# FULL STACK DEVELOPMENT WITH MERN PROJECT DOCUMENTATION

## **INTRODUCTION**

Project Title :: **flightfinder: navigating your air travel options**

Team id : LTVIP2025TMID54243

Team Member:

N.MOHITH VIJAY SAI– over all developer

## PROJECT OVERVIEW

**FlightFinder** is a full-stack flight booking web application built using the **MERN stack** (MongoDB, Express.js, React.js, Node.js). The platform is designed to modernize and simplify the air travel booking process for users by offering a seamless, responsive, and intuitive interface to search, filter, and reserve flight tickets.

This application allows users to:

* Register and securely log in
* Search for flights using parameters like departure city, destination, dates, travel class, and number of passengers
* View real-time results from multiple airlines
* Filter options by direct flights, preferred airlines, and departure time
* Select seats interactively through a seat map
* Make secure payments through an integrated payment gateway
* Receive instant booking confirmation and downloadable e-tickets

In addition to the user-facing interface, the app includes an **admin dashboard** that enables administrators to:

* Add, update, or delete flight listings
* View booking records
* Monitor platform usage

By combining robust backend functionality with a user-friendly frontend, FlightFinder delivers a comprehensive and scalable solution for managing air travel services efficiently.

.

### Description

**FlightFinder** is a next-generation web application that transforms the flight booking experience into a smooth, efficient, and user-friendly process. Designed using the MERN stack, FlightFinder offers an integrated platform where users can search, book, and manage flight reservations with just a few clicks.

The application caters to the needs of a wide range of users—from business professionals booking last-minute trips to families planning their vacations. Users can easily input travel details such as departure city, destination, date, travel class, and number of passengers. The system fetches real-time flight options from a database, displaying them in an organized and filterable format.

To enhance the user experience, FlightFinder includes:

* A responsive and interactive interface built in **React.js**
* Seamless communication with backend APIs using **Axios**
* Robust **Node.js/Express.js** services handling authentication, search, booking, and payment logic
* A secure **MongoDB** database storing user information, flight details, and booking records
* **JWT-based authentication** for session security and role-based access control (user/admin)

Furthermore, FlightFinder offers a visually intuitive **seat selection interface**, enabling users to pick their desired seats before payment. The system integrates with a **mock payment gateway** for demo purposes, simulating a real-world transaction experience.

For administrators, the backend provides a dedicated dashboard to:

* Manage flights
* Monitor user activity
* View and manage bookings

**FlightFinder** brings together modern web technologies to create a fast, secure, and reliable flight reservation system that can be further scaled for real-world deployment.

### Scenario based case-study

### ****John’s Journey: A Seamless Flight Booking Experience****

**John**, a busy business executive based in New York City, needs to attend a technology conference in Paris. Due to his packed schedule, he prefers a quick and reliable method to book his travel without having to visit multiple airline websites.

He opens the **FlightFinder** app on his laptop and begins his booking journey:

### 🛫 Step 1: Entering Travel Preferences

John navigates to the homepage and inputs the following details:

* **Departure City**: New York City
* **Destination**: Paris
* **Departure Date**: April 10
* **Return Date**: April 15
* **Travel Class**: Business
* **Passengers**: 1

He clicks **“Search Flights”** and within seconds, FlightFinder displays a list of available flights from multiple airlines.

### 🧭 Step 2: Filtering the Results

To match his preferences, John uses the filtering options to:

* Show **only direct flights**
* Select flights **departing in the evening**
* Choose his **preferred airline** (based on loyalty programs)

The updated list shows John the most relevant results, with clear indicators of price, flight duration, and seat availability.

### 💺 Step 3: Seat Selection

John selects a flight and proceeds to the seat selection interface. A visual seat map of the business class cabin appears. John chooses a **window seat with extra legroom**, clearly marked as available.

### 💳 Step 4: Secure Booking

He moves on to payment. Using FlightFinder’s **secure payment gateway**, John enters his card details and confirms the booking.

