FULL STACK DEVELOPMENT WITH MERN PROJECT DOCUMENTATION

TeamDetails:

ProjectTitle:: flightfinder navigating your air travel options

Teamid: LTVIP2025TMID55815

TeamMembers:

1.Team Leader: Nagamani– Full Stack Developer & Project Coordinator Responsible for overall planning, coordination, GitHub management,andintegrationoffrontend and backend.

2.TeamMember: Bala Muddana Venkata Dhanush – FrontendDeveloper Works on the React-based UI, handles component design, page routing, and user interactions.

3.TeamMember:MeghanaVallamsetty– Backend Developer Builds RESTful APIs usingNode.js and Express.js, manages authentication and server logic.

4.Team Member: Kandula Peresh Babu – Database AdministratorDesignsandmanagesMongoDBschemas,handles CRUD operations and ensures data consistency.

FlightBooking APP

INTRODUCTION

Introducing SB Flights, the ultimate digital platformdesigned to revolutionize the way you book flight tickets. With SB Flights, your flight travel experience will be elevated to new heights of convenience and efficiency.

Our user-friendlywebappempowerstravelersto effortlesslydiscover, explore, andreserve 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.

Imagine accessing comprehensive details about each flight journey at your fingertips. From departureandarrivaltimesto flight classesandavailableamenities, you'llhaveallthe information you need to make informed decisions. No more guessing or uncertainty– SB Flights ensures that every aspect of your flight travel is crystal clear.

Thebookingprocessisabreeze.Simplyprovide yourname, age,andpreferredtraveldates,along withthedepartureandarrivalcities,andthenumberofpassengers.Onceyousubmityourbooking request, you'llreceive an instant confirmationofyourticket reservation. No more waiting in long queues or dealing with complicated reservation systems – SB Flights makes it quick and hassle-free.

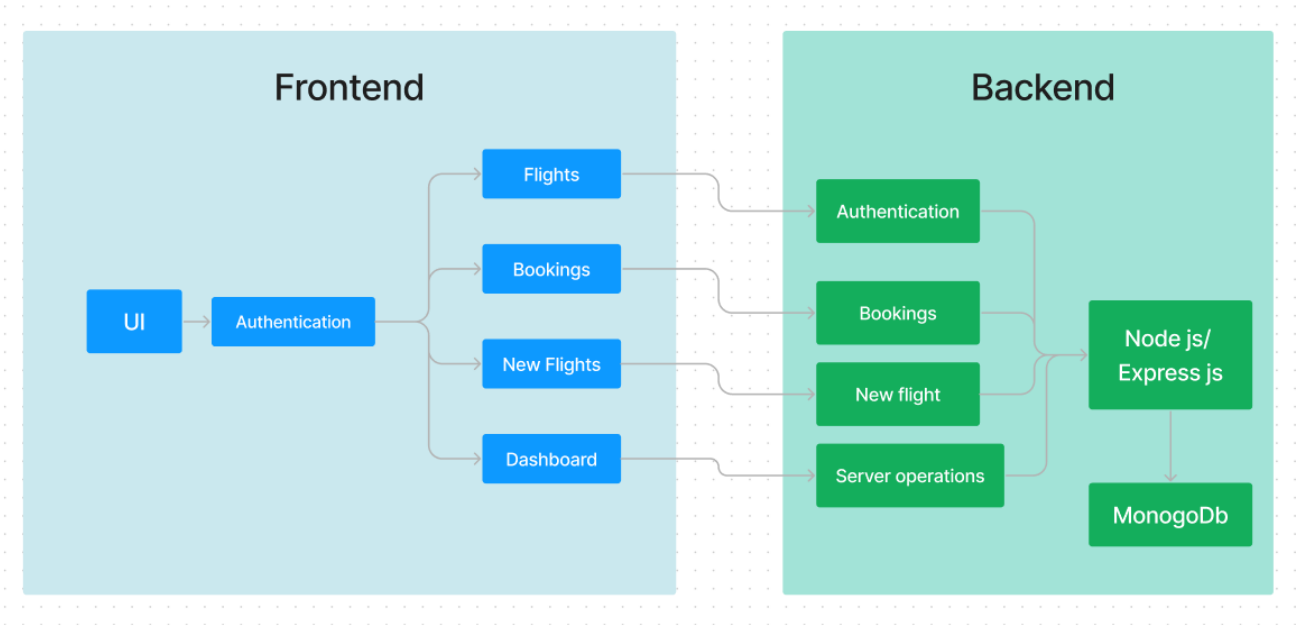
Once your booking is confirmed, our dedicated booking details page becomes your travel companion. It provides a comprehensive overview of your current and previous bookings, allowing you to effortlessly manage your travelplans and stayorganized. With SB Flights, you'll have all your essential travel information at your fingertips, ensuring a stress-free journey.

But SB Flights isn't just for travelers. Flight administrators also benefit from our intuitive admin dashboard.Thisspeciallydesigned dashboard empowersadministratorsto efficientlymanageand overseeticket reservations fortheir flight service. Theycaneasilyviewthelist ofavailable flights for booking and monitor the bookings made by users. With separate login and registration pages for each flight service, privacy and security are always maintained.

