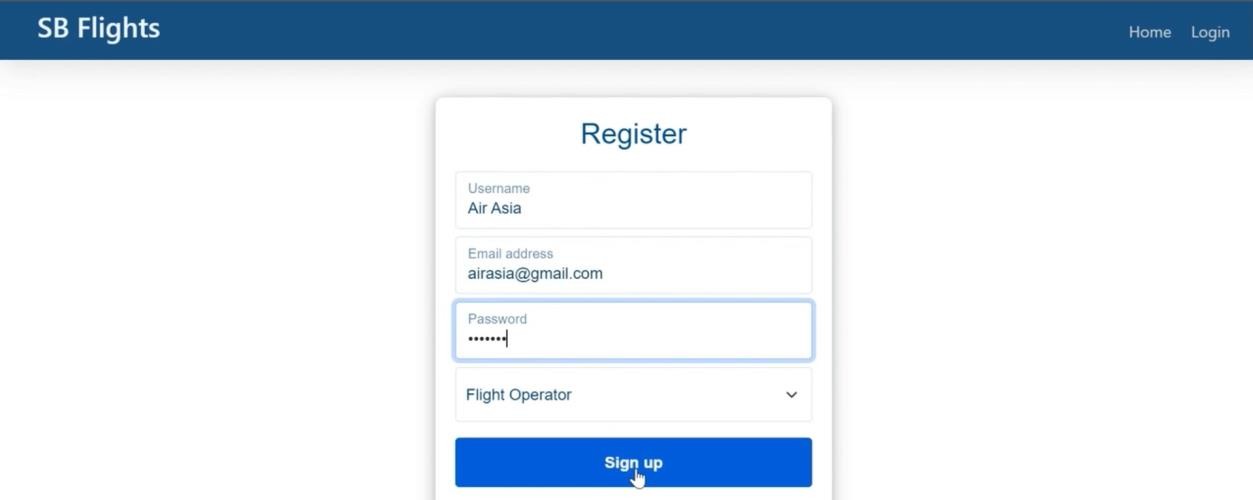
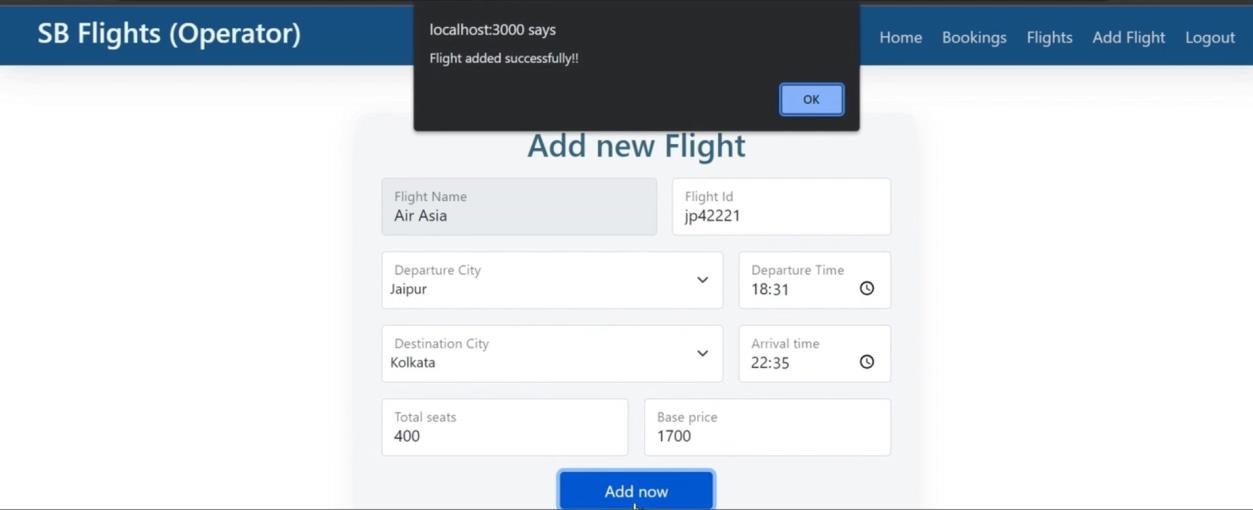
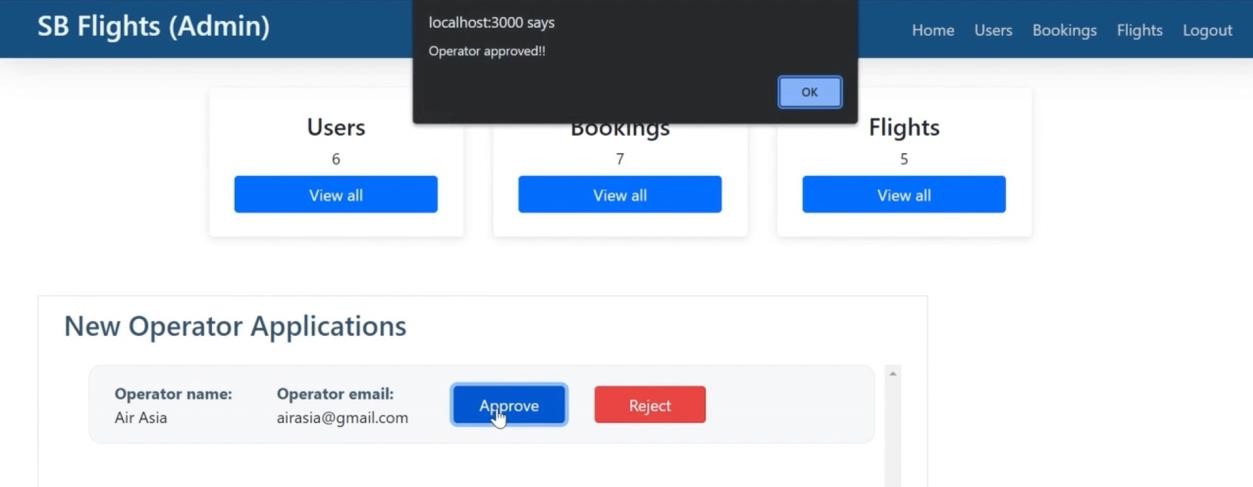
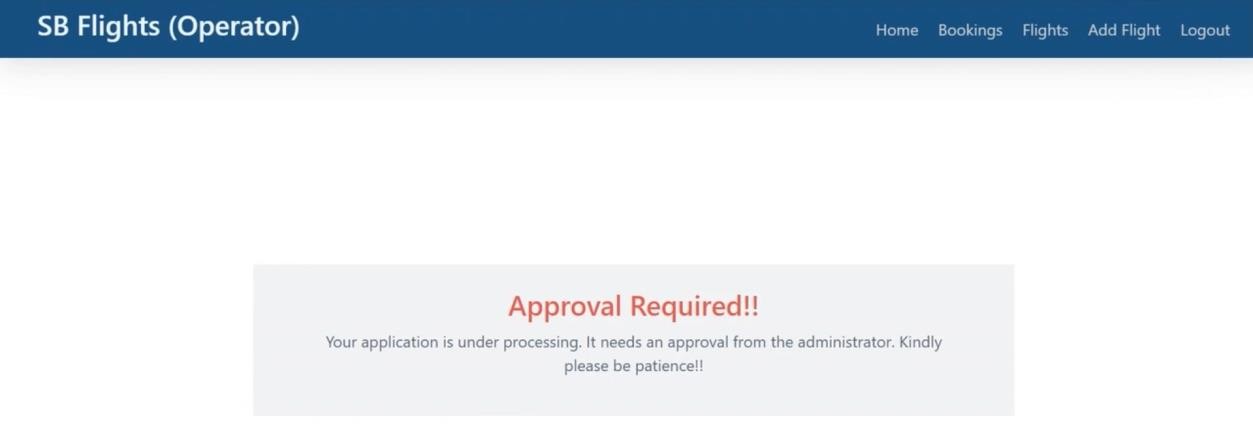












