NAAN MUDHALVAN PROJECT

MERN stack powered by MongoDB

Flight Booking App - Project

TEAM MEMBERS:

S. Harish -311421205025

S. Harikaran -311421205024

P. Malaravan -311421205046

M. Balaji -311421205025

1. Introduction:

Project Title: Flight Booking App

- **Technology Stack**: MERN (MongoDB, Express.js, React, Node.js)

Team Members and Roles

- **S. Harish** - Frontend Developer: Responsible for UI/UX design, creating responsive and interactive interfaces using React.

- **S. Harikaran** - Backend Developer: Manages the backend setup, creates APIs, handles server logic, and ensures secure communication between the frontend and backend.
- **P. Malaravan** - Database Administrator: Manages the MongoDB database, optimizing queries, defining schemas, and ensuring data consistency and security.
- **M. Balaji** - Full Stack Developer: Works across both frontend and backend to ensure seamless integration and functionality throughout the application.

2. Project Overview:

Purpose and Goals

The **Flight Booking App** aims to provide a reliable and intuitive platform for users to search, book, and manage their flight reservations. The goal is to build a comprehensive web application that makes booking flights straightforward and accessible. The application includes both a user interface for customers and an admin interface for managing flight data, aiming to enhance the overall user experience and streamline flight bookings.

Key Features

1. **User Registration and Login**: Enables users to register and securely log in.
2. **Flight Search and Filtering**: Allows users to search for available flights with filtering options for departure and arrival locations, dates, and times.
3. **Seat Selection and Booking**: Offers seat selection options, allowing users to view available seats and book their desired seats.

4. **Booking Management and History**: Lets users manage their bookings and view their booking history.
5. **Admin Dashboard for Flight Management**: Provides admins with a dashboard to add, update, delete flights, and monitor overall booking activity.

3. System Architecture:

Frontend (React)

The frontend is developed using **React** to create a dynamic and responsive user interface. The application uses **React Router** for navigation, making it easy to switch between different pages such as search, booking, and user profile. **Redux** is used for state management, ensuring that application state is maintained across components, which improves the user experience by keeping data such as search results and user login information persistent across pages. **Axios** is used to manage HTTP requests between the frontend and backend.

Backend (Node.js and Express.js)

The backend is built using **Node.js** and **Express.js**, providing a RESTful API for communication with the frontend. The server handles user requests, processes them, and sends responses. Major features of the backend include:

- **Routing**: Different API endpoints handle various tasks such as user login, flight search, and booking, which improves the modularity and maintainability of the server code.

- **Middleware**: Middleware is used for tasks such as authentication, validation, and error handling. These middle wares ensure that requests are properly processed before reaching the core business logic.
- **Data Validation**: Each API endpoint includes validation logic to ensure data integrity.

Database (MongoDB)

MongoDB is used as the database to store and retrieve information. The **Mongoose** library defines schemas, allowing for a well-structured data model. Important schemas include:

- **User Schema**: Stores user details, roles (user or admin), and booking history.
- **Flight Schema**: Contains information about flights, such as flight number, departure and arrival locations, dates, and available seats.
- **Booking Schema**: Tracks bookings with information about users, flight ID, seat number, and booking status.

4. Setup Instructions

Prerequisites

- **Node.js** (v14.x or above) - Required to run both the frontend and backend servers.
- **MongoDB** - Used as the primary database for storing application data. MongoDB Atlas or a local MongoDB installation can be used.
- **Git** - Required to clone the project repository.

Installation Steps

1. **Clone the Repository**:
    - Run the following command to clone the repository to your local machine:

- Copy code

git clone [repository-url]

2. **Install Dependencies**:
    - In both the client and server directories, install dependencies by running:

Bash:

- Copy code
- npm install

3. **Configure Environment Variables**:
    - In the backend directory, create a .env file with the following environment variables:
        - MONGODB_URI for the MongoDB connection string.
        - PORT for the server port.

4. **Database Setup**:
    - Set up a MongoDB database either locally or using MongoDB Atlas.

5. Folder Structure:

Client (Frontend)

The client folder contains all React files and assets for the frontend:

- Contains reusable UI components
- Pages for each route (e.g., Home, Search, Booking)
- Redux store, actions, and reducers
- API integration for Axios requests
- Root component

Server (Backend)

The server folder contains the backend code using Express:

- Defines Mongoose schemas for MongoDB
- API route handlers (e.g., auth, flights, bookings)
- Contains business logic for routes
- Middleware for authentication, validation, etc.
- Main server entry point

6. Running the Application:

Frontend

1. Open the terminal, navigate to the client directory, and start the frontend:

Bash:

- cd client
- npm start

Backend

1. Open another terminal, navigate to the server directory, and start the backend:

Bash:

- cd server
- npm start

7. API Documentation:

Authentication Endpoints

- **POST** /api/auth/login: Logs in a user with email and password.
- **POST** /api/auth/register: Registers a new user with a name, email, and password.

Flight Endpoints

- **GET** /api/flights: Retrieves available flights with optional filters (date, location).
- **GET** /api/flights/:id: Fetches detailed information for a specific flight.

Booking Endpoints

- **POST** /api/bookings: Creates a booking for a flight with specific seat information.
- **GET** /api/bookings/:id: Fetches booking details for a specific booking ID.

8. Authentication:

Overview

JWT (JSON Web Tokens) is used for secure authentication and authorization. On successful login, a JWT token is generated, which the client stores in local storage. The

backend verifies this token for protected routes, ensuring only authorized users can access certain features.

Token Management

- **Login**: Upon login, the server generates a JWT, which the client stores.
- **Authorization**: Each protected route checks for a valid JWT token in the request headers.

9. User Interface:

The user interface provides a clear and responsive experience, including:

1. **Home Page**: Users can search for flights, specifying dates and locations.
2. **Flight Search Results**: Displays available flights with sorting and filtering options.
3. **Booking Page**: Users can select seats, review details, and confirm their booking.
4. **User Dashboard**: Displays the user's booking history and options to manage existing bookings.

10. Testing:

Testing Tools

- **Frontend**: **Jest** and **React Testing Library** for unit and integration tests.
- **Backend**: **Mocha** and **Chai** for testing API endpoints and logic.

Testing Strategy

- **Unit Testing**: Ensures that individual components and functions work as expected.
- **Integration Testing**: Verifies interactions between components and services.
- **End-to-End Testing**: Tests the entire application from the user's perspective, covering the complete flow from search to booking.

11. PROJECT IMPLEMENTATION & EXECUTION :

11.1 FRONT-END IMPLEMENTATION :

Client>src>components>login.jsx:

```
import { render, screen } from '@testing-library/react';
import App from './App';


test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Client>src>components>navbar.jsx:

```
import React, { useContext } from 'react'
import '../styles/Navbar.css';
import { useNavigate } from 'react-router-dom';
```

```
import { GeneralContext } from '../context/GeneralContext';

const Navbar = () => {

    const navigate = useNavigate();
    const usertype = localStorage.getItem('userType');

    const {logout} = useContext(GeneralContext);

  return (
    <>
      <div className="navbar">

      {!usertype ?

