Flightfinder: Navigating Your Air Travel Options

Team ID : LTVIP2026TMIDS76114

Names	Roles
Laveti Durga Sai Krishna	Frontend Development
Saladi Subrahmanyam	Backend Development
Sheik Tanisha Valli	Frontend Development
Silla Manoj Kumar	Database Integration
Bandaru Sri Laxmi Priyanka	Database Schema Design

INTRODUCTION

In today's fast-paced world, travel has become an essential part of our lives, whether it's for business, leisure, or personal reasons. With the advent of technology, booking flights has become more accessible and convenient than ever before, thanks to flight booking apps. A flight booking app is a mobile application that allows users to search, compare, and book flights easily from the comfort of their smartphones or tablets. The primary purpose of a flight booking app is to streamline the process of planning and booking air travel. These apps provide users with a range of features and functionalities that simplify the entire journey, from searching for flights to managing bookings and receiving travel updates.

Description:

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.

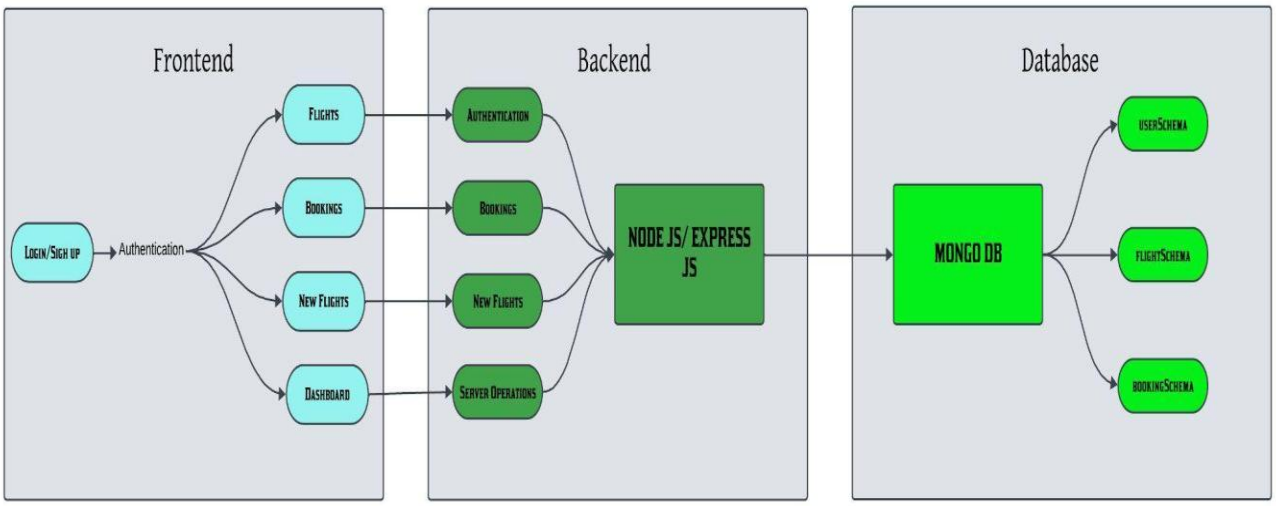
This successful flight booking app combines a user-friendly interface, efficient search and booking capabilities, personalized features, robust security measures, reliable performance, and continuous improvement based on user feedback.

Scenario

- 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.

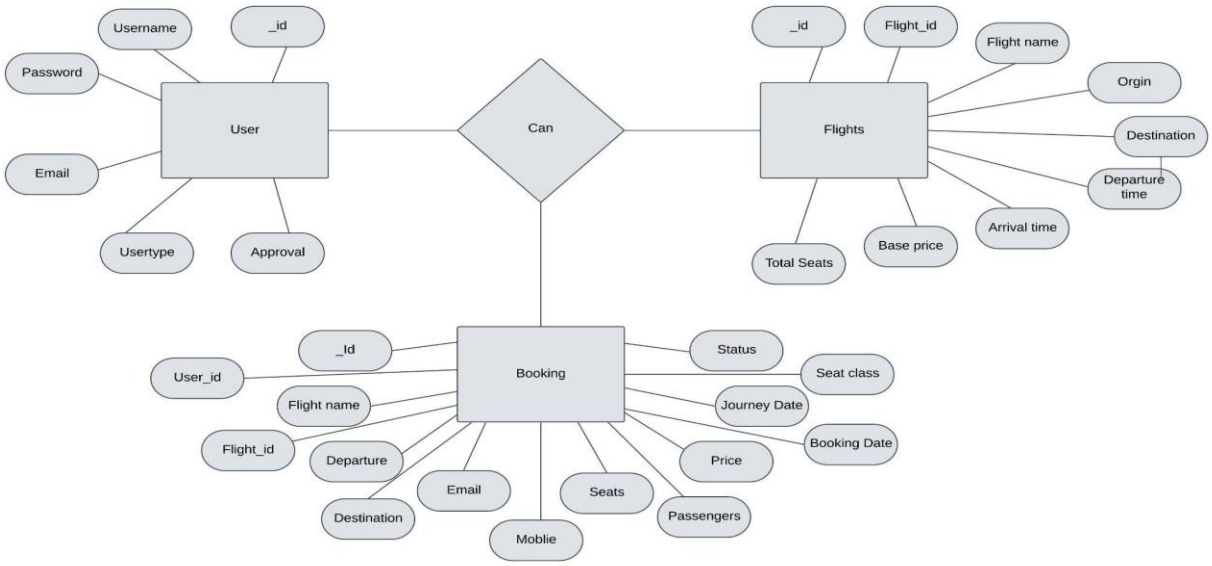
TECHNICAL ARCHITECTURE



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.,

PREREQUISITES:

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.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

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

- Download: https://www.mongodb.com/try/download/community
- Installation instructions: https://docs.mongodb.com/manual/installation/

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

- Installation: Open your command prompt or terminal and run the following command: npm install express

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, follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

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.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: https://gitscm.com/downloads

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

- Visual Studio Code: Download from https://code.visualstudio.com/download
- Sublime Text: Download from https://www.sublimetext.com/download
- WebStorm: Download from https://www.jetbrains.com/webstorm/download

To Connect the Database with Node JS go through the below provided link:

- https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb

To run the existing Flight Booking App project downloaded from github:

Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app.
- Execute the following command to clone the repository:

  Git clone: https://github.com/harsha-vardhan-reddy-07/Flight-Booking-App-MERN

Install Dependencies:

- Navigate into the cloned repository directory:

  cd Flight-Booking-App-MERN

- Install the required dependencies by running the following command:

  npm install

Start the Development Server:

- To start the development server, execute the following command:

  npm run dev or npm run start

- The e-commerce app will be accessible at http://localhost:3000 by default. You can change the port configuration in the .env file if needed.

Access the App:

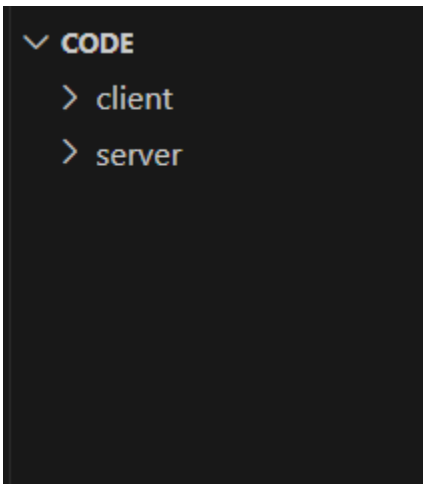
• Open your web browser and navigate to http://localhost:3000

• You should see the flight booking app's homepage, indicating that the installation and the setup was successful.

You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as needed.

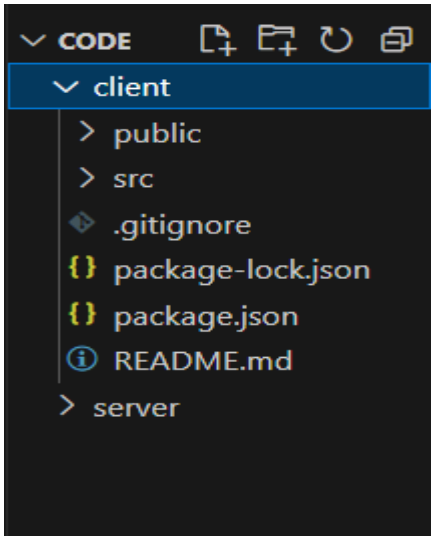
PROJECT STRUCTURE:

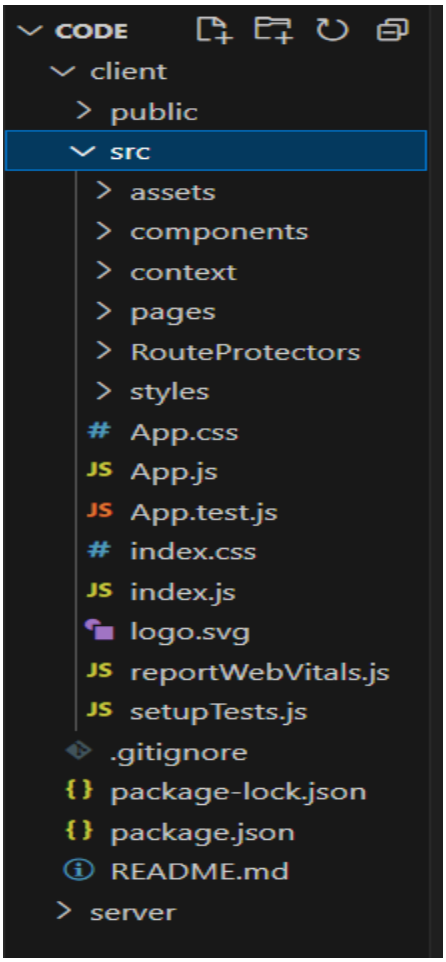
- Inside the Flight Booking app directory, we have the following folders



- **Client directory:**

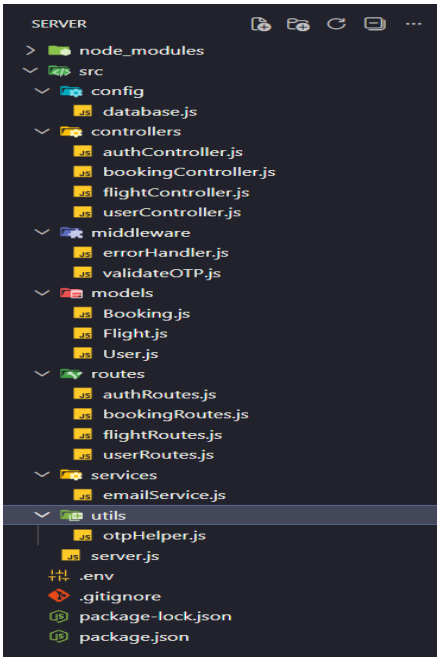
  The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api's.





