**FlightFinder: Navigating Your Air Travel Options**

**TEAM ID** : LTVIP2025TMID59738

**TEAM MEMBERS:**

|  |  |
| --- | --- |
| **MEMBER** | **RESPONSIBILITY** |
| 1. Vellalacheruvu Devi Sahasra | Frontend & Backend Development, Database Integration |
| 2. Tirumalasetti Vineela | Admin & Flight Operator Panel Logic, Role-based Access Implementation |
| 3. Thiruvaipati Lakshmi Navaneeth | Flight Data Seeding, Filtering Logic, Booking Module Support |
| 4. Thippaluri Yaseen Basha | Testing, UI Review, Final Documentation Preparation |

**1. Introduction**

Project Title 🡪 FlightFinder: Navigating Your Air Travel Options

**1.1 Project Overview**

The Flight Booking Web Application is a full-stack project developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js). It is designed to replicate the core functionalities of a real-world flight reservation system. The application supports three user roles:

* User (Passenger) to search and book flights
* Flight Operator to manage assigned flight routes
* Admin to oversee and control the entire system

**1.2 Purpose**

The purpose of this project is to design a scalable and interactive flight booking system that helps users:

* Book flights with ease using a user-friendly interface
* View options with connecting flights if no direct ones are available
* Experience realistic booking logic similar to commercial platforms

At the same time, it provides admin and operator interfaces to manage flight data, ensuring the system remains up-to-date and functional.

**2. Key Features**

* User registration and login with role-based access (User, Admin, Operator)
* Search flights by source, destination, and date
* View both direct and connecting (layover) flight options
* Book flights and store booking details
* Simulated payment system for booking confirmation
* View past bookings in user dashboard
* Admin panel to manage all flights and users
* Operator panel to add and update assigned flights
* Realistic flight data and scheduling
* Clean and responsive UI with React

**3. Architecture**

**Frontend:**

Built using **React.js**, the frontend follows a component-based architecture:

* **Component Structure**: Functional components with hooks (useState, useEffect)
* **Routing**: React Router v6 for navigation
* **State Management**: Context API for user authentication and booking state
* **UI Library**: Tailwind CSS for responsive and modern design
* **HTTP Requests**: Axios used for communication with the backend API

**Backend:**

Developed using **Node.js** and **Express.js**:

* RESTful API structure
* Middleware: cors, dotenv, morgan, express-validator
* **Authentication**: JWT middleware for securing protected routes
* **Project Structure**: Modular with separate folders for Routes, Controllers, and Models

**Database:**

* **MongoDB** is used as the NoSQL database
* **Mongoose** is used as the ORM for schema modeling and query handling

**4. Setup Instructions**

**Prerequisites:**

Make sure you have the following installed:

* Node.js (v16.x or higher)
* npm or yarn
* MongoDB Atlas account or local MongoDB instance
* Git

**Installation Steps:**

1. **Clone the repository:**

git clone https://github.com/Sahasra75/SkyDreams---Flight-Finder

cd Flight-Booking-App

1. **Set up environment variables:**  
   Create .env file in the backend root directory:

PORT=6001

MONGO\_URI=mongodb://localhost:27017/flightbooking

1. **Install dependencies:**

* Backend:

cd server

npm install

* Frontend:

cd client

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:**

cd client

npm start

Runs on: http://localhost:3000

**Backend:**

cd server

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

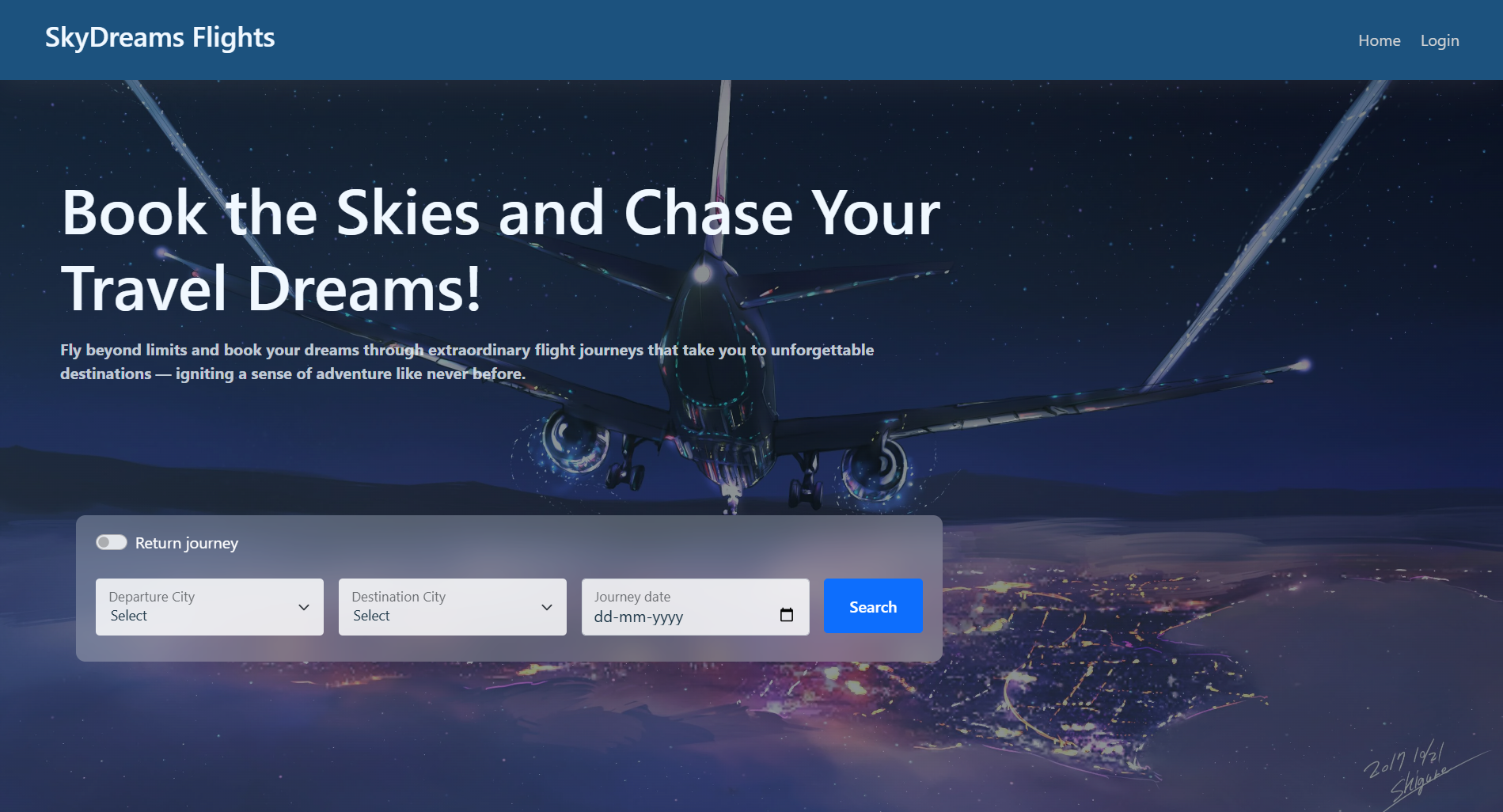
**8. Authentication**

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