          <>
            <h3 >SB Flights</h3>

            <div className="nav-options" >
              <p onClick={()=>navigate('/')}>Home</p>
              <p onClick={()=>navigate('/auth')}>Login</p>
            </div>
```

```
    </>
:

<>
{usertype === 'customer' ?

<>
    <h3 >SB Flights</h3>

    <div className="nav-options" >

        <p onClick={()=>navigate('/')}>Home</p>
        <p onClick={()=>navigate('/bookings')}>Bookings</p>
        <p onClick={logout}>Logout</p>

    </div>
</>
    :  usertype === 'admin' ?

            <>
                <h3 >SB Flights (Admin)</h3>
```

```
                <div className="nav-options" >

                    <p
onClick={()=>navigate('/admin')}>Home</p>
                    <p onClick={()=>navigate('/all-
users')}>Users</p>
                    <p onClick={()=>navigate('/all-
bookings')}>Bookings</p>
                    <p onClick={()=>navigate('/all-
flights')}>Flights</p>
                    <p onClick={logout}>Logout</p>
                </div>
            </>

        : usertype === 'flight-operator' ?
            <>
                <h3 >SB Flights (Operator)</h3>
                <div className="nav-options" >

                    <p onClick={()=>navigate('/flight-
admin')}>Home</p>
                    <p onClick={()=>navigate('/flight-
bookings')}>Bookings</p>
```

```
                    <p
onClick={()=>navigate('/flights')}>Flights</p>
                    <p onClick={()=>navigate('/new-flight')}>Add
Flight</p>
                    <p onClick={logout}>Logout</p>
                  </div>
                </>

              :

                ""

        }
        </>
        }
        </div>

    </>
  )
}
```

Client>src>components>Register.jsx:

```
import React, { useContext } from 'react'
```

```
import { GeneralContext } from '../context/GeneralContext';


const Register = ({setIsLogin}) => {


  const {setUsername, setEmail, setPassword, usertype,
setUsertype, register, setHomeBranch} =
useContext(GeneralContext);


  const handleRegister = async (e) =>{
    e.preventDefault();
    await register()
  }
  return (
    <form className="authForm">
        <h2>Register</h2>
        <div className="form-floating mb-3 authFormInputs">
            <input type="text" className="form-control"
id="floatingInput" placeholder="username"
                                        onChange={(e)=>
setUsername(e.target.value)} />
            <label htmlFor="floatingInput">Username</label>
        </div>
        <div className="form-floating mb-3 authFormInputs">
```

```
        <input type="email" className="form-control"
id="floatingEmail" placeholder="name@example.com"
                                        onChange={(e)=>
setEmail(e.target.value)} />
        <label htmlFor="floatingInput">Email address</label>
      </div>
      <div className="form-floating mb-3 authFormInputs">
        <input type="password" className="form-control"
id="floatingPassword" placeholder="Password"
                                        onChange={(e)=>
setPassword(e.target.value)} />
        <label htmlFor="floatingPassword">Password</label>
      </div>
      <select className="form-select form-select-lg mb-3"
aria-label=".form-select-lg example"
                                        onChange={(e)=>
setUsertype(e.target.value)}>
        <option value="">User type</option>
        <option value="admin">Admin</option>
        <option value="customer">Customer</option>
        <option value="flight-operator">Flight
Operator</option>
      </select>
```

```
      <button className="btn btn-primary" onClick={handleRegister}>Sign up</button>
      <p>Already registered? <span onClick={()=> setIsLogin(true)}>Login</span></p>
    </form>
  )}
export default Register;
```

client>src>context:

```
import React, { createContext, useState } from 'react';
import axios from "axios";
import { useNavigate } from "react-router-dom";

export const GeneralContext = createContext();

const GeneralContextProvider = ({children}) => {

  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [usertype, setUsertype] = useState('');
```

```
  const [ticketBookingDate, setTicketBookingDate] =
useState();


  const inputs = {username, email, usertype, password};


  const navigate = useNavigate();


  const login = async () =>{
    try{
      const loginInputs = {email, password}
        await axios.post('http://localhost:6001/login',
loginInputs)
        .then( async (res)=>{


            localStorage.setItem('userId', res.data._id);
            localStorage.setItem('userType', res.data.usertype);
            localStorage.setItem('username', res.data.username);
            localStorage.setItem('email', res.data.email);


            if(res.data.usertype === 'customer'){
               navigate('/');
```

```
        } else if(res.data.usertype === 'admin'){
            navigate('/admin');
        } else if(res.data.usertype === 'flight-operator'){
          navigate('/flight-admin');
        }
      }).catch((err) =>{
        alert("login failed!!");
        console.log(err);
      });

  }catch(err){
    console.log(err);
  }
}

const register = async () =>{
  try{
    await axios.post('http://localhost:6001/register', inputs)
    .then( async (res)=>{
      localStorage.setItem('userId', res.data._id);
      localStorage.setItem('userType', res.data.usertype);
      localStorage.setItem('username', res.data.username);
```

```
        localStorage.setItem('email', res.data.email);

        if(res.data.usertype === 'customer'){
           navigate('/');
        } else if(res.data.usertype === 'admin'){
           navigate('/admin');
        } else if(res.data.usertype === 'flight-operator'){
          navigate('/flight-admin');
        }

     }).catch((err) =>{
        alert("registration failed!!");
        console.log(err);
     });
  }catch(err){
     console.log(err);
  }
}


const logout = async () =>{
```

```
    localStorage.clear();
    for (let key in localStorage) {
      if (localStorage.hasOwnProperty(key)) {
        localStorage.removeItem(key);
      }
    }

    navigate('/');
  }


  return (
    <GeneralContext.Provider value={{login, register, logout,
username, setUsername, email, setEmail, password,
setPassword, usertype, setUsertype, ticketBookingDate,
setTicketBookingDate}}
>{children}</GeneralContext.Provider>
  )
}

export default GeneralContextProvider
```

client>src>Routeprocters:

```
import { useEffect } from 'react';

const AuthProtector =  ({ children }) => {

  useEffect(() => {

    if (!localStorage.getItem('userType')) {
      window.location.href = '/';
    }
  }, [localStorage]);

  return children;
};

export default AuthProtector;
```

client>scr>Routeprotecters:

