1.Introduction

FLIGHT BOOKING APP

Team Members:

Team Member	Role
SUNIL S	Lead (Oversees and contributes to all MERN stack development.
SANDEEP AKASH B	Developed Frontend
SANTHOSH B	Designed User Interface
NAVEEN RAJ K	Developed Backend
TAMILARASAN M	Test the Interface

//11&9 need to attach

2. Project Overview

Purpose:

The purpose of the Flight Booking Application is to provide an intuitive and seamless platform for booking domestic and international flights. The application aims to offer users a simple, interactive, and secure experience when searching for flights, making bookings, and managing travel details. By leveraging the MERN stack, this app ensures a responsive design that works across various devices and provides real-time flight availability, seat selection, and payment integration.

Goals:

- **User Accessibility**: Enable users to book flights from anywhere at any time on any device.
- **Real-Time Information**: Provide up-to-date flight schedules, availability, and pricing.
- **Security**: Ensure secure transactions for booking and payment.
- **Scalability**: Build a platform that can handle increasing user demand and flight options.

Features:

- **Search and Book Flights**: Users can search for available flights based on criteria like origin, destination, date, and class.
- **Seat Selection**: Users can select their preferred seats for each flight.
- **Payment Integration**: Secure payment gateways to complete bookings.
- **User Profiles**: Users can create profiles to save personal information, booking history, and preferences.
- **Booking Management**: Users can view, modify, and cancel bookings.

- **Flight Status**: Real-time flight status updates, including delays, cancellations, and gate information.
- **Admin Dashboard**: Admins can manage flight schedules, view bookings, and process cancellations.

---

3. Architecture

Frontend Architecture (React)

The frontend is developed using React, with a focus on creating a responsive and user-friendly interface for travelers.

- **Component-Based Structure**: React components for flight search, booking, seat selection, and payment processing.
- **State Management**: Context API or Redux for managing flight search details, user authentication, and booking status.
- **Routing**: React Router for navigating between pages like home, search results, flight details, and user dashboards.
- **Interactivity**: Real-time features such as flight status updates, seat map selection, and booking confirmation using third-party libraries or WebSocket.

Backend Architecture (Node.js & Express.js)

The backend is built with Node.js and Express.js, managing all the server-side logic, API requests, and database interactions.

- **API Design**: RESTful APIs for flight search, booking management, user authentication, and payment processing.

- **Authentication & Authorization**: JWT (JSON Web Tokens) for secure user authentication and role-based access control for admins and customers.
- **Flight Management**: API endpoints to manage flight schedules, availability, and seat selections.
- **Payment Gateway Integration**: Integrates with secure payment systems (e.g., Stripe, PayPal) for transaction processing.

Database Schema and Interactions (MongoDB)

MongoDB is used to manage the flight booking data, including user profiles, flight schedules, bookings, and payment records.

- **User Schema**: Stores user details, booking history, and payment information.
- **Flight Schema**: Includes flight details such as origin, destination, date, time, and available seats.
- **Booking Schema**: Tracks flight bookings, seat selections, and payment status.
- **Payment Schema**: Handles payment records and statuses related to bookings.

---

4. Setup Instructions

Prerequisites:

- **Vite**: A fast build tool for React.
    - Install: npm create vite@latest and choose the React template.
- **Node.js & npm**: Required for server-side JavaScript.

- Install: Node.js

- **Express.js**: Web framework for backend APIs.
  - Install: npm install express
- **MongoDB**: NoSQL database for storing app data.
  - Install: MongoDB
- **Mongoose**: ODM for MongoDB in Node.js.
  - Install: npm install mongoose

Installation Steps:

- **Create Vite + React App**:

  bash

  Copy code

  git clone <repo-url>

- cd frontend
- npm install
- cd ../backend
- npm install

1.

- **Set Up Backend (Express + MongoDB)**:

  cd backend

  npm init -y

  npm install express mongoose

2.

3. **Start Servers**:

- **Frontend**: In the Vite project folder:

  bash

Copy code

npm run dev

  - Access at http://localhost:5173.

- **Backend**: In the backend folder:

  bash

  Copy code

  node server.js

  -

---

5. Folder Structure

Client Side (React Frontend Structure)

- **node_modules**: Contains dependencies for the React app.
- **public**: Static files like index.html.
- **src**: The main source folder containing:
  - **Assets**: Images and other static assets.
  - **Components**: Divided into subfolders for Admin, Common, and User components:
  - **Admin**: Admin functionalities like ManageFlights.jsx and FlightList.jsx.
  - **Common**: Shared components like NavBar.jsx, FlightSearch.jsx, and Footer.jsx.
  - **User**: Components for customer features such as FlightDetails.jsx, BookingHistory.jsx, and UserDashboard.jsx.
- **App.css**: Global CSS styles.
- **App.jsx**: Main app component that sets up routing.

