**FlightFinder :Navigating Your Air Travel Options**

TEAM ID : LTVIP2025TMID59356

Team Members:

|  |  |
| --- | --- |
| **MEMBER** | **RESPONSIBILITY** |
| 1. Makkena Venkataprathyusha | Project setup and Configuration |
| 2.Kummaragunta Deva Priya | Frontend Development |
| 3. Kotha Himavarshini | Database Development |
| 4. Kurra Srikanth | Project Implementation |
| 5. Makkena Venkataprathyusha | Backend Development |

**1. Introduction**

Project Title: FlightFinder-Navigating Your Air Travel Options

**1.1 Project Overview**

1.2 Purpose:  
FlightFinder aims to reduce the complexity often associated with booking air travel, such as navigating multiple airline websites, checking for the best prices, and tracking flight changes. Through features like advanced search filters, booking history, and secure payment integration, the platform enhances the overall travel experience.

**2.Features:**

* Flight Search Engine
* Real-Time Flight Listings
* Smart Filters and Sorting
* User Authentication
* Payment Gateway Integration
* Booking History and Management
* Admin Panel
* Notifications and Alerts

**3. Architecture**

#### **Frontend (Client-Side)**

* **Framework:** React.js
* **Purpose:** Provides the user interface and handles interactions.
* **Key Responsibilities:**
  + Collect user inputs (e.g., flight search, booking details)
  + Display flight listings and booking status
  + Handle routing (using React Router)
  + Communicate with backend APIs via HTTP (Axios or Fetch)
  + Store session/user state (using Redux or Context API)

#### **Backend (Server-Side)**

* **Runtime & Framework:** Node.js with Express.js
* **Purpose:** Handles logic, data processing, and API responses.

**4. Setup Instructions**

Prerequisites:

Ensure you have the following installed:

* **Node.js** (v18 or above)
* **MongoDB** (local or cloud like MongoDB Atlas)
* **npm** or **yarn**
* A code editor (e.g., VS Code)

**Backend:**

bash

cd backend

npm install

Frontend:

bash

cd ../frontend

npm install

**5. Folder Structure**

Client (Frontend):

/frontend

├── public/

├── src/

│ ├── components/

│ │ ├── FlightCard.jsx

│ │ ├── Navbar.jsx

│ │ └── BookingForm.jsx

│ ├── context/

│ │ └── AuthContext.js

│ ├── pages/

│ │ ├── Home.jsx

│ │ ├── Login.jsx

│ │ ├── Register.jsx

│ │ ├── Dashboard.jsx

│ │ └── AdminPanel.jsx

│ ├── App.js

│ └── index.js

└── package.json

**Server (Backend):**

/backend

├── controllers/

│ ├── userController.js

│ ├── flightController.js

│ └── bookingController.js

├── models/

│ ├── User.js

│ ├── Flight.js

│ └── Booking.js

├── routes/

│ ├── userRoutes.js

│ ├── flightRoutes.js

│ └── bookingRoutes.js

├── middleware/

│ └── authMiddleware.js

├── config/

│ └── db.js

├── .env

├── app.js

└── package.json

**6. Running the Application**

**To run both servers:**

**Frontend:**

bash

cd frontend

npm start

Runs on: http://localhost:3000

Backend:

bash

cd backend

npm start

Runs on: http://localhost:6001

**7. API Documentation**

**Base URL:**

http://localhost:6001/api

Auth Endpoints:

POST /auth/register - Register new users (User, Operator, Admin)

POST /auth/login - Login and receive JWT token

**Flight Endpoints:**

GET /flights - View all available flights

POST /flights (Protected) - Add a new flight (Admin/Operator only)

PUT /flights/:id (Protected) - Update flight details

DELETE /flights/:id (Protected) - Remove a flight

Booking Endpoints:

GET /bookings (Protected) - View user’s bookings

POST /bookings (Protected) - Book a flight

Payment:

POST /payment (Simulated) - Mock payment processing during booking

**8. Authentication**

Authentication is implemented using \*JWT (JSON Web Tokens)\*:

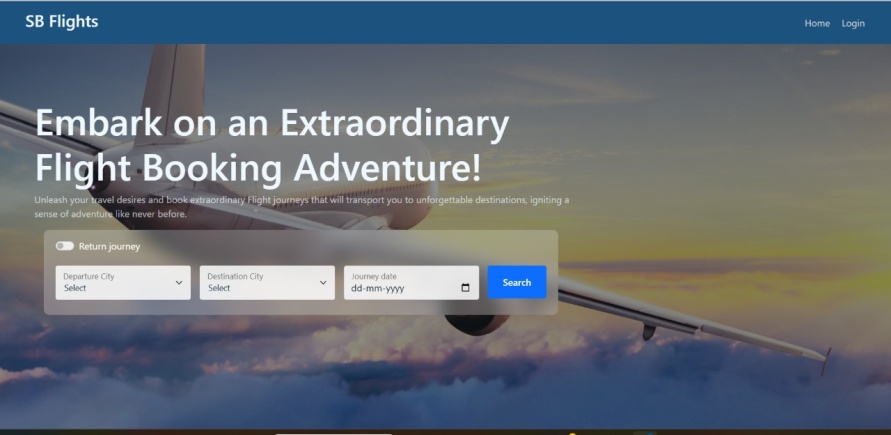
On successful login, a JWT token is generated and stored in localStorage