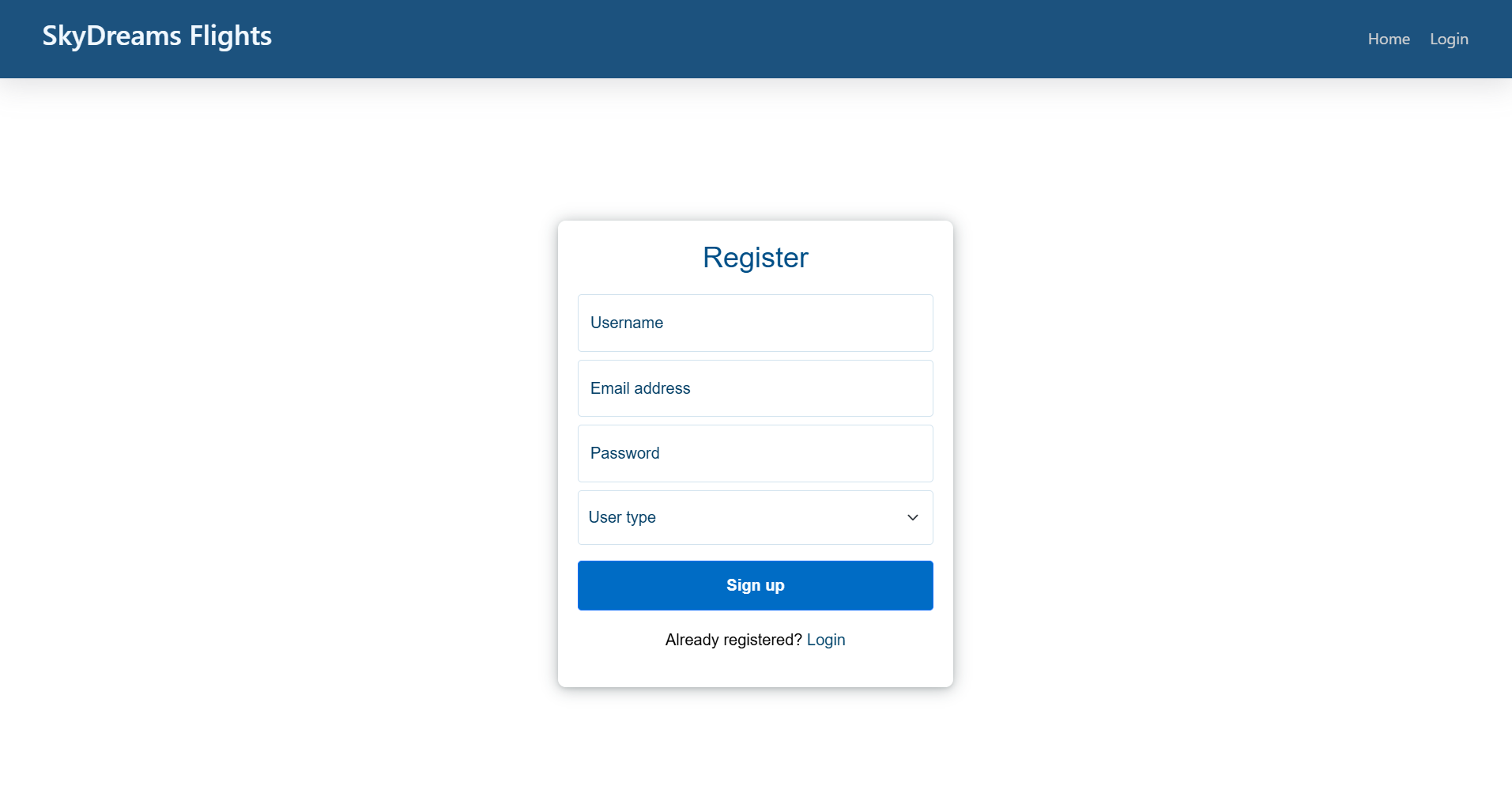
* On successful login, a JWT token is generated and stored in localStorage
* All protected API endpoints require a valid token
* Middleware verifies token and allows role-based access (User, Admin, Operator)

