# Flight Finder: Navigating Your Air Travel Options

**Team ID :** LTVIP2025TMID58325

**Team Leader :** G Satish Kumar

**Team member :** Gabbada S V N Pramod Kumar

**Team member :** Gadi Amulya Satya Mida

**Team member :** Ganesh Chepuri

**INTRODUCTION :**

This Flight Booking APP is the ultimate digital platform designed to revolutionize the way you book flight tickets. With this app your flight travel experience will be elevated to new heights of convenience and efficiency. Our user-friendly web app empowers travelers to effortlessly discover, explore, and reserve flight tickets based on their unique preferences. Whether you're a frequent commuter or an occasional traveler, finding the perfect flight journey has never been easier.

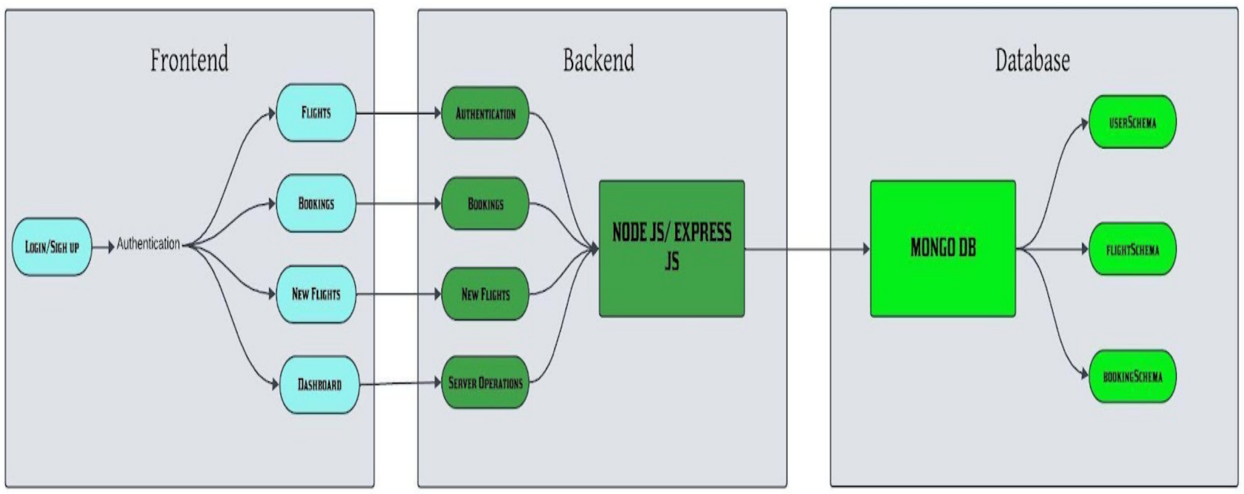
This successful flight booking app combines a user-friendly interface, efficient search and booking capabilities, personalized features, robust security measures, reliable performance, and continuous improvement based on user feedback.

## Project Overview

- John, a frequent traveler and business professional, needs to book a flight for an upcoming conference in Paris. He prefers using a flight booking app for its convenience and features.
- John opens the flight booking app on his smartphone and enters his travel details for Departure as New York City, Destination as Paris, Date of Departure on April 10th and return on April 15th and Class as Business class, Number of passengers as 1
- The app quickly retrieves available flight options based on John's preferences. He sees a range of choices from different airlines, including direct flights and those with layovers. The results show details such as price, airline, duration, and departure times.

- Using the app's filters, John narrows down the options to show only direct flights with convenient departure times. He also selects his preferred airline based on past experiences and loyalty programs.
- After choosing a flight, John proceeds to select his seat in the business class cabin. The app provides a seat map with available seats highlighted, allowing John to pick a window seat with extra legroom.
- John securely enters his payment information using the app's integrated payment gateway. The app processes the payment and generates a booking confirmation with his e-ticket and itinerary details.
- This scenario demonstrates how a flight booking app streamlines the entire travel process for users like John, offering convenience, customization, and real-time assistance throughout their journey.

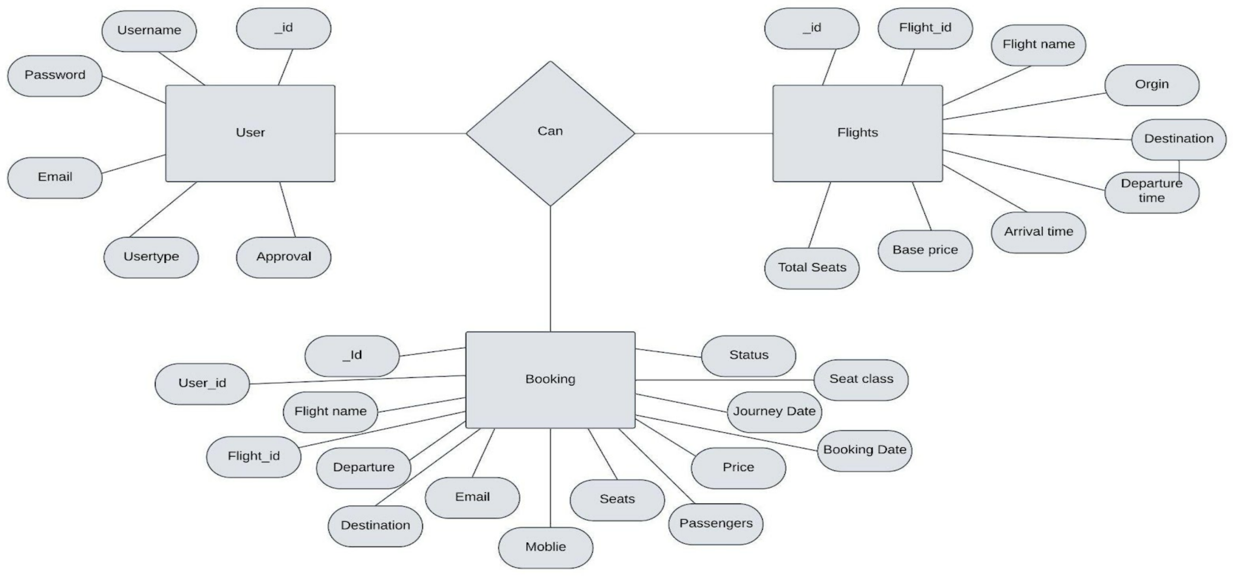
## TECHNICAL ARCHITECTURE



In this architecture diagram:

- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Flight Search, and Booking.
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Flights, Admin and Bookings. It also includes Admin Authentication and an Admin Dashboard.
- The Database section represents the database that stores collections for Users, Flights, and Flight Bookings.

## ER- Diagram



The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

**USER:** Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.

**BOOKING:** Represents a specific flight booking made by a customer. A booking includes a particular flight details and passenger information. A customer can have multiple bookings.

