Full Stack Development with MERN

Flight Finder App

1.Introduction

- **Project Title:** FlightFinder: Navigating Your Air Travel Options
- **Team Members**: Vasupalli Yesu [Team Leader]
  Sai Lakshmi Vedurupaka [Team member]
  Rayudu Surya Sri [Team member]
  Shaik Reshma [Team member]

2. Project Overview

- **Purpose**:
  FlightFinder is a full-featured MERN-stack based flight booking web application designed to provide travelers with a seamless and intuitive experience. Users can browse available flights, compare options, reserve seats, make secure payments, and manage bookings—all in real time. The platform also includes dedicated dashboards for admins and flight operators to efficiently manage backend operations, including flight listings, passenger data, and booking statuses.
- **Key Features**:
  - Users, admins, and operators can register and log in securely. Role-based authorization ensures that each user accesses only the features relevant to their role.
  - Users can search for flights using filters like source, destination, date, class (economy/business), and airline preference, improving discovery and planning.
  - Flight results display real-time availability with essential details such as flight number, duration, stops, layovers, pricing, and departure/arrival times.
  - An interactive seat map allows users to visually select and book their preferred seat (e.g., window, aisle, extra legroom).
  - The booking process includes passenger information input and mock payment processing with instant confirmation and downloadable ticket details.
  - Users can access their booking history, check flight details, cancel bookings, and receive refund status updates through their profile dashboard.
  - Admins can add, update, or remove flights; manage user records; view system-wide bookings; and monitor platform activity through a comprehensive dashboard.
  - Operators can view and manage only the flights assigned to them, with permission to update flight times, availability, and monitor associated bookings.

- A responsive UI ensures usability across all device types—desktop, tablet, and mobile—with consistent design elements and navigation flow.
- Real-time validation and notifications guide users through booking steps and alert them on changes like flight status updates or booking confirmations.
- Admins and users can access role-specific dashboards, offering insights into booking trends, flight utilization, and system health.
- The application is built with scalability in mind, using modular architecture and RESTful APIs to support future enhancements like internationalization, multi-currency payments, and airline loyalty integrations.

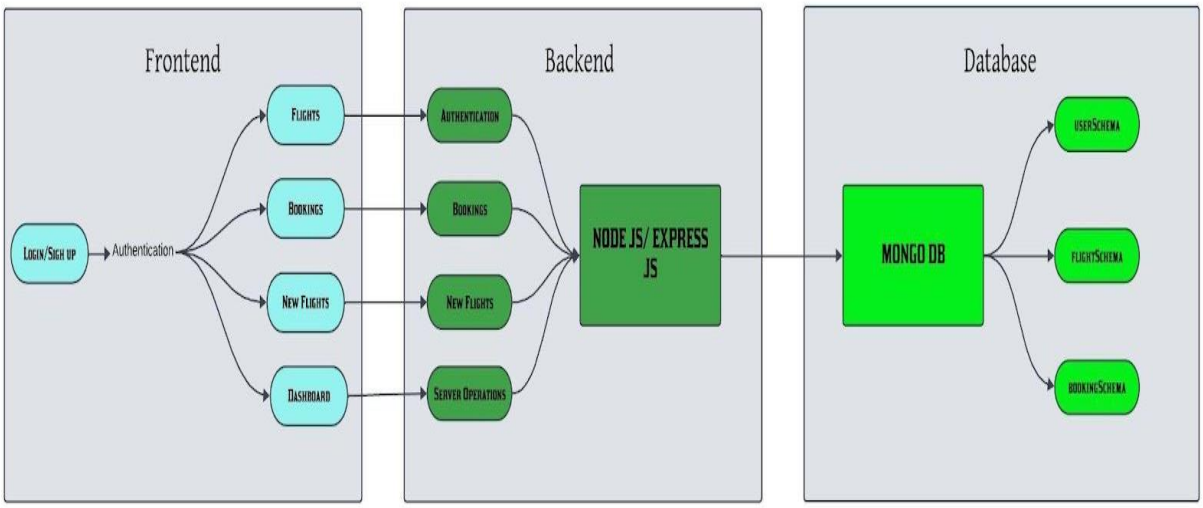
3. System Architecture

- **Client Layer (Frontend)**
  Technology: React.js
  Users: Customers(users), Flight Operators, Admin
  Role:
  Provides an interactive, responsive interface for users to search flights, select preferences, book tickets, and manage travel history. The frontend is built using React.js, with modern UI patterns to offer a fast, mobile-friendly experience.
  - Implements session and state management using React Context API and localStorage to maintain login state and user role access.
  - Uses React Router DOM to handle protected and dynamic routing for different user types (admin/operator/user).
  - API integration is handled using Axios for reliable communication with the backend, supporting GET, POST, PUT, and DELETE requests for flights, users, and bookings.
  - UI components are styled using Bootstrap and custom CSS for full responsiveness across devices.
  - Modal components are used for quick actions such as seat selection and flight filtering, enhancing usability.
  - Custom hooks and utility components simplify code reuse and readability throughout the interface.

- **Server Layer (Backend API)**
  Technology: Node.js + Express.js
  Responsibilities: Handles the business logic, routes, and core operations through modular APIs. The backend is responsible for authentication, flight listing, booking management, and administrative functions.
  - Implements secure, role-based JWT (JSON Web Token) authentication for all user roles (traveler, operator, admin).
  - Modular API structure with dedicated route files for /api/users, /api/flights, /api/bookings, and /api/admin.
  - Middleware is used extensively for input validation (using packages like express-validator), request logging, and error handling with custom status messages.
  - Uses CORS to allow secure cross-origin requests from the frontend.

- Supports RESTful routing, enabling CRUD operations for flights, bookings, and user data.
- Controllers separate route logic from business logic, making the app easier to maintain and scale.

- **Database Layer**
  Technology: MongoDB + Mongoose
  Functions: Stores Stores and manages all persistent data related to users, flights, bookings, and admin configurations.
  - Utilizes Mongoose schemas to define strict models for Users, Flights, and Bookings, including validation rules, data types, and references.
  - Supports entity relationships such as:
    ➤ One user can have multiple bookings
    ➤ One flight can be associated with multiple bookings
    ➤ Admins/operators are tagged to their managed flight entities
  - Indexes are applied to frequently queried fields like departure, destination, flightDate, and userId to optimize performance.
  - Enables seamless scalability by supporting document-based storage ideal for a growing user base and booking volume.
  - Ensures data integrity through built-in schema validation and atomic update operations using Mongoose transactions where needed.
  - Supports environment-based database URI configurations, allowing use of MongoDB Atlas for production and Compass for local development.