### 📧 Step 5: Confirmation

In just a few seconds, John receives:

* A **confirmation message** on the screen
* An **email with the booking details and e-ticket**
* Access to his itinerary and seat number via the **“My Bookings”** page

### 🧑‍💼 John’s Review

Impressed by the ease and speed of the entire process, John saves FlightFinder to his bookmarks and decides to use it for all future travel. The intuitive interface, real-time filters, and professional presentation make his journey stress-free and efficient.

This case study illustrates how **FlightFinder** streamlines the end-to-end travel booking process, delivering convenience and customization for today’s travelers.

### TECHNICAL ARCHITECTURE

The **FlightFinder** application is built on the robust and scalable **MERN stack**, which includes **MongoDB**, **Express.js**, **React.js**, and **Node.js**. The architecture follows a classic **client-server model** and is divided into three main layers: frontend, backend, and database.

### 🔹 1. Frontend (Client Side)

Built with **React.js**, the frontend provides a responsive and dynamic user interface that interacts with backend APIs via **Axios**. Key features include:

* User Authentication (Login/Register)
* Flight Search and Filtering Interface
* Real-time Seat Selection UI
* Booking and Payment Forms
* User Booking History Page

Libraries Used:  
✅ React Router DOM for navigation  
✅ Bootstrap / Material UI for styling  
✅ Axios for API requests

### 🔹 2. Backend (Server Side)

The backend is powered by **Node.js** with the **Express.js** framework. It handles all business logic, authentication, routing, and interaction with the database.

**Main Modules Include:**

* User Management (register, login, profile)
* Flight Management (create, fetch, update, delete flights)
* Booking System (create bookings, assign seats)
* Payment Processing (mock gateway integration)
* Admin Panel Routes (role-based access control)

Security:

* JWT-based authentication and authorization
* Input validation and error handling
* CORS and Helmet for API protection

### 🔹 3. Database (MongoDB)

**MongoDB** is used for storing structured data in collections such as:

* users: Stores user credentials, roles (user/admin), and profiles
* flights: Stores flight details like origin, destination, timings, airline, seat map
* bookings: Links users to flights with seat number and payment status

Access to MongoDB is managed through **Mongoose**, an ODM (Object Data Modeling) library that ensures structured schemas and relationships.

### 🔄 How It Works (Simplified Flow)

sql

CopyEdit

User (React Frontend)

|

[Axios API Request]

|

Express.js (Node Backend)

