Full Stack Development with MERN Project

Flight Finder :Navigating Your Travel Options

1. Introduction

- **Team Members:**

  1. **Team Leader :** Peddi Arjun

     Coordinator

     Builds RESTful APIs using Node.js and Express.js, manages authentication and server logic.

  2. **Team member :** Parasa Naga Veera Vardhan

     Works on the React-based UI, handles component design, page routing, and user interactions.

  3. **Team member :** Parasa Janani Priya

     Designs and manages MongoDB schemas, handles CRUD operations and ensures data consistency.

  4. **Team member :** Parise Lavanya

     Responsible for overall planning, coordination, GitHub management, and integration of frontend and backend.

2. Project Overview

Purpose: Built as a user-friendly web Application,Flight Finder Simplifies the travel planning allowing travelersto browse,comput, and book Flights efficiently.The System improves over time by Machine learning to tailor flight suggestions to individual needs and past booking behaviour

The application is designed to:

- Flight Search by source to destination
- Regestration and Login System

Ultimately, the goal is to replicate the core functionality of platforms like **SkyScanner** using open-source technologies.

Features: For Users:

- **Sign Up / Log In** – Create an account and access your Flight Booking.
- **Browse Flights** – View a list of available Flights
- **Order Confirmation** – Get a message when your Booking is successfully Completed.

For Admin (Future Scope):

- **Add or Update Flight Details** – Admin can manage Flight Details.
- **View Usera** – Admin can see Users Who are Login and Register

3. Architecture

Frontend (HTML,Css,Javascript)

- Built using Html with multiple pages (Home, Login page, Admin Page etc.)
- Uses Javascript for navigation and Context API for managing the cart
- API calls to the backend
- Admin and user info are stored in localStorage

Backend (Node.js + Express.js)

- Handles API routes like register, login, get products, and place orders
- Uses Express middleware for JSON handling and CORS
- Connects to MongoDB using Mongoose

Database (MongoDB)

- Stores user, product, and order data
- Collections:
    - users: name, email, password, address
    - Flight Details: Source,Destination,
    - Booking: userid,Email,payment method

4. Setup Instructions

Prerequisites

- **Node.js & npm** – For running frontend and backend
- **MongoDB** – Local database (use Compass or terminal)
- **Git** – To clone the project
- **VS Code** – Recommended editor

Installation Steps

Clone the Project

git clone : https://github.com/Lavanyaparise/FLIGHTFINDER-.git

Install & Run Backend

```
cd server
npm install
node server.js
```

1. **Install & Run Frontend**
   Open a new terminal:

   ```
   cd client
   npm install
   npm start
   ```

2. **Start MongoDB**
   - Use MongoDB Compass or run mongod in terminal.

Your app will run at:

- Frontend: http://localhost:3000

- Backend API: http://localhost:5000

5. Folder Structure

- **Client(React frontend):**

```
Public/
├── src/
│   ├── Index/
│   │   └── Index.css/        → All page components (Login, Register etc.)
│   ├── Admin/          → Cart context (Users,Flight Booking Details)
│   ├── App.jsx
│   └── index.js
```

- **Server(Node.js backend):**

```
src/
├── script.js/          → Mongoose schemas (Users Details,Flight Booking Details)
├──.env → Main server file
```

6. Running the Application

Frontend :

```
cd client
npm start
```

Runs the React app at: http://localhost:3000

Backend :

```
cd server
npm start   # Or use: node server.js
```

Runs the Node.js server at: http://localhost:5000

7. API Documentation

- **POST /api/register :** Registers a new user.
- **POST /api/login** : Logs in an existing user.
- **GET /api/products** : Retrieves a list of available Flight Details**.**
- **POST /api/orders** : Conform Booking Of Flights

8. Authentication

How Authentication Works:

- Users register by providing their email, password and Some Other Details:

POST /api/register

- They log in with their email and password using:

  POST /api/login

Method Used:

- The current setup uses **basic email and password matching**.
- There is **no token-based authentication** or sessions implemented at this stage.
- After login, the user's details can be stored on the frontend (e.g., in localStorage) to maintain the login state.