4. **Setup Instructions**

- **Install Prerequisites:** Install the following essential tools and technologies before starting the setup:
  - Node.js and npm – Required for running the backend server and React frontend.

- MongoDB (or MongoDB Atlas) – For database storage (either local Compass or cloud-based).
- Git – For cloning the project repository.
- Visual Studio Code – Recommended code editor for development.

- **Clone Repository:** git clone https://github.com/your-username /Flight-Booking-App-MERN.git
- **Start MongoDB:** Run mongod (or connect Atlas DB)
  - If using local MongoDB: Run mongod from your terminal to start the MongoDB server.
  - If using MongoDB Atlas: Make sure your cluster is running and use the appropriate connection URI in your .env file.
- **Setup Backend:** Navigate to the server folder and install all backend dependencies:

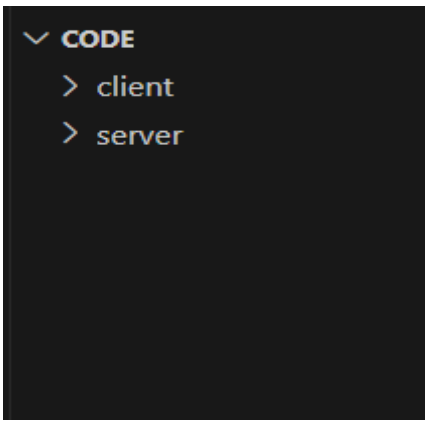
  cd code
  cd server
  node index.js
  The backend server will start on: http://localhost:6001
- **Setup Frontend:** Navigate to the client folder and install frontend packages:

  cd code
  cd client
  npm run start
  The frontend React app will run on: http://localhost:3000
- **Access the App:** After starting both servers, open your browser and go to: http://localhost:3000 – User Interface (React frontend)
- **Backend API:** APIs will be accessible via: http://localhost:6001 – RESTful backend (Express.js)
- **Environment Variables**: Create a .env file inside the server/ directory with the following variables:

  PORT=6001
  MONGO_URI=your_mongodb_connection_string
  JWT_SECRET=your_secret_key

5. Folder Structure

- Inside the Flight Booking app directory, we have the following folders



- **Client (React Frontend)**

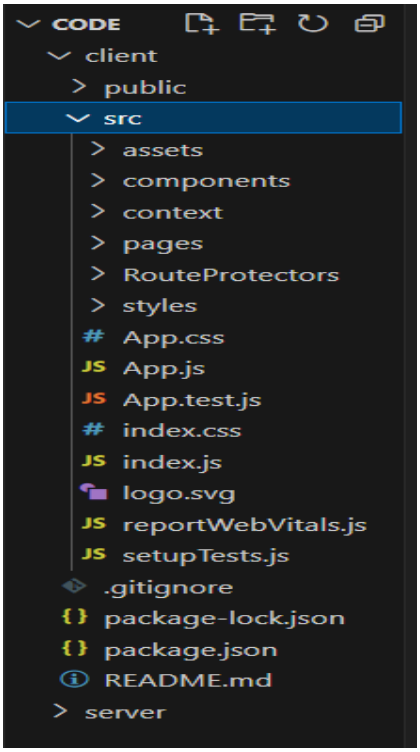
  The client directory houses the complete frontend application built using React.js. It is designed with a modular structure to promote reusability, maintainability, and scalability across different user roles (user, admin, operator).

  public/ – Static files like index.html, favicon, and any static images/icons used in the UI.

  src/ – Main application logic resides here, structured by feature and role:

  - **components**/ – Reusable UI components such as Navbar, Login, and Register.
  - **pages**/ – Role-specific page views Admin,Allbookings,AllFlights,AllUsers,Authenticate,Bookings,Allflights, etc.
  - **context**/ – Global state managed using React Context API for authentication, user sessions, and role-based access control.
  - **styles**/ – Contains CSS/Bootstrap and custom styling files used across components and pages.
  - **App.js** – Main app component that defines route structure, layout, and session routing logic.
  - **index.js** – Application entry point responsible for rendering the root component to the DOM.

  package.json – Lists all frontend dependencies like React, Axios, Bootstrap, React Router DOM, and context-related packages.



- **Server (Node.js + Express Backend)**

The server directory contains the backend logic written using **Node.js**, **Express.js**, and **Mongoose** (for MongoDB). It is structured to separate concerns like routing, business logic, and configuration.

models/ – Mongoose schemas and models for collections such as:

- User.js – Defines schema for user data, including roles (user/admin/operator).
- Flight.js – Contains flight details like airline, date, route, class, and seat map.
- Booking.js – Stores user bookings, associated flight, seat number, and payment reference.

routes/ – API route definitions separated by functionality:

- authRoutes.js – Handles registration and login for all roles.
- flightRoutes.js – APIs to create, retrieve, update, and delete flights.
- bookingRoutes.js – Handles booking creation, history, and cancellation.
- adminRoutes.js – Admin-specific actions like managing users and viewing all data.

controllers/ – Contains core logic for request handling and DB operations:

- Handles flight searches, booking validations, flight status updates, etc.

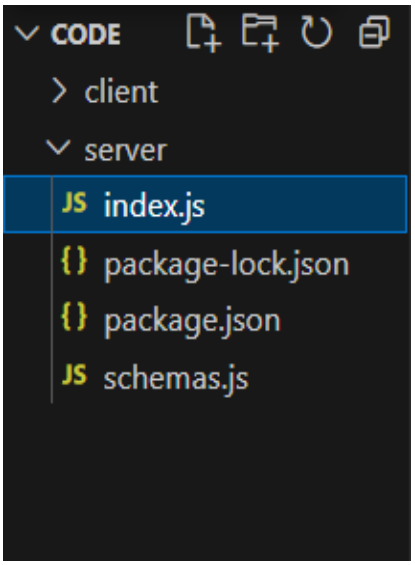
middleware/ – Includes reusable middleware functions:

- authMiddleware.js – Verifies JWT tokens and protects routes.
- errorHandler.js – Centralized error response management.

server.js – Main entry point that initializes the Express app, connects to MongoDB, and registers all routes and middleware.