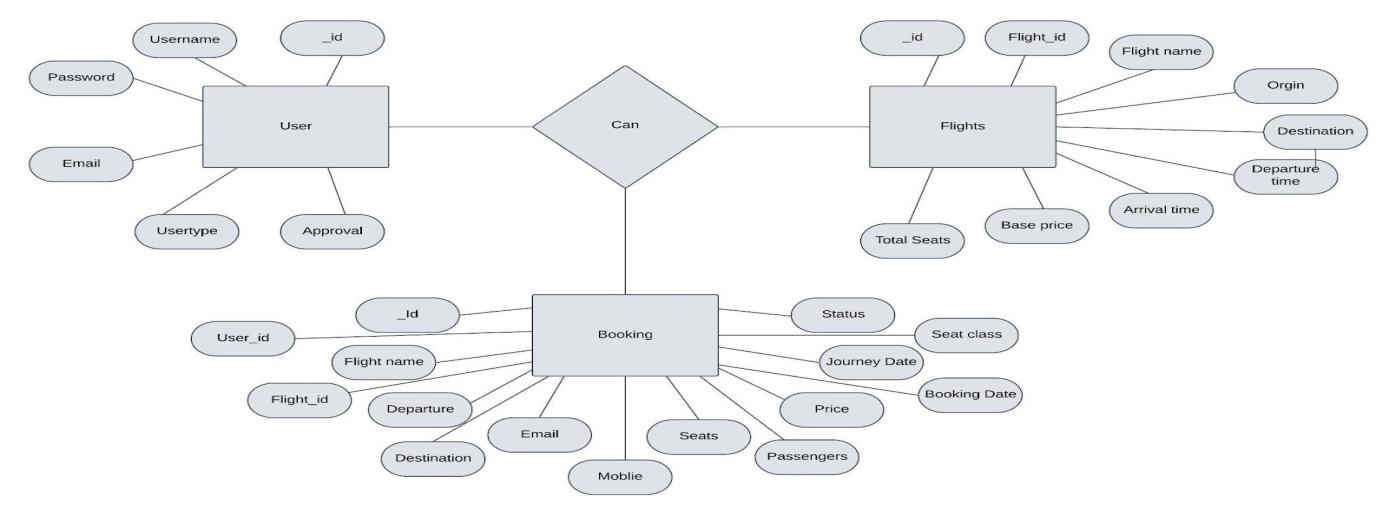
|

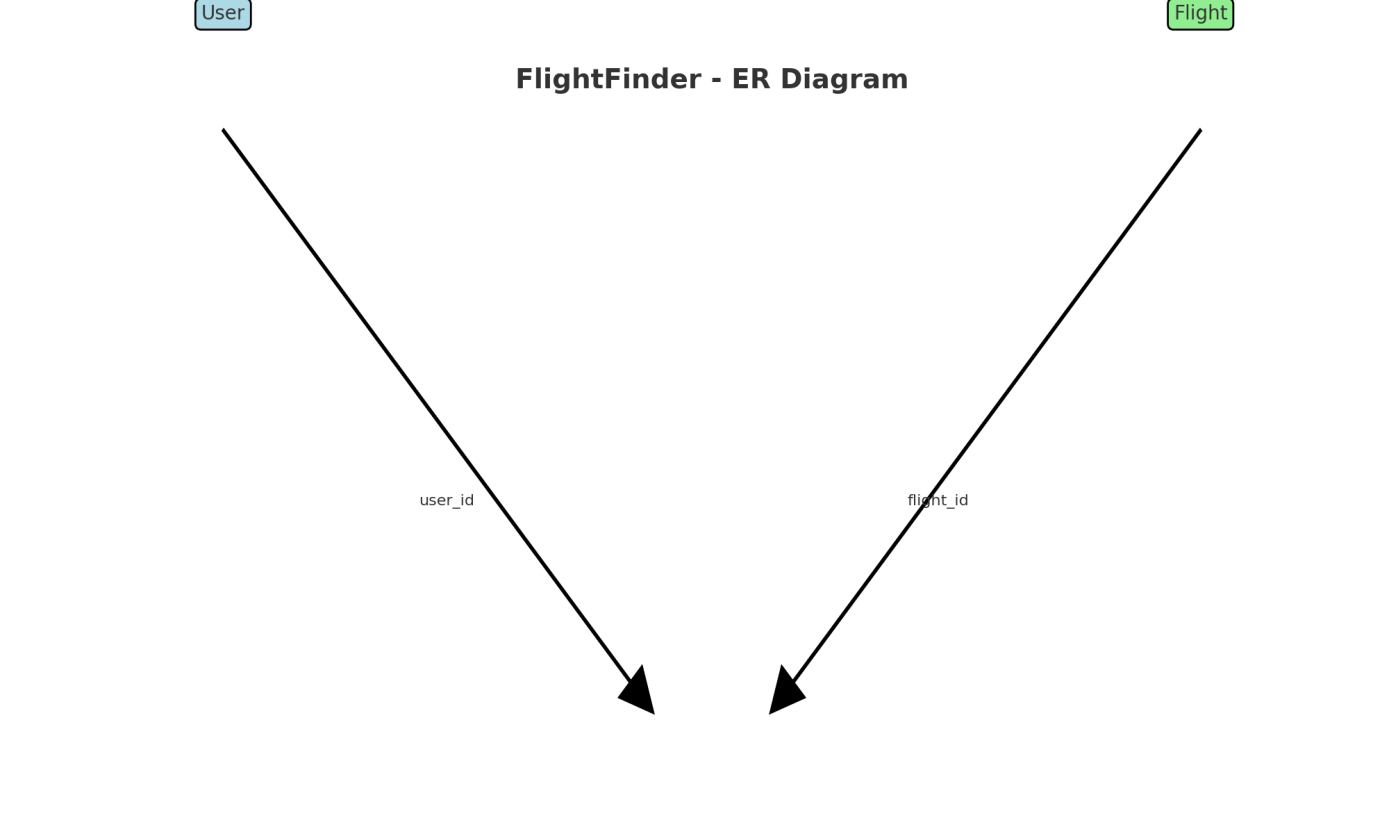
MongoDB (Cloud or Local)

### 🌐 Deployment (Optional)

* **Frontend**: Hosted on Vercel or Netlify
* **Backend**: Deployed on Render or Railway
* **Database**: MongoDB Atlas for cloud-hosted database

### ER DIAGRAM





The **FlightFinder** system is built around three main entities:

### 🧑‍💼 1. ****User****

Represents individuals who use the platform (travelers or admins).

