

Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** Air Bookings Web Application
- **Team Members:** Developed by me only.

2. Project Overview

- **Purpose:** To provide a seamless and user-friendly platform for booking flights, managing schedules, and handling travellers, flight operators, and admin controls in a single MERN-based application.

Features:

- Authentication for User, Flight Operator, and Admin
- Users can search flights, book tickets, view bookings, and cancel reservations
- Flight operators can add flights, update flight details, and manage customer bookings
- Admin can manage all users, flight operators, and view all bookings
- Secure role-based route protection

3. Architecture

• **Frontend:**

- Built with React, React Router, Context API
- Manages UI, cart state, filtering, and authentication
- Axios used for API calls
- UI for flight search, booking, cancellation, operator dashboard, admin dashboard

• **Backend:**

- RESTful API using Express
- Routes for flights, bookings, users, and operators
- Middleware for token verification and admin protection

• **Database:**

- Users (user/operator/admin roles)
- Mongoose schemas define fields and relationships
- A booking links a user to a flight

- Operators link to flights they manage

4. Setup Instructions

• Prerequisites:

- Node.js
- MongoDB
- npm or yarn

• Installation:

- Clone repository
- Navigate to /client and /server folders
- Run npm install in both
- Create .env in server
- Add MongoDB URL, JWT secret, and required variables
- Start both server

5. Folder Structure

• Client:

- /src/components – UI components
- /src/pages – All screens (Home, Login, Cart, Admin pages)
- /src/context – Auth & cart context
- /src/assets – Images & icons
- /src/utils – Helpers, API configs

• Server:

- /routes – Auth, product, user, order routes
- /controllers – Logic for each route
- /models – Mongoose schemas
- /middleware – Auth middleware, admin check
- /config – Database connection
- server.js – App entry point

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
- **Frontend:** `npm start` in the client directory.
- **Backend:** `node inex.js/server.js` in the server directory.

7. API Documentation

Auth Endpoints

- `Router.post("/register",Register)`
- `Router.post("/login",Login)`

Flight Endpoints (Operator/Admin)

- `Router.post('/add-flight',addFlight)`
- `Router.put('/update-flight',updateFlight)`
- `Router.get('/fetch-flights',fetchFlight)`
- `Router.get('/fetch-flight/:id',fetchFlightById)`
- `Router.get('/fetch-bookings',fetchBookings)`

Booking Endpoints

- `Router.post('/book-ticket', verifyToken, bookTicket);`
- `Router.put('/cancel-ticket/:id', verifyToken, cancelTicket);`

Admin Endpoints

- `Router.post('/approve-operator',Approve)`
- `Router.post('/reject-operator',Reject)`
- `Router.get('/fetch-user/:id',fetchUser)`
- `Router.get('/fetch-users',fetchAllUsers)`

8. Authentication

- JWT-based authentication stored in HTTP-only cookies

- Three roles: User, Operator, Admin
- Middleware checks role before allowing access
- Context API manages login states in frontend

9. User Interface

- Flight search & booking interface
- Booking history with cancel option
- Operator dashboard to add/manage flights
- Operator booking management
- Admin dashboard for users, operators, and bookings

10. Testing

- Manual testing using browser
- Postman used to test API endpoints
- Basic validation tests for auth and product routes

11. Screenshots or Demo

<https://github.com/imsathishjk/airbookings>

12. Known Issues

- Some products need optimized loading time
- Authentication may fail if environment variables are misconfigured
- Admin panel UI can be improved

13. Future Enhancements

- Payment gateway integration
- Wishlist feature
- Product reviews & ratings
- Email notifications for orders

- Mobile app version