- **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.

API Documentation:

Authentication Routes

Method	Endpoint	Description	Request Body	Response
POST	`/register`	Register new user	`{username, email, usertype, password}`	User object
POST	`/login`	User login	`{email, password}`	User object

User Management Routes

Method	Endpoint	Description	Authorization	Request Body
GET	`/fetch-users`	Get all users	Admin	-
GET	`/fetch-user/:id`	Get user by ID	Authenticated	-
POST	`/approve-operator`	Approve flight operator	Admin	`{id}`
POST	`/reject-operator`	Reject flight operator	Admin	`{id}`

Flight Management Routes

Method	Endpoint	Description	Authorization	Request Body
POST	`/add-flight`	Create new flight	Operator	`{flightName, flightId, origin, destination, journeyDate, departureTime, arrivalTime, basePrice, totalSeats}`
PUT	`/update-flight`	Update flight details	Operator	`{_id, ...updateFields}`
GET	`/fetch-flights`	Get all flights	Public	Query: `?date=YYYY-MM-DD` (optional)
GET	`/fetch-flight/:id`	Get flight by ID	Public	-

Booking Management Routes

Method	Endpoint	Description	Authorization	Request Body
POST	`/book-ticket`	Create new booking	Customer	`{user, flight, flightName, flightId, departure, destination, email, mobile, passengers, totalPrice, journeyDate, seatClass}`
GET	`/fetch-bookings`	Get all bookings	Authenticated	-
PUT	`/cancel-ticket/:id`	Cancel booking	Customer/Admin	-

PROJECT FLOW:

Milestone 1: Project setup and configuration.

Folder setup:

To start the project from scratch, firstly create frontend and backend folders to install essential libraries and write code.

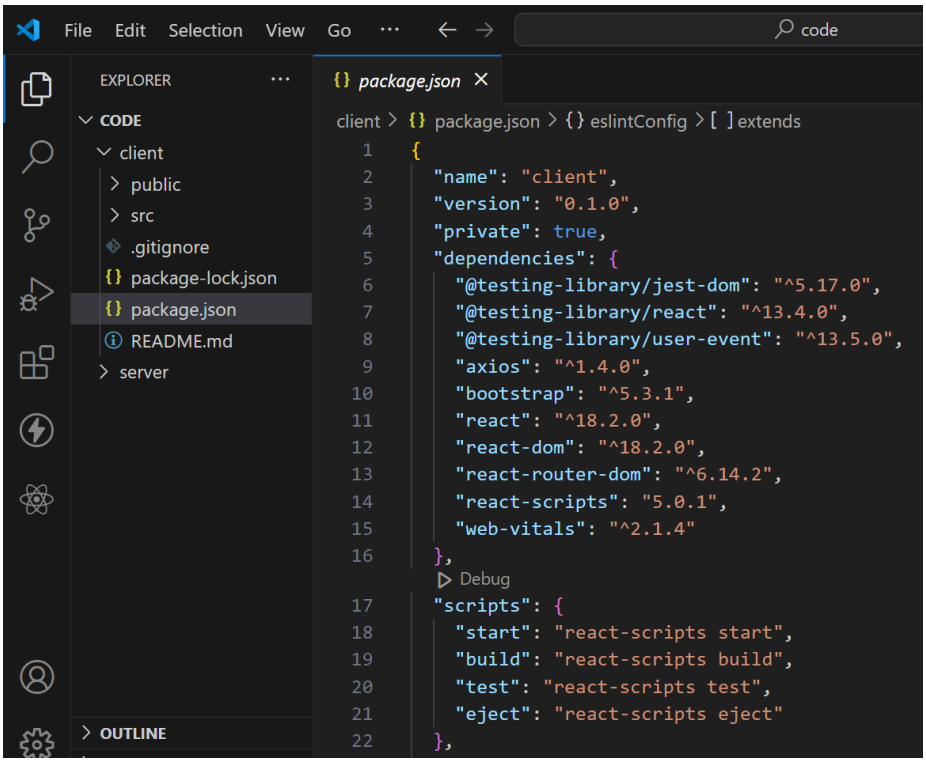
- client
- Server

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.