**Attributes:**

* user\_id (Primary Key)
* name
* email
* password
* role (user/admin)

### ✈️ 2. ****Flight****

Represents available flights in the system.

**Attributes:**

* flight\_id (Primary Key)
* airline
* origin
* destination
* departure\_time
* arrival\_time
* date
* class (economy/business)
* available\_seats
* price

### 🧾 3. ****Booking****

Represents reservations made by users.

**Attributes:**

* booking\_id (Primary Key)
* user\_id (Foreign Key → User)
* flight\_id (Foreign Key → Flight)
* seat\_number
* booking\_date
* status (confirmed/cancelled)
* payment\_status (paid/unpaid)

### 🔗 Relationships:

* One **User** can make many **Bookings**
* One **Flight** can have many **Bookings**
* One **Booking** links exactly one **User** and one **Flight**

### PROJECT STRUCTURE

The **FlightFinder** project is organized into two main parts:

1. client/ – Frontend built with **React.js**
2. server/ – Backend API built with **Node.js** and **Express.js**

This modular structure ensures clean separation of concerns, maintainability, and scalability.

### 📁 Folder Layout

csharp

CopyEdit

flightfinder/

├── client/ # React Frontend

│ ├── public/ # Static files

│ └── src/

│ ├── components/ # Reusable UI elements (Navbar, FlightCard)

│ ├── pages/ # Main pages (Login, Search, Booking)

│ ├── utils/ # API helpers (Axios configs)

│ └── App.js # Main app entry

├── server/ # Express Backend

│ ├── models/ # Mongoose schemas (User, Flight, Booking)

│ ├── routes/ # Route handlers (authRoutes, flightRoutes)

│ ├── controllers/ # Business logic (bookings, user auth)

│ ├── middleware/ # Auth middleware (JWT, error handling)

│ └── server.js # Main entry point

├── .env # Environment variables

├── package.json # Project metadata & scripts

└── README.md # Project documentation

### 🔗 Purpose of Each Folder

* **client/** – Hosts the user interface. It communicates with backend APIs to fetch flight data and submit bookings.
* **server/** – Manages database access, authentication, and business logic.
* **models/** – Defines MongoDB schemas for User, Flight, and Booking.
* **routes/** – Exposes RESTful API endpoints.
* **controllers/** – Contains logic to handle each endpoint's behavior.
* **middleware/** – Adds features like JWT validation and error handling.

**PRE-REQUISTIC:**

To develop and run the **FlightFinder** application locally, you need to have the following tools, technologies, and skills:

### ✅ 1. ****Node.js & npm****

* **Why:** Node.js is a JavaScript runtime that powers the backend. npm (Node Package Manager) is used to install dependencies.
* **Download:** <https://nodejs.org>
* **Verify:**

bash

CopyEdit

node -v

npm -v

### ✅ 2. ****MongoDB****

* **Why:** MongoDB is a NoSQL database used to store user, flight, and booking data.
* **Options:**
  + Use **MongoDB Atlas** for cloud storage
  + Or install **MongoDB Community Edition** locally
* **Download:** <https://www.mongodb.com/try/download/community>

### ✅ 3. ****React.js****

* **Why:** React builds the interactive, component-based frontend.
* **Create a new app:**

bash

CopyEdit

npx create-react-app client

### ✅ 4. ****Express.js****

* **Why:** Express.js powers the backend API and routes.
* **Install in server/:**

bash

CopyEdit

npm install express

### ✅ 5. ****Mongoose****

* **Why:** Mongoose is an ODM library for MongoDB. It helps define schemas for collections.
* **Install:**

bash

CopyEdit

npm install mongoose

### ✅ 6. ****Axios****

* **Why:** Axios is used on the frontend to send HTTP requests to the backend.
* **Install:**

bash

CopyEdit

npm install axios

### ✅ 7. ****Other Required Packages****

In the backend (server/):

bash

CopyEdit

npm install dotenv cors jsonwebtoken bcryptjs

In the frontend (client/):

bash

CopyEdit

npm install react-router-dom bootstrap

### ✅ 8. ****Git****

* **Why:** Version control system to manage your project code and track changes.
* **Download:** <https://git-scm.com/downloads>