```
import React from 'react'
import { Navigate } from 'react-router-dom';

const LoginProtector = ({children}) => {
```

```
    if (localStorage.getItem('userType')){
        if (localStorage.getItem('userType') === 'customer'){
            return <Navigate to='/' replace />
        }else if (localStorage.getItem('userType') === 'admin'){
            return <Navigate to='/admin' replace />
        }
    }

    return children;
}

export default LoginProtector;
```

11.2 BACK-END IMPLEMENTATION :

Server>index.js:

```
import express from 'express';
import bodyParser from 'body-parser';
import mongoose from 'mongoose';
import cors from 'cors';
import bcrypt from 'bcrypt';
import { User, Booking, Flight } from './schemas.js';
```

```
const app = express();


app.use(express.json());
app.use(bodyParser.json({limit: "30mb", extended: true}))
app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));
app.use(cors());

// mongoose setup


const PORT = 6001;
mongoose.connect('mongodb://localhost:27017/FlightBookingMERN', {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    }
).then(()=>{

    // All the client-server activites


    app.post('/register', async (req, res) => {
        const { username, email, usertype, password } = req.body;
```

```
    let approval = 'approved';
    try {

        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already
exists' });
        }

        if(usertype === 'flight-operator'){
            approval = 'not-approved'
        }

        const hashedPassword = await bcrypt.hash(password,
10);
        const newUser = new User({
            username, email, usertype, password:
hashedPassword, approval
        });
        const userCreated = await newUser.save();
        return res.status(201).json(userCreated);

    } catch (error) {
```

```
        console.log(error);
        return res.status(500).json({ message: 'Server Error' });
      }
  });

  app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    try {

      const user = await User.findOne({ email });

      if (!user) {
        return res.status(401).json({ message: 'Invalid email or password' });
      }
      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res.status(401).json({ message: 'Invalid email or password' });
      } else{

        return res.json(user);
```

```
        }

    } catch (error) {
      console.log(error);
      return res.status(500).json({ message: 'Server Error' });
    }
});


// Approve flight operator

app.post('/approve-operator', async(req, res)=>{
    const {id} = req.body;
    try{

        const user = await User.findById(id);
        user.approval = 'approved';
        await user.save();
        res.json({message: 'approved!'})
    }catch(err){
        res.status(500).json({ message: 'Server Error' });
    }
```

```
})

// reject flight operator

app.post('/reject-operator', async(req, res)=>{
    const {id} = req.body;
    try{

        const user = await User.findById(id);
        user.approval = 'rejected';
        await user.save();
        res.json({message: 'rejected!'})
    }catch(err){
        res.status(500).json({ message: 'Server Error' });
    }
})

// fetch user

app.get('/fetch-user/:id', async (req, res)=>{
    const id = await req.params.id;
    console.log(req.params.id)
```

```
    try{
        const user = await User.findById(req.params.id);
        console.log(user);
        res.json(user);

    }catch(err){
        console.log(err);
    }
})

// fetch all users

app.get('/fetch-users', async (req, res)=>{

    try{
        const users = await User.find();
        res.json(users);

    }catch(err){
        res.status(500).json({message: 'error occured'});
    }
})
```

```
// Add flight


app.post('/add-flight', async (req, res)=>{
    const {flightName, flightId, origin, destination,
departureTime,
                    arrivalTime, basePrice, totalSeats} =
req.body;
    try{


        const flight = new Flight({flightName, flightId, origin,
destination,
                            departureTime, arrivalTime, basePrice,
totalSeats});
        const newFlight = flight.save();


        res.json({message: 'flight added'});


    }catch(err){
        console.log(err);
    }
})
```

```
    // update flight

    app.put('/update-flight', async (req, res)=>{
        const {_id, flightName, flightId, origin, destination,
                departureTime, arrivalTime, basePrice, totalSeats}
= req.body;
        try{

            const flight = await Flight.findById(_id)

            flight.flightName = flightName;
            flight.flightId = flightId;
            flight.origin = origin;
            flight.destination = destination;
            flight.departureTime = departureTime;
            flight.arrivalTime = arrivalTime;
            flight.basePrice = basePrice;
            flight.totalSeats = totalSeats;

            const newFlight = flight.save();

            res.json({message: 'flight updated'});
```

```
    }catch(err){
        console.log(err);
    }
})

// fetch flights

app.get('/fetch-flights', async (req, res)=>{

    try{
        const flights = await Flight.find();
        res.json(flights);

    }catch(err){
        console.log(err);
    }
})

// fetch flight

app.get('/fetch-flight/:id', async (req, res)=>{
```

```
    const id = await req.params.id;
    console.log(req.params.id)
    try{
        const flight = await Flight.findById(req.params.id);
        console.log(flight);
        res.json(flight);

    }catch(err){
        console.log(err);
    }
})

// fetch all bookings

app.get('/fetch-bookings', async (req, res)=>{

    try{
        const bookings = await Booking.find();
        res.json(bookings);

    }catch(err){
        console.log(err);
```

```
    }
  })

  // Book ticket

  app.post('/book-ticket', async (req, res)=>{
    const {user, flight, flightName, flightId,  departure, destination,
            email, mobile, passengers, totalPrice, journeyDate, journeyTime, seatClass} = req.body;
    try{
      const bookings = await Booking.find({flight: flight, journeyDate: journeyDate, seatClass: seatClass});
      const numBookedSeats = bookings.reduce((acc, booking) => acc + booking.passengers.length, 0);

      let seats = "";
      const seatCode = {'economy': 'E', 'premium-economy': 'P', 'business': 'B', 'first-class': 'A'};
      let coach = seatCode[seatClass];
      for(let i = numBookedSeats + 1; i< numBookedSeats + passengers.length+1; i++){
        if(seats === ""){
```

```
            seats = seats.concat(coach, '-', i);
        }else{
            seats = seats.concat(", ", coach, '-', i);
        }
    }
    const booking = new Booking({user, flight, flightName,
flightId, departure, destination,
                                email, mobile, passengers, totalPrice,
journeyDate, journeyTime, seatClass, seats});
    await booking.save();

    res.json({message: 'Booking successful!!'});
  }catch(err){
    console.log(err);
  }
})

// cancel ticket

app.put('/cancel-ticket/:id', async (req, res)=>{
  const id = await req.params.id;
  try{
```

```
            const booking = await
Booking.findById(req.params.id);
            booking.bookingStatus = 'cancelled';
            await booking.save();
            res.json({message: "booking cancelled"});

        }catch(err){
            console.log(err);
        }
    })




        app.listen(PORT, ()=>{
            console.log(`Running @ ${PORT}`);
        });
    }
).catch((e)=> console.log(`Error in db connection ${e}`));
```

Server>schemas.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'}
});
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);
```