SB Flights is hereto enhance yourtravelexperience byproviding aseamless and convenient way to bookflight tickets. Withouruser-friendlyinterface, efficient booking management, and robust administrative features, weensureahassle-freeandenjoyable flightticket bookingexperiencefor both users and flight administrators alike.

GetreadytoembarkonaneweraofflighttravelwithSBFlights–yourtickettoeffortlessbooking and unforgettable journeys.

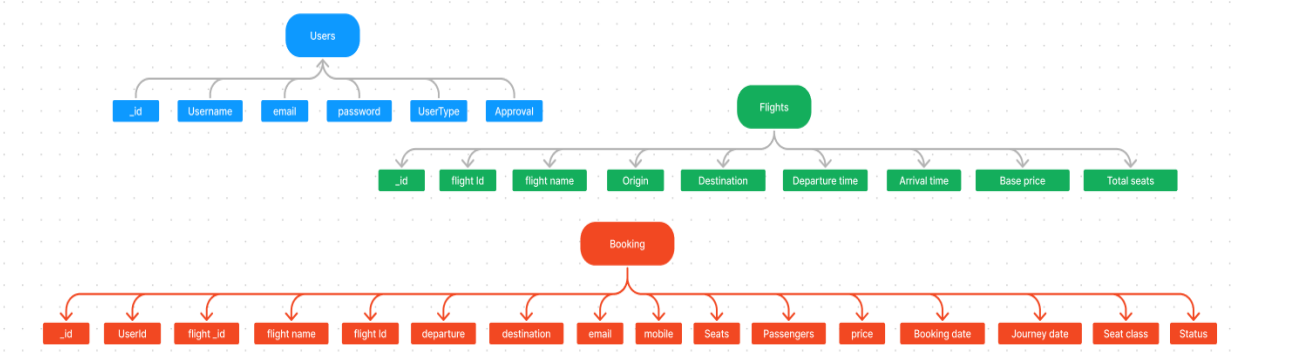
TECHINICALARCHITECTURE:



Inthis architecture diagram:

- Thefrontend isrepresentedbythe"Frontend"section, includinguser interface components such as User Authentication, Flight Search, and Booking.
- Thebackend is represented bythe "Backend" section, consisting ofAPI endpoints forUsers,Flights,AdminandBookings.ItalsoincludesAdminAuthenticationand an Admin Dashboard.
- TheDatabasesectionrepresentsthedatabasethat storescollections for Users, Flights, and Flight Bookings.

ERDIAGRAM:



The flight bookingER-diagramrepresentstheentitiesandrelationships involved ina 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: Representstheindividualsor entitieswhobookflights. Acustomer canplacemultiplebookingsand make multiple payments.

BOOKING: Representsaspecificflightbookingmadebyacustomer. Abookingincludesa particular flight details and passenger information. A customer can have multiple bookings.

FLIGHT: Represents aflight that isavailablefor booking. Here, thedetailsofflight willbeprovidedandthe users can book them as much as the available seats.

ADMIN: Adminisresponsibleforallthebackendactivities.Adminmanagesallthebookings,addsnewflights, etc.,

Features:

1. **Extensive Flight Listing:** SB Flights offers an extensive list of flight services, providing a wide range of routes and options for travelers. You can easily browse through the list and explore different flight journeys, including departure and arrival times, flight classes, and available amenities, to find the perfect travel option for your journey.

2. **Book Now Button:** Each flight listing includes a convenient "Book Now" button. When you find aflight journeythat suits your preferences, simplyclick onthe buttontoproceed withthe reservation process.

3. **Booking Details**: Upon clicking the "Book Now" button, you will be directed to a booking details page. Here, you can provide relevant information such as your preferred travel dates, departureandarrivalstations,thenumberofpassengers,andanyspecialrequirementsyoumay have.

4. **Secure and Efficient Booking Process:** SB Flights ensures a secure and efficient booking process.Yourpersonalinformationwillbehandled withtheutmostcare,andwestrivetomake the reservation process as quick and hassle-free as possible.

5. **Confirmation and Booking Details Page:** Once you have successfully made a reservation, youwillreceiveaconfirmationmessage.Youwillthenberedirectedtoabookingdetailspage, where you can review all the relevant information about your booking, including your travel dates, departure and arrival stations, the number of passengers, and any special requirements you specified.

In addition to these user-facing features, SB Flights provides a powerful admin dashboard, offering administratorsarange of functionalities to efficiently manage the system. With the admin dashboard, admins can add and manage multiple flight services, view the list of available flights, monitor user activity, and access booking details for all flight journeys.

SB Flights is designed to enhance your flight travel experience by providing a seamless and user-friendly way to book flight tickets. With our efficient booking process, extensive flight listings, and robust admin dashboard, we ensure a convenient and hassle-free flight ticket booking experience for both users and flight administrators alike.