**FLIGHT**: Represents a flight that is available for booking. Here, the details of flight will be provided and the users can book them as much as the available seats.

**ADMIN**: Admin is responsible for all the backend activities. Admin manages all the bookings, adds new flights,etc.

## PRE-REQUISITES

To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application

**Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

**MongoDB**: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.

**Express.js**: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing,middleware, and API development.

**React.js**: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces

**HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling,and JavaScript for client-side interactivity is essential.

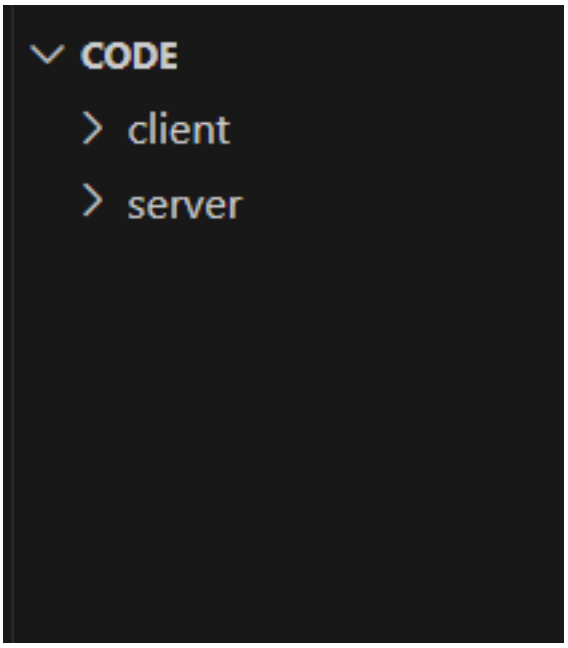
**Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

**Front-end Framework:** Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

**Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

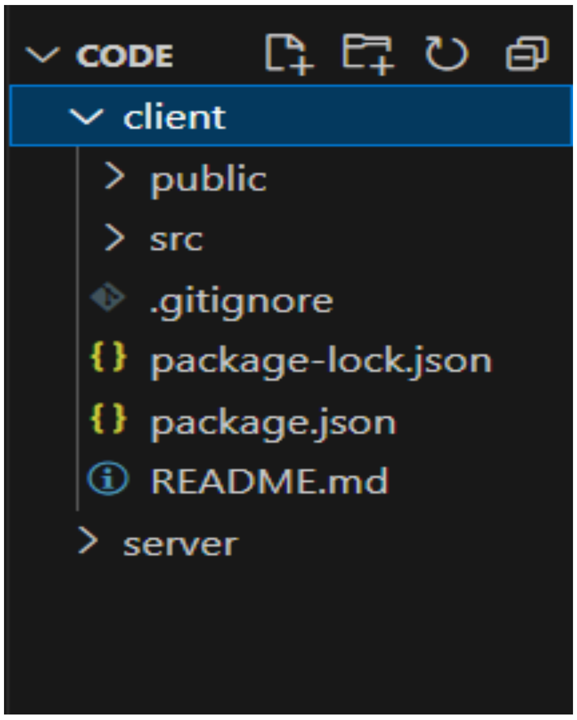
## FOLDER STRUCTURE

- Inside the Flight Booking app directory, we have the following folders

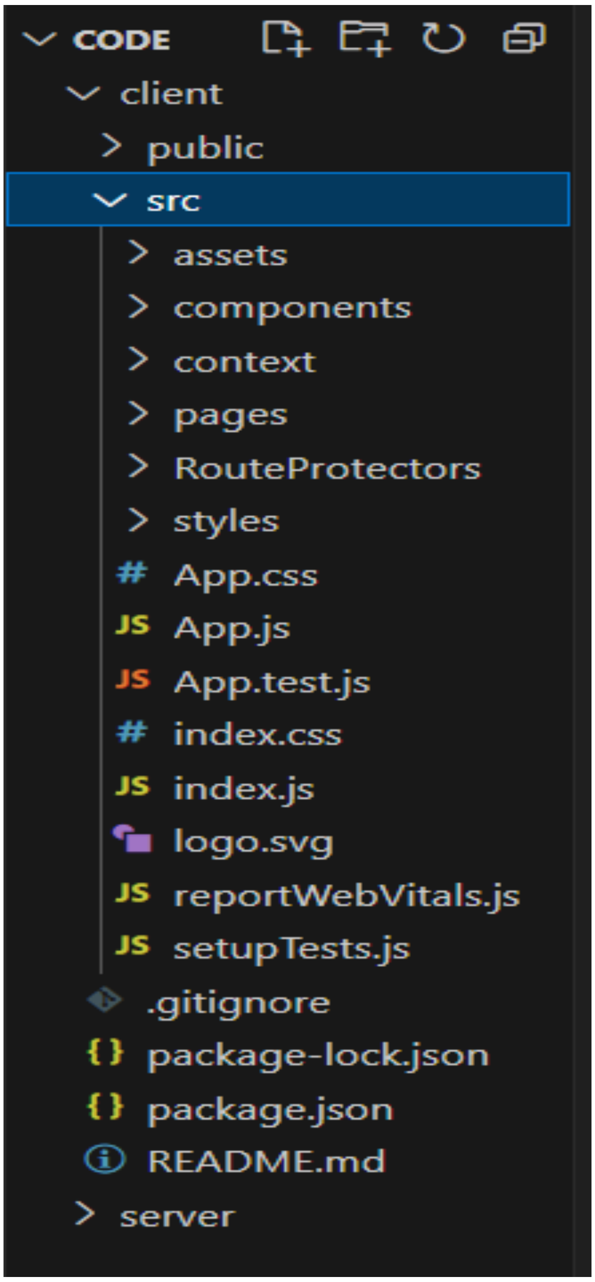


- **Client directory:**

  The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api’s.



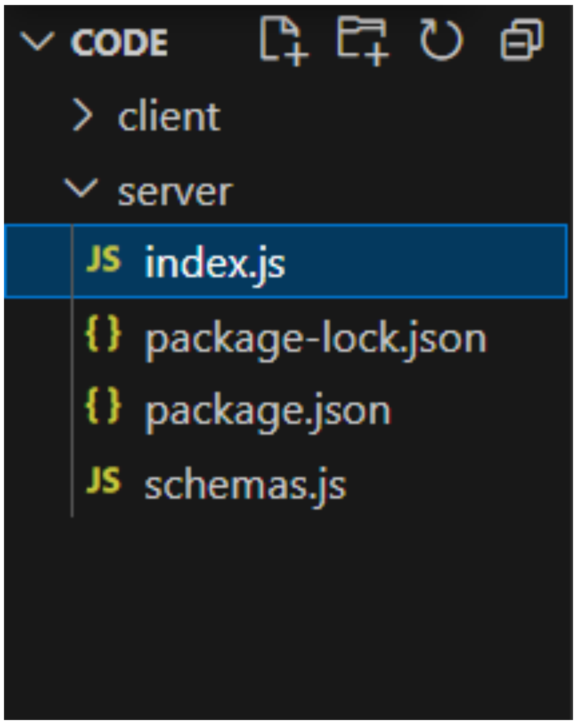
```
CODE
  client
    public
    src
    .gitignore
    package-lock.json
    package.json
    README.md
  server
```



```
CODE
  client
    public
    src
      assets
      components
      context
      pages
      RouteProtectors
      styles
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      reportWebVitals.js
      setupTests.js
    .gitignore
    package-lock.json
    package.json
    README.md
  server
```

- **Server directory:**

  The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with Api.