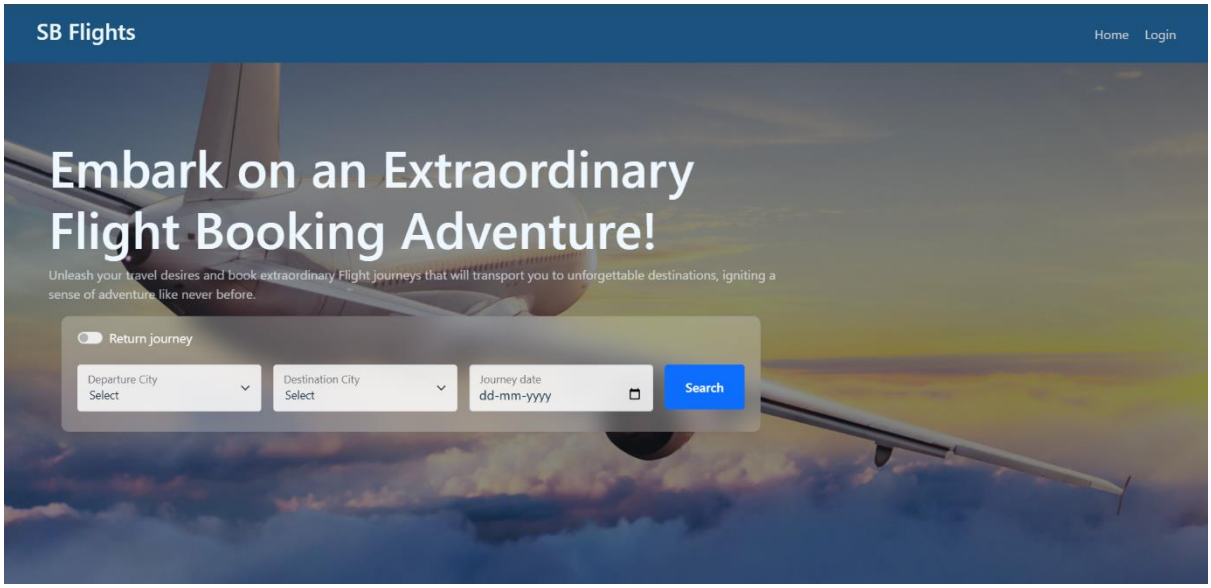
12. Known Issues:

- **Duplicate Results**: Occasionally, the flight search results may display duplicates due to filtering issues. This is planned to be fixed in future updates.
- **Seat Selection Lag**: The seat selection interface may occasionally become slow on older or slower devices, which could be optimized in future versions.

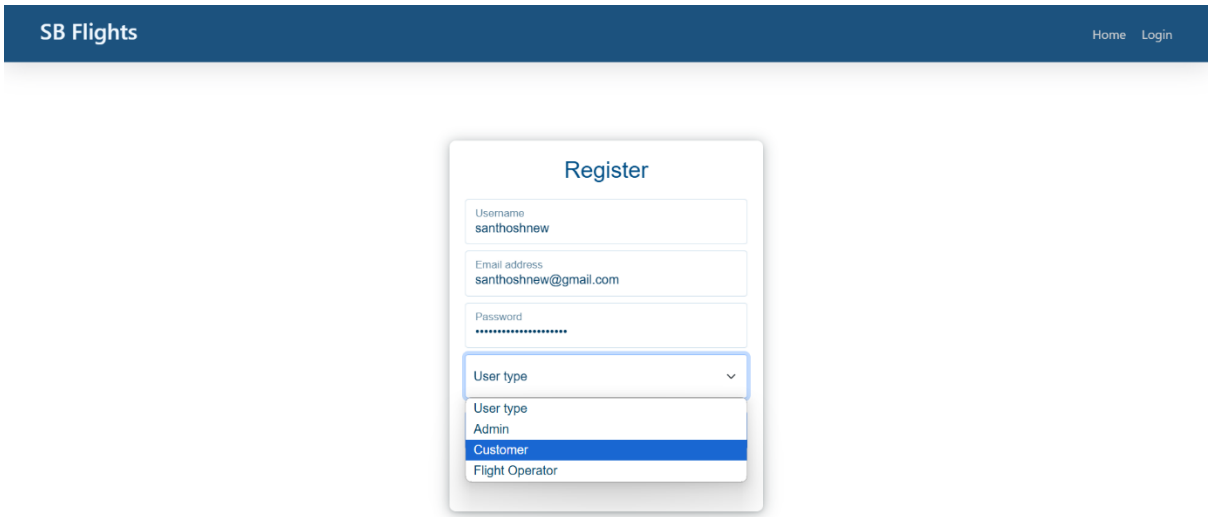
13.SCREENSHOTS:

SCREENSHOTS OF FLIGHT BOOKING APP:

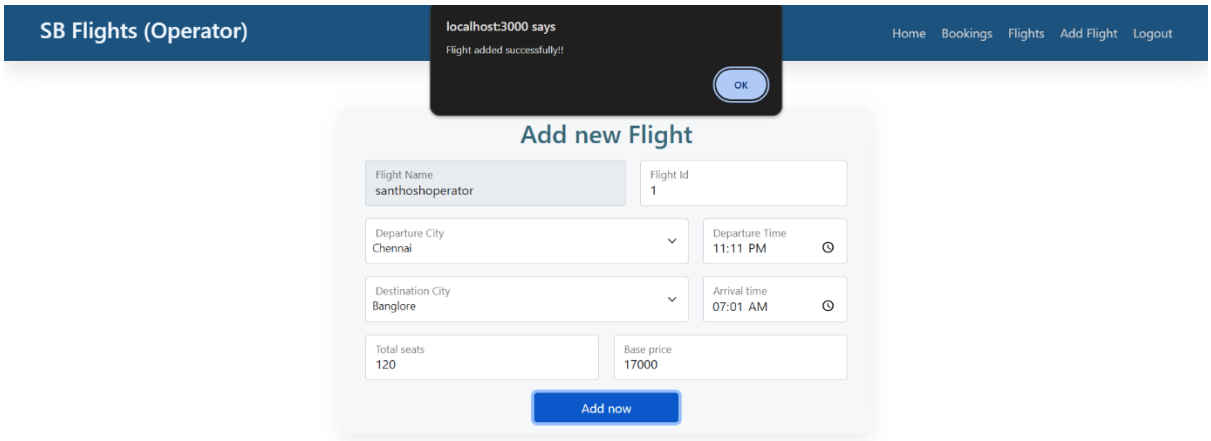
USER PAGE, LANDING PAGE, INTERACTION PAGE:


SB Flights
Home
Login
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.
Return journey
Departure City
Select
Destination City
Select
Journey date
dd-mm-yyyy
Search