PREREQUISITES:

Todevelopafull-stackflight bookingappusingReact JS,Node.js, andMongoDB,thereare severalprerequisites you should consider. Here arethekeyprerequisites for developing such an application:

Node.jsandnpm: InstallNode.js, which includesnpm(NodePackageManager),onyour developmentmachine. Node.js is required to run JavaScript on the server side.

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

MongoDB: Setup a MongoDBdatabasetostore hotel andbookinginformation.Install MongoDB locally oruse a cloud-based MongoDB service.

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

Express.js: Express.js isawebapplicationframeworkforNode.js.InstallExpress.jstohandle server-side routing,middleware, and API development.

- Installation:Openyourcommandpromptor terminal andrun thefollowing command: **npm installexpress**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to createinteractiveand reusableUIcomponents, making it easier to build dynamicand responsive web applications.ToinstallReact.js,aJavaScriptlibraryforbuildinguserinterfaces,followtheinstallation guide: https://reactjs.org/docs/create-a-new-react-app.html

HTML,CSS,andJavaScript: BasicknowledgeofHTMLforcreatingthestructureofyourapp, 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 Mongoosetoconnect your Node.js server with the MongoDBdatabase and perform CRUD (Create, Read, Update, Delete) operations.

Front-endFramework: UtilizeAngulartobuildtheuser-facingpartof theapplication, including productslistings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changesthroughoutthedevelopmentprocess.PlatformslikeGitHuborBitbucketcanhost your repository.

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

DevelopmentEnvironment: ChooseacodeeditororIntegratedDevelopment Environment (IDE) that suits yourpreferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- VisualStudioCode:Downloadfromhttps://code.visualstudio.com/download
- SublimeText:Downloadfromhttps://www.sublimetext.com/download
- WebStorm:Downloadfromhttps://www.jetbrains.com/webstorm/download

ToConnecttheDatabasewithNodeJSgothroughthebelowprovidedlink:

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

TorunttheexistingFlight BookingAppprojectdownloaded fromgithub:

Follow below steps:

Clonetherepository:

- Openyourterminalor commandprompt.
- Navigatetothedirectorywhereyouwanttostorethee-commerceapp.
- Executethefollowingcommandtoclonetherepository:

  Gitclone: https://github.com/Dhanushmuddana/flightfinder.git

Install Dependencies:

- Navigateintotheclonedrepositorydirectory:
  cdFlight-Booking-App-MERN

- Installtherequireddependenciesbyrunningthefollowingcommand:
  npminstall

StarttheDevelopmentServer:

- Tostartthedevelopmentserver,executethefollowingcommand:
  npmrundevornpmrunstart

- Thee-commerceappwillbeaccessibleat http://localhost:3000 bydefault.Youcan change the portconfiguration in the .env file if needed.

AccesstheApp:

- Openyourwebbrowserandnavigateto http://localhost:3000.
- Youshouldseetheflightbookingapp'shomepage,indicatingthattheinstallation and setup weresuccessful.

Youhavesuccessfullyinstalledandsetuptheflightbookingapponyourlocalmachine.You can now proceedwith further customization, development, and testing as needed.

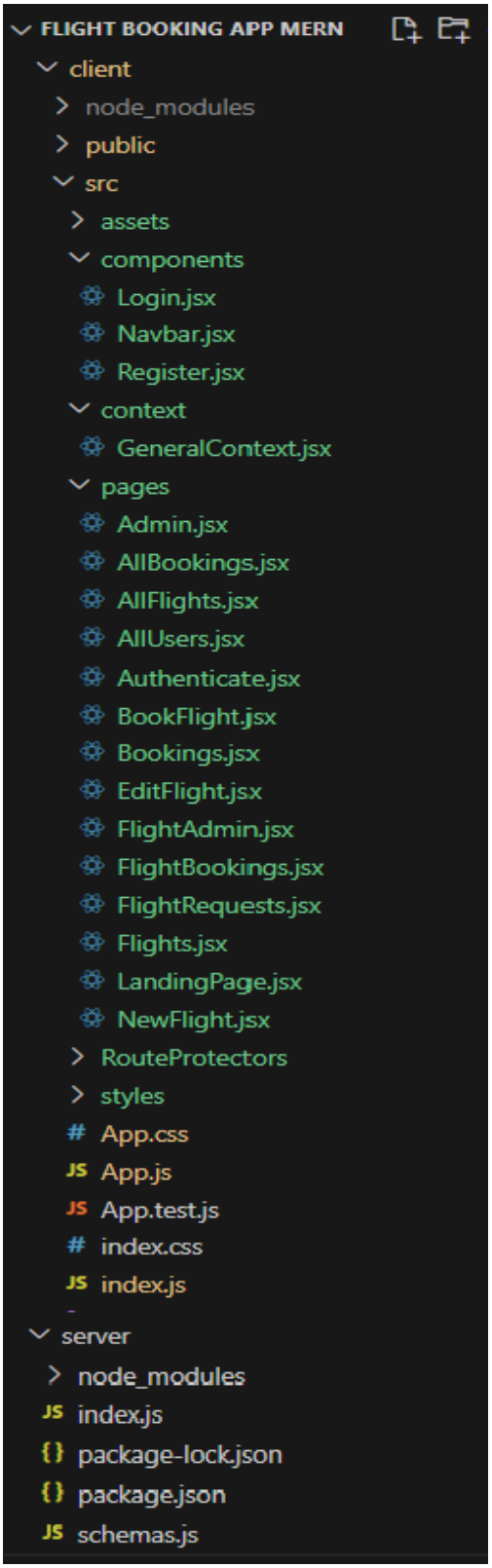
USER&ADMINFLOW:

1. **UserFlow:**
    - Usersstartbyregisteringforanaccount.
    - Afterregistration,theycanloginwiththeir credentials.
    - Oncelogged in, theycancheck fortheavailabilityofflightsintheir desiredroute and dates.
    - Userscanselectaspecificflightfromthelist.
    - Theycanthenproceed byentering passengerdetailsandotherrequireddata.
    - Afterbooking,theycanviewthedetailsoftheirbooking.

2. **FlightOperatorFlow:**
    - Flightoperatorstartbyloggininwiththeir credentials.
    - Onceloggedin,theyaredirectedtotheFlightoperatorDashboard.
    - FlightOperatorcanaccesstheDashboard,wheretheycanviewbookings,add new flight routes, etc.,

3. **Admin Flow:**
    - Adminsstart bylogginginwiththeircredentials.
    - Onceloggedin, theyaredirectedtotheAdminDashboard.
    - AdminscanaccesstheFlight Booking AdminDashboard,wheretheycanview bookings, approve new flight operators, etc.,

PROJECTSTRUCTURE:



ThisstructureassumesaReactappand followsamodularapproach.Here'sabrief explanation of the main directories and files:

- src/components:Containscomponentsrelatedtotheapplication such as, register, login, home, bookings, etc..
- src/pageshasthefilesfor allthepagesintheapplication.

ProjectFlow:

Milestone1:ProjectSetupandConfiguration:

1. **Installrequiredtoolsandsoftware:**
   - Node.js.
   - MongoDB.
   - ReactJs.
   - Git.
2. **Createprojectfoldersandfiles:**
   - Client folders.
   - Serverfolders

Milestone2:BackendDevelopment:

1. **Setupexpressserver:**
   - Installexpress.
   - Createindex.jsfile.
   - DefineAPI’s
2. **ConfigureMongoDB:**
   - InstallMongoose.
   - Createdatabaseconnection.
3. **ImplementAPIendpoints:**
   - ImplementCRUDoperations.
   - TestAPIendpoints.

Milestone3:WebDevelopment:

1. **SetupReact Application:**
   - CreateReactappinclientfolder.
   - Installrequiredlibraries
   - Createrequiredpagesandcomponentsandadd routes.
2. **DesignUIcomponents:**
   - CreateComponents.
   - Implementlayoutandstyling.
   - Addnavigation.
3. **Implementfrontendlogic:**
   - IntegrationwithAPIendpoints.
   - Implementdatabinding.

Createdatabase in cloud link:-
https://cloud.mongodb.com/v2/6864ef939feb171d6264913f#/clusters/starterTemplates

Backend:

1. **SetUpProject Structure:**
   - Createanewdirectoryforyourproject andsetupapackage.jsonfileusingnpminit command.
   - InstallnecessarydependenciessuchasExpress.js,Mongoose,andotherrequired packages.
2. **DatabaseConfiguration:**
   - SetupaMongoDBdatabaseeither locallyorusingacloud-basedMongoDBservice like MongoDB Atlas or use locally with MongoDB compass.
   - Createadatabaseanddefinethenecessarycollectionsforflights,users,bookings, and other relevant data.
3. **CreateExpress.jsServer:**
   - SetupanExpress.jsservertohandleHTTPrequestsandserveAPIendpoints.
   - Configuremiddlewaresuchasbody-parserforparsingrequestbodiesandcorsfor handling cross-origin requests.

4. **DefineAPIRoutes:**
   - CreateseparateroutefilesfordifferentAPIfunctionalitiessuchasflights,users, bookings, and authentication.
   - Definethenecessaryroutesforlistingflights,handlinguserregistrationandlogin, managing bookings, etc.
   - ImplementroutehandlersusingExpress.jstohandlerequestsandinteractwiththe database.

5. **ImplementDataModels:**
   - DefineMongooseschemasforthedifferentdataentitieslikeflights,users,and bookings.
   - Createcorresponding MongoosemodelstointeractwiththeMongoDB database.
   - ImplementCRUDoperations(Create,Read,Update,Delete)foreachmodelto perform database operations.
   -

6. **UserAuthentication:**
   - Createroutesandmiddleware foruserregistration, login,andlogout.
   - Setupauthenticationmiddlewaretoprotectroutesthatrequireuserauthentication.

7. **HandlenewFlightsandBookings:**
   - Createroutesandcontrollerstohandlenewflightlistings,includingfetchingflight data from the database and sending it as a response.
   - Implementbookingfunctionalitybycreatingroutesandcontrollerstohandle booking requests, including validation and database updates.