### ✅ 9. ****Code Editor****

Use a code editor like **Visual Studio Code**:

* Download: <https://code.visualstudio.com>

**Application flow:**

The **FlightFinder** application is designed to deliver a smooth, end-to-end experience for users searching and booking flights, and for admins managing flight data. Here's how each role flows through the system:

### 👤 ****User Responsibilities (Traveler)****

1. **Register / Login**
   * User signs up or logs in via secure JWT-based authentication.
2. **Flight Search**
   * Inputs travel details: source, destination, date, class, passengers.
3. **View & Filter Flights**
   * Sees a list of available flights with options to filter:
     + Direct flights only
     + Preferred airlines
     + Departure/arrival times
4. **Flight Selection**
   * Chooses a flight and proceeds to seat selection.
5. **Seat Selection**
   * Interactively picks a seat (window, aisle, etc.) using a visual seat map.
6. **Booking & Payment**
   * Enters payment info via a secure (mock or real) payment gateway.
   * Confirms booking.
7. **Receive Confirmation**
   * Booking confirmation and e-ticket are displayed and emailed.
   * Booking is saved to user’s history.

### 🧑‍💼 ****Admin Responsibilities****

1. **Admin Login**
   * Accessed via a separate login route with admin privileges.
2. **Manage Flights**
   * Add new flights
   * Update existing flights
   * Delete or disable outdated flights
3. **View Bookings**
   * Monitor all user bookings
   * Check availability and seating data
4. **Monitor System Activity**
   * Review active users
   * Access logs or analytics (if implemented)

### 🔁 Data Flow Summary

csharp

CopyEdit

[User Interface (React)]

⬇

[Express.js API Endpoints]

⬇

[MongoDB (users, flights, bookings)]

### Example Use Flow:

pgsql

CopyEdit

John → Login → Search Flight → Select Flight → Choose Seat → Pay → Booked → Get Ticket

Admin → Login → Add Flights → View All Bookings → Manage Schedule

This flow ensures a well-structured, secure, and user-friendly experience for both travelers and system administrators.

**Project Flow:**

The **FlightFinder** system is designed with a modular, role-based workflow that supports both end-users (passengers) and system administrators. The application flow ensures a seamless experience across flight search, booking, and administrative tasks.

### 👨‍💼 1. ****User Flow (Passenger)****

pgsql

CopyEdit

Register/Login ➜ Search Flights ➜ Filter & Select ➜ Choose Seat ➜ Make Payment ➜ Get E-Ticket

**Steps:**

1. **Authentication**: Users register or log in securely via the frontend.
2. **Search Flights**: Enter source, destination, travel date, class, and passengers.
3. **Flight Results**: Browse flights, apply filters (direct only, preferred airline, time).
4. **Flight Details**: Choose a flight and view pricing, duration, and available seats.
5. **Seat Selection**: Select a seat using a dynamic seat map.
6. **Payment**: Proceed to payment via integrated gateway (Stripe or mock).
7. **Confirmation**: Booking is stored in the database, and an e-ticket is generated.

### 🧑‍💼 2. ****Admin Flow****

pgsql

CopyEdit

Login ➜ Add/Edit Flights ➜ View Bookings ➜ Manage Users ➜ Monitor Activity

**Steps:**

1. **Admin Login**: Only authorized admins can access backend tools.
2. **Flight Management**: Add new flights or update/delete existing ones.
3. **Booking Records**: Access full booking history for all users.
4. **Schedule Updates**: Modify dates, times, and pricing.
5. **System Oversight**: Monitor real-time data and usage statistics (if enabled).