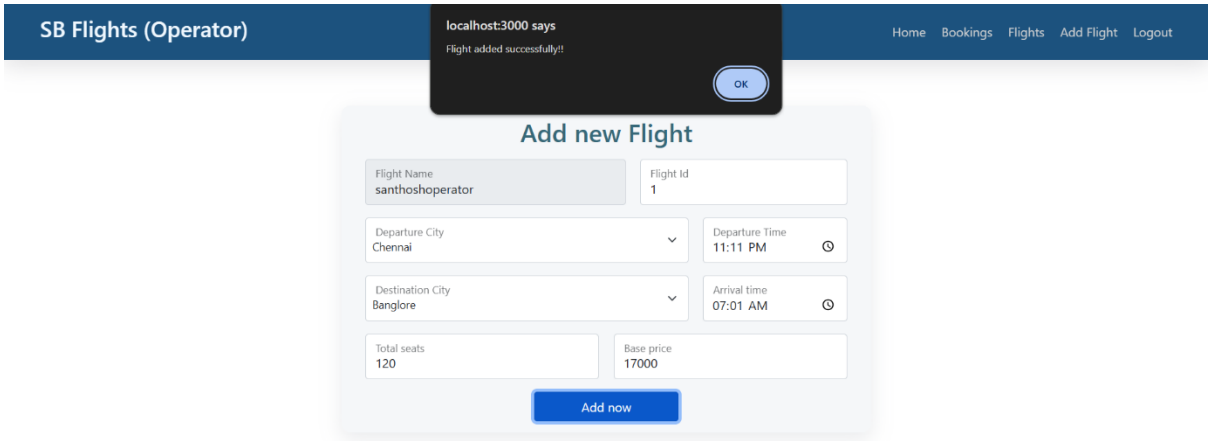
USER LOGIN:

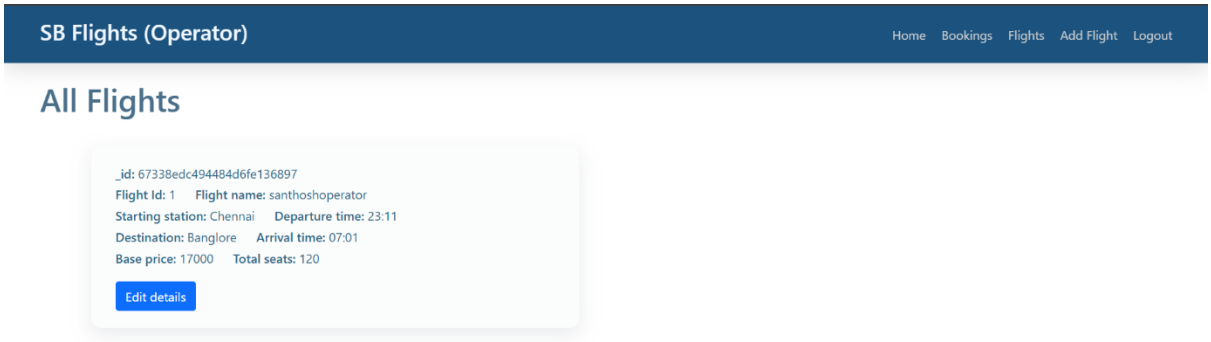

SB Flights
Home
Login
Register
Username
santhoshnew
Email address
santhoshnew@gmail.com
Password
User type
User type
Admin
Customer
Flight Operator

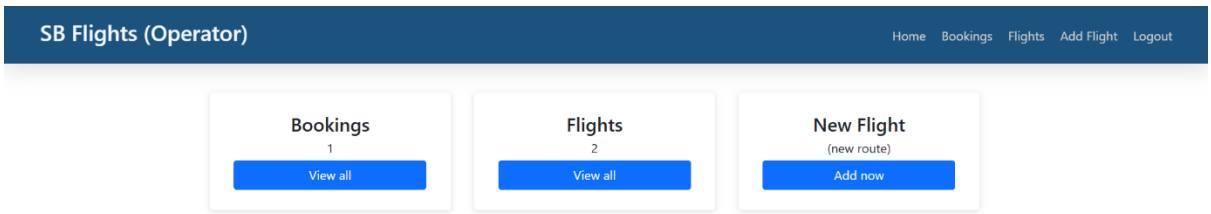
LOGIN AS FLIGHT OPERATOR:



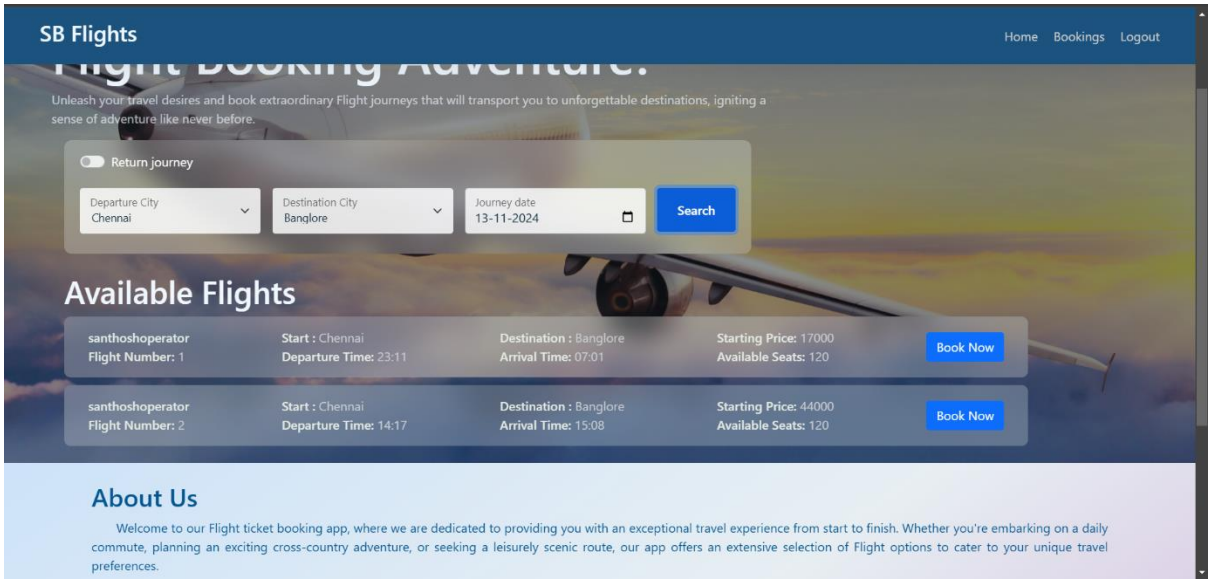
CONTROLL OVER FLIGHTS AS OPERATOR:

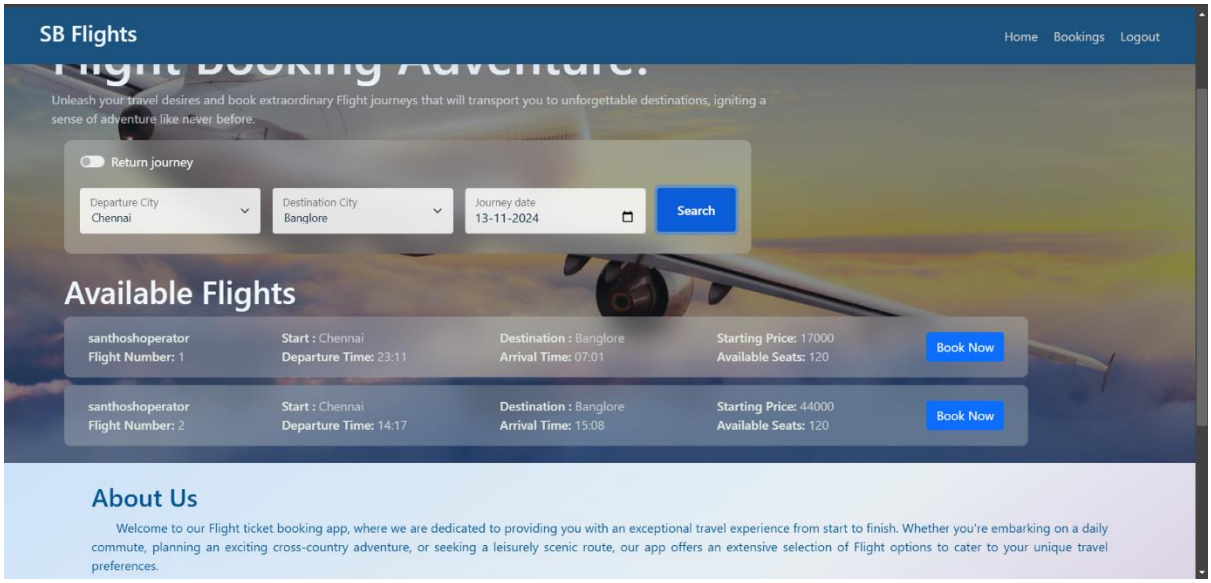


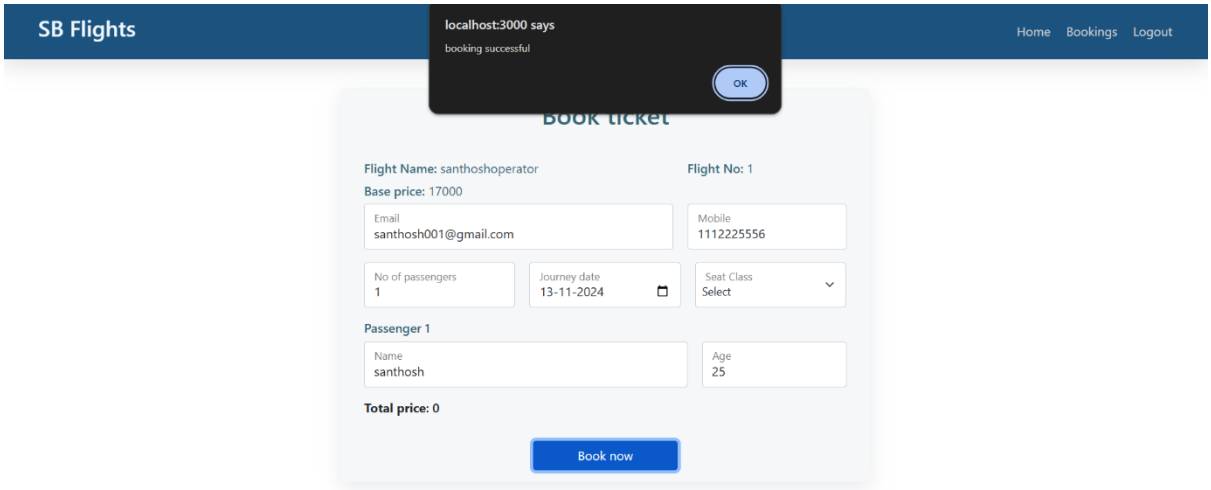


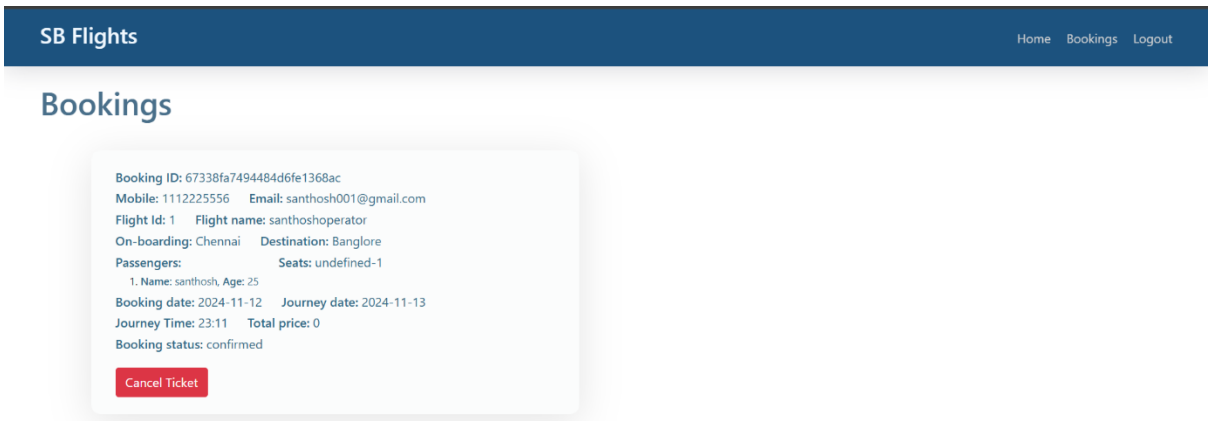


SEARCHING SEATS AND TICKETS:

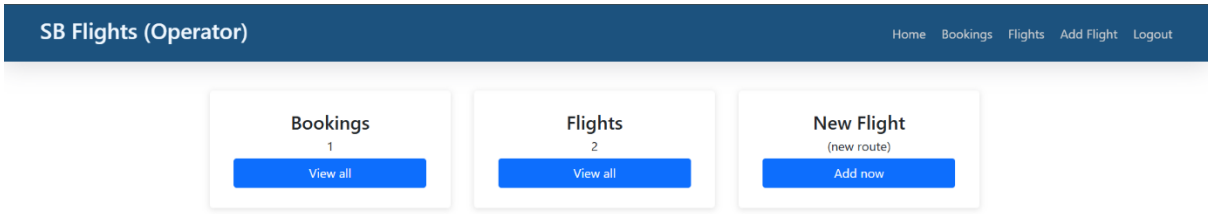

SB Flights
Home
Bookings
Logout
Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.
Return journey
Departure City
Chennai
Destination City
Banglore
Journey date
13-11-2024
Search
Available Flights
santhoshoperator
Flight Number: 1
Start : Chennai
Departure Time: 23:11
Destination : Banglore
Arrival Time: 07:01
Starting Price: 17000
Available Seats: 120
Book Now
santhoshoperator
Flight Number: 2
Start : Chennai
Departure Time: 14:17
Destination : Banglore
Arrival Time: 15:08
Starting Price: 44000
Available Seats: 120
Book Now
About Us
Welcome to our Flight ticket booking app, where we are dedicated to providing you with an exceptional travel experience from start to finish. Whether you're embarking on a daily commute, planning an exciting cross-country adventure, or seeking a leisurely scenic route, our app offers an extensive selection of Flight options to cater to your unique travel preferences.

