Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** FLIGHT BOOKING APPLICATION
- **Team Members:**
  - **Sai Prasana R - Project Manager (Team Lead)**
  - **Subash Chandar S - Frontend Developer**
  - **Praveen Kumar T - Backend Developer**
  - **Suman V - Database**

2. Project Overview

- **Purpose:**
  - A flight booking app is a mobile or web application that allows users to search for, book, and manage their flight reservations. These apps typically provide features for comparing flight prices, choosing seating preferences, booking tickets, managing itineraries, and tracking flight statuses.
- **Features:**
  - **Flight Search & Filters**:
    - Search flights by origin, destination, date, and number of passengers.
  - **Real-time Pricing & Availability**:
    - Display up-to-date prices and availability for various airlines.
  - **Booking & Payment**:
    - Secure payment gateway integration (credit card, PayPal, etc.).
  - **Flight Information**:
    - Detailed flight information, including flight numbers, departure/arrival times, layover details, and terminal information.
  - **Booking Management**:
    - Option to modify or cancel bookings directly through the app.
    - Option to select seats, add baggage, and upgrade to business or first class.
  - **User Accounts & Profiles**:
    - Users can create accounts, saving frequent flyer details, travel preferences, and payment methods for a smoother experience.
  - **Review & Ratings**:
    - Users can leave reviews for airlines, airports, and flights.

3. Architecture

❖ Frontend:

- The frontend architecture of this flight booking application is structured using the React library, following a modular, component-based design approach to enhance maintainability and scalability. At the core, it utilizes functional components with React hooks like useState and useEffect for state management and side effects. The application is segmented into multiple pages and features, each encapsulated within its respective components, ensuring separation of concerns.
- **Routing & Navigation**
  - The app leverages React Router for navigation between different views, such as the Admin Dashboard, User Management, Authentication, Flight Management, and Booking views.
- **Data Handling & API Integration**
  - The frontend communicates with a backend server using Axios for HTTP requests. It fetches data from various endpoints (like fetch-users, fetch-bookings, and fetch-flights) to dynamically render content.
- **Component Hierarchy**

  The main components include:
  - Admin Dashboard: Displays cards for user, booking, and flight counts. It also handles new operator approval and rejection requests using dedicated API calls.
  - All Users: Lists all registered users and segregates them by user types (customers vs. flight operators) for easy management.
  - Authenticate: Toggles between login and registration views to handle user authentication.
  - Flight Admin: Provides a dashboard for flight operators, showing their bookings, flights, and allowing new flight additions.
  - Flight Bookings: Displays flight-specific bookings for operators, with features to cancel bookings.
- **Styling & Layout**
  - Styling is managed using CSS modules (e.g., Admin.css, allUsers.css, FlightAdmin.css), ensuring styles are scoped to individual components, thus avoiding global namespace pollution.
- **State Management & User Feedback**
  - Local component state is extensively used to manage user interactions and application state (like tracking approval statuses, counts, and form inputs).

❖ Backend:

- The backend architecture for the Flight Booking application is built using the MERN stack (MongoDB, Express.js, React, and Node.js). It leverages Express.js as the web server framework to handle HTTP requests and MongoDB as the database for storing application data, with Mongoose used for object data modeling (ODM).

- **Authentication and Authorization:**
  - The backend includes user authentication using bcrypt for password hashing. Users can register and log in, with different types of users (e.g., regular users and flight operators).
- **CRUD Operations and API Endpoints:**
  - User Management: There are endpoints for user registration, login, fetching individual user details, and listing all users.
  - Flight Management: The system supports adding, updating, and retrieving flight details. Flights can be fetched individually or as a list.
  - Booking System: Users can book flights, with the system dynamically assigning seat numbers based on the class (e.g., economy, business). It tracks the number of seats booked and generates seat identifiers accordingly. Bookings are stored in the database, with the ability to fetch all bookings or cancel them via specific endpoints.
- **Database Structure:**
  - The application uses three main schemas User, Flight, and Booking managed through Mongoose models
- **Error Handling and Logging:**
  - Error handling is incorporated throughout the routes, with error messages and status codes returned for invalid operations or server issues.

❖ Database:

- **Flight Schema**
  - The Flight collection is used to store information about all the flights managed by different operators.
  - The Flight collection contains detailed flight information, including the flight name, a unique flight ID, the origin and destination cities, departure and arrival times, base price, and the total number of seats available.
- **Booking Schema**
  - The Booking collection is crucial for managing passenger reservations.
  - The Booking collection records all flight reservations made by users.
- **Relationships & Structure**
  - Users are linked to Bookings through their unique user IDs, enabling the system to fetch all bookings made by a specific user.
  - Flights are associated with Bookings via the flight ID, which helps track which passengers are booked on specific flights.
  - Indexes on fields like email (in User) and flightId (in Flight) improve query performance, especially for searches and lookups.

- This database structure supports scalability and ensures efficient data retrieval, making it well-suited for a flight booking system with multiple users and operators.

4. Setup Instructions

❖ Prerequisites:

- **Node.js:** Version 14 or higher, required to run the backend and install packages.
- **MongoDB:** A local instance or cloud-based MongoDB (such as MongoDB Atlas) for storing project data.
- **npm:** Node package manager to handle dependencies for both frontend and backend.