### 🧾 3. ****Backend Flow (API)****

1. Frontend sends requests using Axios.
2. Express.js routes handle endpoints:
   * /api/users (register, login)
   * /api/flights (search, create, update)
   * /api/bookings (create, fetch, cancel)
3. Controllers process the logic and access MongoDB using Mongoose.
4. Responses are returned to the frontend in JSON format.

### 📊 Summary Diagram (Text Version)

csharp

CopyEdit

[User Interface - React.js]

↓

[Axios HTTP Requests]

↓

[Backend API - Express.js]

↓

[Mongoose Models - User, Flight, Booking]

↓

[MongoDB Atlas - Cloud Database]

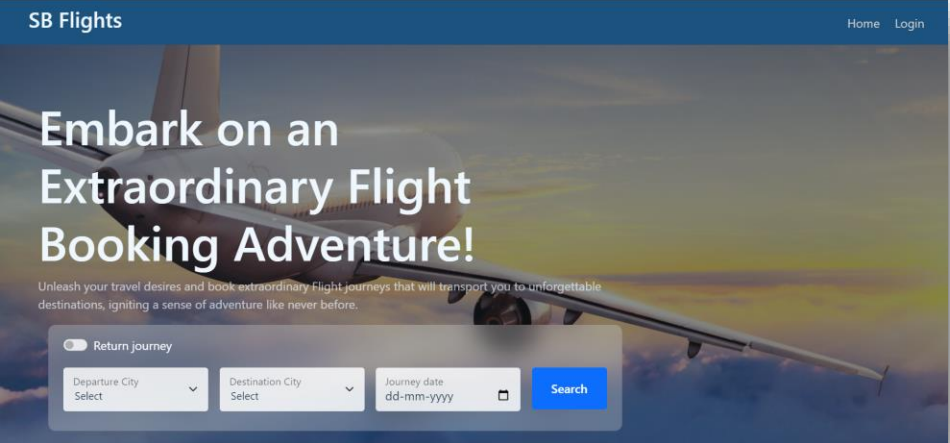
This flow ensures:

* A modular full-stack design
* Clean separation of responsibilities
* Smooth user-to-database communication
* Admin-level control without affecting user experience

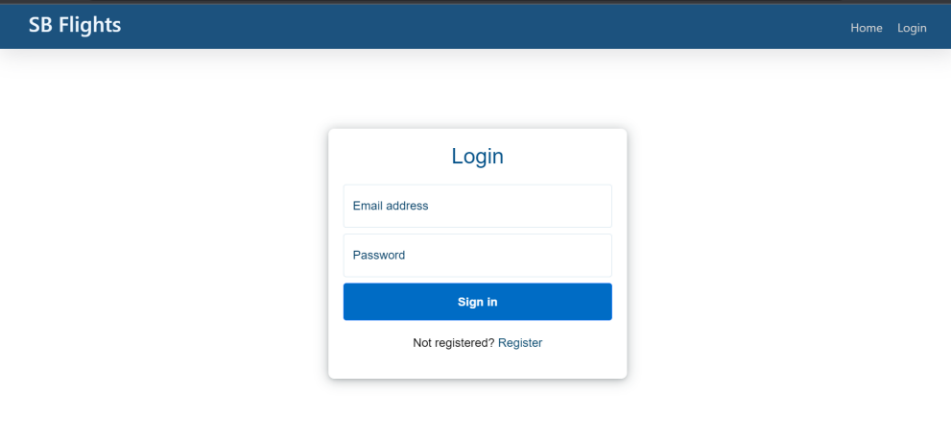
### Project Implementation

Finally, after finishing coding the projects we run the whole project to test it’s working process and look for bugs. Now, let’s have a final look at the working of our video conference application