**9. Screenshots or Demo**

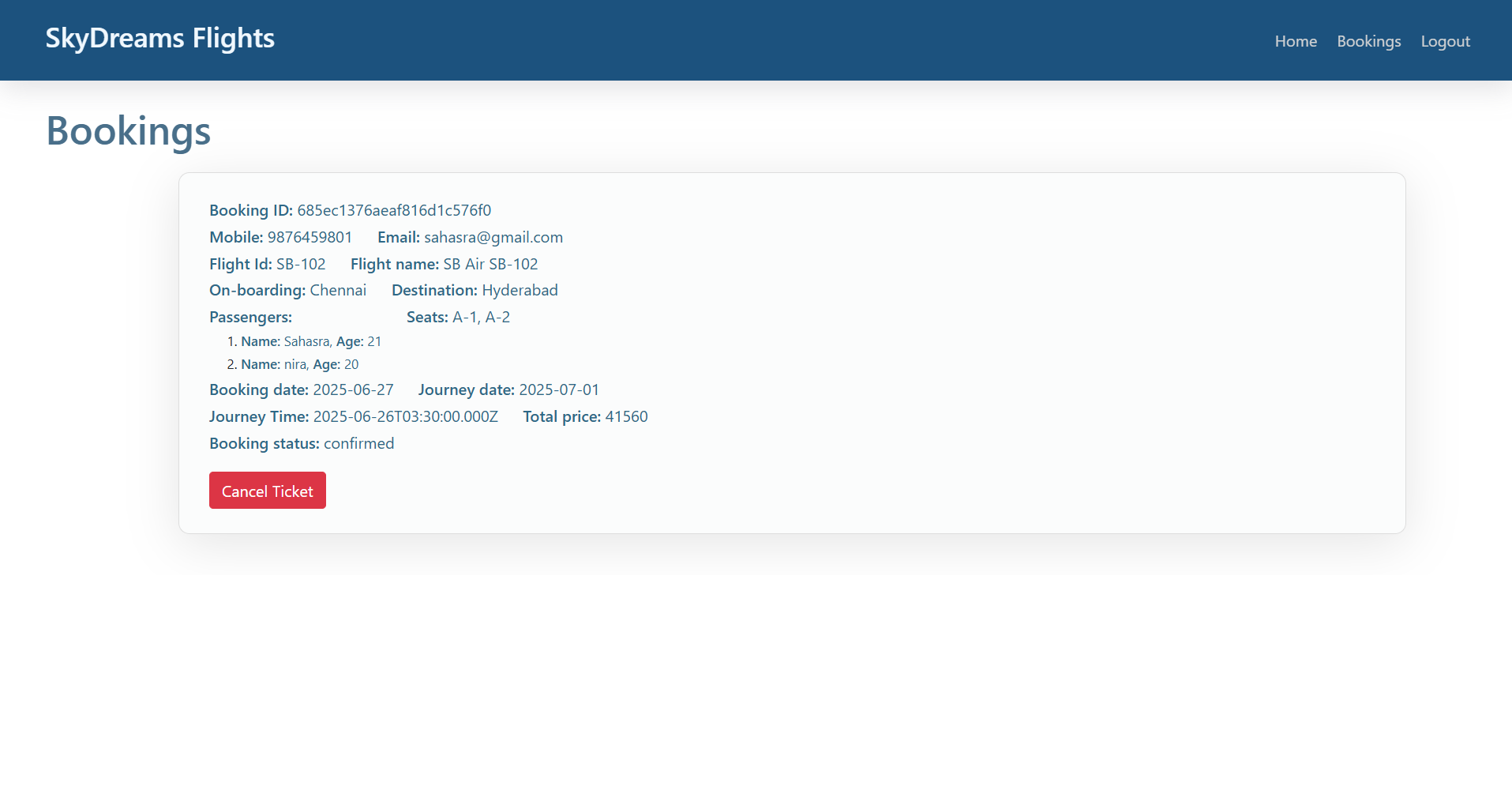
HOME PAGE :