## APPLICATION FLOW

- **USER:**
  - Create their account.
  - Search for his destination.
  - Search for flights as per his time convenience.
  - Book a flight with a particular seat.
  - Make his payment.
  - And also cancel bookings.
- **ADMIN**
  - Manages all bookings.
  - Adds new flights and services.
  - Monitor User activity.

## Project Setup And Configuration

**Installation of required tools:**

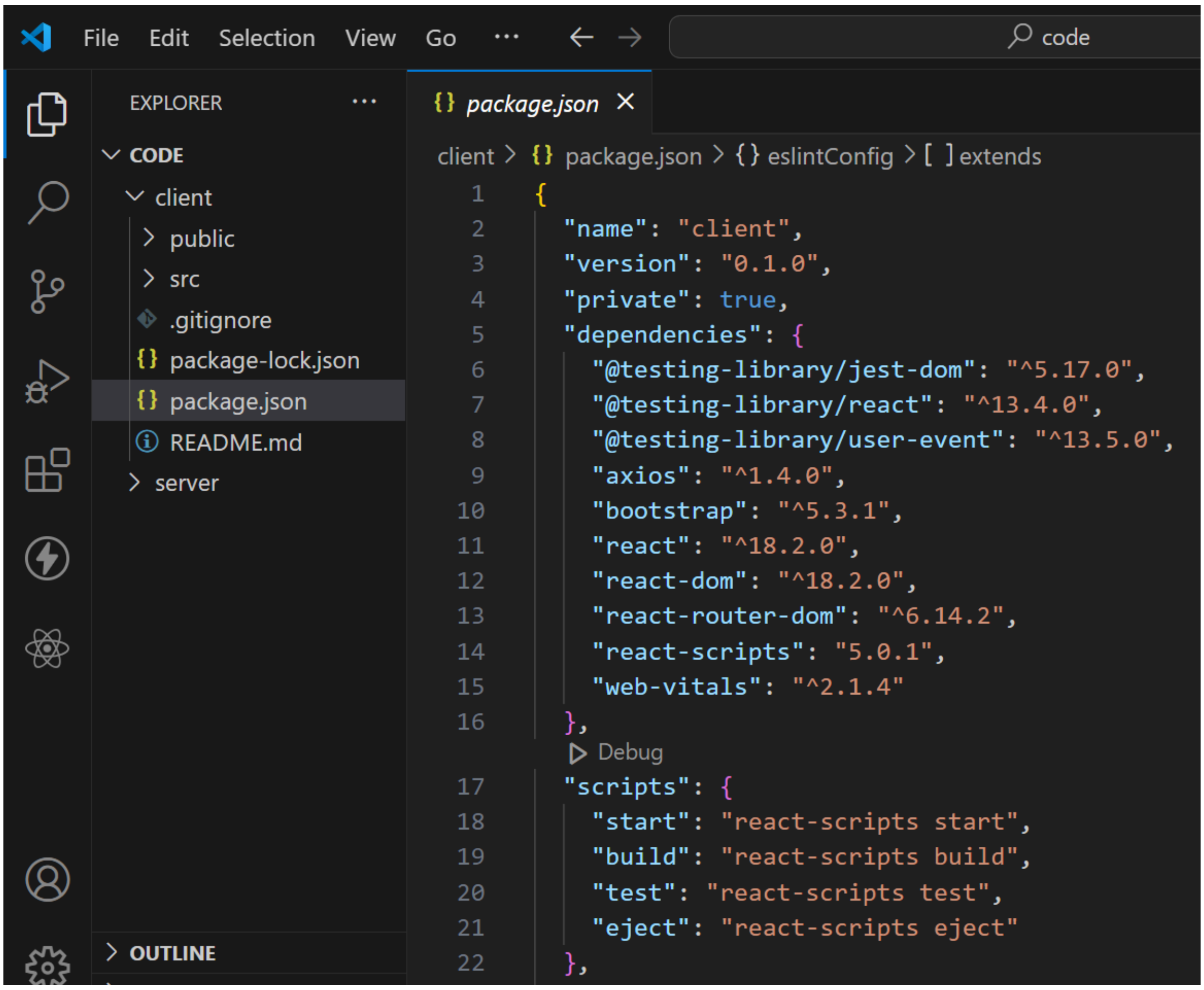
Now, open the frontend folder to install all the necessary tools we use.

For frontend, we use:

- React Js

- Bootstrap
- Axios

After installing all the required libraries, we'll be seeing the package.json file similar to the one below.



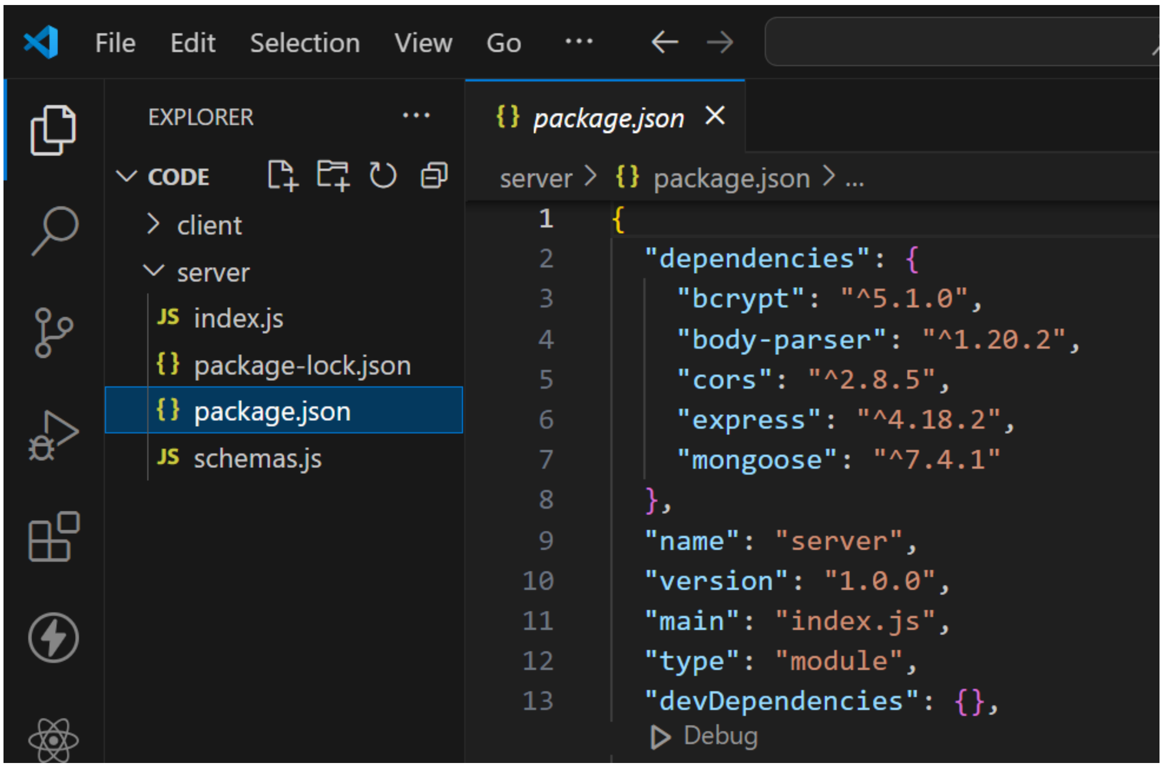
```json
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.14.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

Now, open the backend folder to install all the necessary tools that we use in the backend.

For backend, we use:

- bcrypt
- body-parser
- cors
- express
- mongoose

After installing all the required libraries, we'll be seeing the package.json file similar to the one below.