.env – Holds environment variables like PORT, MONGO_URI, and JWT_SECRET for secure configuration management.

package.json – Lists backend dependencies such as Express, Mongoose, dotenv, bcryptjs, cors, and jsonwebtoken.



6. Running the Application

To run the FlightFinder project locally, follow the steps below for both the frontend and backend environments:

- **Frontend (React):**
  - Open a terminalor command prompt window.
  - Navigate to the client folder using the following command: cd client
  - Install frontend dependencies (if not already done): npm install
  - Start the React development server: npm run start
  - The frontend will run at: http://localhost:3000
- **Backend (Node.js + Express):**
  - Open a new terminal or split your terminal window.
  - Navigate to the server folder: cd server
  - Install backend dependencies (if not already done): npm install
  - Start the backend server: node index.js
  - The backend will run at: http://localhost:=5000

7. API Documentation

- **User APIs**

  POST: /api/register
  Registers a new user (traveler, admin, or operator).
  Request Body:
  {
  "name": "John Doe",
  "email": "john@example.com",
  "password": "123456",
  "role": "user"
  }

  POST: /api/login
  Authenticates a user and returns a JWT token.
  Request Body:
  {
  "email": "john@example.com",
  "password": "123456"
  }

- **Flight APIs**

  POST : /api/flights
  Adds a new flight (Admin only).
  Request Body:
  {
  "flightNumber": "AI302",
  "airline": "Air India",
  "departure": "New York",
  "arrival": "Paris",

```
  "departureTime": "2024-08-10T09:00:00Z",
  "arrivalTime": "2024-08-10T17:00:00Z",
  "duration": "8h",
  "price": 780,
  "class": "Business",
  "seatsAvailable": 20
}
```


GET : /api/flights
Fetches all available flights with optional filters (date, source, destination, class).
GET /api/flights/:id
Fetches detailed information for a specific flight.
PUT /api/flights/:id
Updates flight information (Admin only).
DELETE /api/flights/:id
Deletes a flight from the system (Admin only).

- **Booking APIs**
  POST : /api/bookings
  Creates a new booking.
  Request Body:


```
{
  "userId": "usr123",
  "flightId": "flt789",
  "seatNumber": "12A",
  "class": "Business",
  "paymentMethod": "Online"
}
```


  GET: /api/bookings/:userId
  Returns all bookings made by a specified user.
  DELETE /api/bookings/:bookingId
  Cancels a specific booking.

- **Admin APIs**
  GET : /api/admin/users
  Fetches all registered users.

  GET /api/admin/bookings
  Fetches all flight bookings across the platform.

  GET /api/admin/flights
  Fetches all flights in the system.

  POST /api/admin/add-flight
  Alternative route to add a new flight (Admin only).

  PUT /api/admin/update-flight/:flightId
  Edits existing flight details (Admin only).

- **Operator APIs**
  GET /api/operator/flights
  Returns all flights assigned to the logged-in flight operator.

  PUT /api/operator/update-flight/:flightId
  Allows operator to update status or details of their assigned flights.

8. Authentication

- **User Registration (POST /register)**
  New users (travelers, flight operators, or admins) register by submitting their name, email, password, and role.
  User credentials are securely stored in the MongoDB database. Passwords are hashed using bcryptjs before saving to ensure encryption and protect against breaches.
- **User Login (POST /login)**
  When a user attempts to log in, their provided email and password are validated against stored user records in the database.
  If the credentials are correct, the backend generates a **JWT token** along with the user's ID and role (either "user", "operator", or "admin").
  The token and user details are stored on the frontend using localStorage or via the React Context API for session persistence.
- **Protected Routes (JWT Middleware)**
  Sensitive backend routes (e.g., booking flights, managing flights, viewing user dashboards) are protected using **JWT authentication middleware**.
  To access these routes, the frontend must include the token in the request headers as:
  Authorization: Bearer <your_token_here>
  Unauthorized or expired tokens result in a 401 Unauthorized response.
- **Logout**
  When a user logs out, the frontend application clears the JWT token and user session data from localStorage or Context.
  This ensures that the session ends immediately and the user is redirected to the login or landing page.

9. User Interfaces

- **Overview:**
  The FlightFinder application features a modern, responsive user interface designed to provide an intuitive experience across all user roles—Travelers, Flight Operators, and Admins. Built using React.js and styled with Bootstrap and custom CSS, the UI is optimized for performance and accessibility.
- **Homepage & Search:**
  The homepage welcomes users with a sleek search bar allowing them to input destinations, airlines, or travel dates. Flight listings are displayed in visually distinct cards with departure, arrival, airline logo, class type, duration, and pricing.

- **User Dashboard:**
  Once logged in, users are taken to a personalized dashboard where they can:

View upcoming or past bookings
Cancel or manage existing bookings
Download e-tickets or check seat details

- **Flight Booking Flow:**
  The booking process includes:
  Search flights
  Select flight and class
  Choose seat via interactive seat map
  Review and confirm booking
  Each step uses modal overlays, validation feedback, and transition animations to guide the user.

- **Admin Dashboard:**
  Admins are provided a clean dashboard with access to:
  User management table (add/edit/remove users)
  Flight management (add/update/delete flights)
  View total bookings and revenue metrics

- **Operator Dashboard:**
  Flight operators access a limited panel where they can:
  View flights assigned to them
  Modify flight timings or seat availability5
  Track related bookings
- **Responsiveness & Accessibility:**
  All pages are responsive for mobile, tablet, and desktop viewports
  Keyboard navigation and ARIA labels used for screen reader support
  Color contrast and font hierarchy maintain visual clarity

10. Testing

Testing Strategy: The project follows a **manual, functional, and end-to-end testing** strategy to validate core functionality, user journeys, API reliability, and UI responsiveness across all user roles (User, Admin, and Operator).