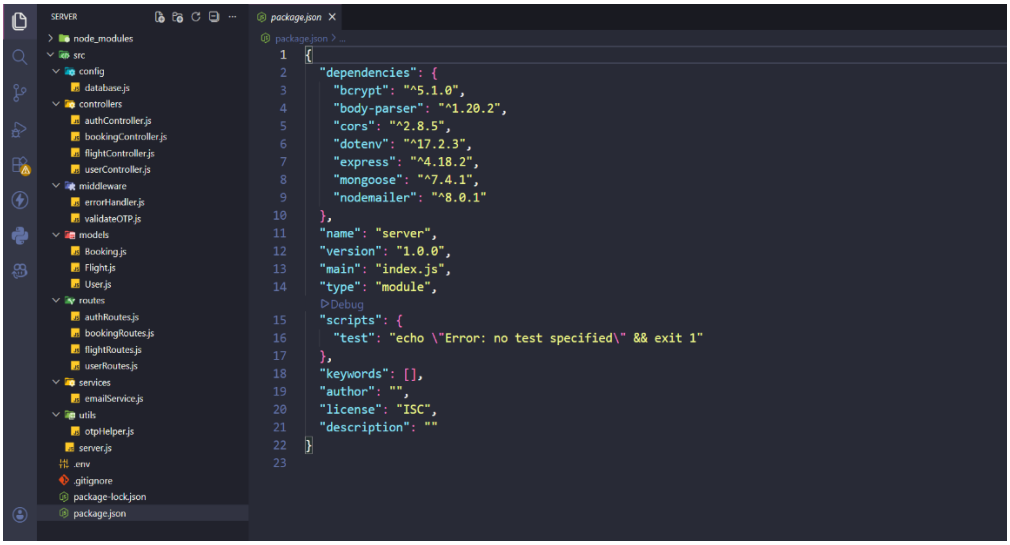


Now, open the backend folder to install all the necessary tools that we use in the backend.

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.



Milestone 2: Backend Development:

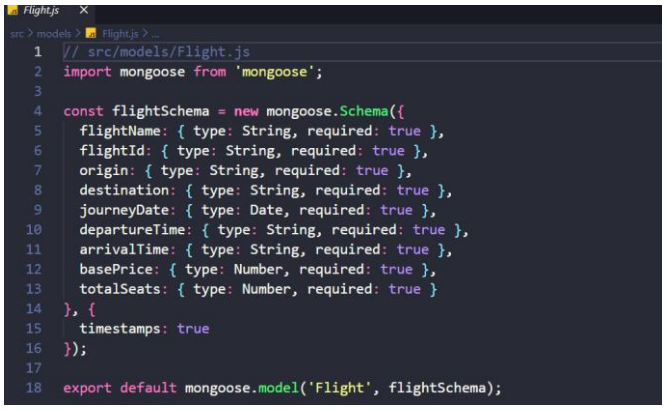
1. **Database Configuration:**
    - Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
    - Create a database and define the necessary collections for flights, users, bookings, and other relevant data.
2. **Create 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.

Milestone 3: Database development

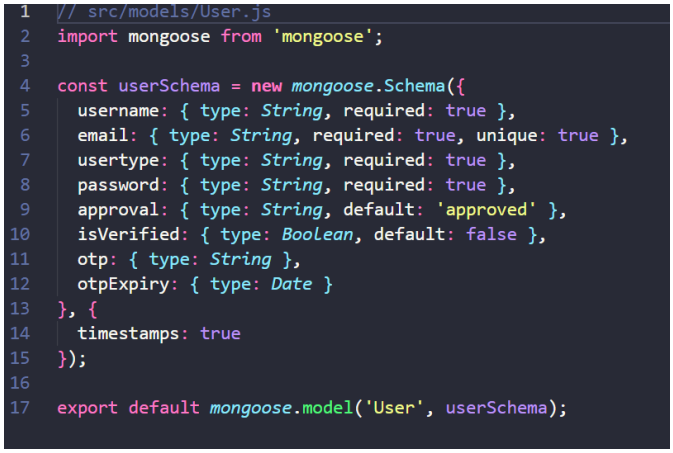
- **Configure schema**

  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.

```
// src/models/Booking.js
import mongoose from 'mongoose';

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' }
}, {
  timestamps: true
});

export default mongoose.model('Booking', bookingSchema);
```



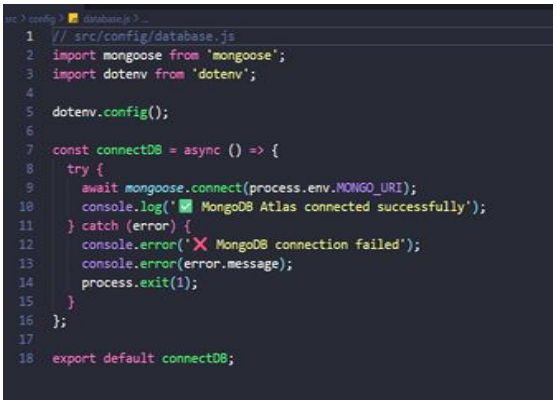
```
// src/models/Flight.js
import mongoose from 'mongoose';

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 },
  journeyDate: { type: Date, required: true },
  departureTime: { type: String, required: true },
  arrivalTime: { type: String, required: true },
  basePrice: { type: Number, required: true },
  totalSeats: { type: Number, required: true }
}, {
  timestamps: true
});

export default mongoose.model('Flight', flightSchema);
```



```
// src/models/User.js
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' },
  isVerified: { type: Boolean, default: false },
  otp: { type: String },
  otpExpiry: { type: Date }
}, {
  timestamps: true
});

export default mongoose.model('User', userSchema);
```