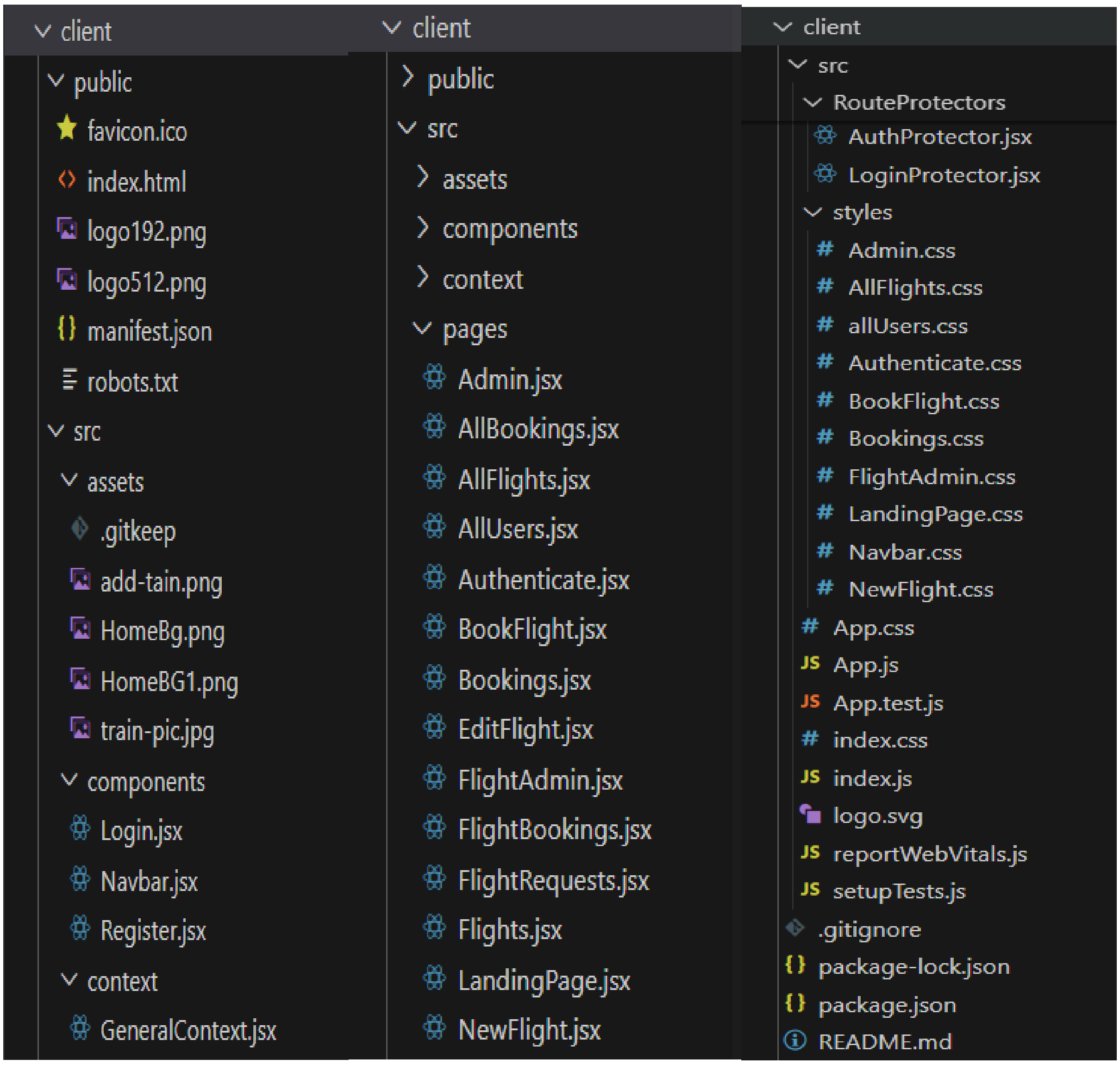
❖ Installation:

1. **Clone the repository:**
   - Clone the project repository to your local machine:
     - git clone
2. **Navigate to the project directory.**
3. **Install project dependencies:**
   - The project contains two main directories: client for the frontend and server for the backend. You will need to install dependencies for each separately.
   - First, install dependencies for the frontend:
     - cd client
     - npm install
   - Then, navigate to the server directory and install backend dependencies:
     - cd server
     - npm install
   - Set up environment variables for database configuration.
4. **Run the application:**
   - Start the backend server:
     - cd server
     - node index.js
   - In a new terminal, start the frontend:
     - cd client
     - npm start
   - The server will be accessible on http://localhost:6001 and the client on http://localhost:3000.

5. Folder Structure

❖ **Client:**

- **node_modules:** Contains all the npm packages required for the frontend project.
- **src/components:** Contains reusable UI components like Login, Navbar, and Register.
- **src/context:** Includes GeneralContext.js for shared state management.
- **src/assets:** Directory in a project where static files like images, fonts, and stylesheets are stored for use in the application.
- **src/routerprotector:** File or component used to guard routes in an app and redirecting unauthorized users to login.
- **src/pages:** Main application pages such as AdminProjects, Freelancer, Client, and Landing.
- **styles:** Contains CSS and styling files that ensure consistent design.



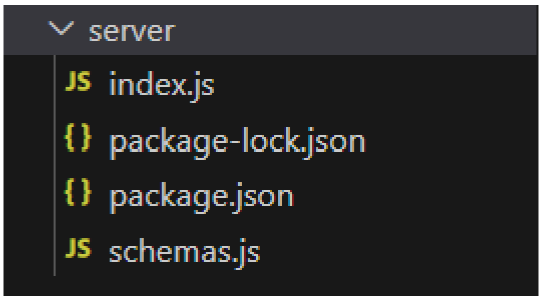
```
client
  public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
  src
    assets
      .gitkeep
      add-tain.png
      HomeBg.png
      HomeBG1.png
      train-pic.jpg
    components
      Login.jsx
      Navbar.jsx
      Register.jsx
    context
      GeneralContext.jsx
```

```
client
  public
  src
    assets
    components
    context
    pages
      Admin.jsx
      AllBookings.jsx
      AllFlights.jsx
      AllUsers.jsx
      Authenticate.jsx
      BookFlight.jsx
      Bookings.jsx
      EditFlight.jsx
      FlightAdmin.jsx
      FlightBookings.jsx
      FlightRequests.jsx
      Flights.jsx
      LandingPage.jsx
      NewFlight.jsx
```

```
client
  src
    RouteProtectors
      AuthProtector.jsx
      LoginProtector.jsx
    styles
      Admin.css
      AllFlights.css
      allUsers.css
      Authenticate.css
      BookFlight.css
      Bookings.css
      FlightAdmin.css
      LandingPage.css
      Navbar.css
      NewFlight.css
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

❖ **Server:**

- **index.js:** Server entry point that initializes Express and configures middleware.