- **index.css**: Global styles.
- **main.jsx**: The entry point for React rendering.

Server Side (Node.js Backend Structure)

- **Config**: Database connection setup (Connect.js).
- **Controllers**: Defines logic for handling API requests:
    - flightController.js: Manages flight-related API requests.
    - userController.js: Manages user registration and login.
- **Middlewares**: Includes authMiddleware.js for user authentication.
- **Routers**: API routes:
    - flightRoutes.js: Routes for flight search, details, and booking.
    - userRoutes.js: Routes for user login, registration, and profile management.
- **Schemas**: Mongoose schema definitions:
    - flightModel.js: Schema for flights.
    - userModel.js: Schema for users.
    - bookingModel.js: Schema for bookings.
    - paymentModel.js: Schema for payment processing.
- **.env**: Environment variables for sensitive information (e.g., API keys).
- **index.js**: Main entry point for the backend application.
- **package.json**: Project dependencies.

---

6. Run the Application

1. **Start the Frontend Server (Vite + React)**:

- In the frontend directory:
  bash

Copy code

cd frontend

- npm run dev
    - Access at http://localhost:5173.

2. **Start the Backend Server (Node.js + Express)**:

- In the backend directory:

  bash

  Copy code

  cd backend

- npm start
    -

---

7. API Documentation

1. User Login

- **Endpoint**: /api/user/login
- **Method**: POST
- **Description**: Authenticates a user and generates a JWT token upon successful login.

Request Parameters:

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Response:

```
{
  "success": true,
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "userData": {
    "id": "12345",
    "name": "John Doe",
    "email": "user@example.com"
  }
}
```

Usage Example:

```
axiosInstance.post('http://localhost:5000/api/user/login', data)
  .then((res) => {
    if (res.data.success) {
      alert(res.data.message);
      localStorage.setItem("token", res.data.token);
      localStorage.setItem("user", JSON.stringify(res.data.userData));
      navigate('/dashboard');
      setTimeout(() => {
        window.location.reload();
      }, 1000);
    } else {
      alert(res.data.message);
    }
```

```
 })
 .catch((err) => {
   if (err.response && err.response.status === 401) {
     alert("User doesn't exist");
   }
   navigate("/login");
 });
```

---

8. Authentication & Authorization

Authentication: Managed with JWT. Upon login, a token is generated and sent to the client, where it is stored in localStorage for subsequent requests.

Authorization: JWTs include user roles (e.g., admin, customer). Role-based access is used to control access to resources like flight booking

- **Authentication Flow:**
    1. When a user logs in with their credentials (email and password), the backend verifies these credentials against the database.
    2. If the login is successful, the server generates a JWT containing the user's information (ID, role, etc.) and sends it back to the client.
    3. The client stores the JWT (typically in localStorage or sessionStorage) and includes it in the Authorization header for future requests to protected routes (e.g., booking a flight or viewing the booking history).
- **Authorization Flow:**
    1. For every subsequent request to a protected resource (e.g., accessing flight bookings), the client sends the JWT in the Authorization header of the HTTP request.

    2. The server checks if the JWT is valid and has not expired. If the token is valid, the server grants access to the requested resource.
    3. The server can also check for the user’s role (e.g., admin, customer) and ensure they have the required permissions to perform the requested action (e.g., admins can add flights, customers can book flights).
- **Token Expiration:** JWTs typically have an expiration time to improve security. When the token expires, the user is required to log in again to obtain a new token. Alternatively, a refresh token mechanism can be implemented to allow seamless token renewal without requiring the user to log in again.

---

9. Testing

Testing Strategy:

The testing strategy for the Flight Booking Application combines both manual and automated testing to ensure that the platform works smoothly and offers a great user experience.

- **Unit Testing:**
    - Purpose: To test individual functions and components in isolation to ensure they behave as expected.
    - Tool Used: Jest for testing React components and Node.js functions.
    - Example: Testing flight search functionality, seat selection, and user login processes.
- **End-to-End (E2E) Testing:**
    - Purpose: To simulate real-world user interactions and test the full flow of the application, from searching for flights to booking and payment.
    - Tool Used: Cypress for end-to-end testing. This helps in testing the whole flow of booking a flight, selecting a seat, and completing payment.
- **Performance Testing:**

  - Purpose: To ensure the platform performs well under different load conditions, such as high traffic or large flight inventories.
  - Tool Used: Google Lighthouse for measuring key performance metrics like page load speed, responsiveness, and SEO.
- **User Acceptance Testing (UAT):**
  - Purpose: To ensure the platform meets the requirements and expectations of end users.
  - Test scenarios: Search flights, book tickets, cancel bookings, etc.