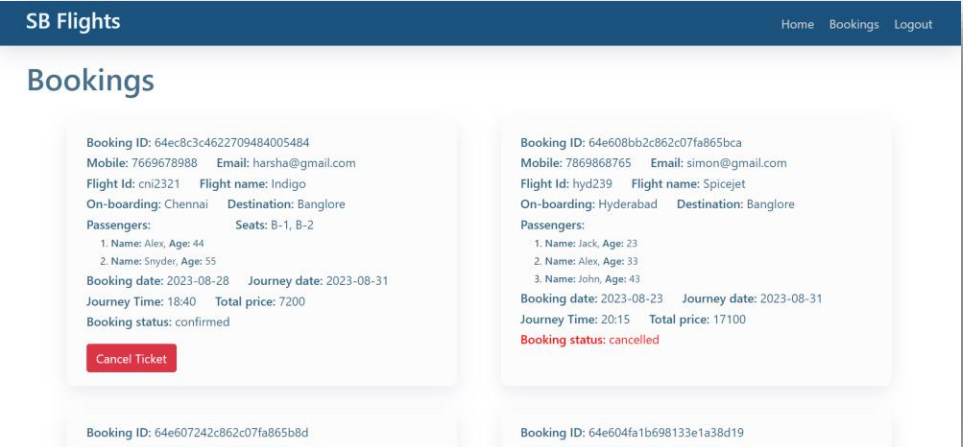
* **Landing page UI**

****

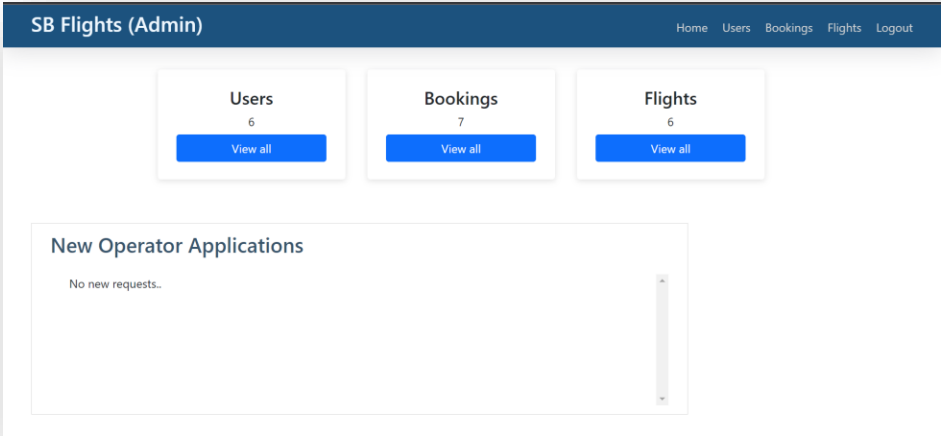
* **Authentication**

****

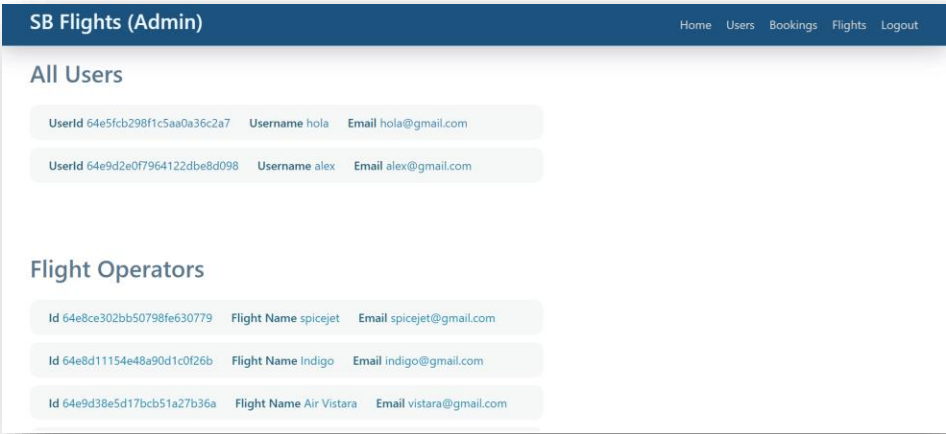
* **User bookings**

****

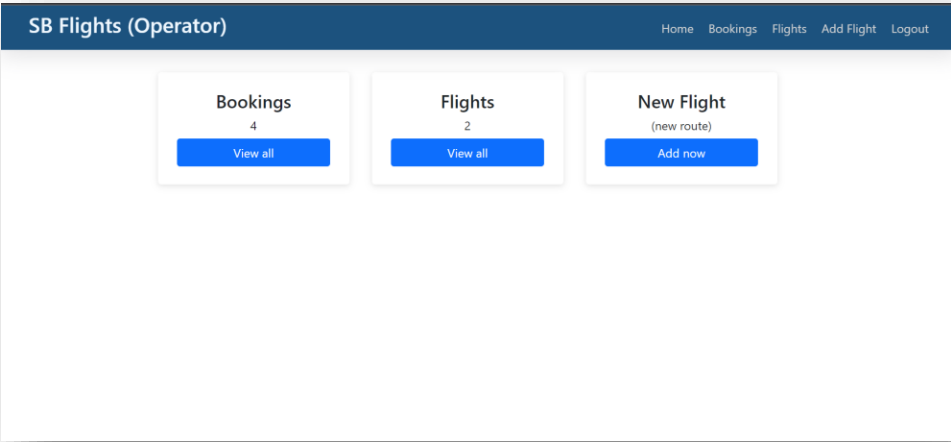
* **Admin Dashboard**

****

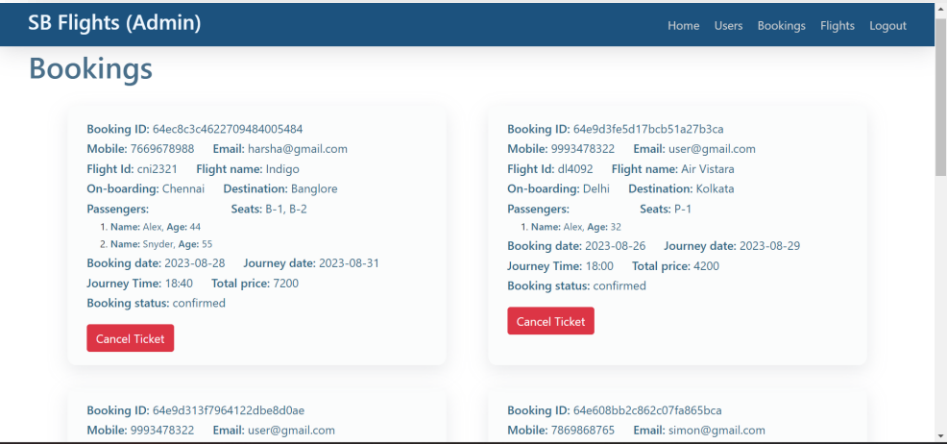