```
server
  index.js
  package-lock.json
  package.json
  schemas.js
```

6. Running the Application

❖ **Frontend:**

- Run npm start within the client directory to launch the React app.
- The application will launch on a local development server, typically at http://localhost:3000.

❖ **Backend:**

- Run node index.js within the server directory to start the Express server.
- The backend runs on a different port http://localhost:6001.

7. API Documentation

❖ **User Authentication**

- POST /api/auth/register: Register a new user.
- POST /api/auth/login: Log in and get a JWT token.
- GET /api/auth/me: Get current user details (requires JWT).
- POST /api/auth/logout: Log out the user.

❖ **Flight Search & Booking**

- GET /api/flights/search: Search available flights.
- GET /api/flights/{flight_id}: Get detailed flight info.
- POST /api/bookings: Create a new booking.
- GET /api/bookings/{booking_id}: Get booking details.
- PUT /api/bookings/{booking_id}: Update an existing booking.
- DELETE /api/bookings/{booking_id}: Cancel a booking.

❖ **Payment**

- POST /api/payments: Process payment for a booking.
- GET /api/payments/{payment_id}: Get payment status.

- **Notifications**
  - GET /api/notifications: Get flight and booking notifications.

- **Admin Endpoints**
  - POST /api/admin/flights: Add a new flight.
  - PUT /api/admin/flights/{flight_id}: Update a flight.
  - DELETE /api/admin/flights/{flight_id}: Delete a flight.

8. Authentication

- **Authentication**:
  - Users log in with email and password; the server issues a **JWT token** that is stored on the client and sent with requests in the Authorization header.
- **Authorization**:
  - The backend checks the user's role from the JWT to determine access (e.g., only admin can manage flights).
- **Token Expiration**:
  - JWT tokens expire and can be refreshed using a **refresh token** for continued access.

9. User Interface

- **Login Screen**:
  - Users log in with email and password.
- **Signup Screen**:
  - Users register with name, email, and password.
- **Home/Search Screen**:
  - Users search for flights (departure/arrival cities, dates, passengers).
- **Flight Details Screen**:
  - Shows detailed flight info (times, price, seat options).
- **Booking Confirmation Screen**:
  - Confirms booking with flight details and booking ID.
- **User Profile Screen**:
  - Users can view/edit their profile and check booking history.
- **Admin Dashboard Screen**:
  - Admins manage flights, bookings, and view analytics.
- **Payment Screen**:
  - Users enter payment info to complete the booking.

10. Testing

- ### Unit Testing (Frontend):
  - **Tests**: Verify individual components (e.g., login form, flight search) work as expected.

- ### Unit Testing (Backend):
  - **Tests**: Ensure API endpoints (e.g., login, flight search) return correct data and status codes.

- ### Integration Testing:
  - **Tests**: Verify frontend and backend integration (e.g., flight search interacting with API).