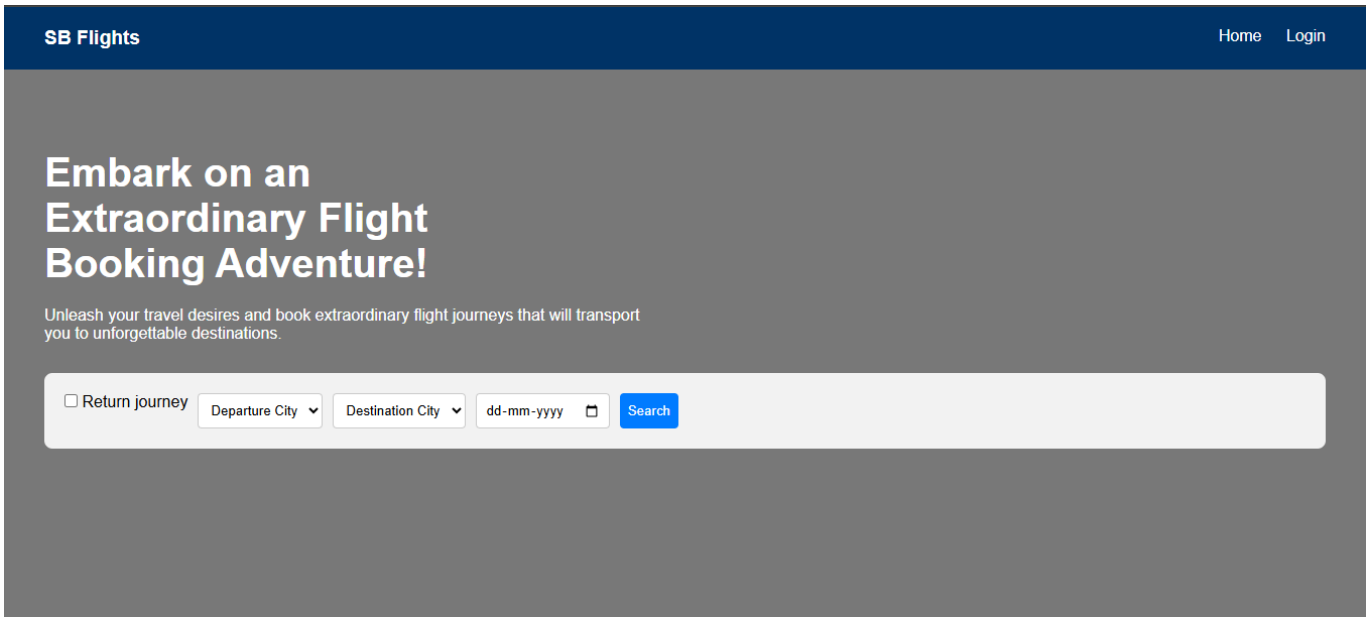
Recommendations for Improvement:

To enhance security in the future, it is recommended to:

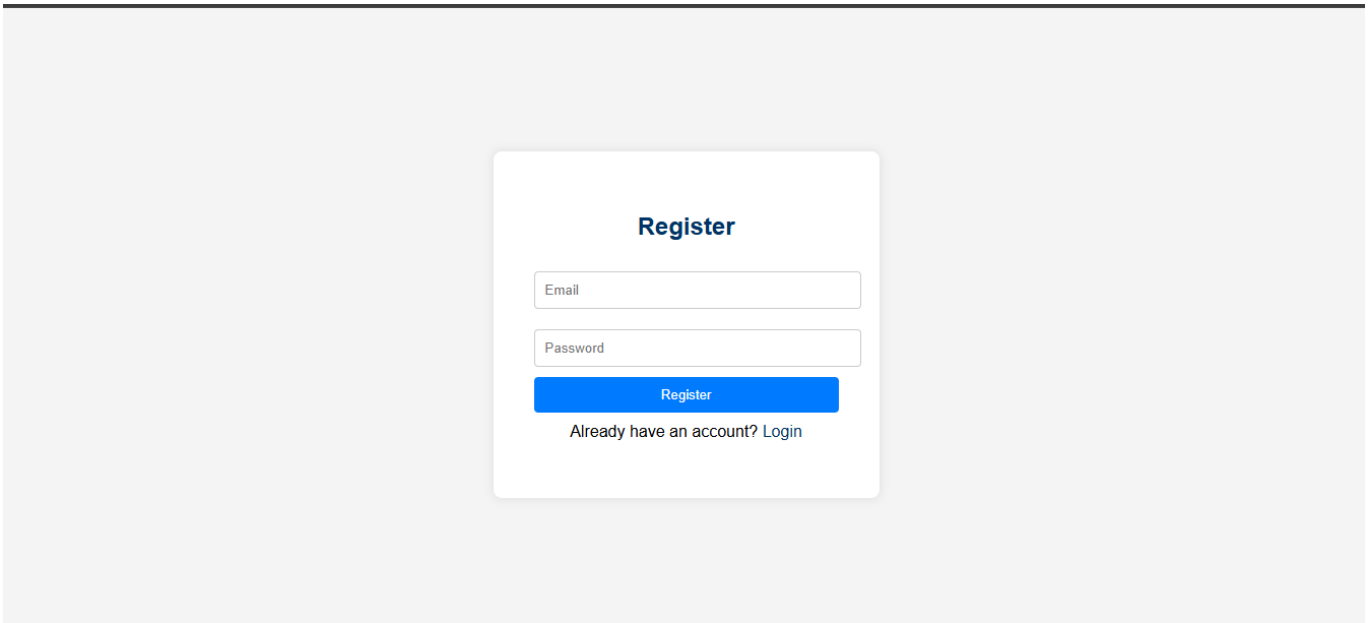
- Use **middleware** to protect private API routes.
- Store tokens securely (e.g., in localStorage or HTTP-only cookies).