```json
{
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^7.4.1"
  },
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "devDependencies": {},
  ▷ Debug
```

## Backend Development

1. **Database Configuration:**
    - Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
    - Create a database and define the necessary collections for flights, users, bookings, and other relevant data.
2. **Create Express.js Server:**
    - Set up an Express.js server to handle HTTP requests and serve API endpoints.
    - Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.
3. **Define API Routes:**
    - Create separate route files for different API functionalities such as flights, users, bookings, and authentication.
    - Define the necessary routes for listing flights, handling user registration and login managing bookings, etc.
    - Implement route handlers using Express.js to handle requests and interact with the database.
4. **Implement Data Models:**
    - Define Mongoose schemas for the different data entities like flights, users, and bookings.
    - Create corresponding Mongoose models to interact with the MongoDB database. Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
5. **User Authentication:**
    - Create routes and middleware for user registration, login, and logout.

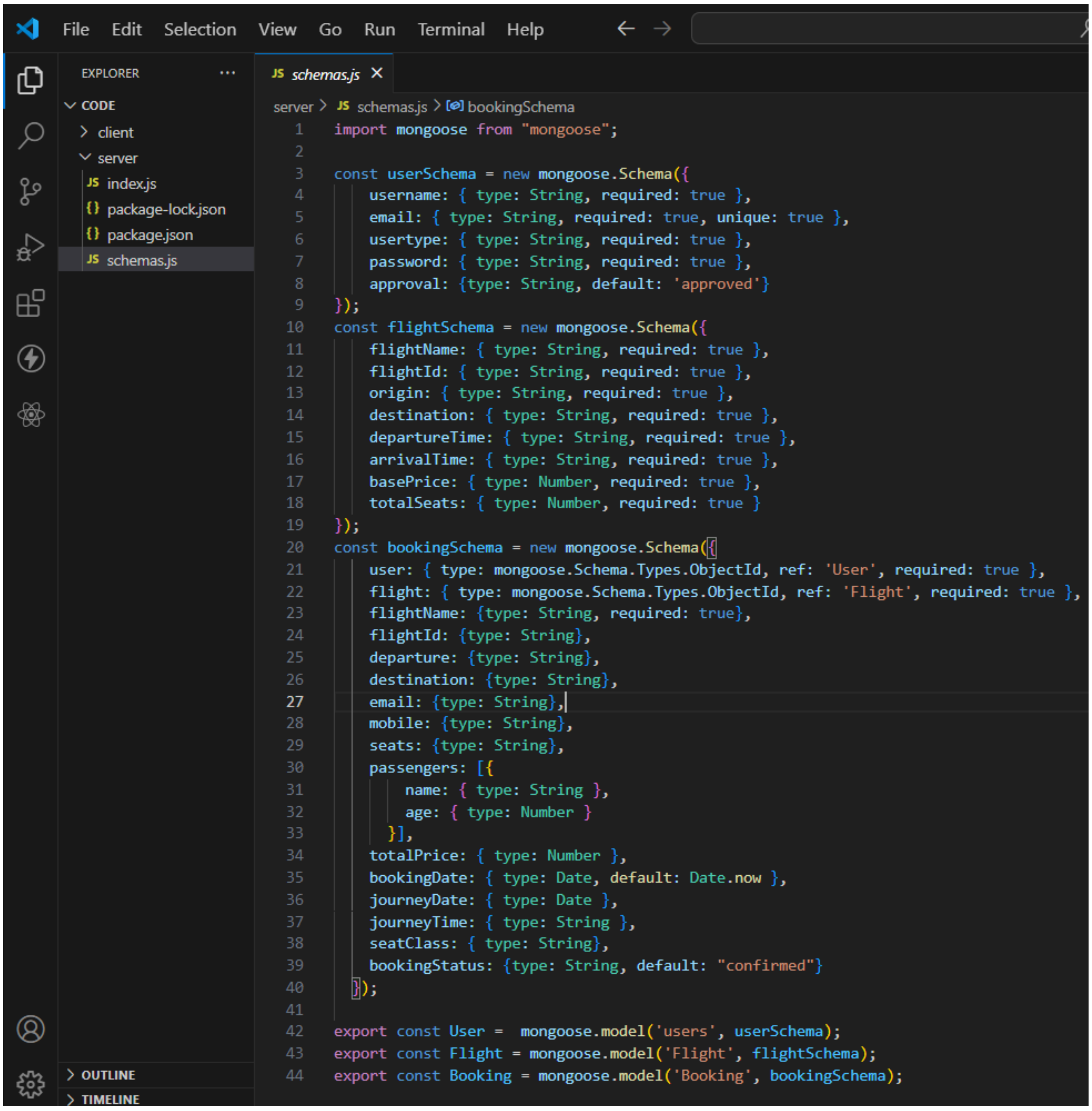
- Set up authentication middleware to protect routes that require user authentication.

6. **Handle new Flights and Bookings:**
    - Create routes and controllers to handle new flight listings, including fetching flight data from the database and sending it as a response.
    - Implement booking functionality by creating routes and controllers to handle booking requests, including validation and database updates.
7. **Admin Functionality:**
    - Implement routes and controllers specific to admin functionalities such as adding flights, managing user bookings, etc.
    - Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.
8. **Error Handling:**
    - Implement error handling middleware to catch and handle any errors that occur during the API requests.
    - Return appropriate error responses with relevant error messages and HTTP status codes.

## Database Development

- **Configure schema**

Firstly, configure the Schemas for MongoDB database, to store the data in such a pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.



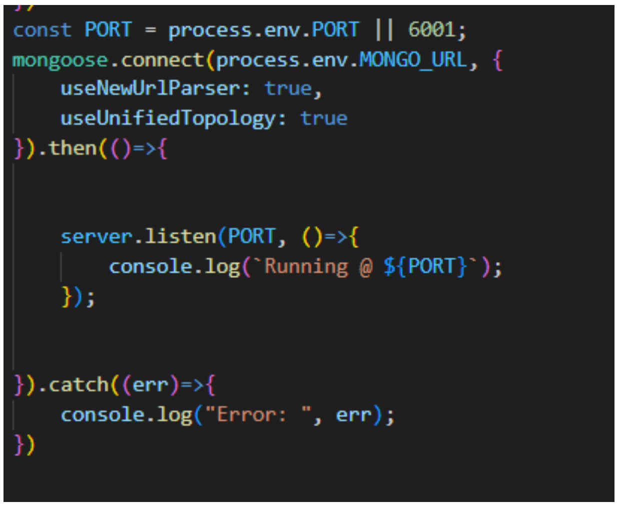
```javascript
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    username: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    usertype: { type: String, required: true },
    password: { type: String, required: true },
    approval: {type: String, default: 'approved'}
});
const flightSchema = new mongoose.Schema({
    flightName: { type: String, required: true },
    flightId: { type: String, required: true },
    origin: { type: String, required: true },
    destination: { type: String, required: true },
    departureTime: { type: String, required: true },
    arrivalTime: { type: String, required: true },
    basePrice: { type: Number, required: true },
    totalSeats: { type: Number, required: true }
});
const bookingSchema = new mongoose.Schema({
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    flight: { type: mongoose.Schema.Types.ObjectId, ref: 'Flight', required: true },
    flightName: {type: String, required: true},
    flightId: {type: String},
    departure: {type: String},
    destination: {type: String},
    email: {type: String},
    mobile: {type: String},
    seats: {type: String},
    passengers: [{
        name: { type: String },
        age: { type: Number }
    }],
    totalPrice: { type: Number },
    bookingDate: { type: Date, default: Date.now },
    journeyDate: { type: Date },
    journeyTime: { type: String },
    seatClass: { type: String},
    bookingStatus: {type: String, default: "confirmed"}
});