* **All users**

****

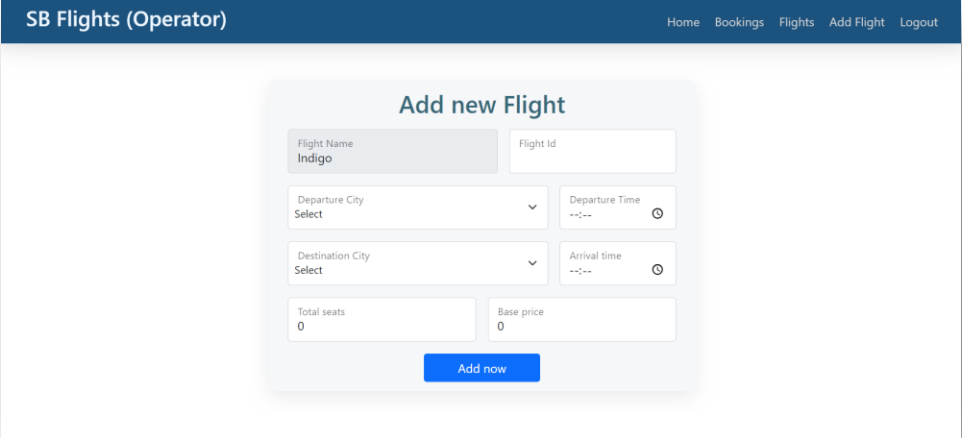
* **Flight Operator**

****

* **All Bookings**

****

* **New Flight**



Use the code in :<https://github.com/MOHITH24-NOWLURI/FLIGHT-BOOKING-FLATFORM.git>

Demo video:

<https://drive.google.com/file/d/1Q0XwKtAz7EkaKNJv3_gbo6mZE9nfuBTK/view>