- **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.



```
// src/config/database.js
import mongoose from 'mongoose';
import dotenv from 'dotenv';

dotenv.config();

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('✅ MongoDB Atlas connected successfully');
  } catch (error) {
    console.error('❌ MongoDB connection failed');
    console.error(error.message);
    process.exit(1);
  }
};

export default connectDB;
```

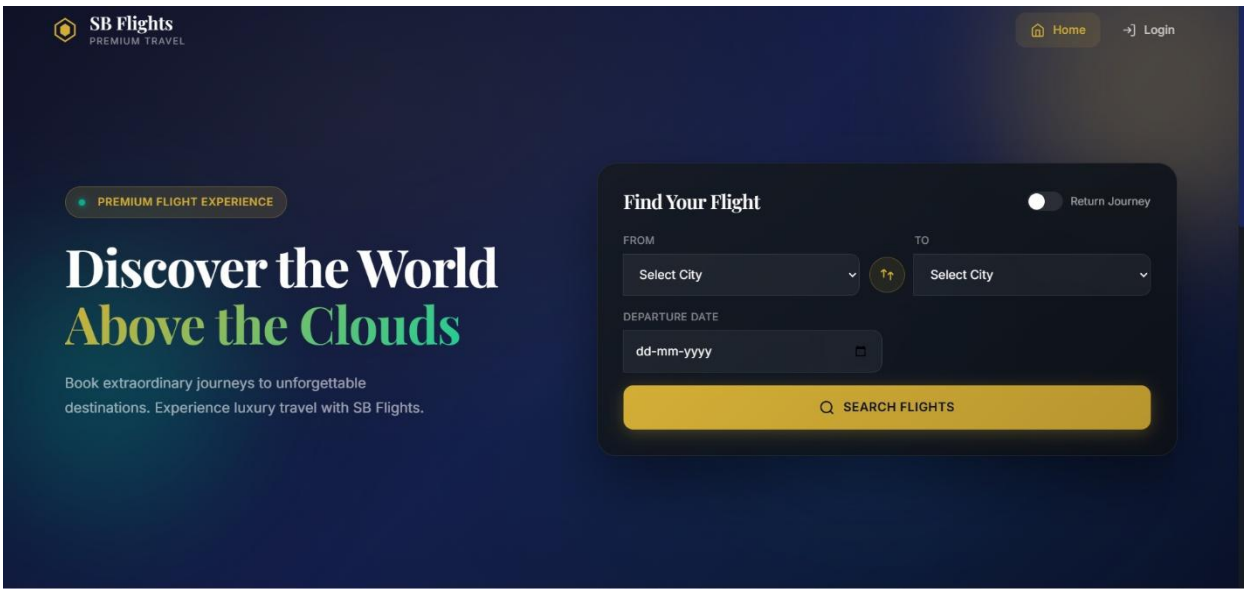
Milestone 4: Frontend development.

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.