- **Unit Testing**: Core backend functions were tested manually to ensure individual logic units behave as expected:

  - JWT-based authentication (token creation and validation)
  - Password hashing with `bcryptjs`
  - Flight availability logic based on filters (departure, destination, date)
  - Booking validation including seat availability and duplicate check prevention

- Admin privilege checks for protected operations like flight management

- **Integration Testing**: The integration between backend REST APIs and the frontend React components was thoroughly validated for smooth end-to-end functionality:

  - User registration/login and state persistence via Context API
  - Flight search requests and response rendering
  - Booking workflows connected to database transactions
  - Real-time seat selection integrated with backend availability checks
  - Admin dashboard functionality (flight creation, viewing users/bookings)

- **End-to-End Testing**: Complete workflows for each role were tested step-by-step:

  - User Journey:
    Register → Login → Search Flights → Select Seat → Book Flight → View/Cancel Bookings
  - Admin Journey:
    Login → Add/Edit/Delete Flights → View All Users → View System Bookings
  - Operator Journey:
    Login → View Assigned Flights → Modify Details → Track Related Bookings

- **UI/UX Testing**:
  Manual UI testing was conducted on a variety of screen sizes and devices:

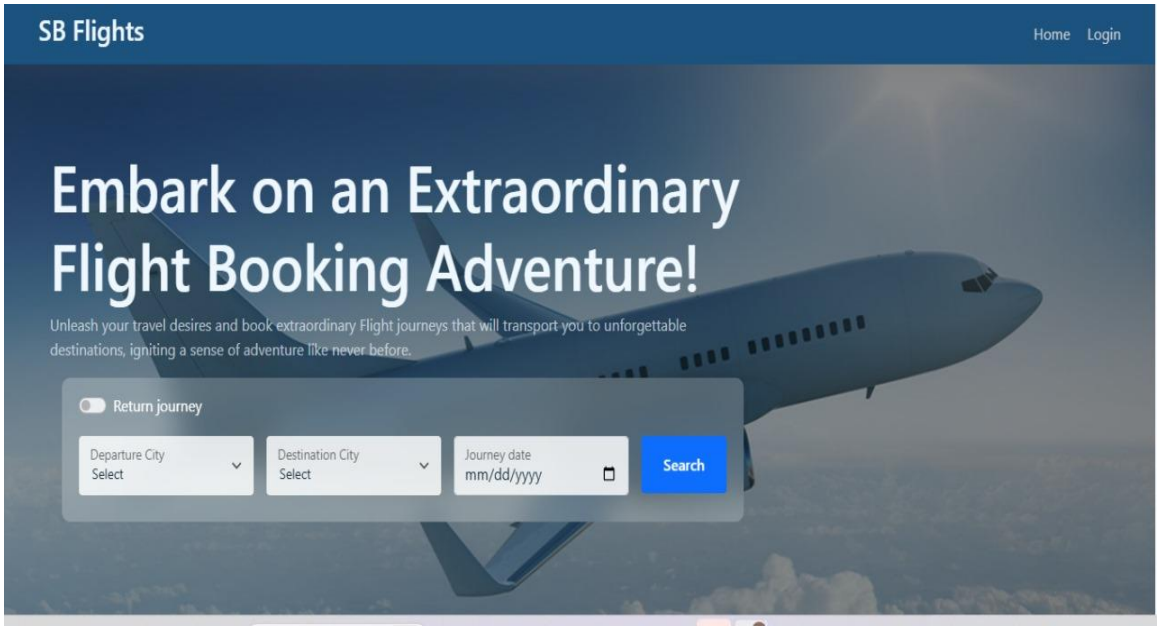
  - Desktop, tablet, and mobile responsiveness
  - Navigation flow between login, dashboards, and booking pages
  - Button accessibility, field validation messages, and modal interactions
  - Visual consistency across light and dark modes (if supported)

  Additional efforts were made to:

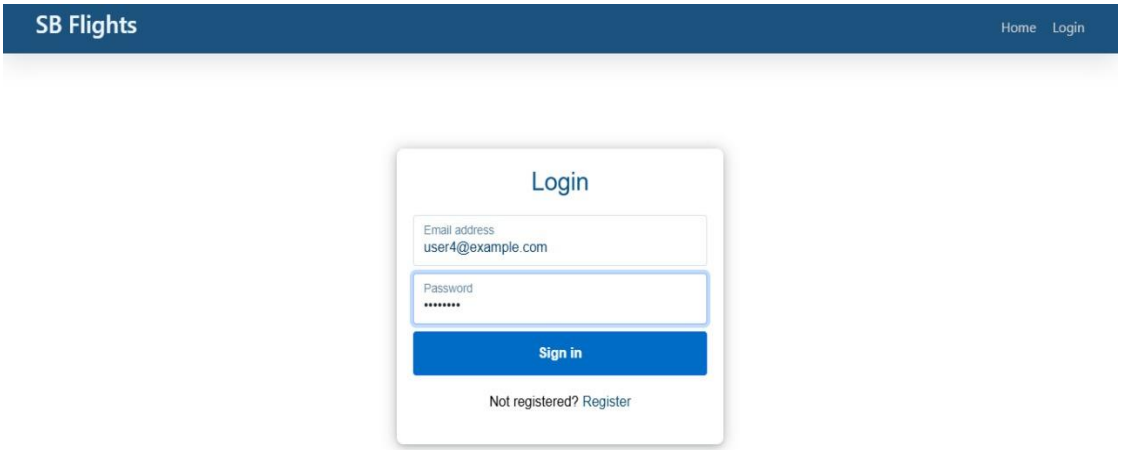
  - Validate error messages for invalid inputs or unauthorized actions
  - Ensure real-time updates (e.g., booking confirmation, seat lockout) are reflected instantly