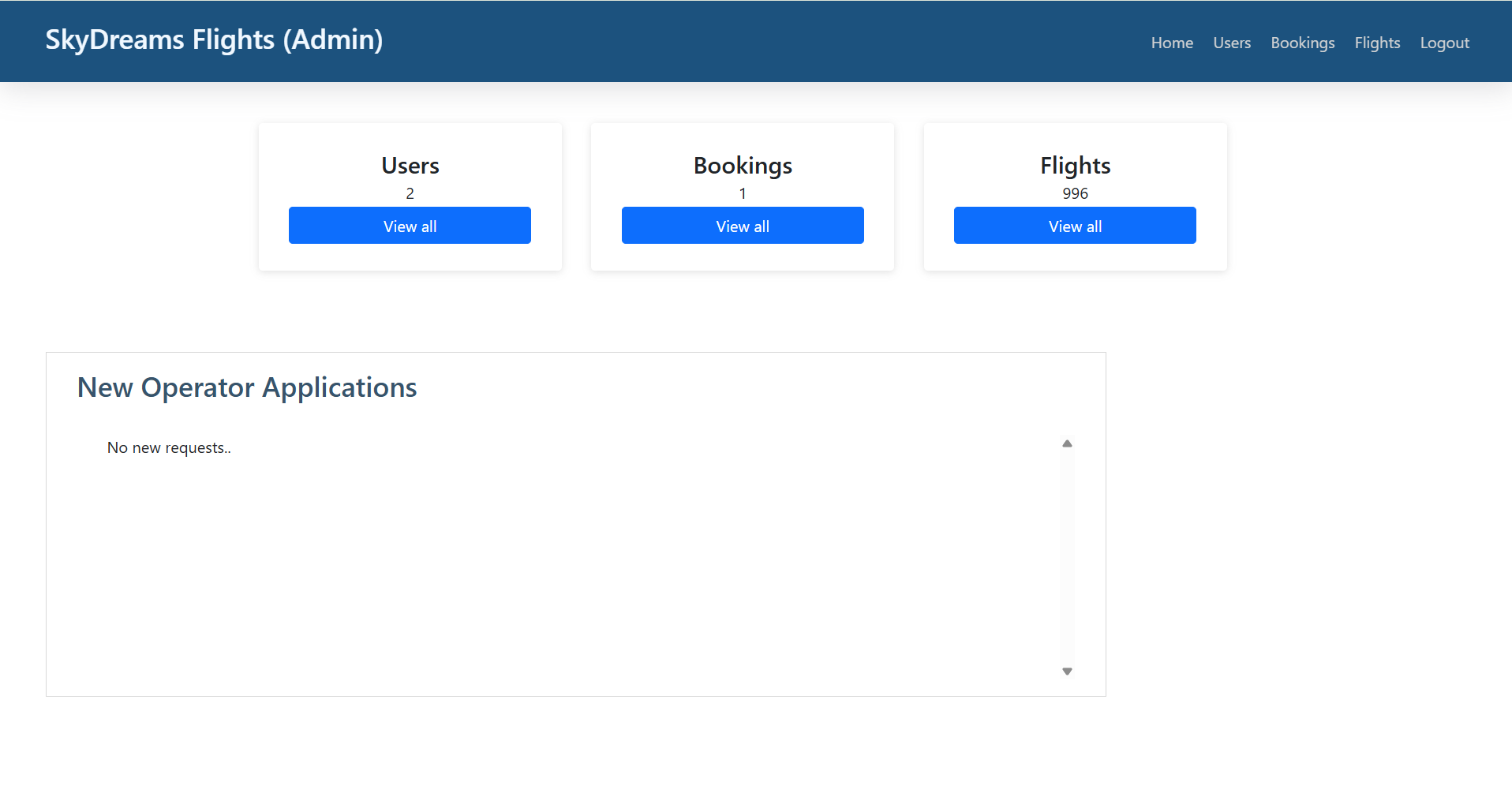
REGISTER PAGE :