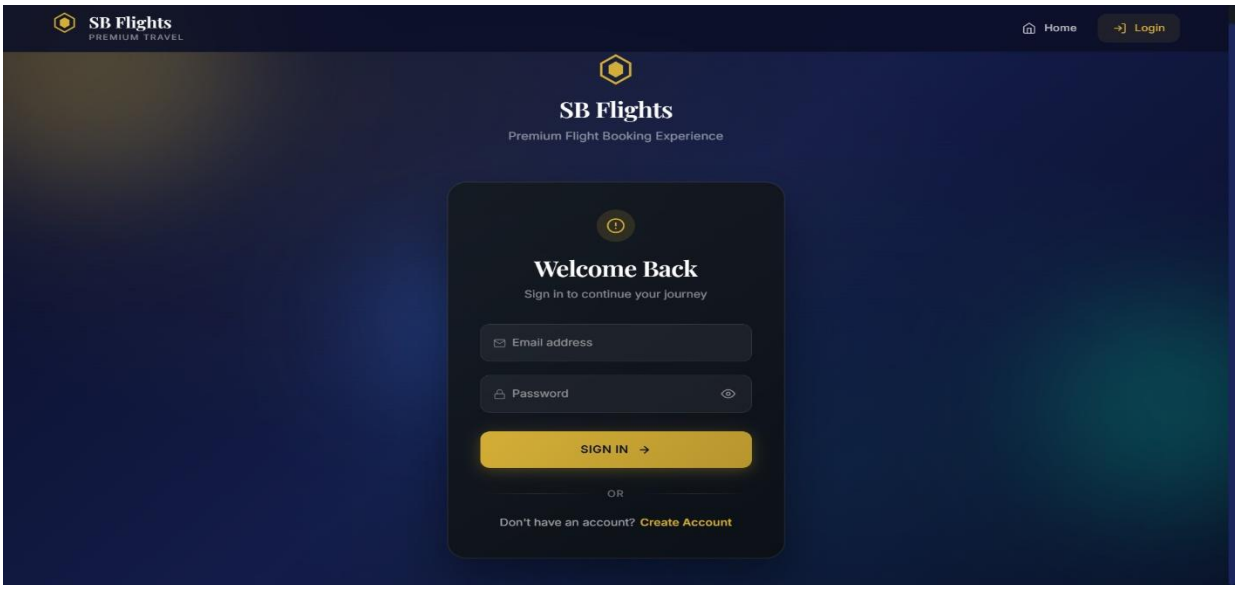
User Interface Screenshots

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 Flight Finder application

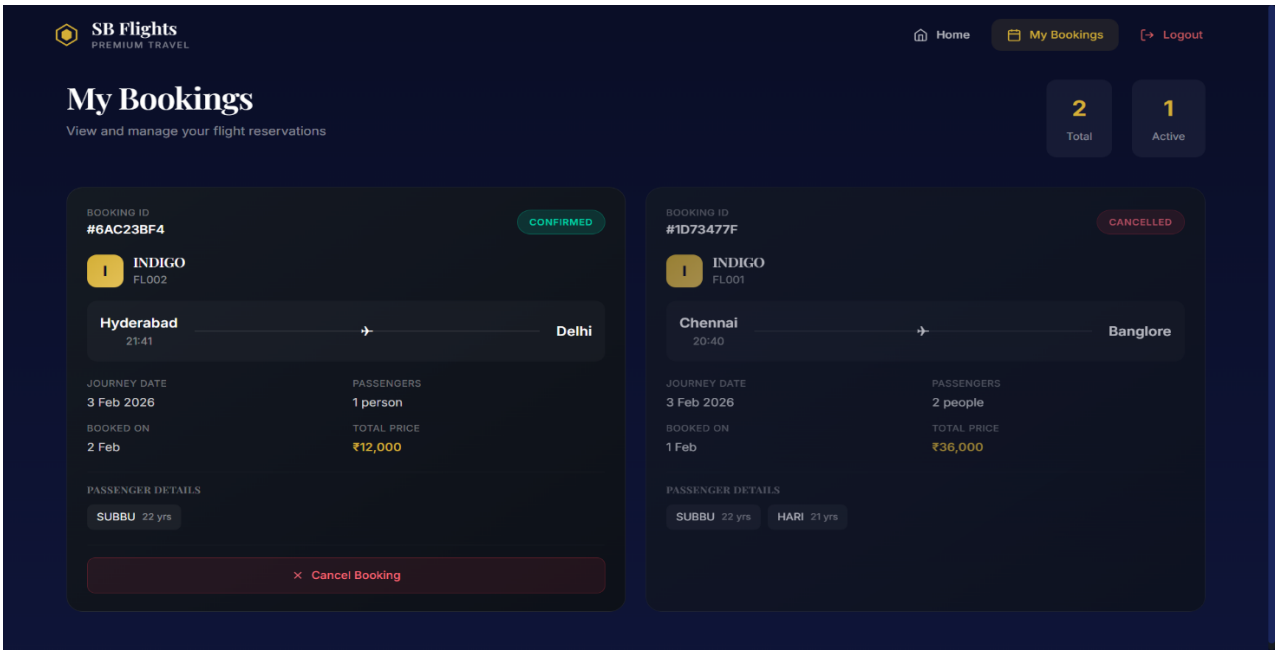
- **Landing page UI**



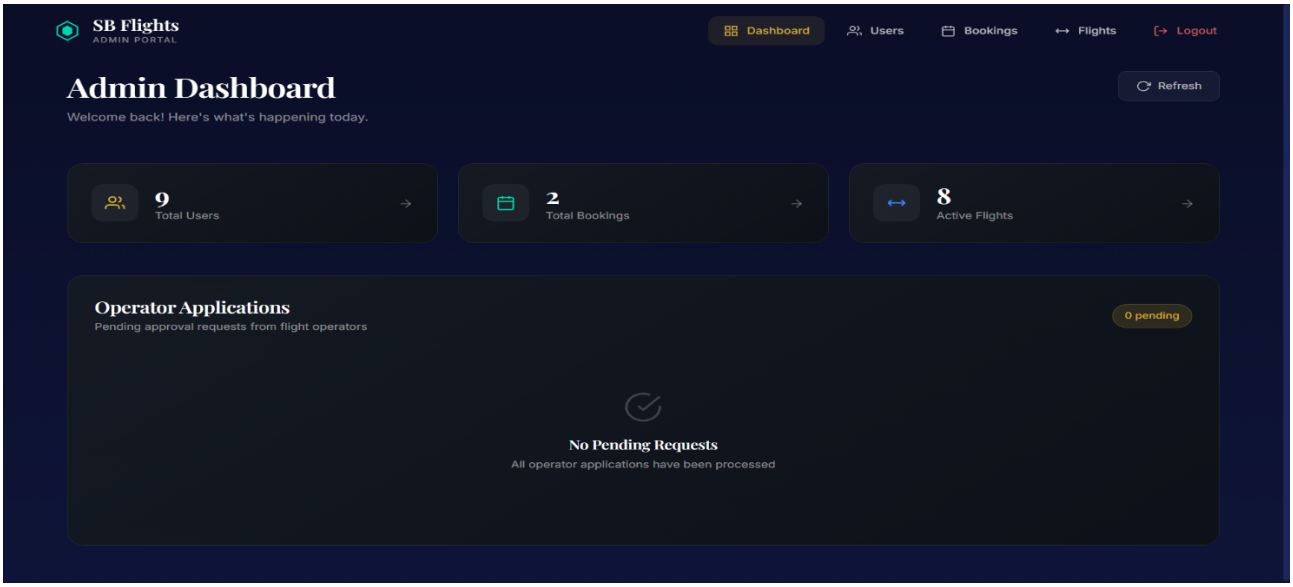
- **Authentication**