export const User =  mongoose.model('users', userSchema);
export const Flight = mongoose.model('Flight', flightSchema);
export const Booking = mongoose.model('Booking', bookingSchema);
```

- **Connect database to backend**

  Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.



```
const PORT = process.env.PORT || 6001;
mongoose.connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{


    server.listen(PORT, ()=>{
        console.log(`Running @ ${PORT}`);
    });


}).catch((err)=>{
    console.log("Error: ", err);
})
```

## Frontend Development

1. **Login/Register**

- Create a Component which contains a form for taking the username and password.
- If the given inputs matches the data of user or admin or flight operator then navigate it to their respective home page

2. **Flight Booking (User):**

- In the frontend, we implemented all the booking code in a modal. Initially, we need to implement flight searching feature with inputs of Departure city, Destination, etc.,
- Flight Searching code: With the given inputs, we need to fetch the available flights. With each flight, we add a button to book the flight, which redirects to the flight-Booking page.

3. **Fetching user bookings:**

- In the bookings page, along with displaying the past bookings, we will also provide an option to cancel that booking.

4. **Add new flight(Admin):**

- Now, in the admin dashboard, we provide functionality to add new flights.
- We create a html form with required inputs for the new flight and then send an httprequest to the server to add it to the database.

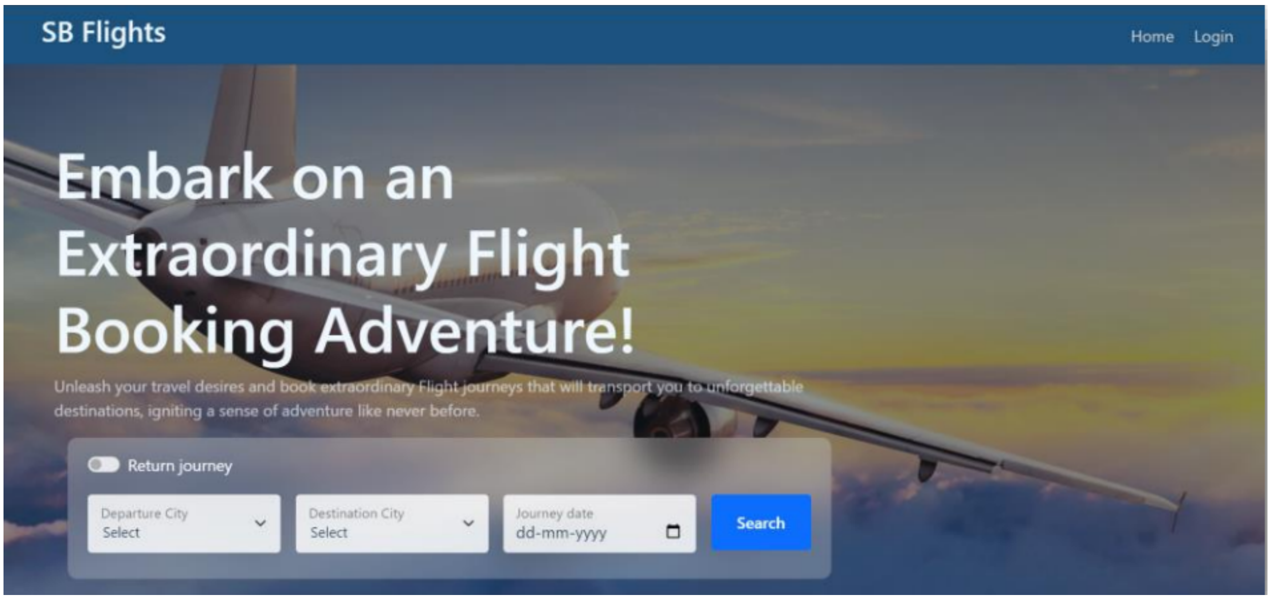
5. **Update Flight:**

- Here, in the admin dashboard, we will update the flight details in case if we want to make any edits to it
- Along with this, implement additional features to view all flights, bookings, and users in the admin dashboard.

## 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

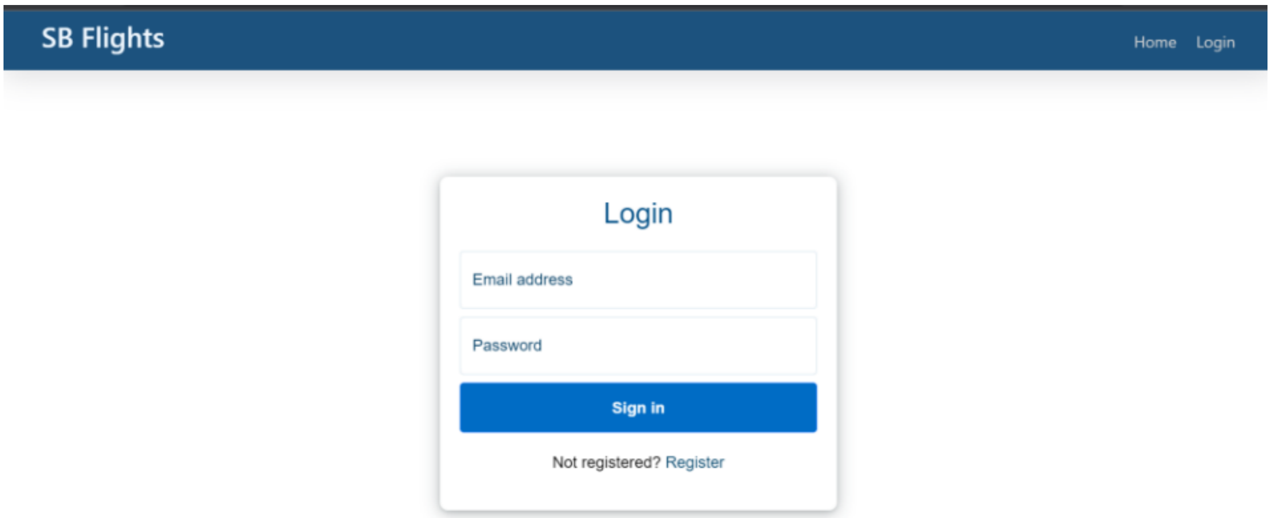
- **Landing page UI**



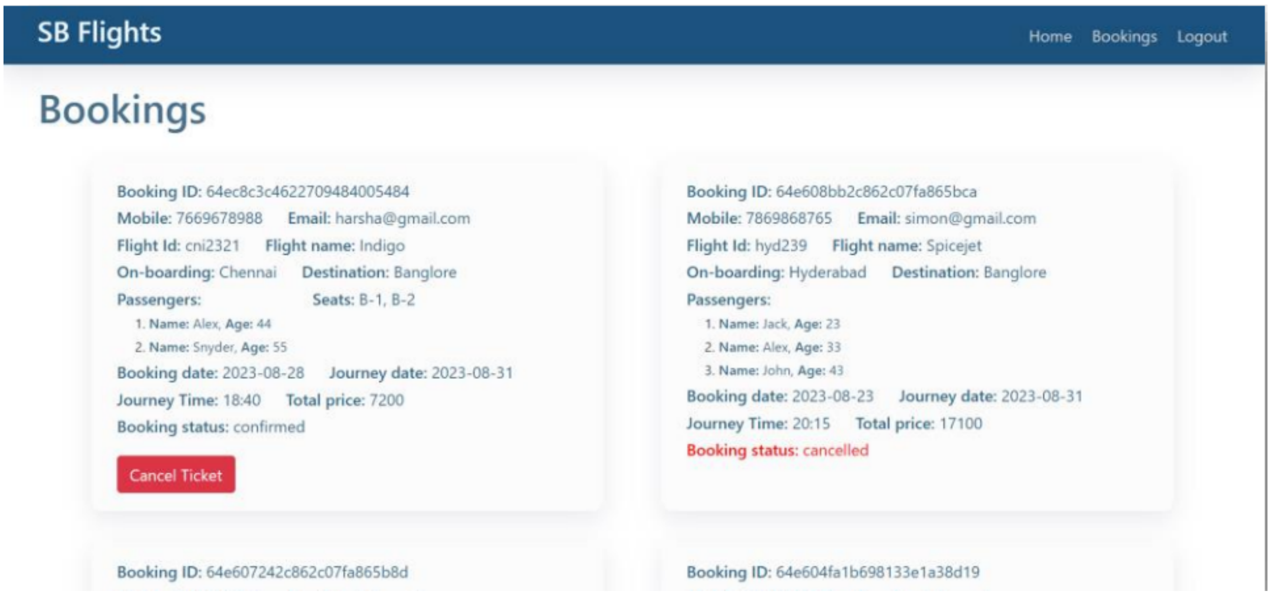
Embark on an Extraordinary Flight Booking Adventure!

Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.

- **Authentication**



- **User bookings**



SB Flights

Home Bookings Logout

Bookings

Booking ID: 64ec8c3c4622709484005484
Mobile: 7669678988 Email: harsha@gmail.com
Flight Id: cni2321 Flight name: Indigo
On-boarding: Chennai Destination: Banglore
Passengers: Seats: B-1, B-2
1. Name: Alex, Age: 44
2. Name: Snyder, Age: 55
Booking date: 2023-08-28 Journey date: 2023-08-31
Journey Time: 18:40 Total price: 7200
Booking status: confirmed

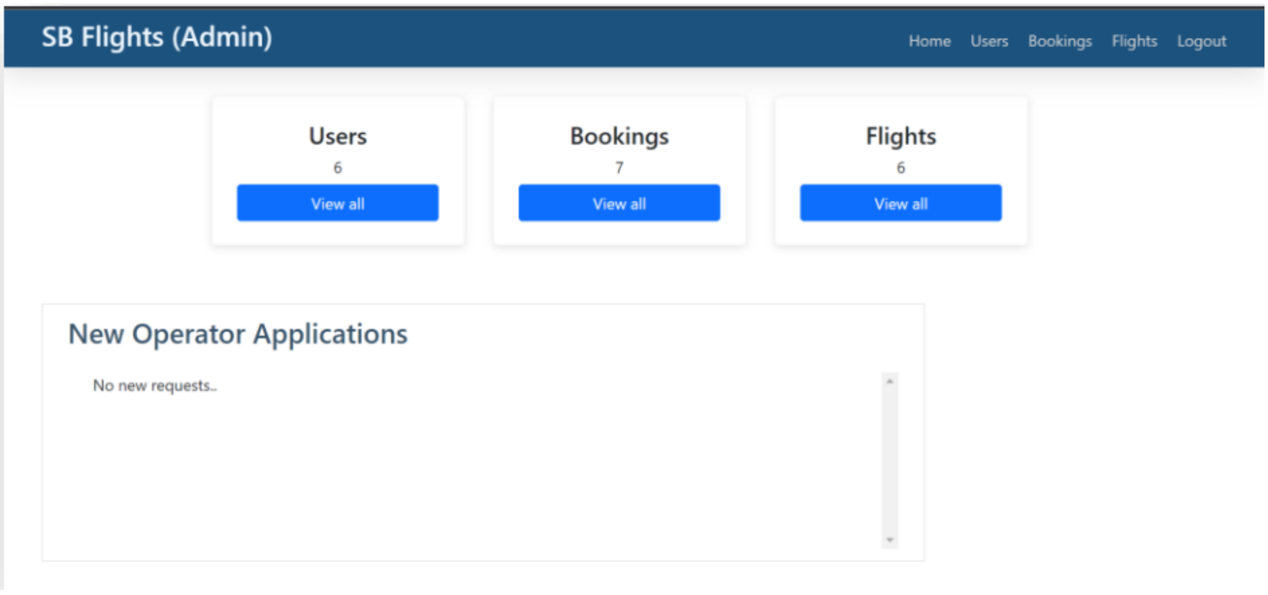