- ### End-to-End Testing:
  - **Tests**: Simulate user flow from login to booking confirmation to ensure everything works together.

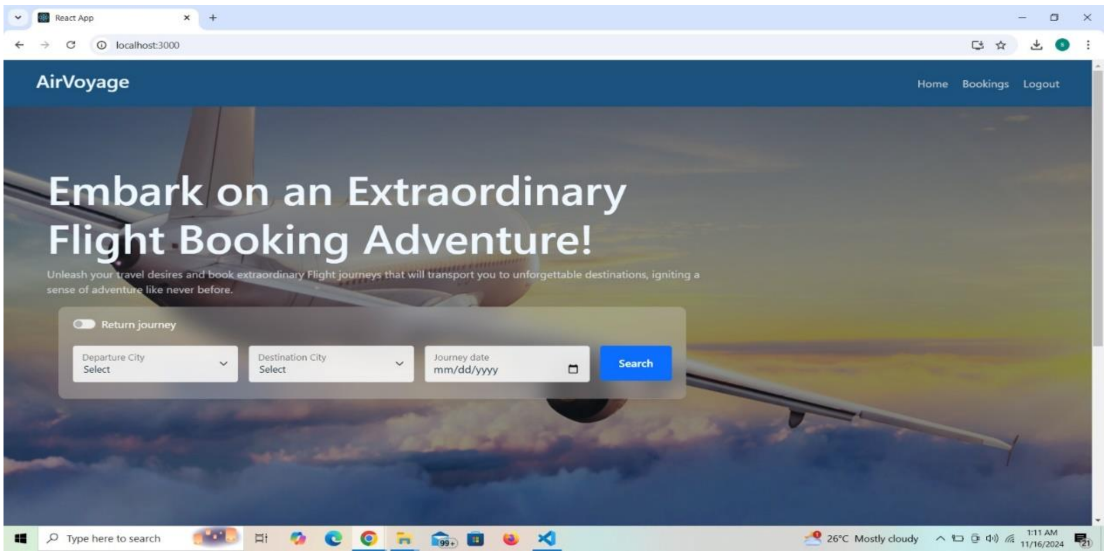
- ### UI Testing:
  - **Tests**: Verify the UI looks and works correctly on different devices and browsers.

- ### Performance Testing:
  - **Tests**: Ensure fast response times for API calls and good page load speed.

- ### Security Testing:
  - **Tests**: Check for vulnerabilities (e.g., token handling, input validation).

11. Screenshots or Demo

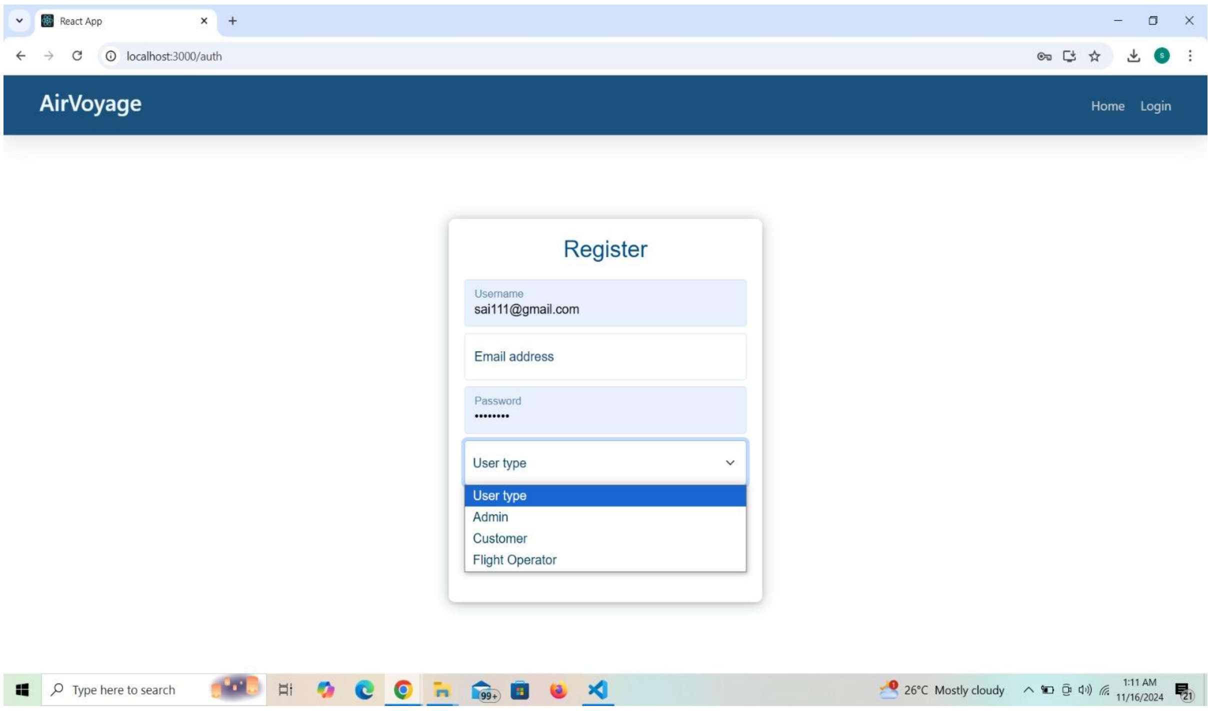
- ### Home Page



Embark on an Extraordinary Flight Booking Adventure!

Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.

❖ Register Page


React App

localhost:3000/auth

AirVoyage

Home Login

Register

Username
sai111@gmail.com

Email address

Password
••••••••

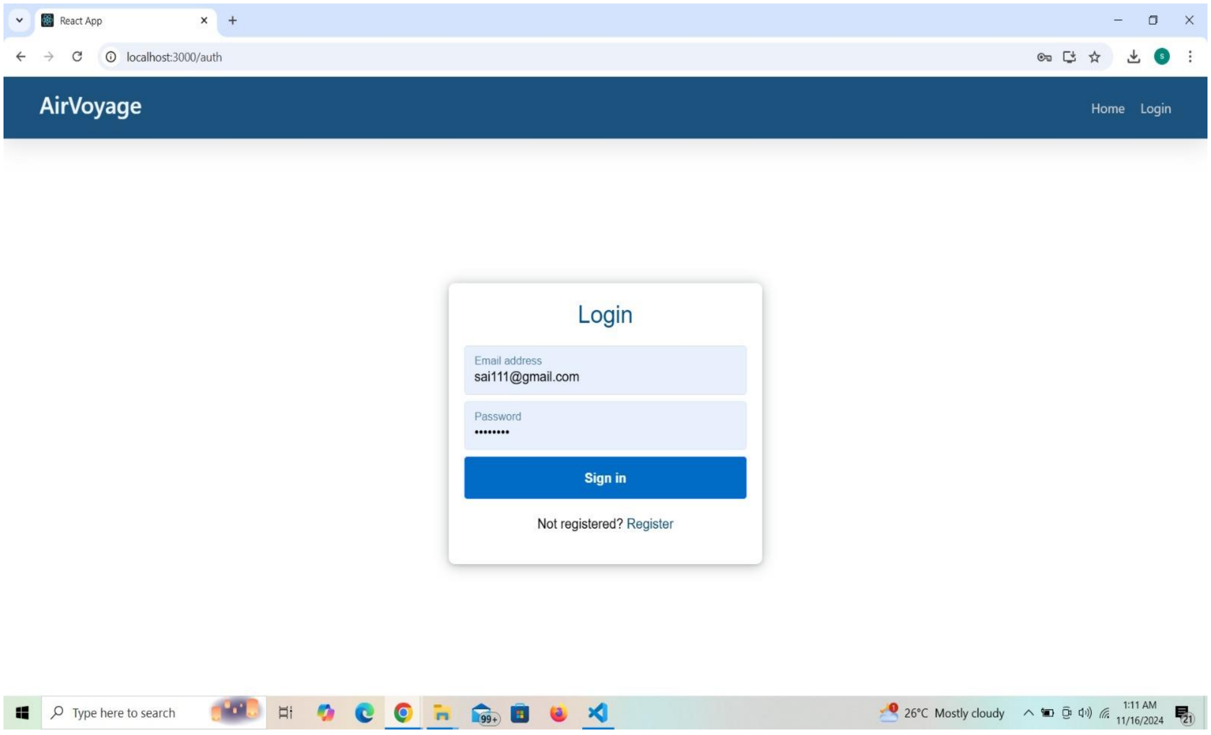
User type

User type
Admin
Customer
Flight Operator

Type here to search

26°C Mostly cloudy 1:11 AM 11/16/2024

❖ Login Page


React App

localhost:3000/auth

AirVoyage

Home Login

Login

Email address
sai111@gmail.com

Password
••••••••

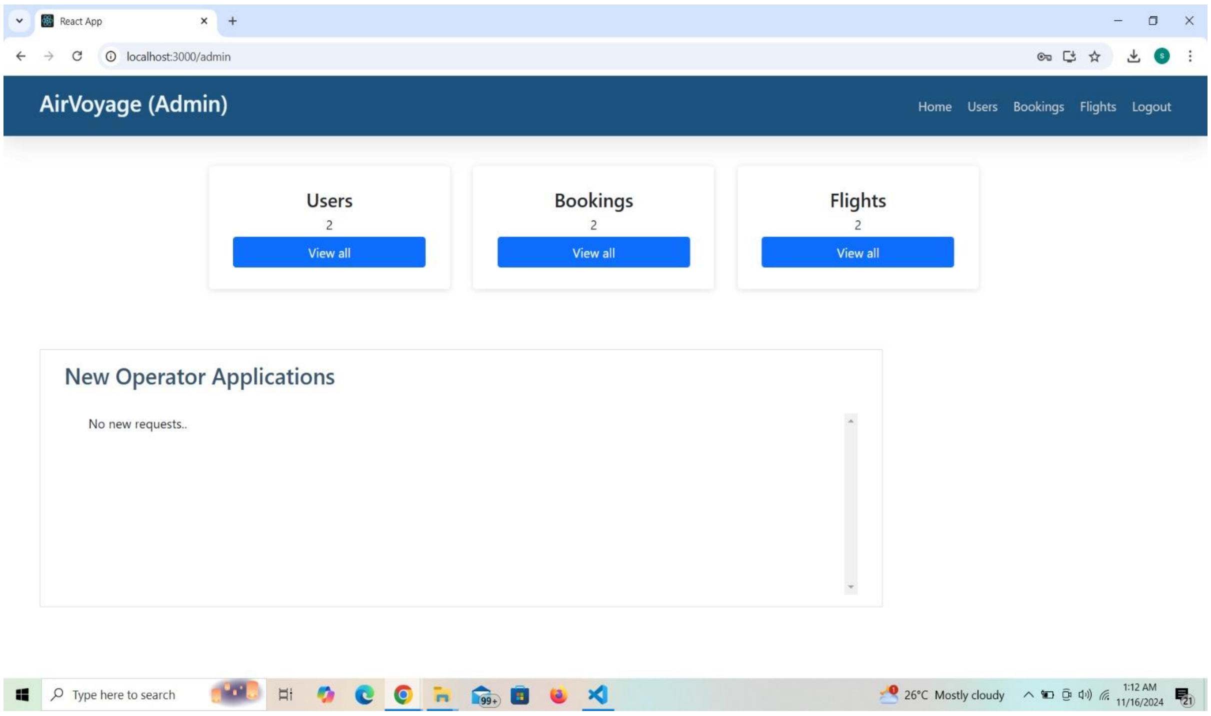
Sign in

Not registered? Register

Type here to search

26°C Mostly cloudy 1:11 AM 11/16/2024

❖ Admin’s Home page



Users
2

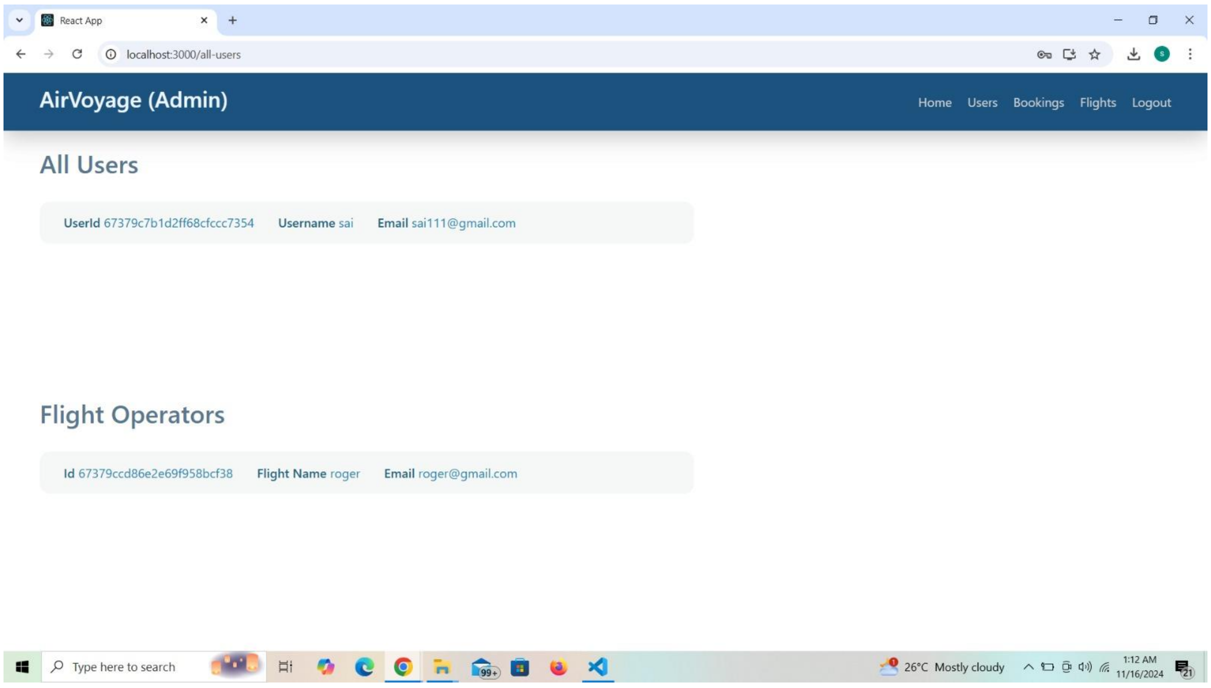
Bookings
2

Flights
2

New Operator Applications

No new requests..

❖ Admin’s Users Page



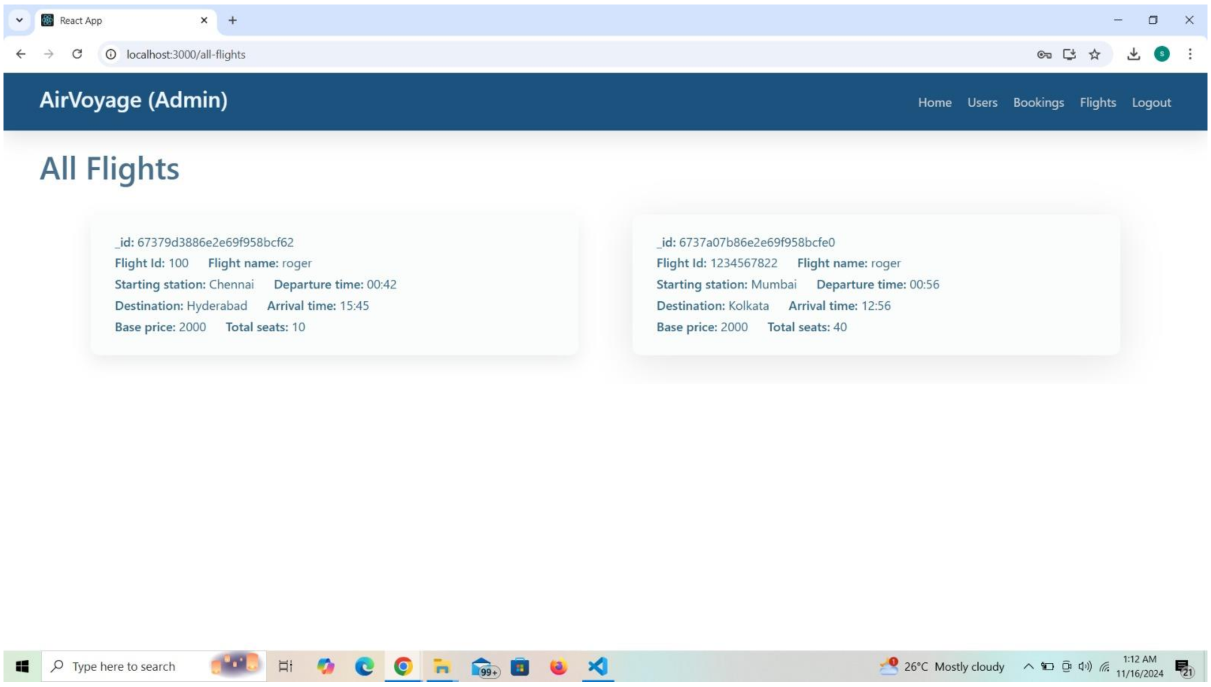