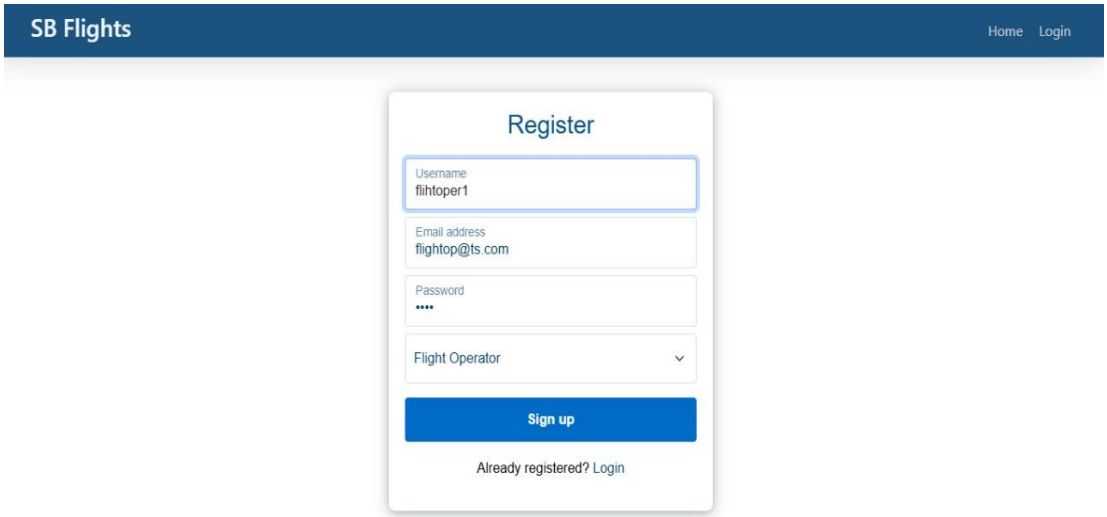
11. Screenshots / Demo

- **Landing page UI**

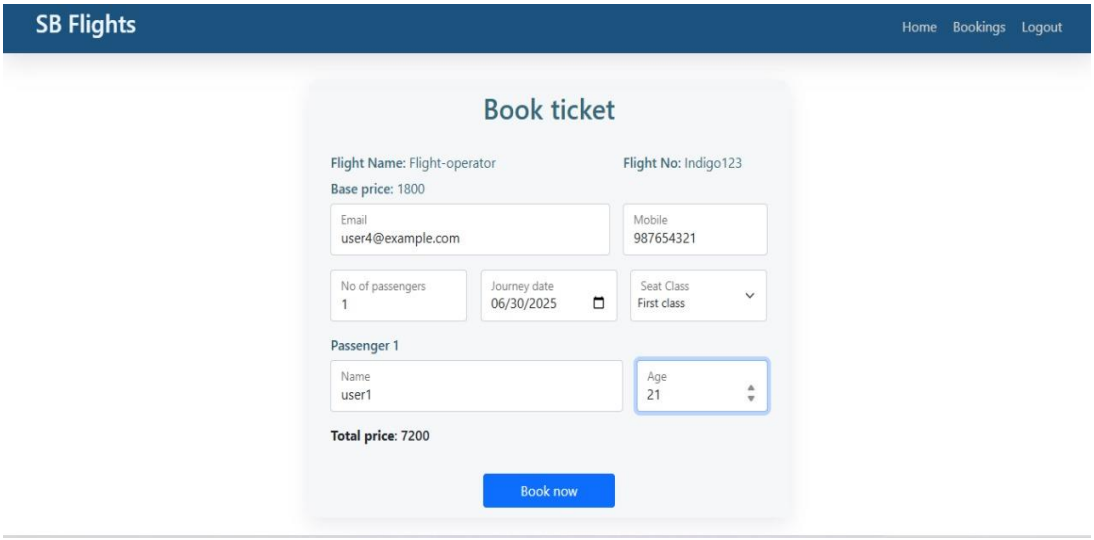


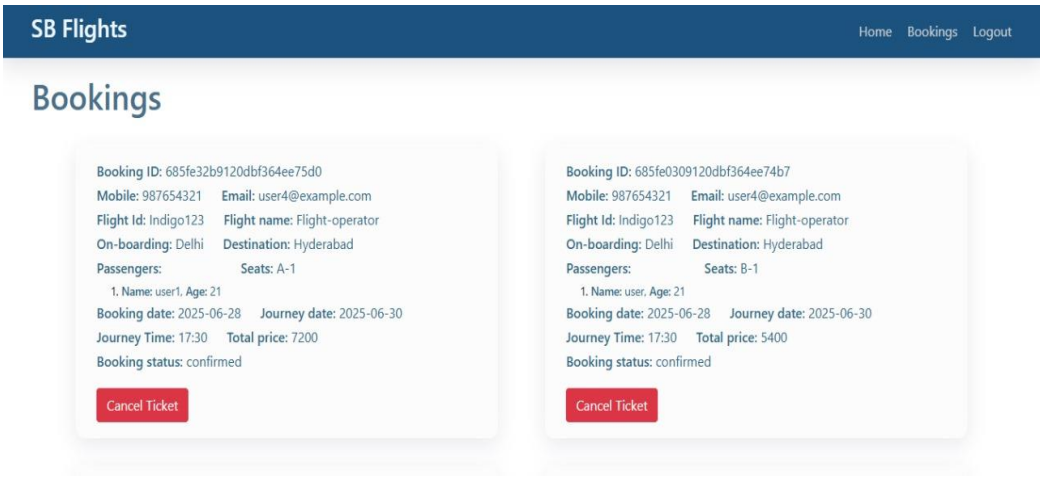
- **Authentication**



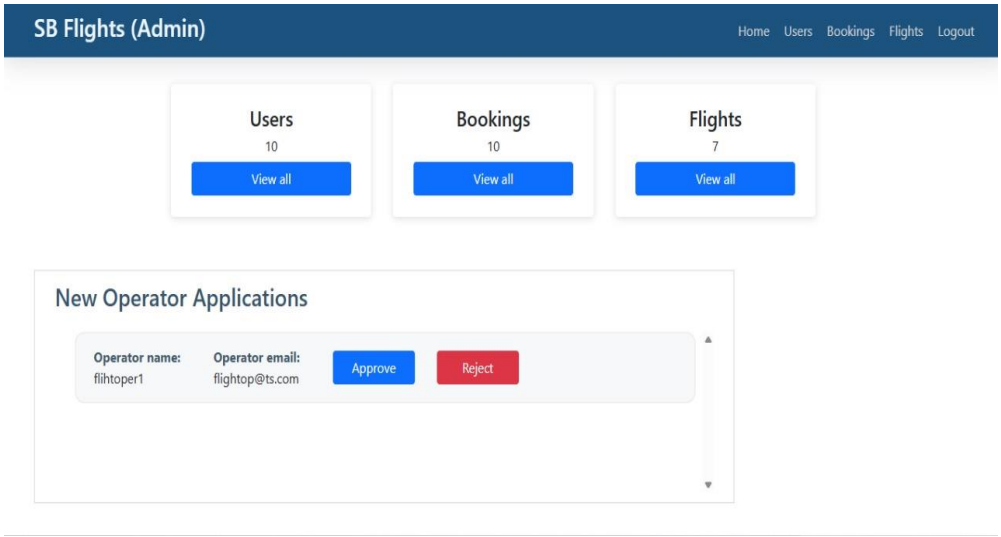


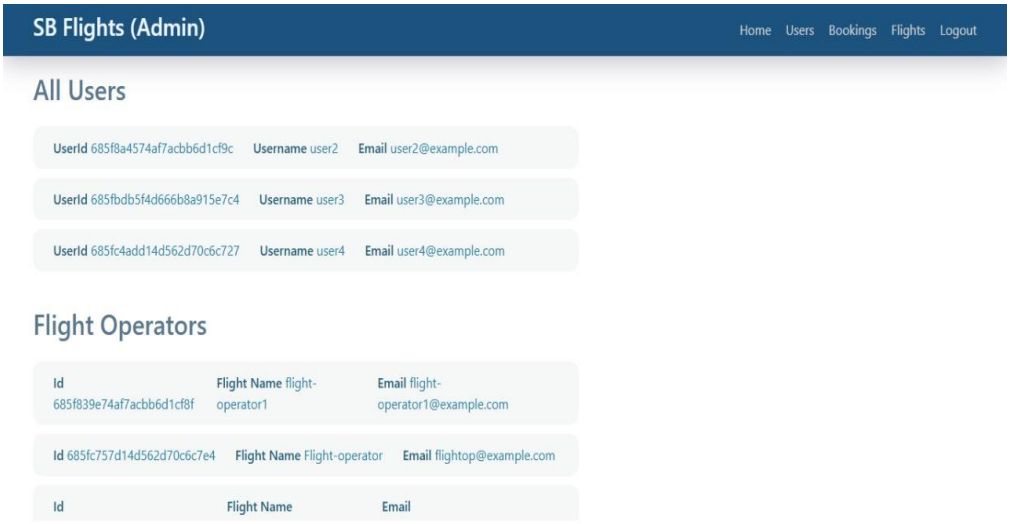
- **User bookings**

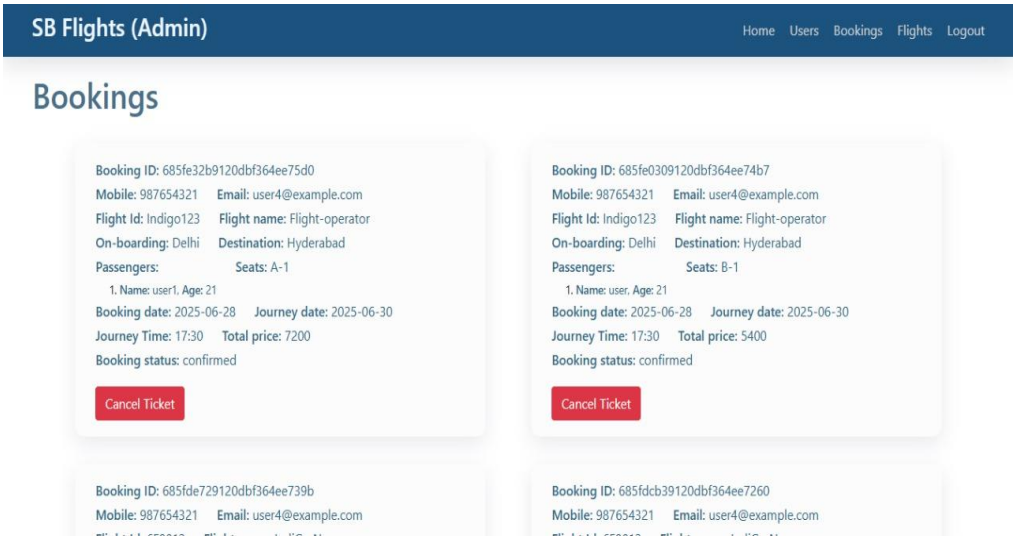