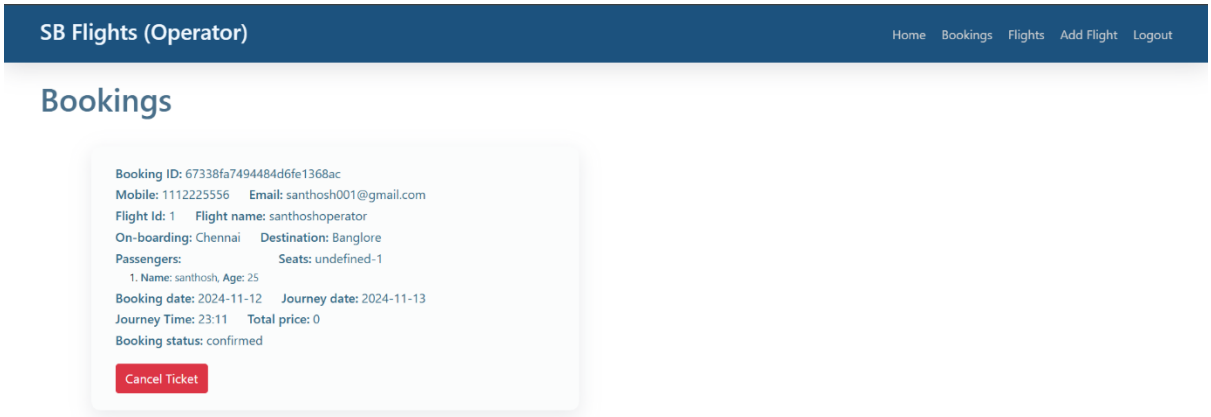

SB Flights
Home
Bookings
Logout
Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.
Return journey
Departure City
Chennai
Destination City
Banglore
Journey date
13-11-2024
Search
Available Flights
santhoshoperator
Flight Number: 1
Start : Chennai
Departure Time: 23:11
Destination : Banglore
Arrival Time: 07:01
Starting Price: 17000
Available Seats: 120
Book Now
santhoshoperator
Flight Number: 2
Start : Chennai
Departure Time: 14:17
Destination : Banglore
Arrival Time: 15:08
Starting Price: 44000
Available Seats: 120
Book Now
About Us
Welcome to our Flight ticket booking app, where we are dedicated to providing you with an exceptional travel experience from start to finish. Whether you're embarking on a daily commute, planning an exciting cross-country adventure, or seeking a leisurely scenic route, our app offers an extensive selection of Flight options to cater to your unique travel preferences.