8. **Admin Functionality:**
   - Implementroutesandcontrollersspecifictoadminfunctionalitiessuchasadding flights, managing user bookings, etc.
   - Addnecessaryauthenticationandauthorizationcheckstoensureonlyauthorized admins can access these routes.

9. **ErrorHandling:**
   - Implementerrorhandling middlewaretocatchandhandleanyerrorsthatoccur during the API requests.
   - ReturnappropriateerrorresponseswithrelevanterrormessagesandHTTPstatus codes.

Schemausecase:

1. **UserSchema:**
   - Schema: userSchema
   - Model:‘User’
   - TheUserschemarepresentstheuserdataandincludesfieldssuchasusername, email, and password.
   - Itisusedtostoreuserinformationforregistrationandauthenticationpurposes.
   - Theemailfieldismarkedasuniquetoensurethateachuser hasauniqueemailaddress**.**

2. **FlightSchema:**
   - Schema:flightSchema
   - Model:‘Flight’
   - TheFlightschemarepresentsthehoteldataandincludesfieldssuchasFlightName, Flight Id, Origin, Destination, Price, seats, etc.,
   - Itisusedtostoreinformationaboutflightsavailableforbookings.

3. **BookingSchema:**
   - Schema: BookingsSchema
   - Model:‘Booking’
   - TheBookingschemarepresentsthebookingdataand includes fieldssuchasuserId, flight Name, flight Id, Passengers, Coach Class, Journey Date, etc.,
   - Itisusedtostoreinformationabouttheflightbookings made byusers.
   - Theuser Idfieldisareferencetotheuser whomadethe booking.

CodeExplanation:

Serversetup:

Letusimport alltherequiredtools/librariesandconnect the database.

index.js

server > index.js > then() callback > app.post('/register') callback

```
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

```

Schemas:

Nowletus definetherequiredschemas

schemas.js

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);
```

UserAuthentication:

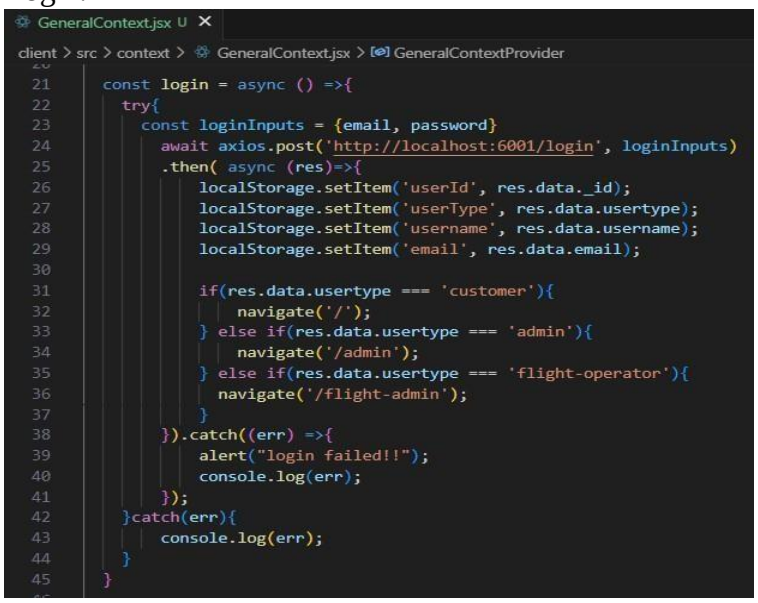
- **Backend**

  Now,herewedefinethefunctionstohandlehttprequestsfromtheclientforauthentication.

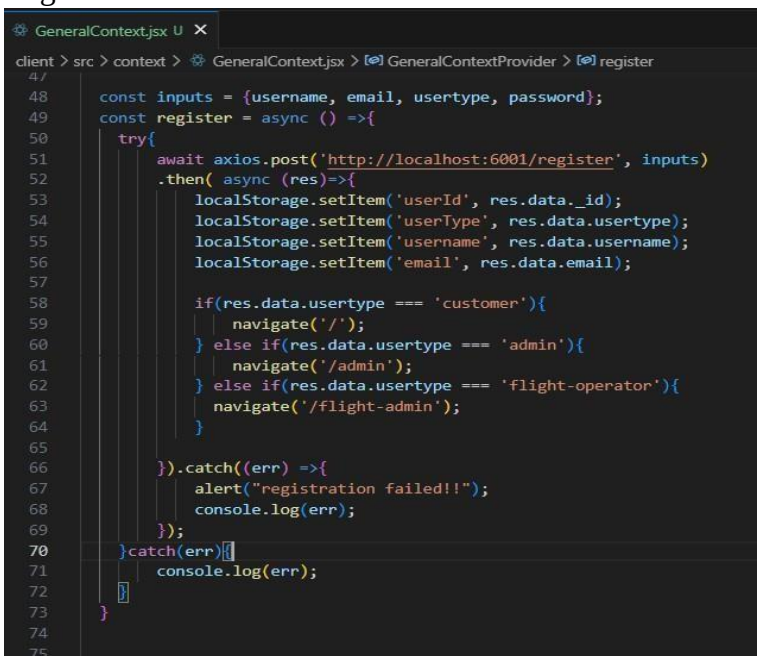
```
JS index.js
server > JS index.js > then() callback
26
27
28      app.post('/register', async (req, res) => {
29          const { username, email, usertype, password } = req.body;
30          let approval = 'approved';
31          try {
32              const existingUser = await User.findOne({ email });
33              if (existingUser) {
34                  return res.status(400).json({ message: 'User already exists' });
35              }
36              if(usertype === 'flight-operator'){
37                  approval = 'not-approved'
38              }
39              const hashedPassword = await bcrypt.hash(password, 10);
40              const newUser = new User({
41                  username, email, usertype, password: hashedPassword, approval
42              });
43              const userCreated = await newUser.save();
44              return res.status(201).json(userCreated);
45          } catch (error) {
46            console.log(error);
47            return res.status(500).json({ message: 'Server Error' });
48          }
49      });
50
51      app.post('/login', async (req, res) => {
52          const { email, password } = req.body;
53          try {
54              const user = await User.findOne({ email });
