**FlightFinder – Smart Flight Booking System**

**Team Members:**

***TEAM ID:*** LTVIP2026TMIDS65671

|  |  |
| --- | --- |
| **MEMBER** | **RESPONSIBILITY** |
| 1. Bogolu Sri Siva Kumar | Testing & Documentation. |
| 2. Annavarapu Tharun Sai | Frontend & Backend. |
| 3. Dudekula Saida Vali | Resource Gathering & Database Connection. |
| 4. Boddupalli Deepak | Frontend & Documentation. |

**1. Introduction**

***Project Title:*** FlightFinder – Smart Flight Booking System

**2. Project Overview**

1. ***Purpose:***  
   FlightFinder is a full-stack web application designed to simplify and modernize the flight search and booking process. The goal is to eliminate the need for checking multiple websites by providing a centralized platform where users can search flights, compare options, book tickets, and manage reservations efficiently.
2. ***Features:***

● User registration and secure login (User & Admin)  
● Search flights by source, destination, and date  
● Real-time seat availability updates  
● Book and cancel flight tickets  
● View booking history  
● Admin dashboard for managing flights and seat capacity  
● Role-based authentication using JWT  
● RESTful API integration between frontend and backend  
● Future-ready payment gateway integration  
● Responsive UI for desktop and mobile users

**3. Architecture**

1. ***Frontend:***

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

● Component Structure: Functional components with React Hooks (useState, useEffect)  
● Routing: React Router v6 for navigation  
● State Management: Context API for authentication and booking state  
● UI Styling: CSS / Bootstrap / Tailwind (based on your implementation)  
● HTTP Requests: Axios for backend API communication

1. ***Backend:***  
   The backend is built using **Node.js and Express.js**:

● RESTful API endpoints for flights, users, and bookings  
● Middleware: cors, dotenv, express.json()  
● Authentication: JWT-based middleware for protected routes  
● Modular folder structure: Routes, Controllers, Models  
● Seat allocation and booking logic handling

1. ***Database:***

**MongoDB (NoSQL)** is used with **Mongoose ORM** for schema modelling Collections include:

● Users (User/Admin roles)  
● Flights (source, destination, time, seat capacity)  
● Bookings (user reference, flight reference, seat details)

MongoDB Atlas is used for cloud-based scalable storage.

**4. Setup Instructions**

1. ***Prerequisites:***

Ensure you have the following installed:

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

1. ***Installation Steps:***
2. Clone the repository
   * git clone <https://github.com/your-username/FlightFinder.git>
   * cd FlightFinder
3. Set up environment variables:

Create .env file in the server directory

* PORT=5000
* MONGO\_URI=your\_mongo\_url\_here
* JWT\_SECRET=mysecretpassword

1. Install dependencies:

#Backend

cd server

npm install

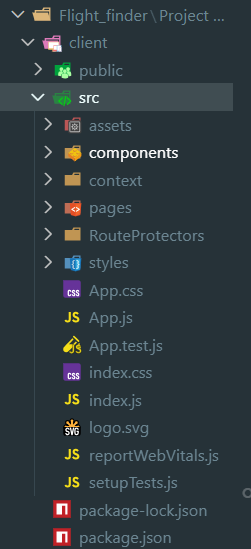
#Frontend

cd ../client

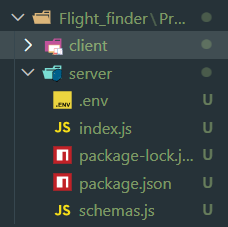
npm install

***5*.Folder Structure:**

***Frontend:***

****

***Backend:***

****

**6. Running the Application**

To start both servers:

***Frontend:***

Navigate to the client directory:

cd client

Start the development server:

npm start

The frontend application runs on: http://localhost:3000

***Backend:***

Navigate to the server directory:

cd server

Start the backend server:

npm start

The backend application runs on: http://localhost:5000

**7. API Documentation**

**Base URL:**

<http://localhost:5000/api>

**Authentication Endpoints**

• **POST /auth/register** – Register a new user (User/Admin)  
• **POST /auth/login** – Login user and return JWT token