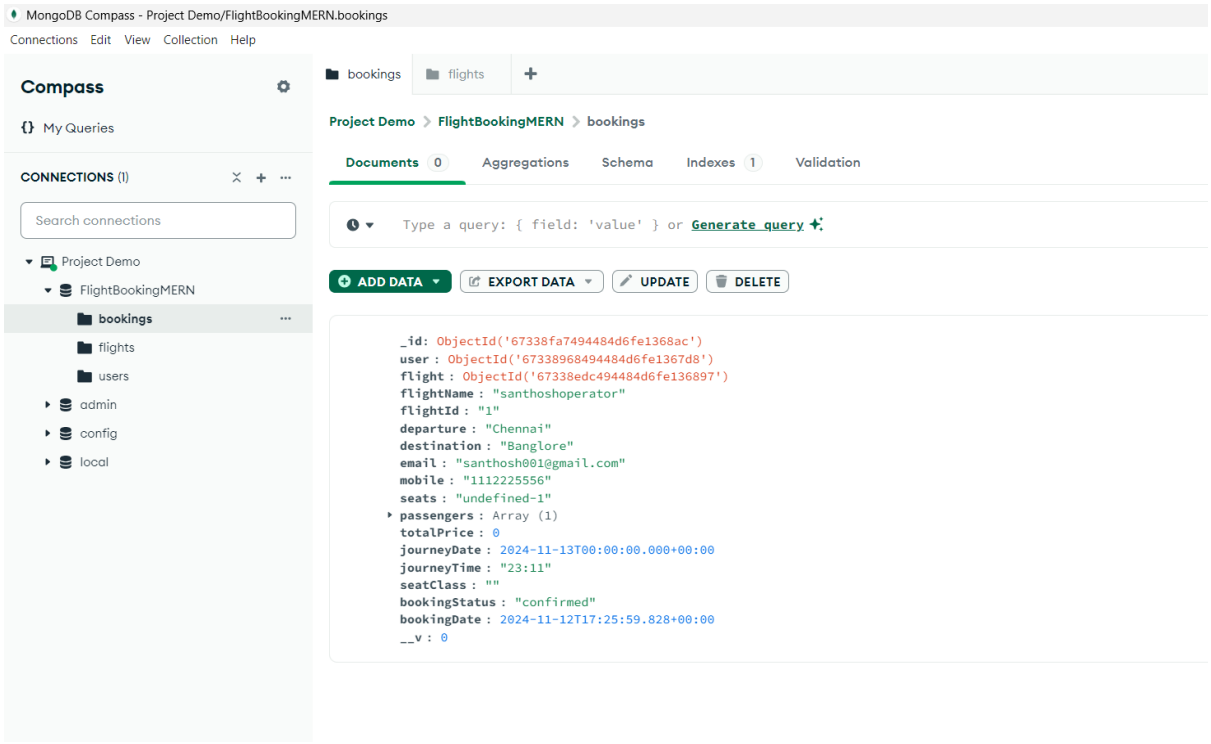


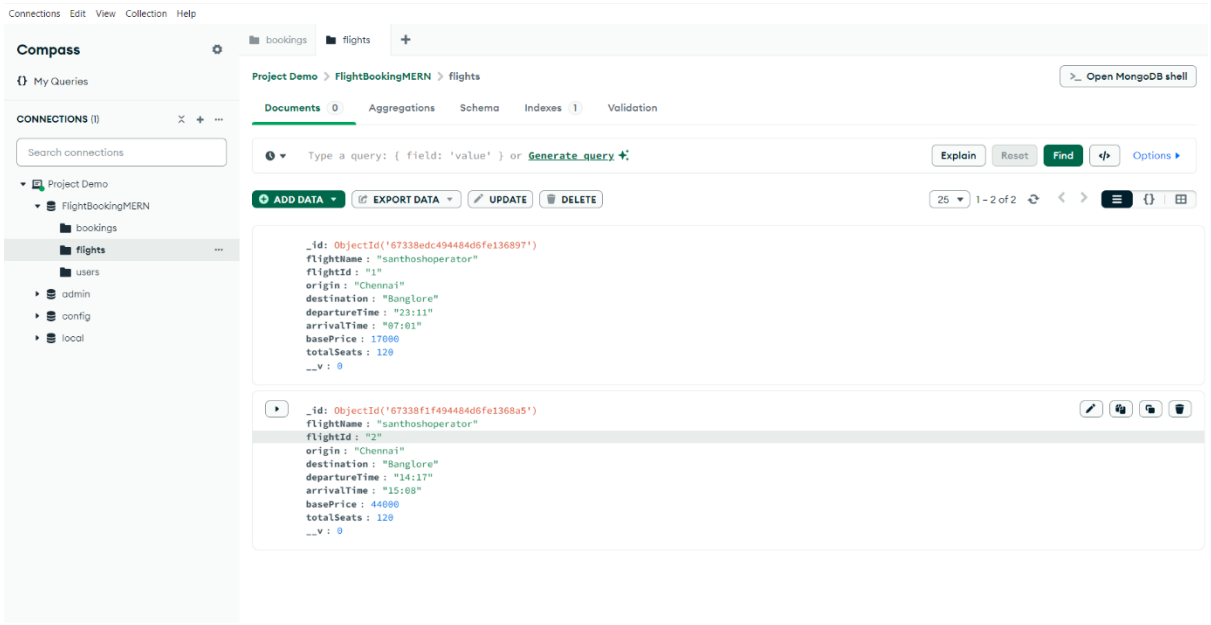
FLIGHT MANAGEMENT:

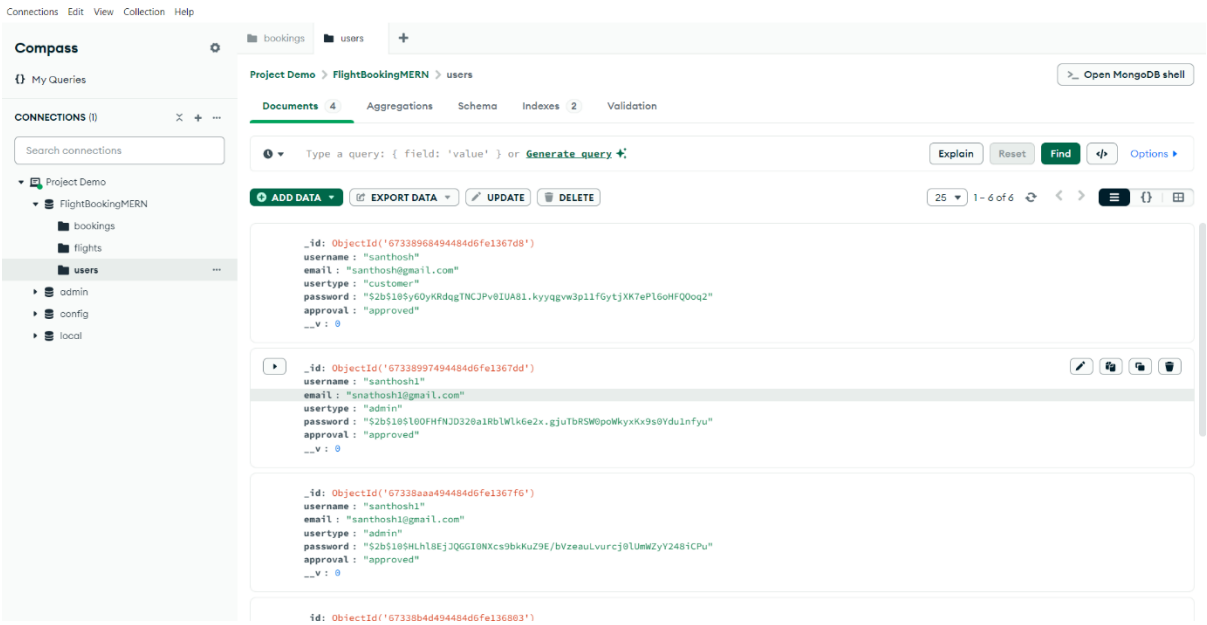




MONGODB COMPASS DATA:


MongoDB Compass - Project Demo/FlightBookingMERN.bookings
Connections Edit View Collection Help
Compass
My Queries
CONNECTIONS (1)
Search connections
Project Demo
FlightBookingMERN
bookings
flights
users
admin
config
local
bookings
flights
Project Demo > FlightBookingMERN > bookings
Documents 0
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
ADD DATA
EXPORT DATA
UPDATE
DELETE
_id: ObjectId('67338fa7494484d6fe1368ac')
user : ObjectId('67338968494484d6fe1367d8')
flight : ObjectId('67338edc494484d6fe136897')
flightName : "santhoshoperator"
flightId : "1"
departure : "Chennai"
destination : "Banglore"
email : "santhosh001@gmail.com"
mobile : "1112225556"
seats : "undefined-1"
passengers : Array (1)
totalPrice : 0
journeyDate : 2024-11-13T00:00:00.000+00:00
journeyTime : "23:11"
seatClass : ""
bookingStatus : "confirmed"
bookingDate : 2024-11-12T17:25:59.828+00:00
__v : 0


Connections Edit View Collection Help
Compass
My Queries
CONNECTIONS (1)
Search connections
Project Demo
FlightBookingMERN
bookings
flights
users
admin
config
local
bookings
flights
Project Demo > FlightBookingMERN > flights
Open MongoDB shell
Documents 0
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
25
1 - 2 of 2
_id: ObjectId('67338edc494484d6fe136897')
flightName : "santhoshoperator"
flightId : "1"
origin : "Chennai"
destination : "Banglore"
departureTime : "23:11"
arrivalTime : "07:01"
basePrice : 17000
totalSeats : 120
__v : 0
_id: ObjectId('67338f1f494484d6fe1368a5')
flightName : "santhoshoperator"
flightId : "2"
origin : "Chennai"
destination : "Banglore"
departureTime : "14:17"
arrivalTime : "15:08"
basePrice : 44000
totalSeats : 120
__v : 0



14. Future Enhancements:

Planned Features

1. **Loyalty Program**: Implement a reward program to encourage frequent usage by offering benefits like discounts or reward points.
2. **Multi-Language Support**: Add support for multiple languages to make the app accessible to a global audience.
3. **Real-Time Flight Updates**: Provide real-time information for booked flights, including delays or gate changes.