55              if (!user) {
56                  return res.status(401).json({ message: 'Invalid email or password' });
57              }
58              const isMatch = await bcrypt.compare(password, user.password);
59              if (!isMatch) {
60                  return res.status(401).json({ message: 'Invalid email or password' });
61              } else{
62                  return res.json(user);
63              }
64          } catch (error) {
65            console.log(error);
66            return res.status(500).json({ message: 'Server Error' });
67          }
68      });
```

- **Frontend**

Login:



Register:



Logout:

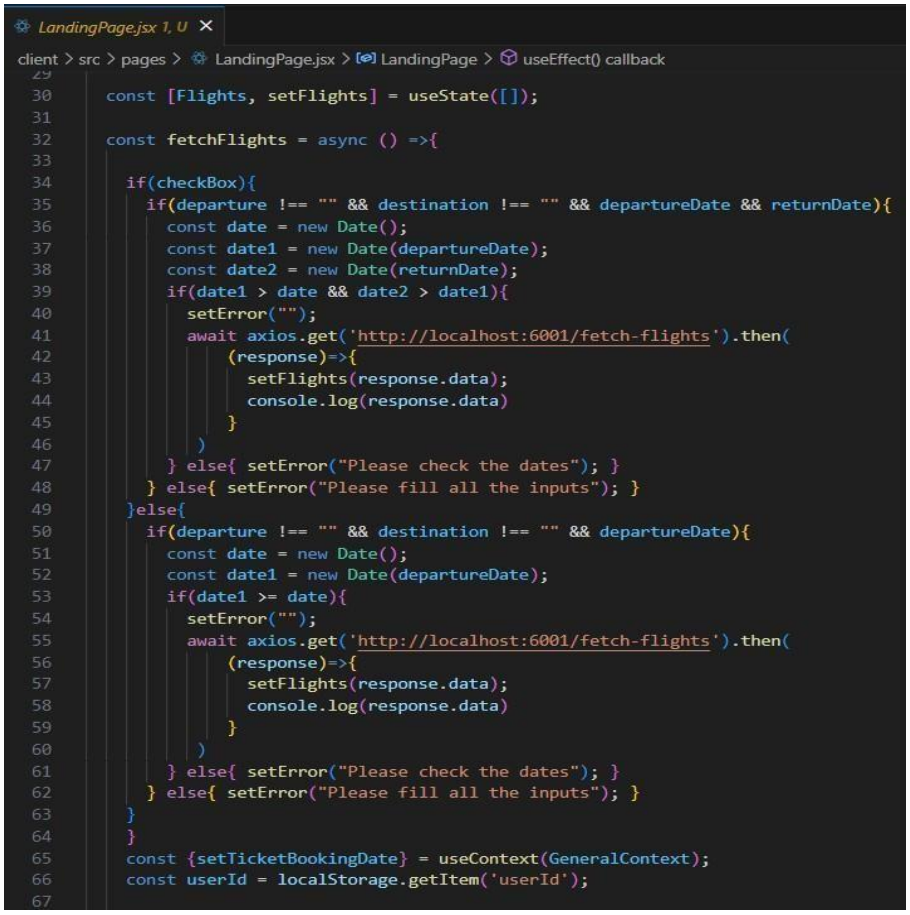
FlightBooking (User):

- **Frontend**

  Inthefrontend,weimplementedallthebookingcodeinamodal.Initially,weneedto implement flight searching feature with inputs of Departure city, Destination, etc.,

  FlightSearchingcode:

  Withthegiveninputs, weneedto fetchtheavailableflights. Witheachflight, weaddabutton to book the flight, which re-directs to the flight-Booking page.

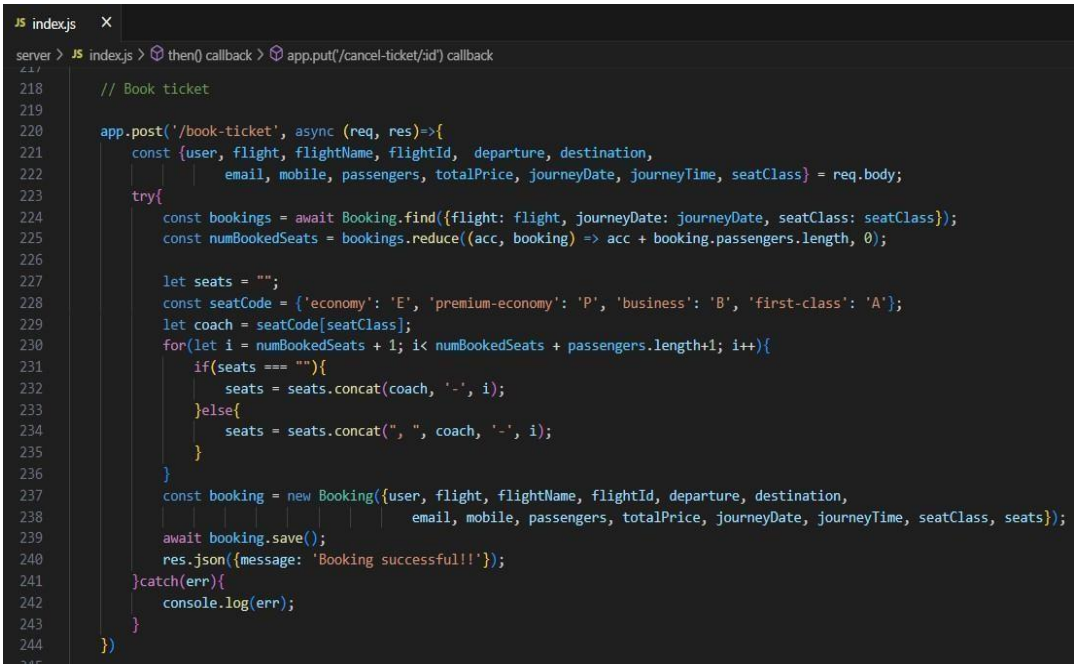


Onselectingthesuitableflight,wethenre-directtotheflight-booking page.

- **Backend**

Inthebackend,wefetchalltheflightsandthenfilterthemintheclientside.

Then,onconfirmation,webooktheflight ticketwiththeentered details.



Fetchinguserbookings:

- **Frontend**

Inthebookingspage, alongwithdisplayingthepastbookings, wewillalso provide an option to cancel that booking.

Bookings.jsx U

client > src > pages > Bookings.jsx > Bookings