All protected API endpoints require a valid token

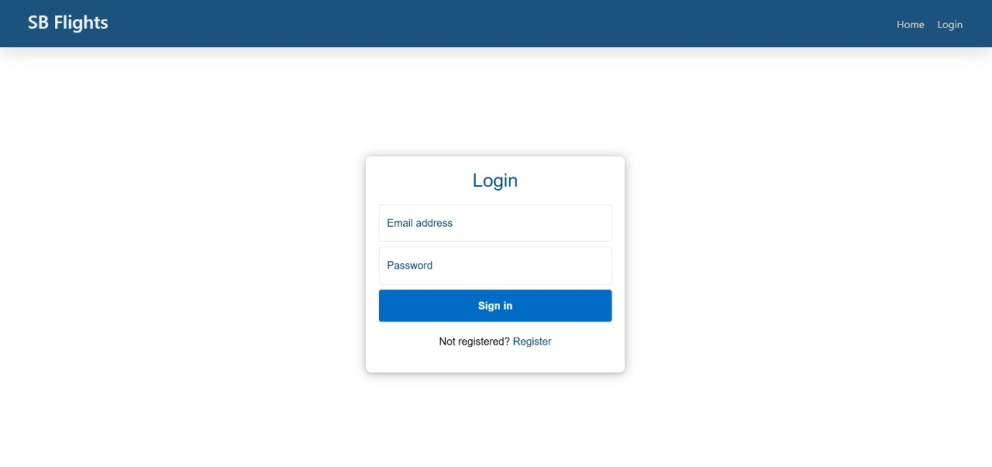
Middleware verifies token and allows role-based access (User, Admin, Operator)

**9. Screenshots or Demo**

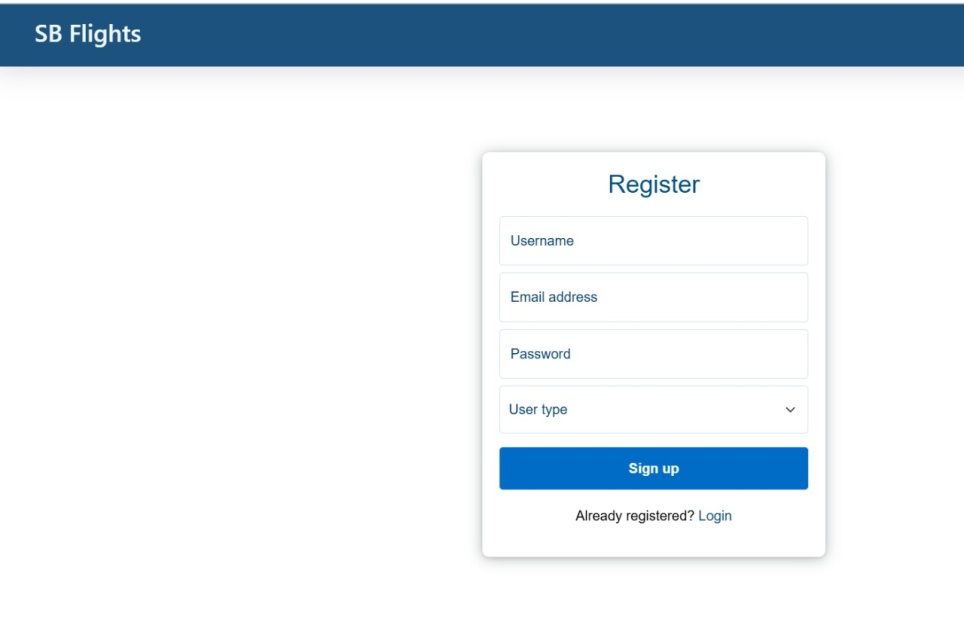
HOME PAGE:



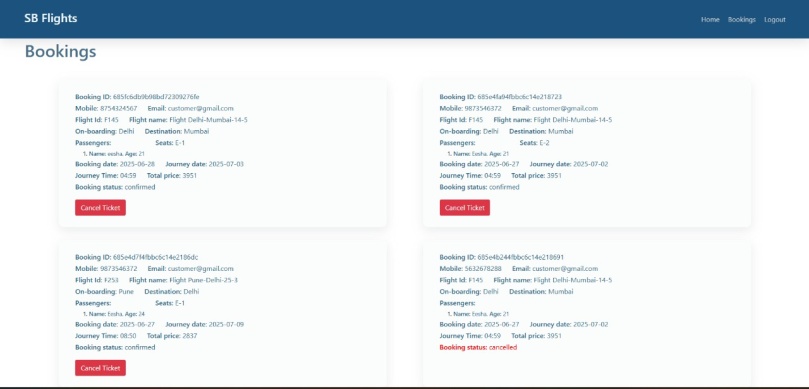
LOGIN PAGE:



REGISTER PAGE :



DASHBOARD:



**10. Testing**

Testing Strategy:

Frontend components are used to test UI components and their interaction with backend APIs.

Backend APIs are manually test REST APIs (GET, POST, PUT, DELETE).

End-to-end testing is performed using Cypress to simulate real-user interactions.

**11. Known Issues**

Without integration with a live flight API (e.g., Amadeus), results may be based on static or outdated data.

Payments may not process real transactions during development/testing phase.

**12. Future Enhancements**

To further elevate the user experience and functionality of FlightFinder, several future enhancements are planned. One major improvement is the integration of **live flight tracking**, which will allow users to view real-time updates on delays, gate changes, and flight positions.

Global usability will be improved through **multi-language and currency support**, making FlightFinder accessible to a broader audience. An **admin analytics dashboard** will be developed to help administrators monitor key metrics such as user activity and revenue trends. Lastly, offering **offline ticket access** and **third-party travel packages** (e.g., hotels, insurance) will enhance the platform’s versatility as an all-in-one travel solution.