**Flight Endpoints**

• **GET /flights** – Get all available flights (with optional filters: source, destination, date)  
• **POST /flights** (Admin – Protected) – Create a new flight  
• **PUT /flights/:id** (Admin – Protected) – Update flight details  
• **DELETE /flights/:id** (Admin – Protected) – Delete a flight

**Booking Endpoints**

• **GET /bookings** (Protected) – Get all bookings of logged-in user  
• **POST /bookings** (Protected) – Create a new booking  
• **DELETE /bookings/:id** (Protected) – Cancel a booking

**User Endpoints**

• **GET /users/me** (Protected) – Get logged-in user profile  
• **GET /users** (Admin – Protected) – Get all registered users

**8. Authentication**

Authentication in FlightFinder is handled using **JWT (JSON Web Tokens)**:

• Upon successful login, the system generates a JWT token.

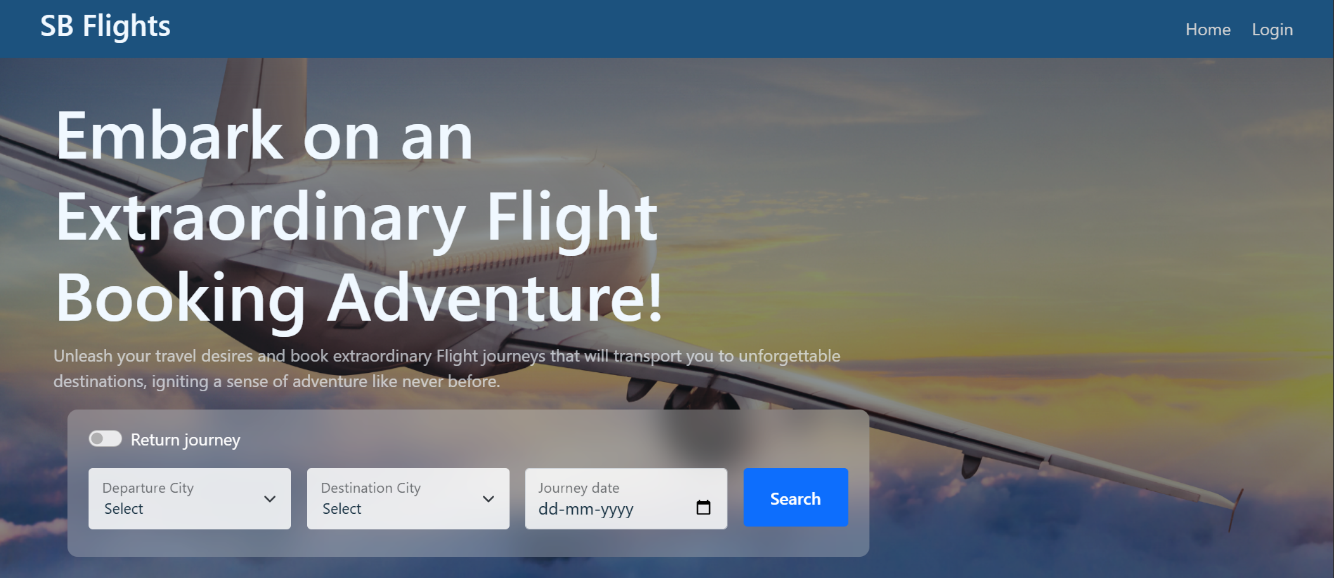
• The token is sent to the client and stored in the browser's localStorage.

• All protected routes require a valid JWT token for access.