All Users

UserId 67379c7b1d2ff68cfccc7354 **Username** sai **Email** sai111@gmail.com

Flight Operators

Id 67379ccd86e2e69f958bcf38 **Flight Name** roger **Email** roger@gmail.com

❖ Admin’s All Flights Page


React App

localhost:3000/all-flights

AirVoyage (Admin)

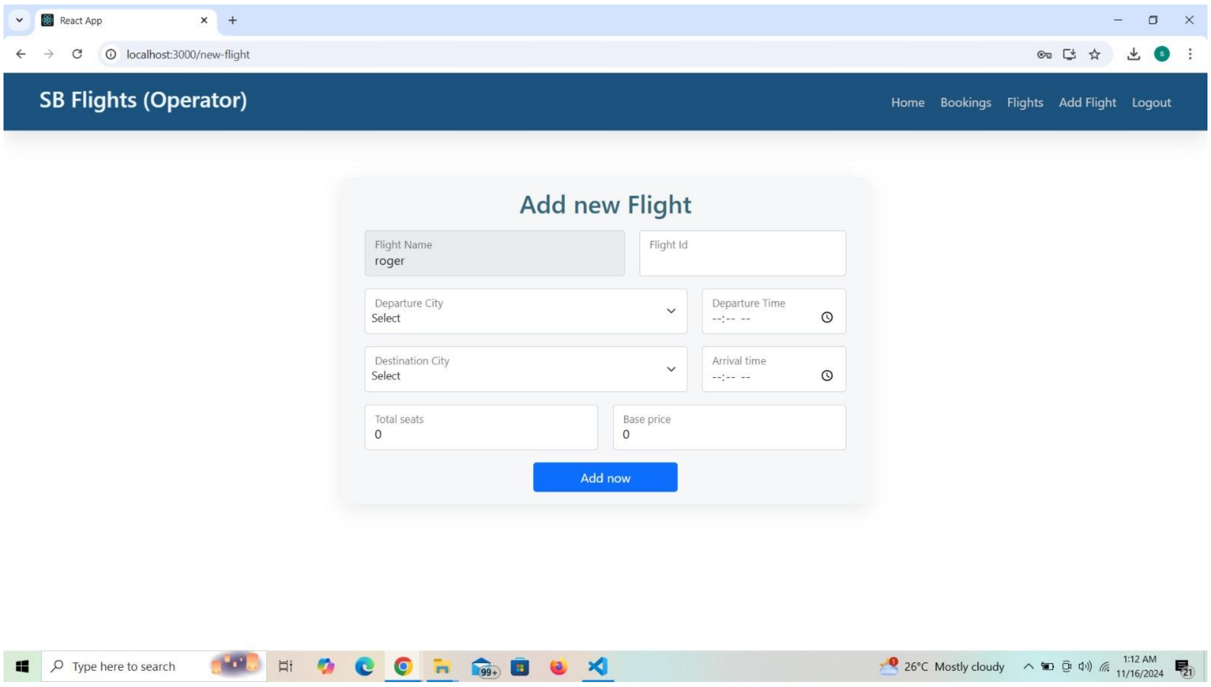
Home Users Bookings Flights Logout

All Flights

_id: 67379d3886e2e69f958bcf62
Flight Id: 100 Flight name: roger
Starting station: Chennai Departure time: 00:42
Destination: Hyderabad Arrival time: 15:45
Base price: 2000 Total seats: 10

_id: 6737a07b86e2e69f958bcfe0
Flight Id: 1234567822 Flight name: roger
Starting station: Mumbai Departure time: 00:56
Destination: Kolkata Arrival time: 12:56
Base price: 2000 Total seats: 40

❖ Operator’s Add Flight Page


React App

localhost:3000/new-flight

SB Flights (Operator)

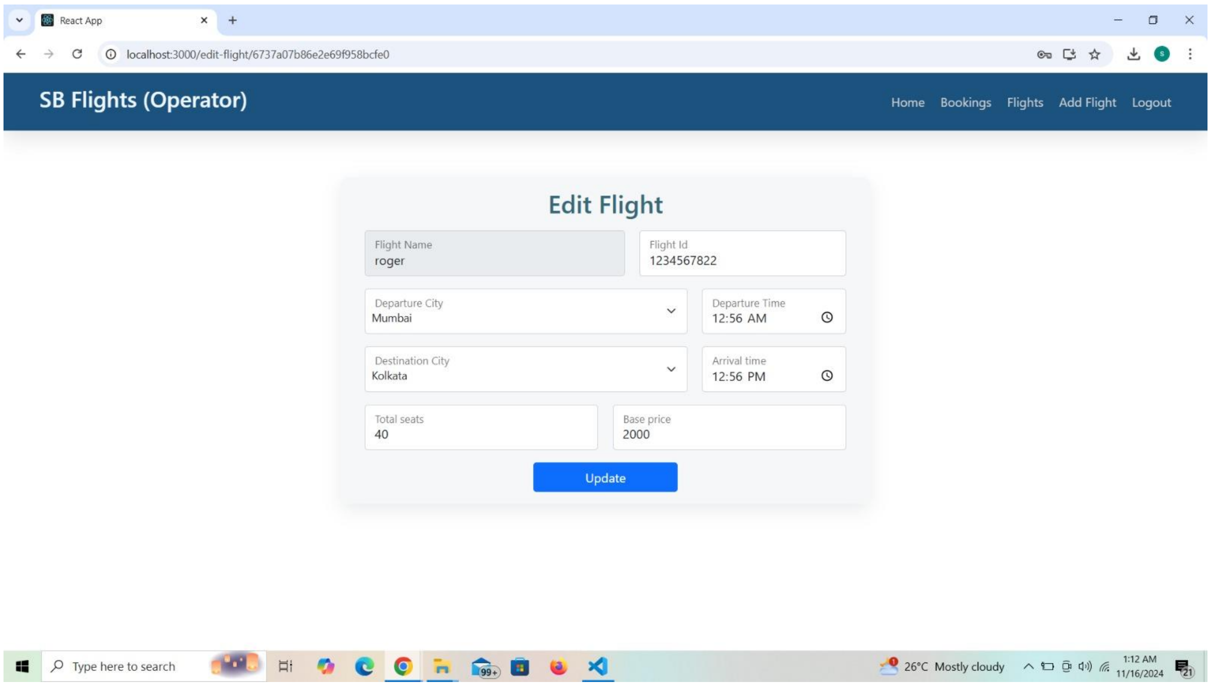
Home Bookings Flights Add Flight Logout

Add new Flight

Flight Name: roger
Flight Id:
Departure City: Select
Departure Time: --:-- --
Destination City: Select
Arrival time: --:-- --
Total seats: 0
Base price: 0

Add now

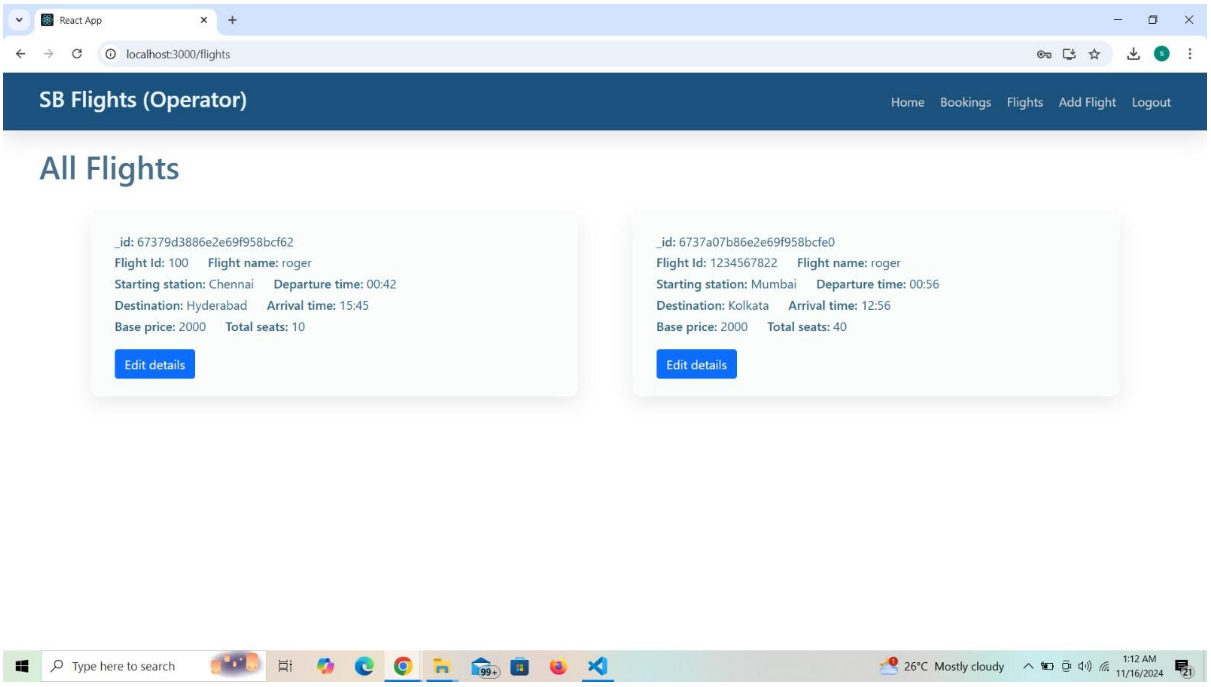
❖ Operator’s Edit Page


localhost:3000/edit-flight/6737a07b86e2e69f958bcfe0

Edit Flight

Flight Name: roger
Flight Id: 1234567822
Departure City: Mumbai
Departure Time: 12:56 AM
Destination City: Kolkata
Arrival time: 12:56 PM
Total seats: 40
Base price: 2000

❖ Operator’s Dashboard

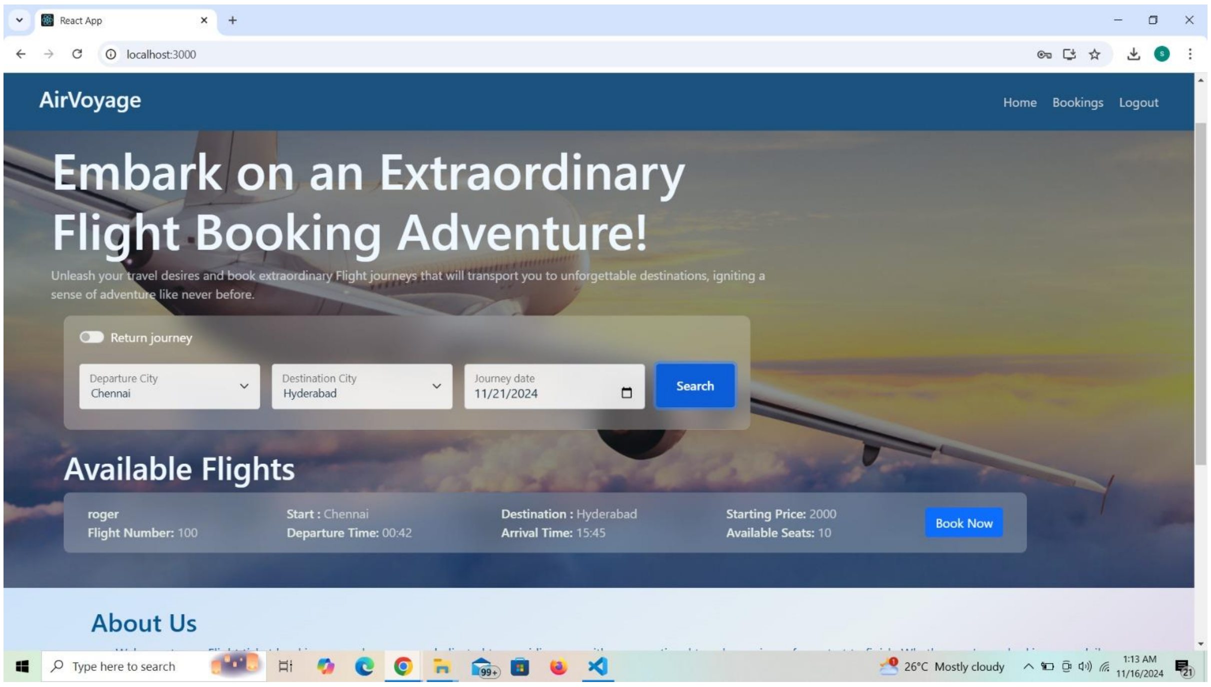

localhost:3000/flights