Cancel Ticket

Booking ID: 64e608bb2c862c07fa865bca
Mobile: 7869868765 Email: simon@gmail.com
Flight Id: hyd239 Flight name: Spicejet
On-boarding: Hyderabad Destination: Banglore
Passengers:
1. Name: Jack, Age: 23
2. Name: Alex, Age: 33
3. Name: John, Age: 43
Booking date: 2023-08-23 Journey date: 2023-08-31
Journey Time: 20:15 Total price: 17100
Booking status: cancelled

Booking ID: 64e607242c862c07fa865b8d

Booking ID: 64e604fa1b698133e1a38d19

- **Admin Dashboard**



SB Flights (Admin)

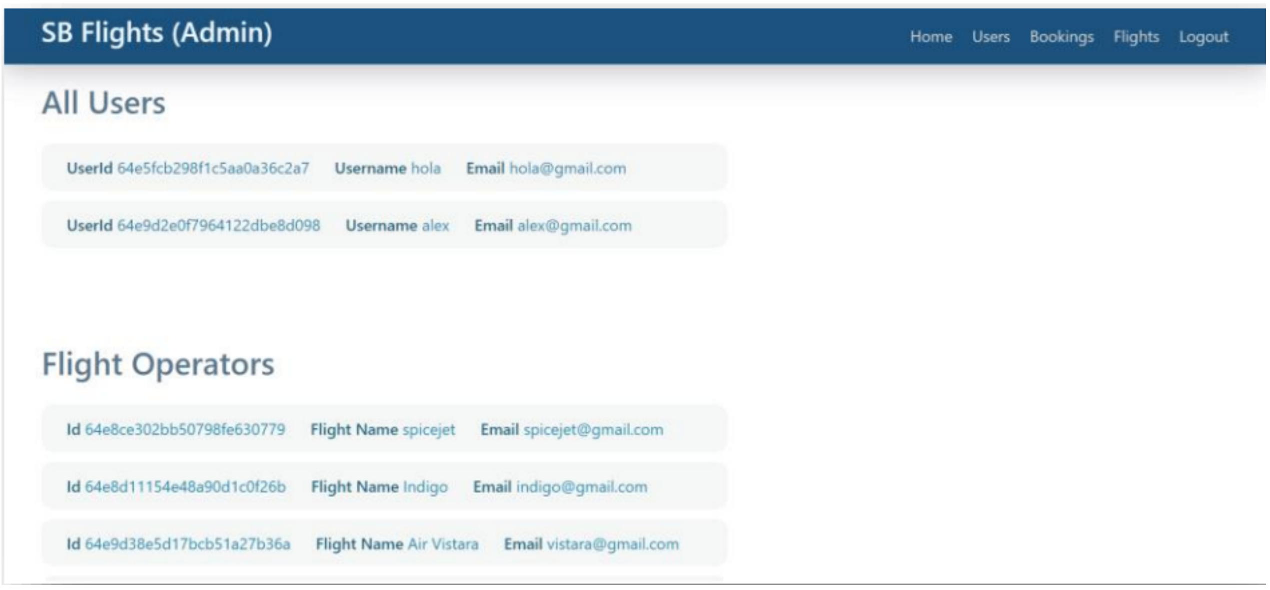
Home Users Bookings Flights Logout

| Users | Bookings | Flights |
|---|---|---|
| 6 | 7 | 6 |
| View all | View all | View all |

New Operator Applications

No new requests..

- **All users**



All Users

**UserId** 64e5fcb298f1c5aa0a36c2a7 **Username** hola **Email** hola@gmail.com

**UserId** 64e9d2e0f7964122dbe8d098 **Username** alex **Email** alex@gmail.com

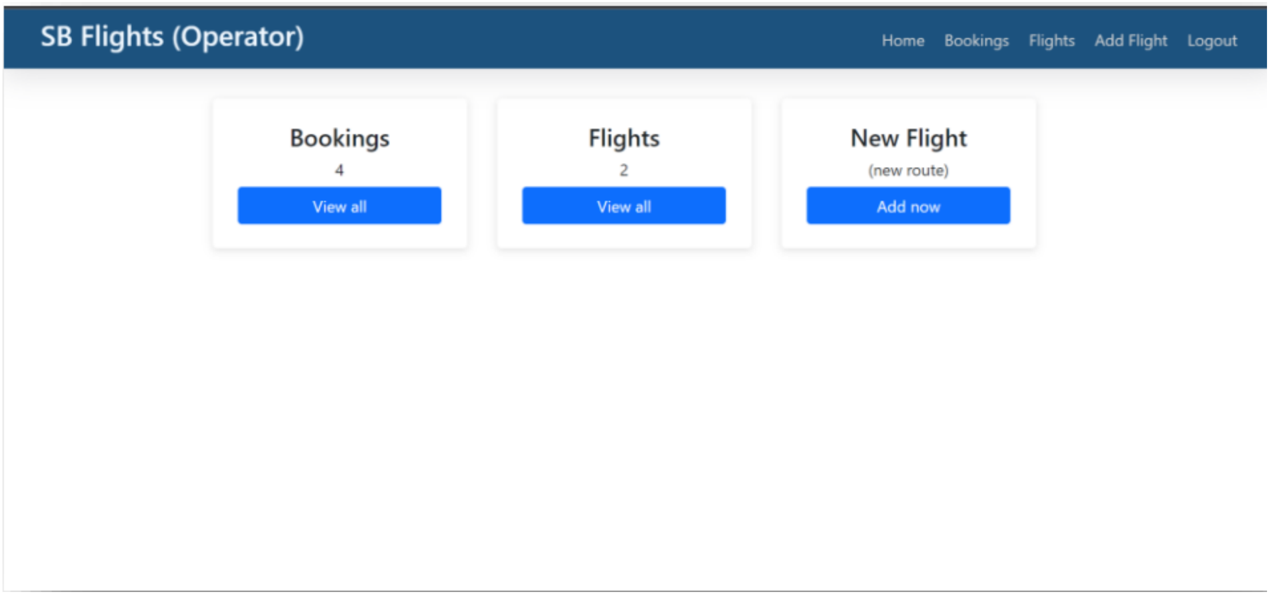
Flight Operators

**Id** 64e8ce302bb50798fe630779 **Flight Name** spicejet **Email** spicejet@gmail.com

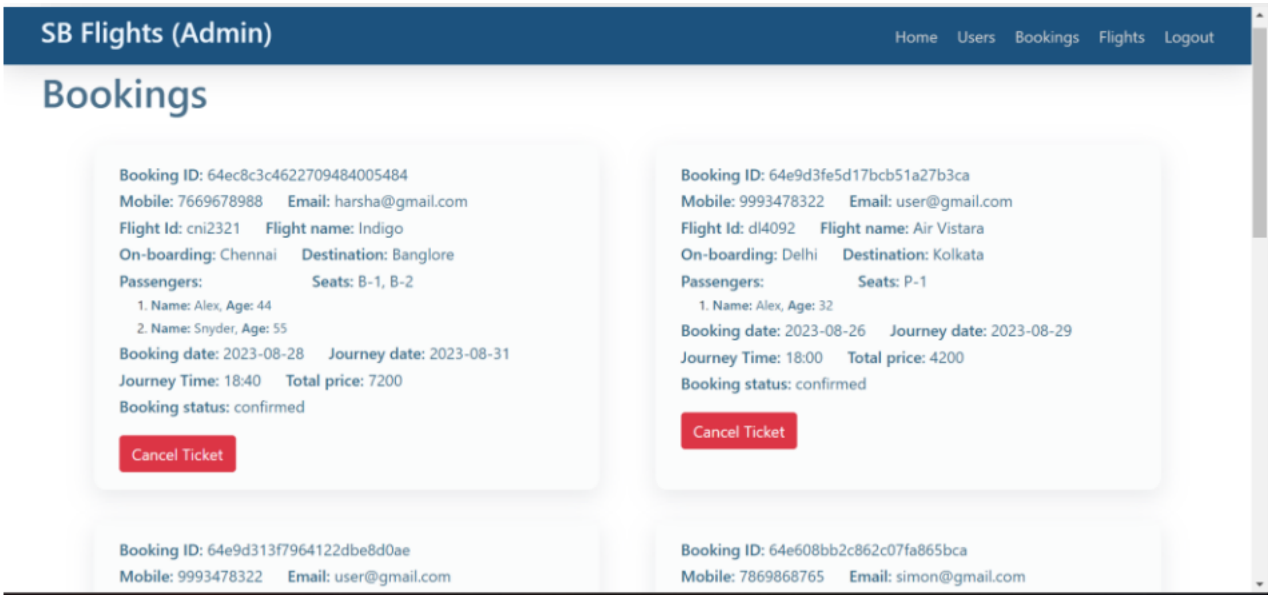
**Id** 64e8d11154e48a90d1c0f26b **Flight Name** Indigo **Email** indigo@gmail.com

**Id** 64e9d38e5d17bcb51a27b36a **Flight Name** Air Vistara **Email** vistara@gmail.com

- **Flight Operator**