**11. Screenshots or Demo**

[**https://drive.google.com/drive/folders/1xJjbDr8GeL1zn6-TK4KeZXQCiFxtX6LD?usp=sharing**](https://drive.google.com/drive/folders/1xJjbDr8GeL1zn6-TK4KeZXQCiFxtX6LD?usp=sharing)

**12. Future Enhancements**

To continually improve the functionality, user experience, and scalability of the Flight Ticket Booking , the following enhancements are planned:

**1. Mobile Application**

* Develop native mobile applications for **iOS** and **Android** using technologies like **React Native** or **Flutter**.
* Include push notifications for order updates, promotional offers, and reminders for reordering frequently purchased items.

**2. Subscription and Loyalty Programs**

* Introduce a subscription model for regular customers (e.g., weekly or monthly deliveries for essentials).
* Implement a loyalty rewards system:
  + Earn points for every purchase.
  + Redeem points for discounts or free products.

**3. Voice Search and Assistant Integration**

* Add voice search functionality to enable users to find products using voice commands.
* Integrate with virtual assistants like **Google Assistant** or **Alexa** for hands-free shopping and order tracking.

**4. Enhanced Analytics for Admins**

* Develop a dashboard with advanced analytics to help administrators track:
  + Sales trends.
  + Inventory levels.
  + Customer behavior and preferences.
* Use predictive analytics to forecast demand and optimize inventory management