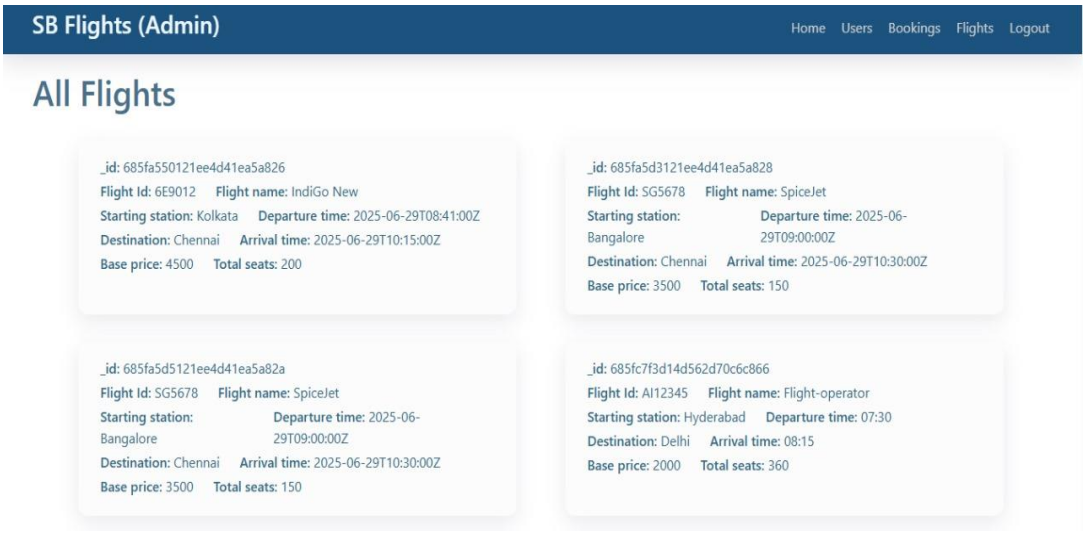


- **Admin Dashboard**

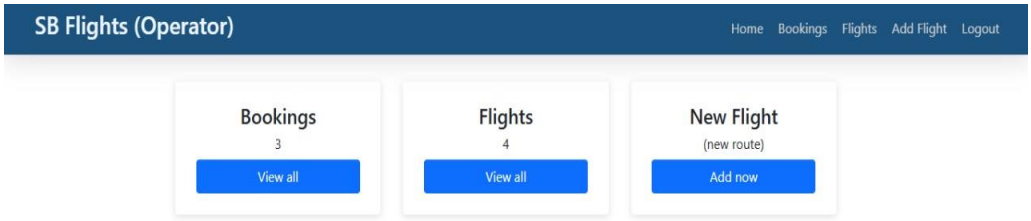

SB Flights (Admin)
Home Users Bookings Flights Logout
Users
10
View all
Bookings
10
View all
Flights
7
View all
New Operator Applications
Operator name:
flihtoper1
Operator email:
flightop@ts.com
Approve
Reject