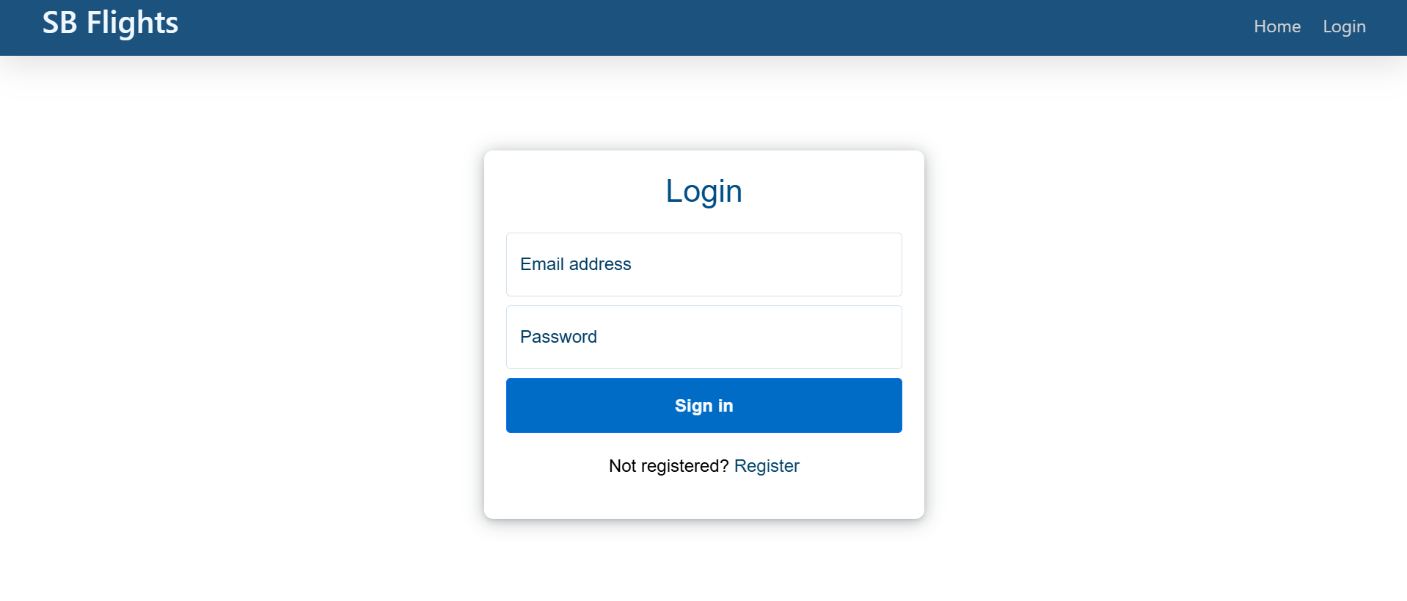
• The token must be included in the Authorization header as:  
Bearer <token>

• Backend middleware verifies the token before granting access.