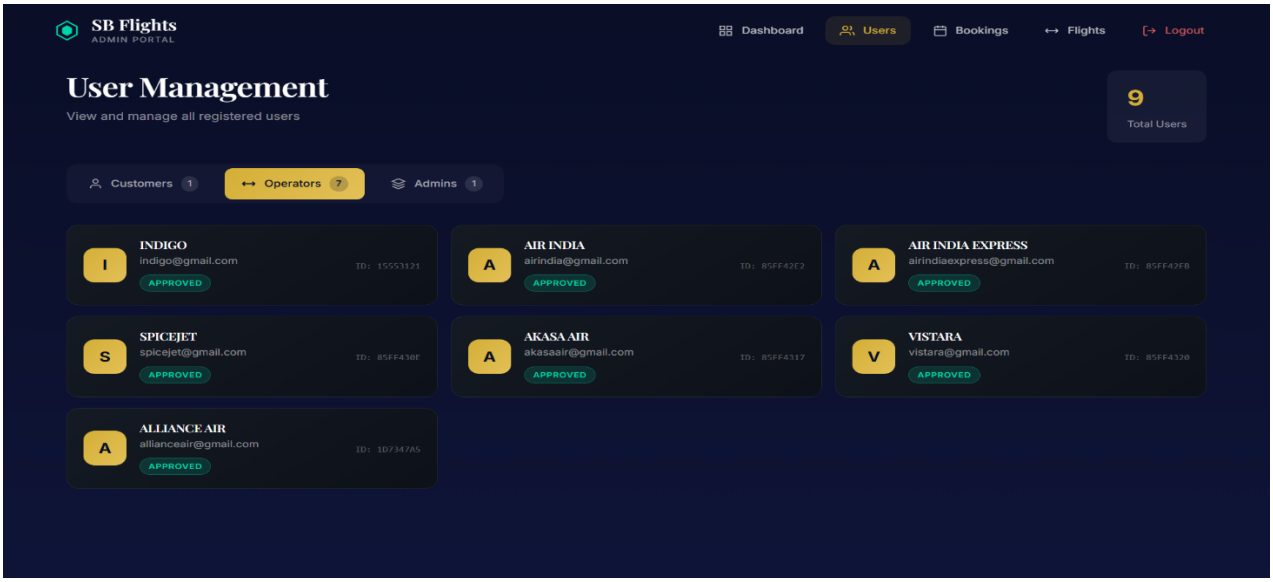
- **User bookings**



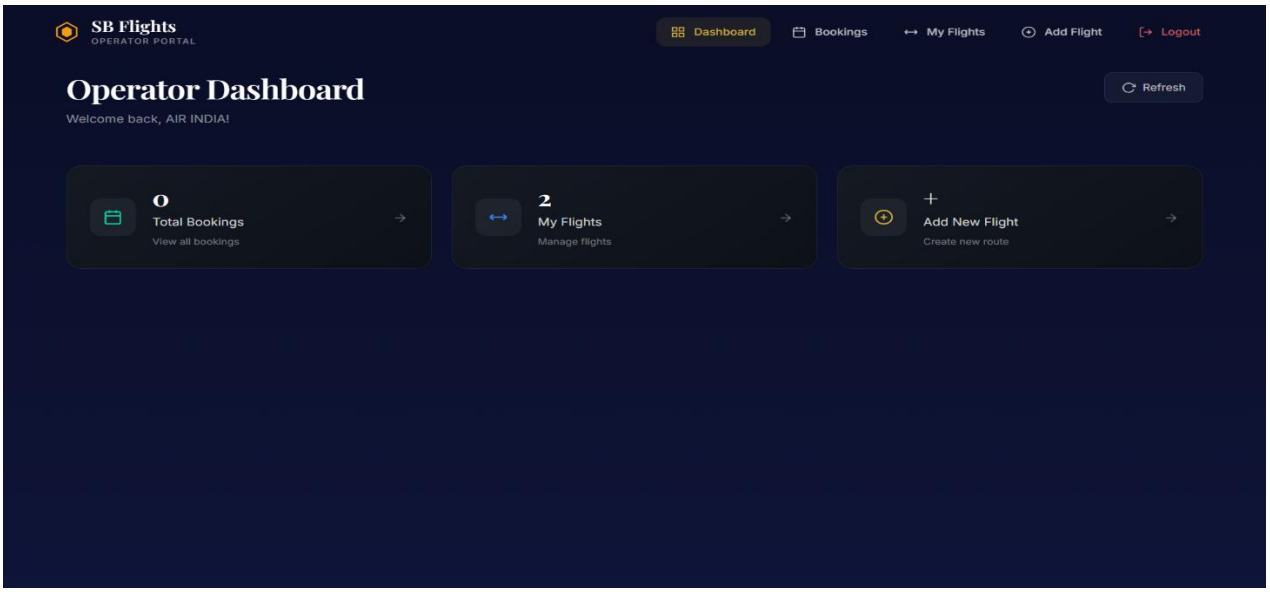
- **Admin Dashboard**



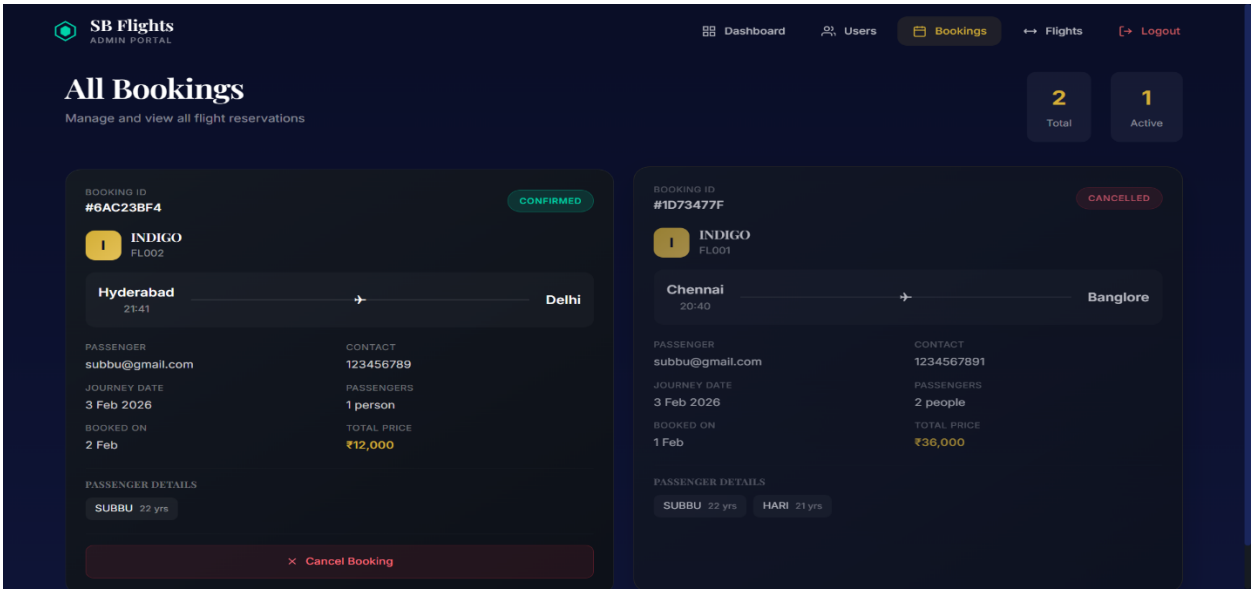
- **All users**



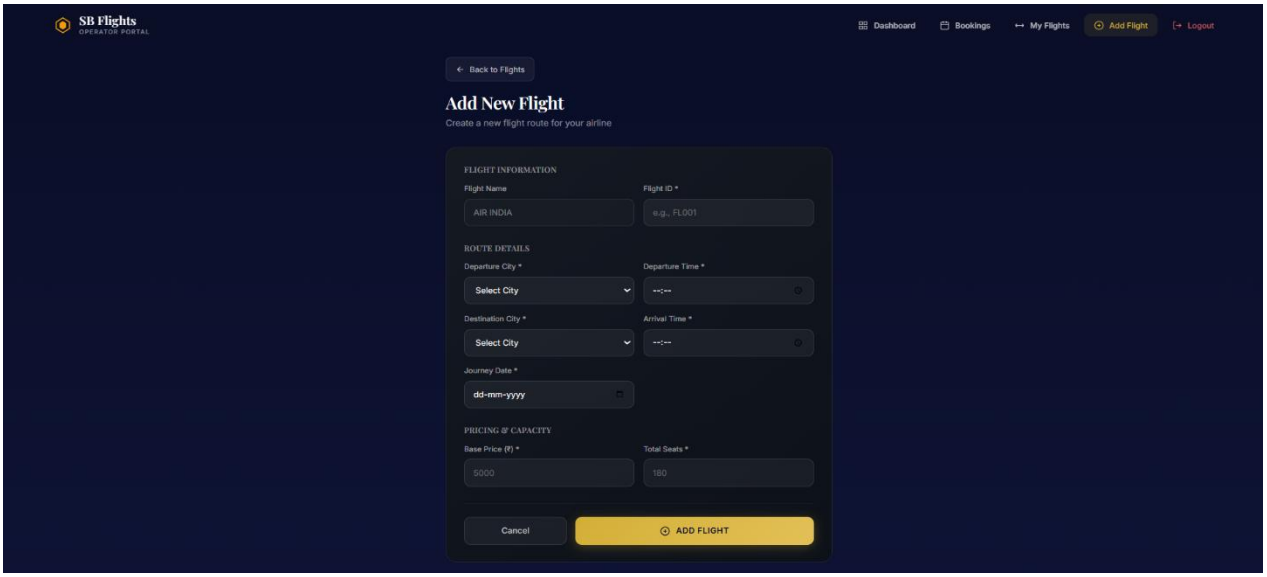
- **Flight Operator**



- **All Bookings**



- **New Flight**



The demo of the app is available at:
https://drive.google.com/file/d/1h285CpdYYGbDBQH7M-KCZArhUie81IPC/view?usp=drive_link

The Github Url of the project:
https://github.com/subbu666/Flightfinder-Navigating-Your-Air-Travel-Options