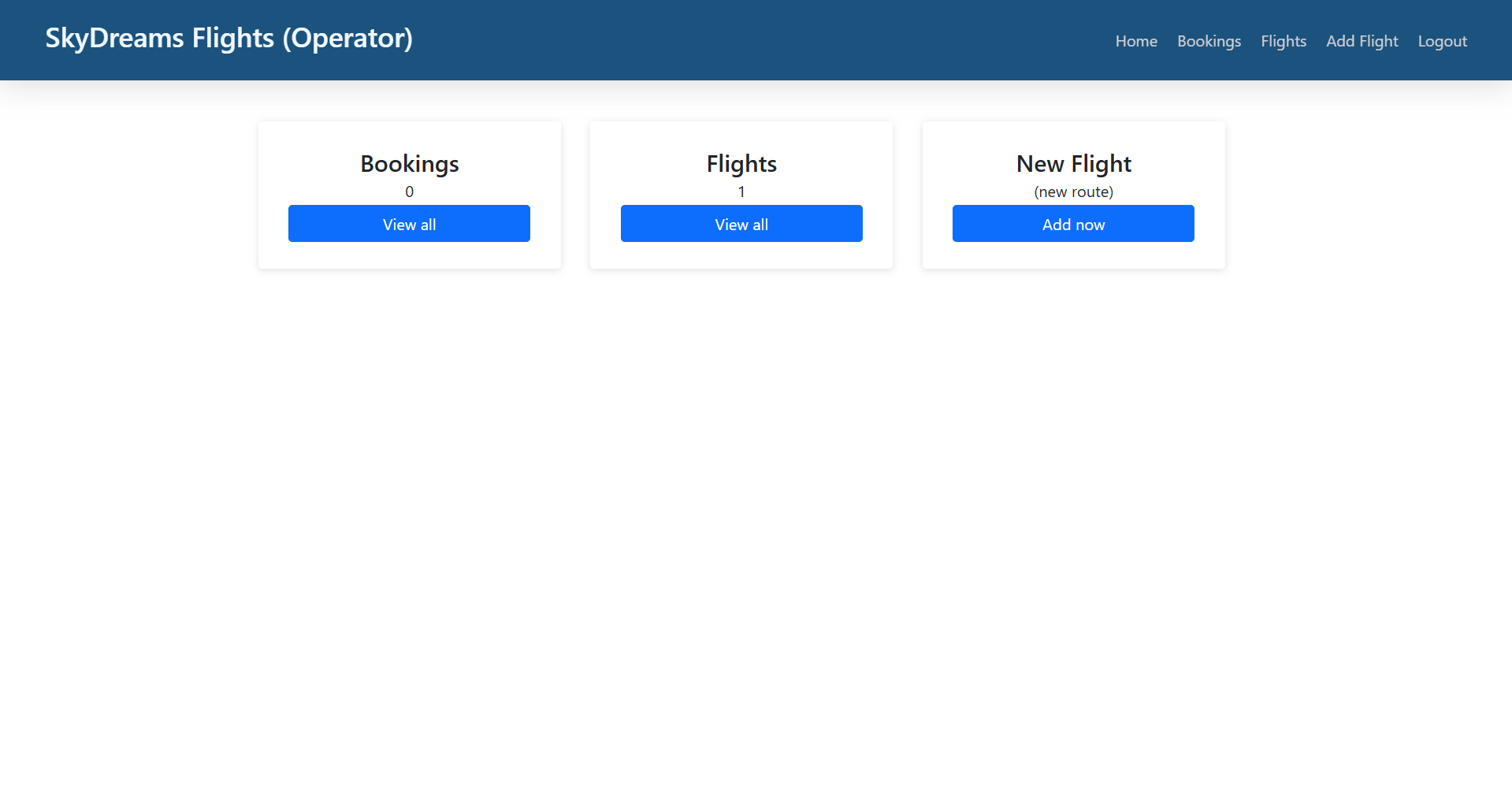
USER BOOKINGS:



ADMIN PAGE:



FLIGHT OPERATOR PAGE:



**10. Testing**

**🔹 Testing Strategy:**

* Frontend Testing:  
  Components tested using Jest and React Testing Library to ensure pages like login, registration, and flight search behave as expected.
* Backend API Testing:  
  Used Postman to test key APIs like registration, login, flight fetch, booking, and admin/operator routes.
* Manual Testing:  
  End-to-end flow (login → search → book → view bookings) was manually tested across different roles (User, Admin, Flight Operator).

**11. Known Issues**

* Flight status or booking history may not refresh automatically without reloading the page.
* Payment is simulated only; no real payment gateway integration yet.
* Admin/operator dashboards have limited responsiveness on small mobile screens.
* No email confirmation implemented after booking.

**12. Future Enhancements**

* Integrate a real payment gateway (like Razorpay or Stripe).
* Add email or SMS notifications after successful bookings.
* Implement seat selection and seat availability logic.
* Add real-time flight tracking (for admin/operator panel).
* Improve mobile responsiveness for all dashboards.
* Support multi-language interface for wider accessibility.