All Flights

_id: 67379d3886e2e69f958bcf62
Flight Id: 100 **Flight name:** roger
Starting station: Chennai **Departure time:** 00:42
Destination: Hyderabad **Arrival time:** 15:45
Base price: 2000 **Total seats:** 10

_id: 6737a07b86e2e69f958bcfe0
Flight Id: 1234567822 **Flight name:** roger
Starting station: Mumbai **Departure time:** 00:56
Destination: Kolkata **Arrival time:** 12:56
Base price: 2000 **Total seats:** 40

❖ Flight Search Page



Embark on an Extraordinary Flight Booking Adventure!

Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.

Return journey

Departure City: Chennai
Destination City: Hyderabad
Journey date: 11/21/2024

Search

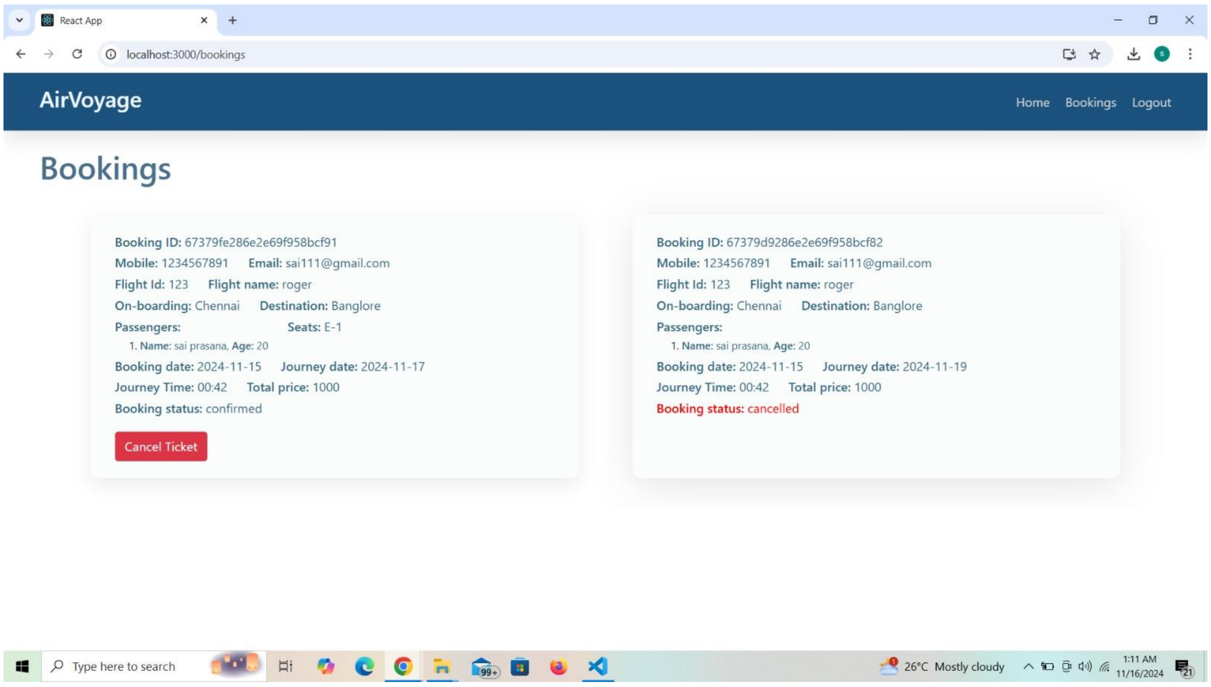
Available Flights

roger **Start :** Chennai **Destination :** Hyderabad **Starting Price:** 2000
Flight Number: 100 **Departure Time:** 00:42 **Arrival Time:** 15:45 **Available Seats:** 10

Book Now

About Us

❖ Booking Page



Bookings

Booking ID: 67379fe286e2e69f958bcf91
Mobile: 1234567891 **Email:** sai111@gmail.com
Flight Id: 123 **Flight name:** roger
On-boarding: Chennai **Destination:** Banglore
Passengers: **Seats:** E-1
1. **Name:** sai prasana, **Age:** 20
Booking date: 2024-11-15 **Journey date:** 2024-11-17
Journey Time: 00:42 **Total price:** 1000
Booking status: confirmed

Cancel Ticket

Booking ID: 67379d9286e2e69f958bcf82
Mobile: 1234567891 **Email:** sai111@gmail.com
Flight Id: 123 **Flight name:** roger
On-boarding: Chennai **Destination:** Banglore