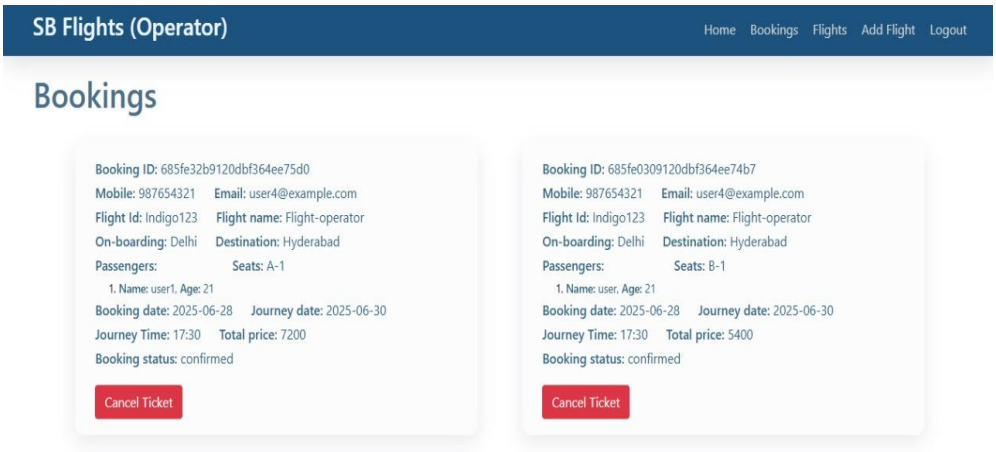

SB Flights (Admin)
Home Users Bookings Flights Logout
All Users
UserId 685f8a4574af7acbb6d1cf9c Username user2 Email user2@example.com
UserId 685fbdb5f4d666b8a915e7c4 Username user3 Email user3@example.com
UserId 685fc4add14d562d70c6c727 Username user4 Email user4@example.com
Flight Operators
Id 685f839e74af7acbb6d1cf8f Flight Name flight-operator1 Email flight-operator1@example.com
Id 685fc757d14d562d70c6c7e4 Flight Name Flight-operator Email flightop@example.com
Id Flight Name Email

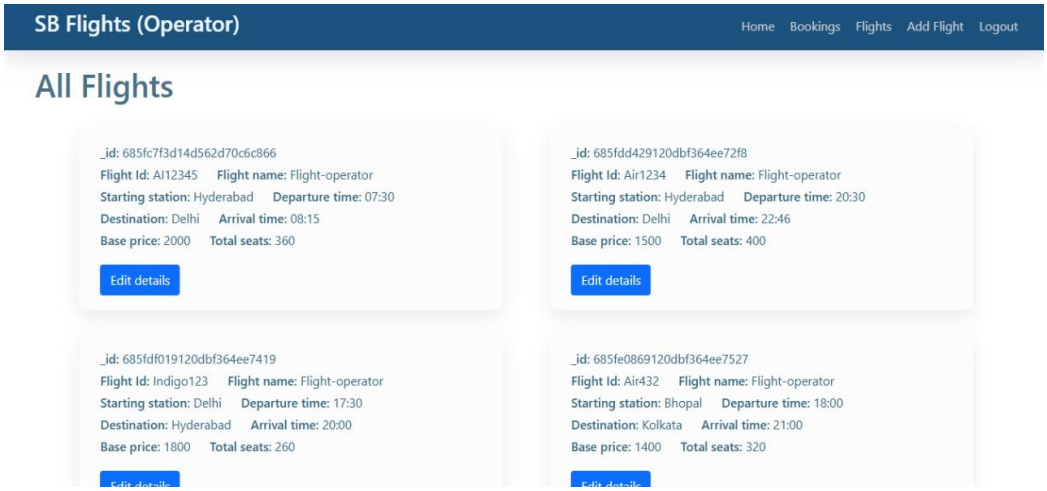

SB Flights (Admin)
Home Users Bookings Flights Logout
Bookings
Booking ID: 685fe32b9120dbf364ee75d0
Mobile: 987654321 Email: user4@example.com
Flight Id: Indigo123 Flight name: Flight-operator
On-boarding: Delhi Destination: Hyderabad
Passengers: Seats: A-1
1. Name: user1, Age: 21
Booking date: 2025-06-28 Journey date: 2025-06-30
Journey Time: 17:30 Total price: 7200
Booking status: confirmed
Cancel Ticket
Booking ID: 685fe0309120dbf364ee74b7
Mobile: 987654321 Email: user4@example.com
Flight Id: Indigo123 Flight name: Flight-operator
On-boarding: Delhi Destination: Hyderabad
Passengers: Seats: B-1
1. Name: user, Age: 21
Booking date: 2025-06-28 Journey date: 2025-06-30
Journey Time: 17:30 Total price: 5400
Booking status: confirmed
Cancel Ticket
Booking ID: 685fde729120dbf364ee739b
Mobile: 987654321 Email: user4@example.com
Booking ID: 685fdcb39120dbf364ee7260
Mobile: 987654321 Email: user4@example.com