9. User Interface

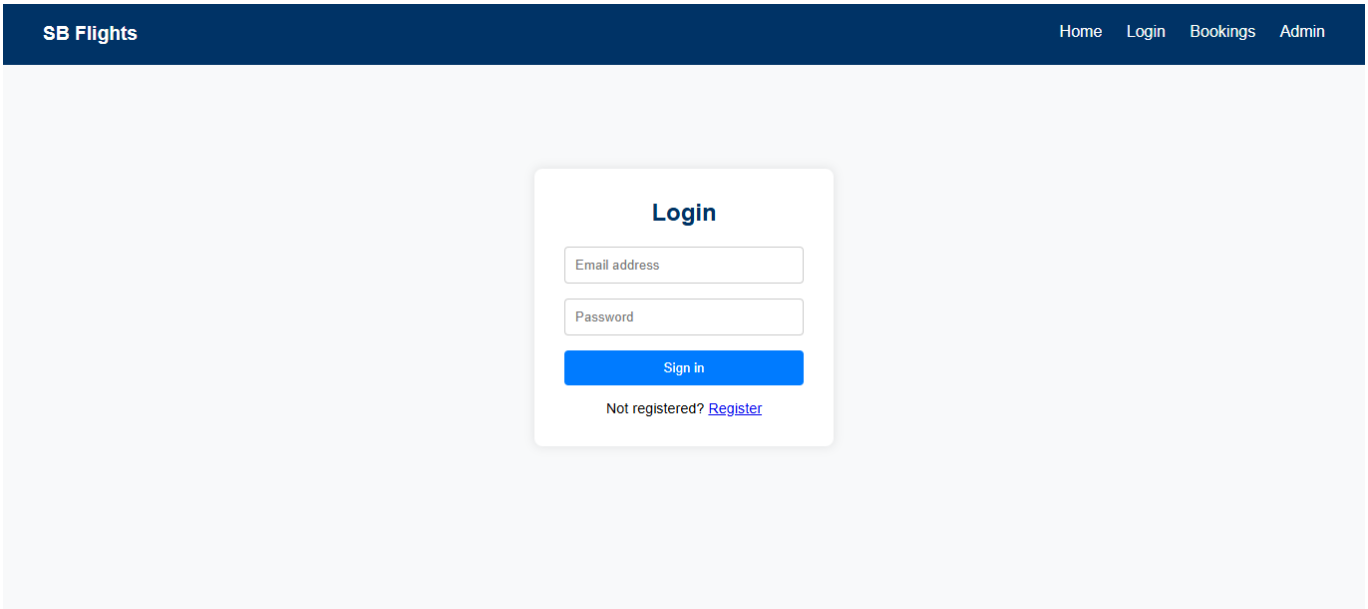
Home page:



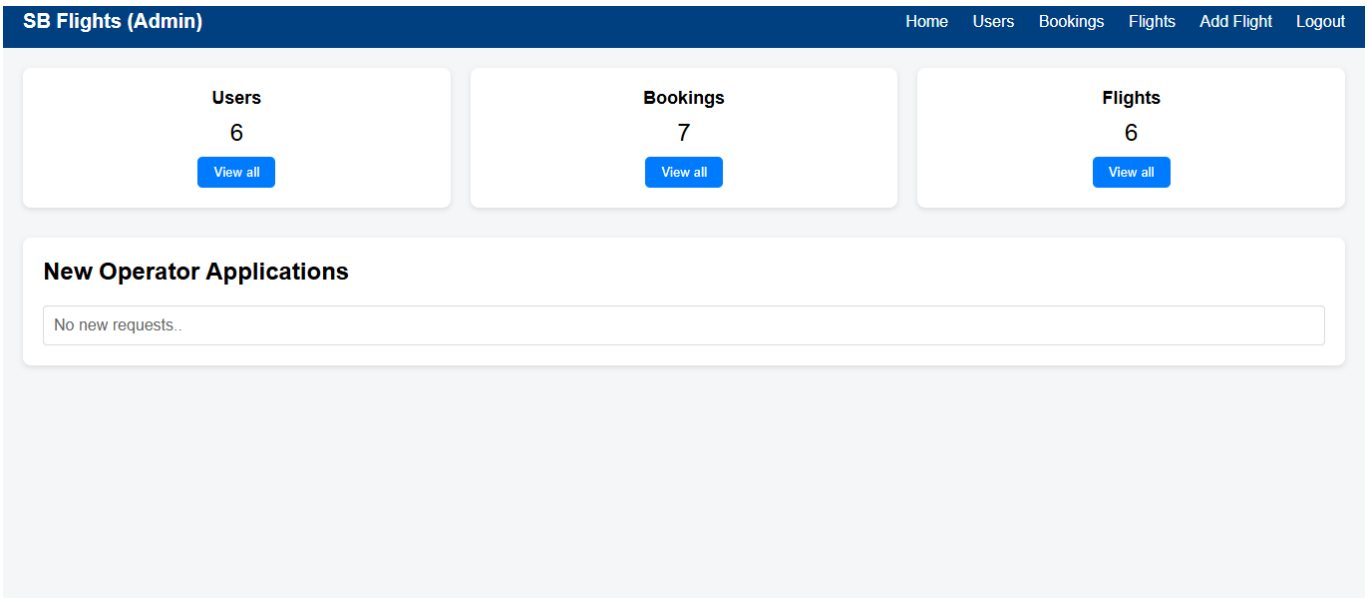
Registration page:



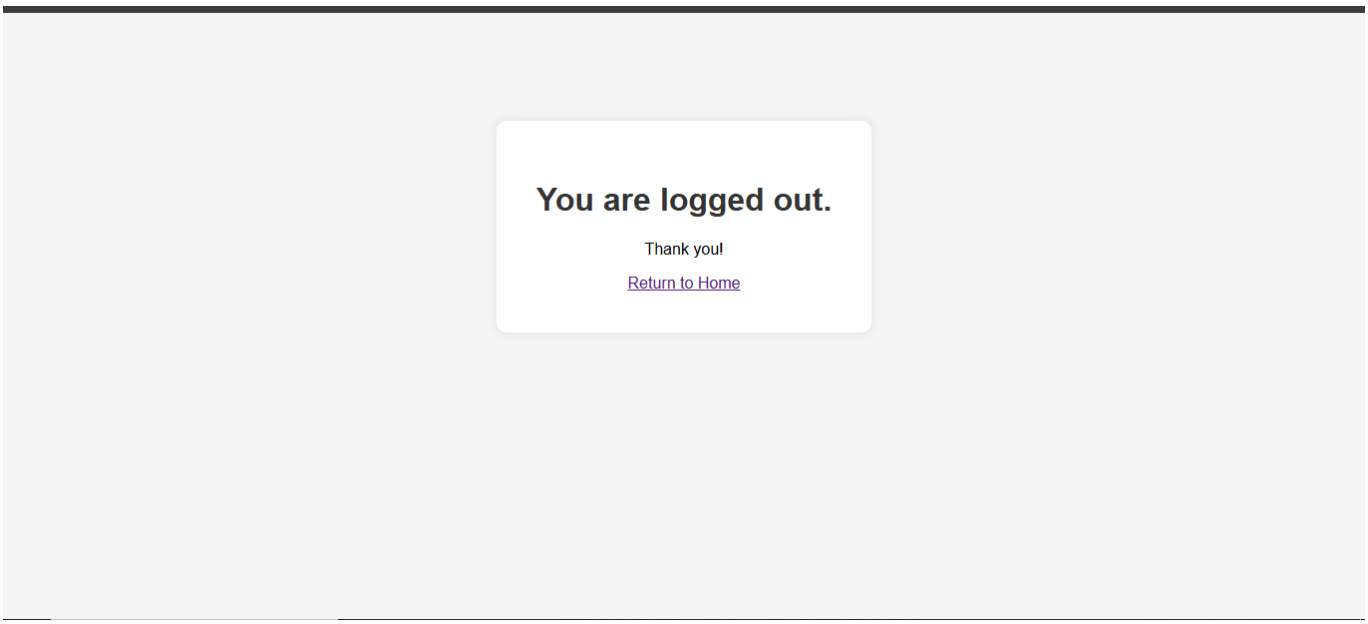
Login page:



Admin dashboard:



Exit Page After Logout:



10. Testing

- **Manual testing** was done by using the app (register, login,Booking flow).
- **Postman** was used to test backend APIs.
- **Browser DevTools** helped inspect React components and API requests.

11.Known Issues

- **No authentication tokens** – Login does not use JWT or sessions, so user sessions are not fully secure.
- **No order history** – Users cannot view past orders after placing them.
- **Cart resets on logout** – Cart is stored in localStorage and clears when browser data is cleared or user logs out.
- **No automated testing** – All testing is manual; no test scripts are in place.
- **No real-time updates** – Admin actions like order status changes aren't reflected instantly on user side.

11. Future Enhancements

- Payment integration with Razorpay/Stripe
- Role-based admin access