- **All Bookings**



## Bookings

**Booking ID:** 64ec8c3c4622709484005484
**Mobile:** 7669678988 **Email:** harsha@gmail.com
**Flight Id:** cni2321 **Flight name:** Indigo
**On-boarding:** Chennai **Destination:** Banglore
**Passengers:** **Seats:** B-1, B-2
1. **Name:** Alex, **Age:** 44
2. **Name:** Snyder, **Age:** 55

**Booking date:** 2023-08-28 **Journey date:** 2023-08-31
**Journey Time:** 18:40 **Total price:** 7200
**Booking status:** confirmed

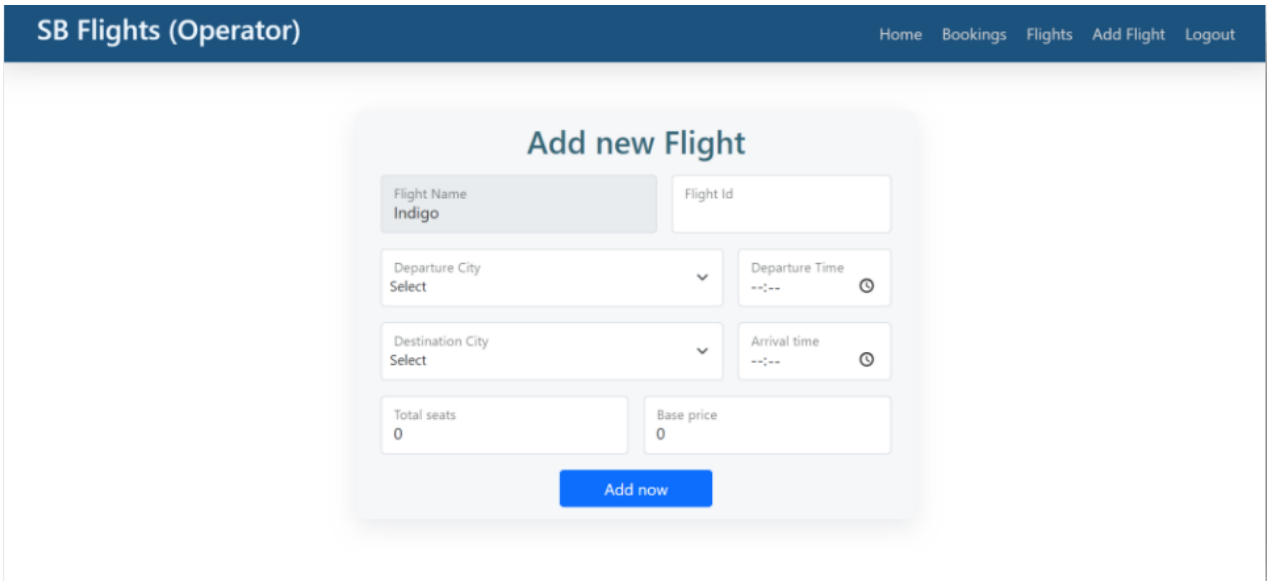
**Booking ID:** 64e9d3fe5d17bcb51a27b3ca
**Mobile:** 9993478322 **Email:** user@gmail.com
**Flight Id:** dl4092 **Flight name:** Air Vistara
**On-boarding:** Delhi **Destination:** Kolkata
**Passengers:** **Seats:** P-1
1. **Name:** Alex, **Age:** 32

**Booking date:** 2023-08-26 **Journey date:** 2023-08-29
**Journey Time:** 18:00 **Total price:** 4200
**Booking status:** confirmed

**Booking ID:** 64e9d313f7964122dbe8d0ae
**Mobile:** 9993478322 **Email:** user@gmail.com

**Booking ID:** 64e608bb2c862c07fa865bca
**Mobile:** 7869868765 **Email:** simon@gmail.com

- **New Flight**



## Add new Flight

| | |
|---|---|
| Flight Name: Indigo | Flight Id |
| Departure City: Select | Departure Time: --:-- |
| Destination City: Select | Arrival time: --:-- |
| Total seats: 0 | Base price: 0 |