Passengers:
1. **Name:** sai prasana, **Age:** 20
Booking date: 2024-11-15 **Journey date:** 2024-11-19
Journey Time: 00:42 **Total price:** 1000
Booking status: cancelled

❖ Demo Video Link

Link: https://ufile.io/roephg67

12. Known Issues

- **Authentication Issues:**
    - Incorrect handling of JWT tokens or session data.
- **Slow API Responses:**
    - Backend APIs take too long to respond, affecting user experience.

13. Future Enhancements

- **AI-Powered Recommendations**:
    - Use machine learning to suggest personalized flights based on user preferences and booking history.
- **Price Prediction & Alerts**:
    - Predict price changes and notify users of potential price drops or rises.
- **Blockchain for Secure Ticketing**:
    - Use blockchain to ensure secure, transparent, and fraud-proof ticket transactions.
- **Voice Integration**:
    - Allow users to book flights via voice commands using Alexa or Google Assistant.

14. Conclusion

The MERN stack flight booking app offers a scalable and user-friendly platform for searching, booking, and managing flights. It combines MongoDB, Express.js, React, and Node.js to provide real-time updates, secure authentication, and seamless user experiences. Key features include flight search, booking management, and notifications. Future enhancements like AI recommendations, voice integration, and blockchain for secure ticketing will further improve the app, m0aking it a comprehensive solution for modern travelers.