**9. Screenshots or Demo**

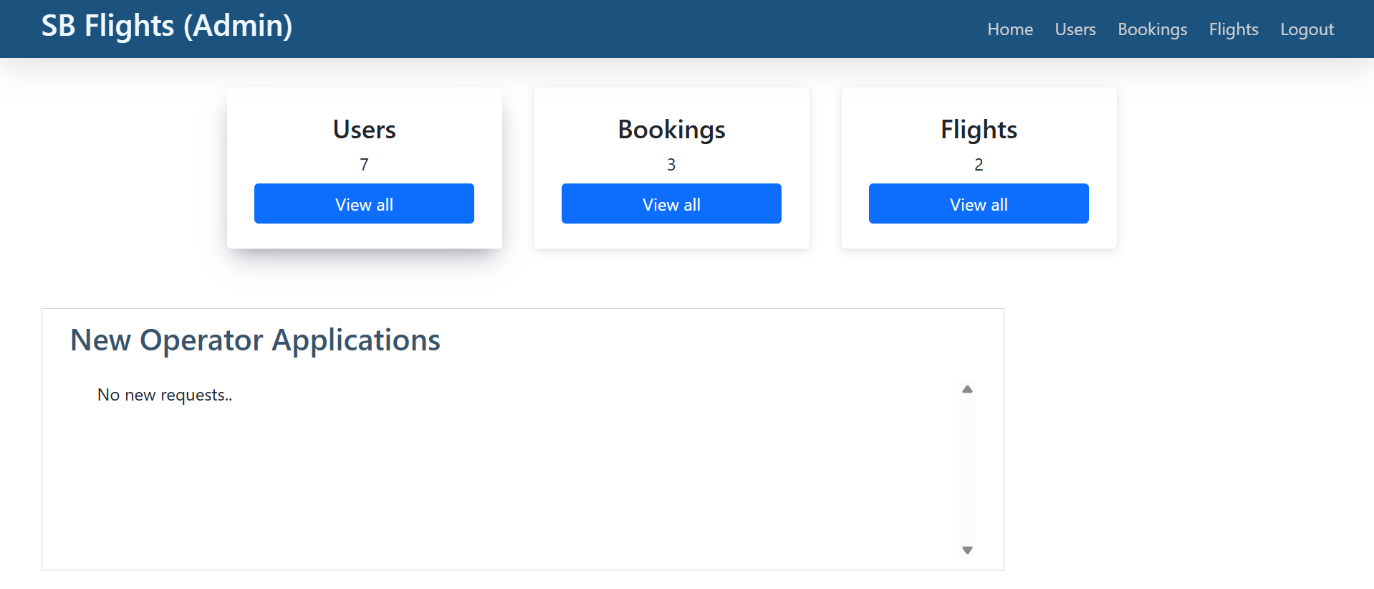
 Home Page

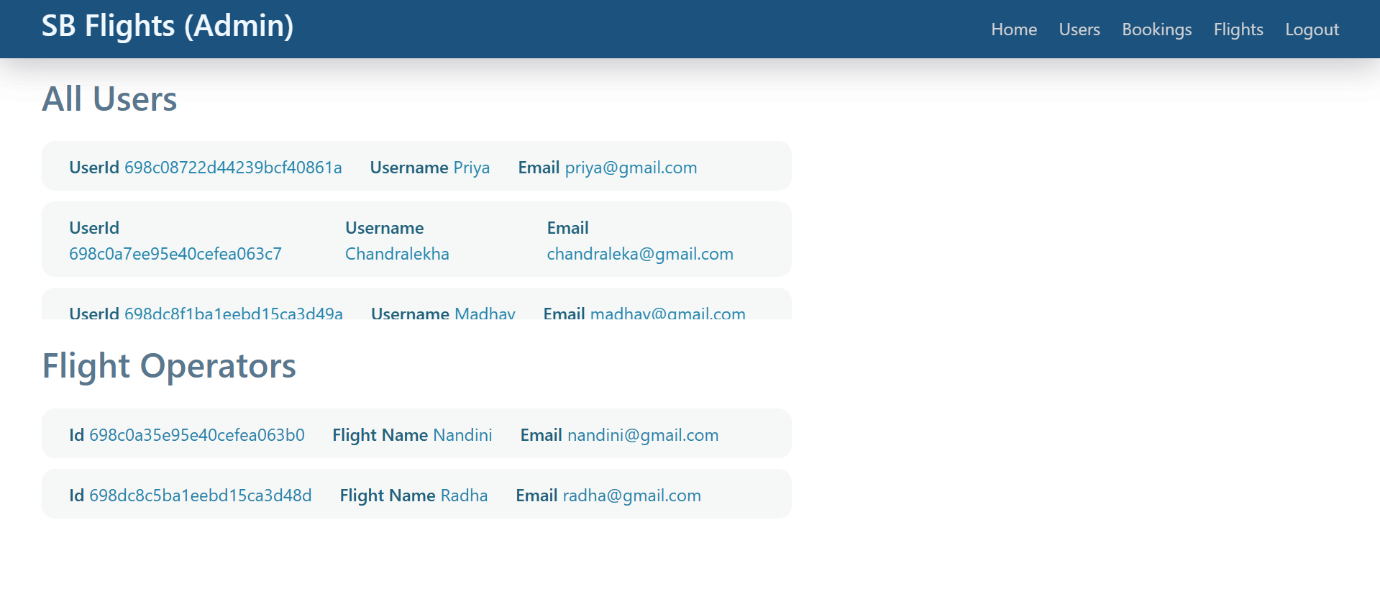
Login Page:

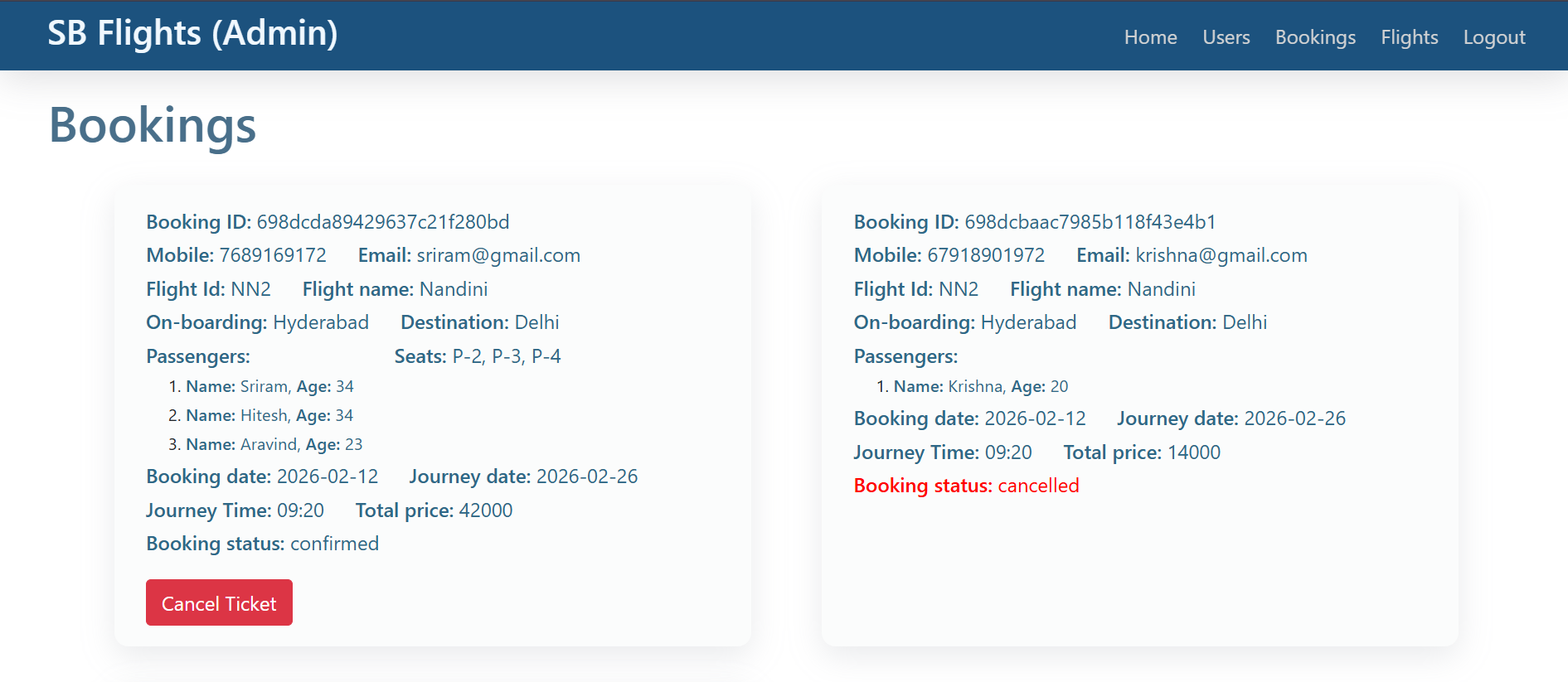


Register Page:

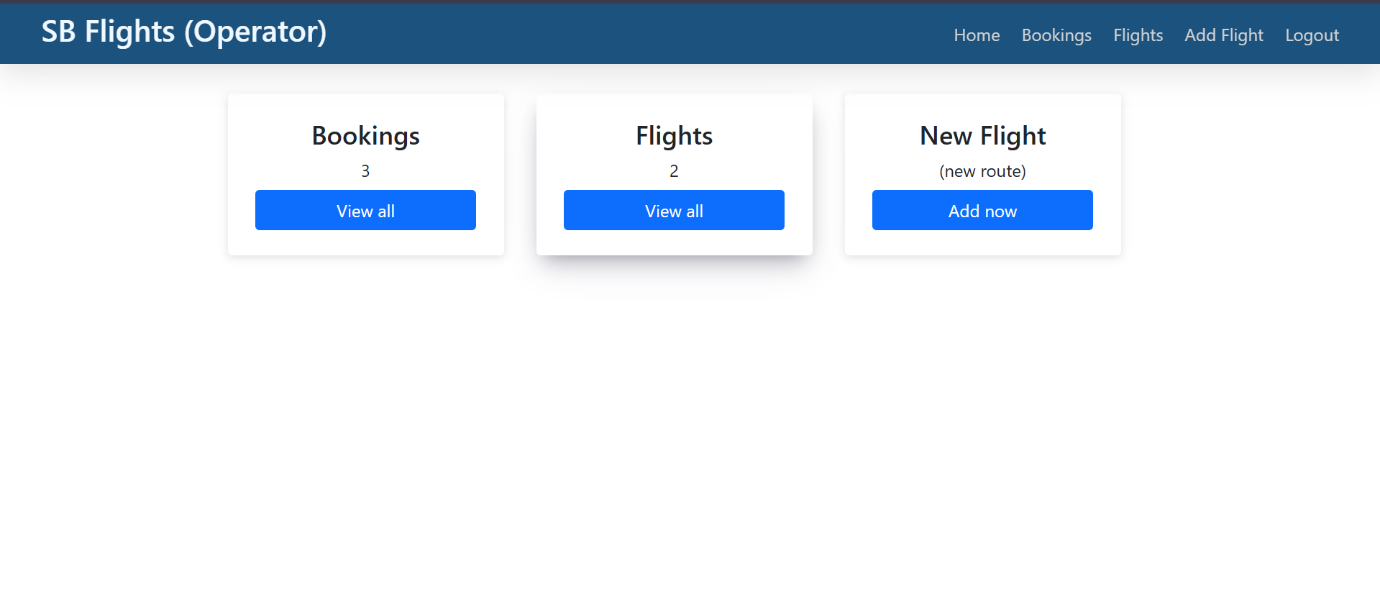
Admin Operations:

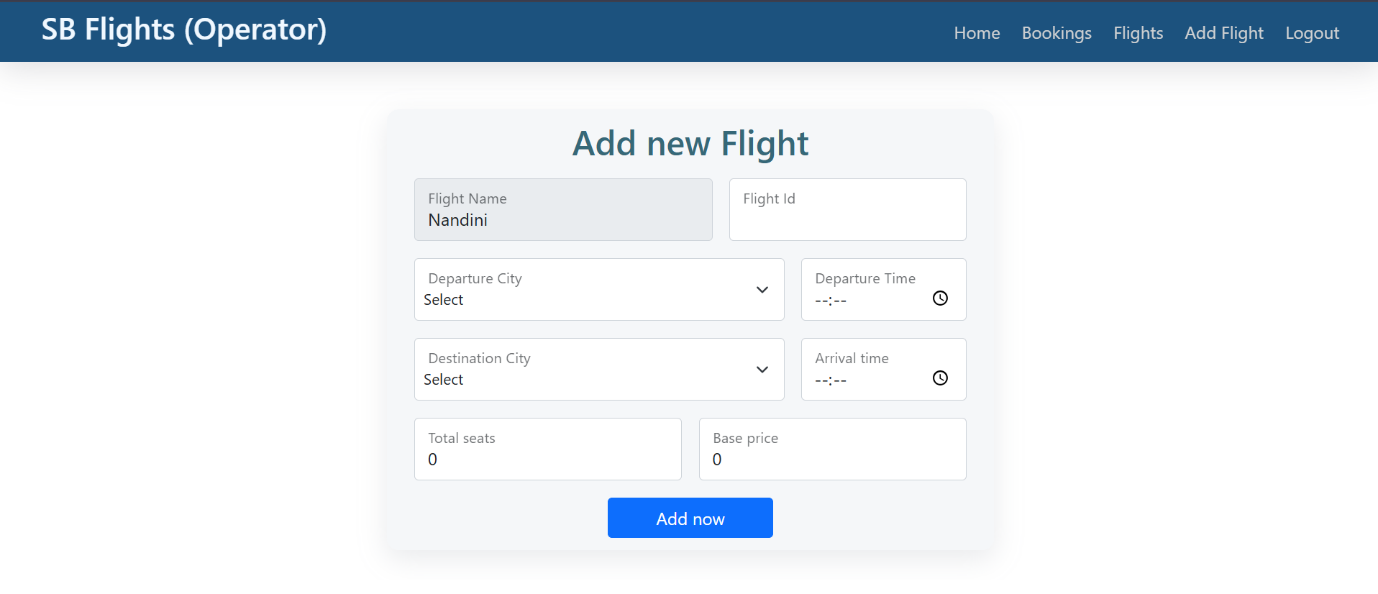


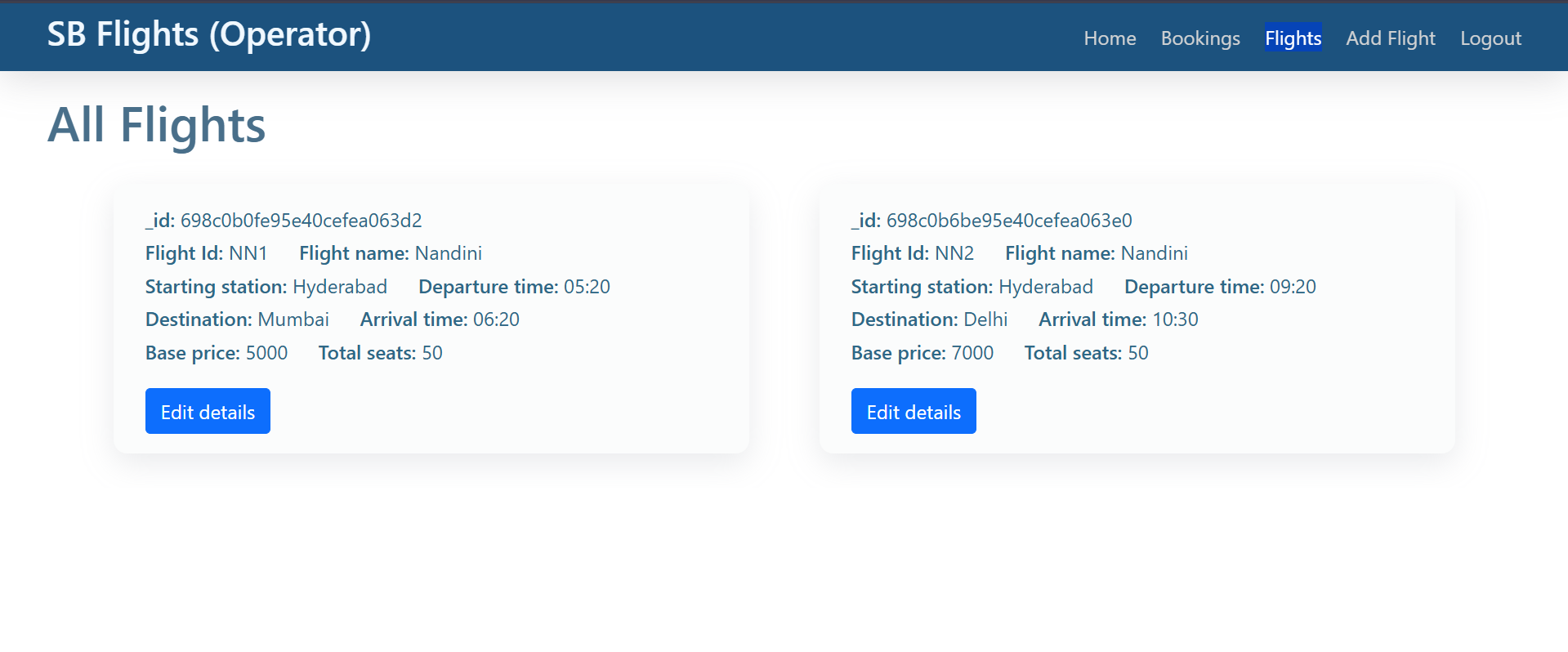




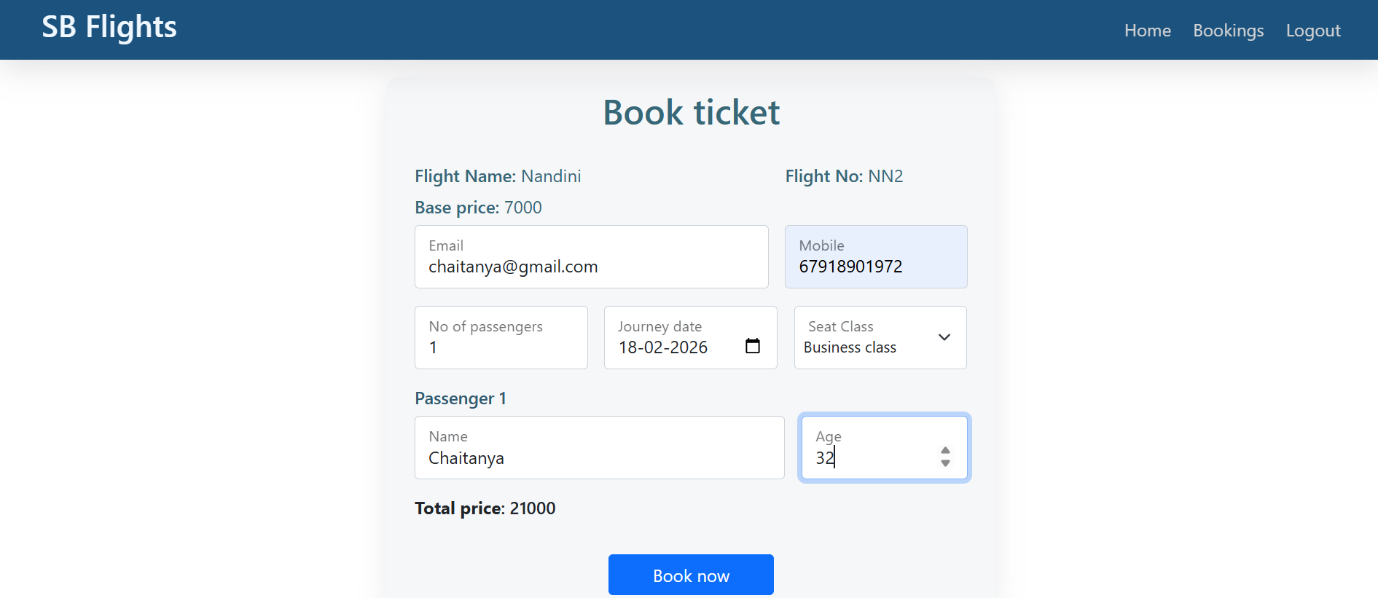
Flight Operator







User operations:



**10. Testing**

***Testing Strategy:***

• Frontend components are tested using **Jest** and **React Testing Library** to verify UI rendering, state updates, and user interactions.

• Backend APIs are tested using **Mocha and Chai** to validate responses, authentication, booking logic, and error handling.

• API endpoints are also tested using **Postman** for manual validation of requests and responses.

• End-to-end testing is performed using **Cypress** to simulate real user actions such as registration, login, flight search, and booking.

• Authentication and role-based access control are tested to ensure protected routes work correctly.

**11. Known Issues**

• Payment gateway integration is currently not implemented (future enhancement ready).

• Booking cancellation does not yet support refund processing logic.

• Admin dashboard responsiveness may be limited on smaller mobile screens.

• No email/SMS notification system implemented yet (manual confirmation only).

.

**12. Future Enhancements**

• Integrate secure online payment gateway (Razorpay / Stripe).

• Implement real-time seat updates using WebSockets.

• Add email and SMS notifications for booking confirmation and cancellation.

• Introduce price prediction or smart flight recommendations using AI/ML.

• Add hotel and travel package booking features.

• Develop a mobile application using React Native.

• Add multi-language and multi-currency support for global users.

• Implement refund and cancellation policy automation.