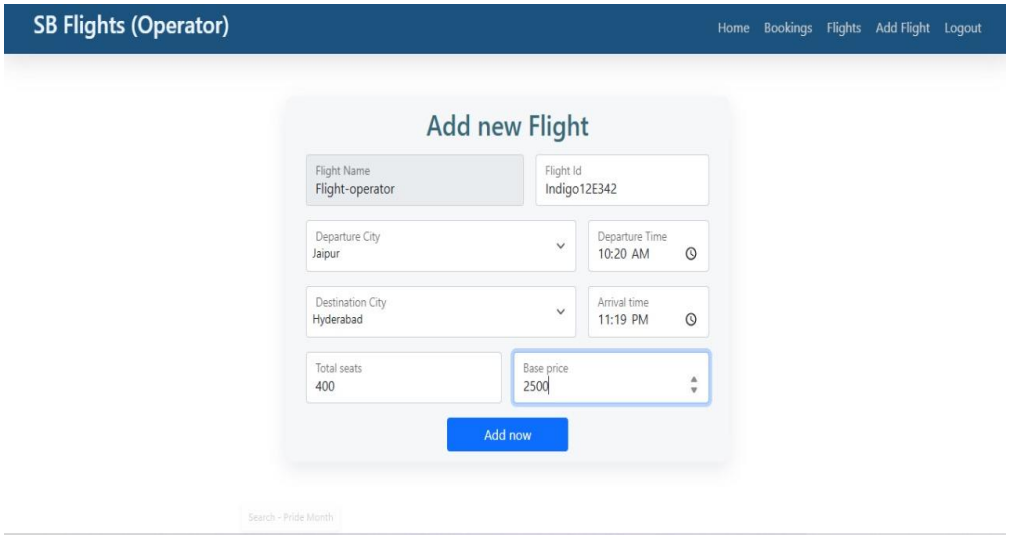
- **Flight Operator**







- **New Flight**



12. Known Issues

- **Booking Confirmation Delay on Slow Networks**
  Description:
  When users complete a flight booking, especially with slower or unstable internet connections, the confirmation screen may take a few seconds to load. Although the booking is successfully stored in the backend, the delay in frontend response can cause users to mistakenly attempt a rebooking, leading to duplicate entries or confusion.

- **Seat Selection Not Persisting on Refresh or Navigation**
  Description:

The selected seat is temporarily held in memory (React state or context) during the booking process. If a user refreshes the page, navigates back, or accidentally closes the tab before final confirmation, the seat selection resets. This disrupts the user flow and requires re-selection, which can be frustrating especially during peak booking times.

- **Admin Dashboard: Lack of Analytics & Visual Reports**
  Description:
  While admins can view lists of users, bookings, and flights, there are no visual data insights like pie charts, bar graphs, or trend lines for bookings per day, class popularity, or top destinations. This makes operational decision-making less data-driven and reduces the value of the admin panel for business oversight.

- **Minimal Form Validation Feedback**
  Description:
  Forms across the application (registration, login, flight addition) show only generic error messages such as “Something went wrong” or “Invalid credentials.” Users are not given specific feedback such as “Password must be 8 characters,” or “Email already in use,” which may lead to failed attempts or unnecessary support requests.

- **Lack of Image/File Upload Support for Flights**
  Description:
  Flight entries added by admin or operators currently don’t support uploading airline logos, aircraft images, or branding visuals. This leads to a text-heavy interface with minimal visual differentiation between flights, lowering the user’s visual engagement and trust.

- **No Email or SMS Notifications for Bookings**
  Description:
  After booking a flight, users receive on-screen confirmation only. There are no automated email or SMS notifications sent to confirm booking details or e-tickets. This limits user assurance, especially for future-dated bookings, and reduces professionalism.

- **Order Status (Booking) Sync Delay Between Roles**
  Description:
  In the admin/operator panels, booking updates (like seat confirmation, booking cancellation) sometimes do not reflect instantly in the user dashboard due to caching or delayed API refresh. This causes mismatches in displayed booking status, especially when multiple users operate on the same booking.

- **Limited Mobile-Specific Optimizations**
  Description:
  Although the app is responsive, some mobile-specific behaviors like touch scrolling in

seat selection, long-form layouts, and navigation bar minimization are not yet optimized, affecting mobile usability.

13. Future Enhancements

- **Flight Image and Airline Branding Upload**
  Enable admins and operators to upload official airline logos or aircraft images through a file upload component rather than manual URLs. This will eliminate broken links and enhance brand visibility on the flight listing cards, making the interface more trustworthy and visually engaging.
- **Advanced Filters for Flight Search**
  Introduce smart filters on the flight listing page to let users narrow down flights based on:

  - Price range
  - Preferred airlines
  - Flight duration
  - Number of stops (non-stop, 1 stop, etc.)
  - Time of departure (morning, evening, night)
    These filters will significantly improve flight discovery and help users find relevant results faster.

- **Frontend Role-Based Route Guarding**
  Strengthen frontend route security to prevent unauthorized access to restricted pages. Only users with valid roles (user, admin, operator) should access their corresponding dashboards and pages. This will prevent issues like an operator accessing admin-only tools or a guest viewing flight management panels.
- **Real-Time In-App Notifications**
  Introduce socket-based real-time notification support for:

  - Booking confirmation
  - Seat availability changes
  - Flight delays or rescheduling alerts
  - Admin announcements for operators
    This will increase user engagement and reduce the need for constant manual refreshes.

- **Visual Booking Progress Tracker**
  Create a dynamic, step-by-step visual indicator for users to track their booking journey: Booking Confirmed → E-Ticket Generated → Boarding Open → In Transit → Completed
  This improves transparency and reduces support inquiries about booking status.
- **User Reviews & Flight Ratings**
  Allow users to rate airlines and specific flights after completion of their trip. This would help future passengers make informed decisions and promote service quality across airlines. Admins can monitor and manage inappropriate reviews through moderation tools.

- **Admin Analytics Dashboard**
  Extend the admin panel with rich visual dashboards using chart libraries like Chart.js or Recharts. Metrics could include:

  - Daily/weekly booking counts
  - Top booked routes
  - Most popular airlines
  - Revenue estimates
  - Active vs. inactive users

- **Dedicated Mobile App Support**
  Design and develop native mobile applications using React Native or Flutter for Android and iOS platforms. This would allow users and operators to manage bookings, view schedules, and receive push notifications on the go.
- **Popular Flights Carousel on Homepage**
  Display trending or frequently booked flights in a visually appealing carousel format. This highlights hot routes or discounted fares and helps users explore commonly traveled destinations.
- **Secure Payment Gateway Integration**
  Integrate a robust online payment solution such as Razorpay, Stripe, or PayPal. This will enable users to securely pay for tickets via UPI, credit/debit cards, or net banking—enhancing trust and convenience while moving beyond mock payment flows.

The demo of the app is available at:

https://drive.google.com/file/d/1xIymAUjZ4cz-YRc2H6rOQFhqRKdA3PY9/view

“Happy Coding"