---

10. Known Issues

1. Authentication Token Expiry

- Issue: Users may be logged out unexpectedly due to the expiration of their JWT token.
- Impact: Users will need to log in again frequently.
- Solution: Implement a refresh token mechanism or adjust the expiration time for more convenience.

2. Payment Gateway Failures

- Issue: Users may experience failed payments when booking flights.
- Impact: Payments may not be processed correctly, leading to incomplete bookings.
- Solution: Check the payment gateway settings, validate API keys, and ensure that third-party payment services are properly integrated.

3. Flight Availability Syncing

- Issue: Flight availability and seat inventory may not update in real time.
- Impact: Users might see incorrect flight availability or seat selection.
- Solution: Ensure that flight availability data is synchronized between the frontend and backend, and implement real-time updates.

4. Booking Confirmation Delays

- Issue: Confirmation emails or messages may be delayed or not sent after booking a flight.
- Impact: Users may be unaware of their booking status, causing confusion.
- Solution: Investigate SMTP server configurations and ensure email delivery is properly configured.

5. Mobile Responsiveness Issues

- Issue: Some features may not render correctly on mobile devices.
- Impact: Users may have a suboptimal experience while booking flights on mobile.
- Solution: Ensure that all pages are fully responsive, and test across multiple devices to ensure compatibility.

---

11. Future Enhancements

1. Real-Time Flight Updates

- Feature: Allow users to receive real-time notifications on flight statuses (e.g., delays, cancellations, gate changes).
- Benefit: Helps users stay informed about any changes to their bookings and improves the user experience.

2. Multi-Currency Support

- Feature: Enable users to select their preferred currency for payment.
- Benefit: Expands the reach of the application to international users and makes the booking process easier for them.

3. Personalized Recommendations

- Feature: Use machine learning algorithms to recommend flights based on user preferences and past booking behavior.
- Benefit: Improves user experience by showing more relevant flight options.

4. Integration with Travel Agencies

- Feature: Integrate with third-party travel agencies to display more flight options and offer bundle deals (e.g., flight + hotel).
- Benefit: Provides a broader selection of flights and services to users.

5. Loyalty Program

- Feature: Implement a loyalty program where users can earn points for each booking, which can later be redeemed for discounts.
- Benefit: Encourages repeat bookings and builds customer loyalty.

6. Virtual Assistant

- Feature: Integrate an AI-powered chatbot or virtual assistant to help users find flights, make bookings, and answer FAQs.
- Benefit: Enhances user support by providing quick and automated assistance.

7. Enhanced User Profiles

- Feature: Allow users to manage preferences like seat selection, meal preferences, and travel history within their profiles.
- Benefit: Personalizes the user experience and saves time during future bookings.

8. Social Sharing

- Feature: Allow users to share their flight bookings and itineraries on social media.
- Benefit: Encourages word-of-mouth marketing and provides an additional way for users to share their travel plans.

9. Multi-Language Support

- Feature: Support multiple languages for international users.
- Benefit: Broadens the platform's accessibility and usability across diverse regions.

10. Advanced Flight Search

- Feature: Enhance the search functionality with filters like flexible dates, non-stop flights, baggage options, etc.
- Benefit: Provides users with more control over their search and booking experience.

11. Travel Insurance Integration

- Feature: Offer optional travel insurance as part of the booking process.
- Benefit: Adds value for users, ensuring they have peace of mind in case of cancellations or other travel-related issues.

---

12. Accredited Certifications

- Feature: Provide users with travel booking certifications, including e-tickets and boarding passes, which are shareable.
- Benefit: Gives users proof of booking and provides a seamless experience across the booking and flight check-in process.

13. Subscription Models

- Feature: Offer subscription-based plans with benefits like discounted flights, priority booking, or access to exclusive deals.
- Benefit: Generates recurring revenue and enhances customer retention.

14. Admin Dashboard Enhancements

- Feature: Improve the admin dashboard with analytics, booking trends, user data, and flight availability insights.
- Benefit: Gives administrators better control and insights into flight management, booking statistics, and performance.

15.VIDEO DEMO LINK

https://drive.google.com/drive/folders/1Ennx4JvebXpU6EvOpbnX0wQoz8KdpIS8?usp=sharing