```jsx
const [bookings, setBookings] = useState([]);

const userId = localStorage.getItem('userId');

useEffect(()=>{
  fetchBookings();
}, [])

const fetchBookings = async () =>{
  await axios.get('http://localhost:6001/fetch-bookings').then(
    (response)=>{
      setBookings(response.data);
    }
  )
}
const cancelTicket = async (id) =>{
  await axios.put(`http://localhost:6001/cancel-ticket/${id}`).then(
    (response)=>{
      alert("Ticket cancelled!!");
      fetchBookings();
    }
  )
}
```

- **Backend**

  Inthebackend, wefetchallthebookingsandthenfilterfortheuser. Otherwise, wecan fetch bookings only for the user.

index.js

server > index.js > then() callback

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

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

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

Thenwedefineafunctiontodeletethebookingoncancellingitonclientside.

index.js

server > index.js > ...

```jsx
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);
    }
})
```

Addnewflight:

Now,intheadmindashboard,weprovideafunctionalitytoaddnewflight.

- **Frontend**

  Wecreateahtmlformwithrequired inputs forthenew flight and thensendanhttp request to the server to add it to database.

NewFlight.jsx U

client > src > pages > NewFlight.jsx > NewFlight

```
30
31      const [flightName, setFlightName] = useState(localStorage.getItem('username'));
32
33      const [flightId, setFlightId] = useState('');
34      const [origin, setOrigin] = useState('');
35      const [destination, setDestination] = useState('');
36      const [startTime, setStartTime] = useState('');
37      const [arrivalTime, setArrivalTime] = useState('');
38      const [totalSeats, setTotalSeats] = useState(0);
39      const [basePrice, setBasePrice] = useState(0);
40
41      const handleSubmit = async () =>{
42
43        const inputs = {flightName, flightId, origin, destination,
44                        departureTime: startTime, arrivalTime, basePrice, totalSeats};
45
46        await axios.post('http://localhost:6001/add-Flight', inputs).then(
47          async (response)=>{
48            alert('Flight added successfully!!');
49            setFlightName('');
50            setFlightId('');
51            setOrigin('');
52            setStartTime('');
53            setArrivalTime('');
54            setDestination('');
55            setBasePrice(0);
56            setTotalSeats(0);
57          }
58        )
59      }
60
```

- **Backend**

  Inthebackend,onreceivingtherequestfromtheclient,wethenaddtherequestbody to the flight schema.

index.js

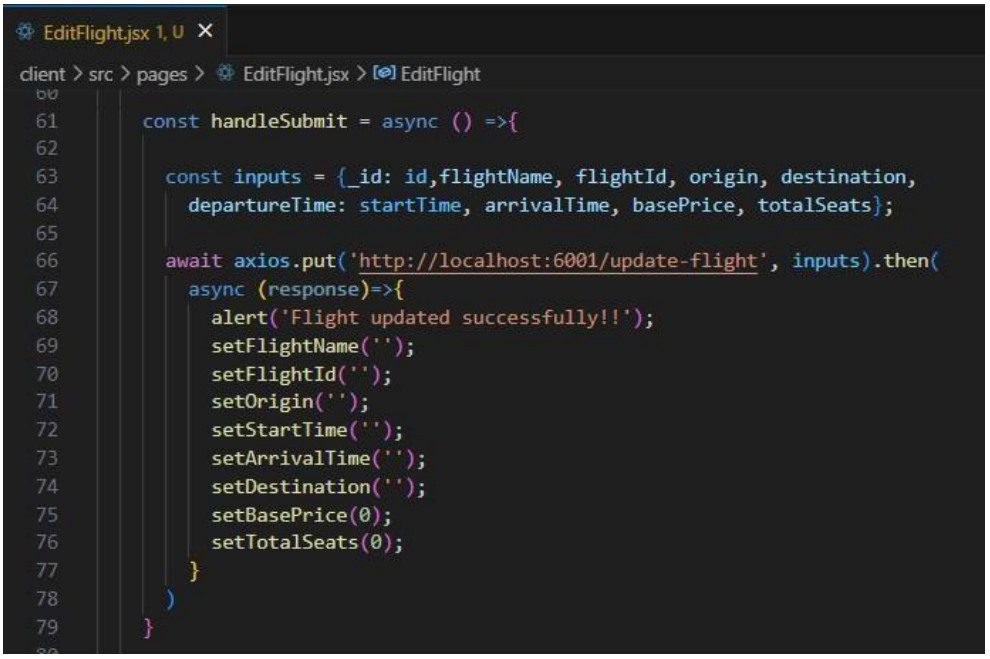
server > index.js > then() callback

```
131     // Add flight
132
133     app.post('/add-flight', async (req, res)=>{
134         const {flightName, flightId, origin, destination, departureTime,
135                                 arrivalTime, basePrice, totalSeats} = req.body;
136         try{
137             const flight = new Flight({flightName, flightId, origin, destination,
138                                         departureTime, arrivalTime, basePrice, totalSeats});
139             const newFlight = flight.save();
140             res.json({message: 'flight added'});
141         }catch(err){
142             console.log(err);
143         }
144     })
```

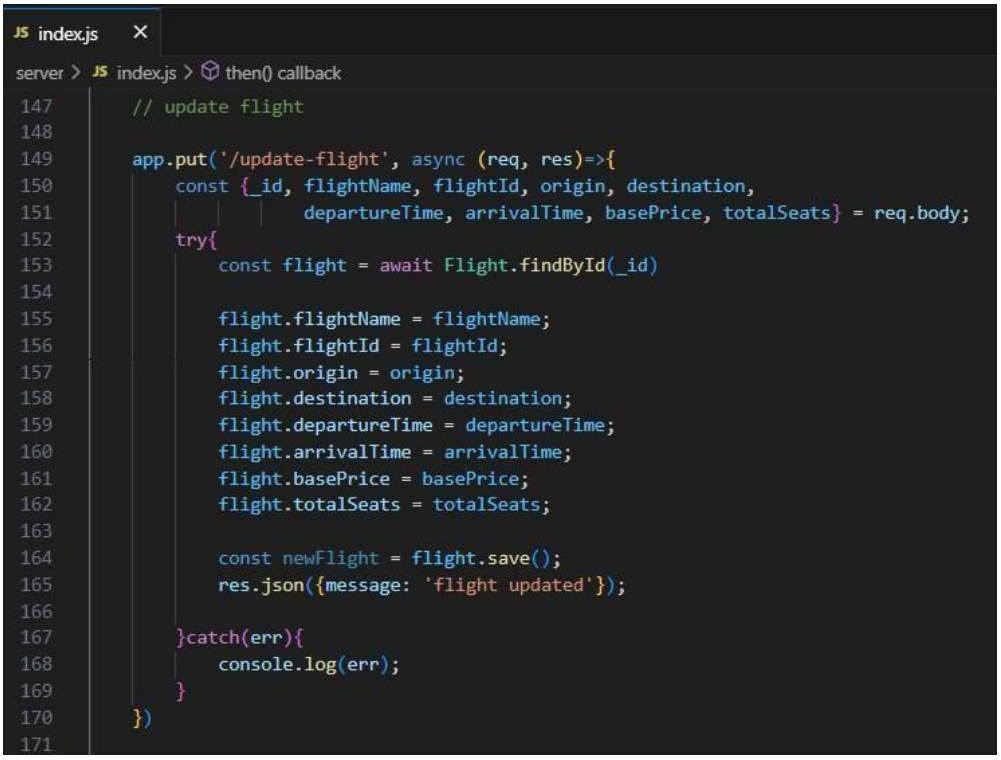
UpdateFlight:

Here,intheadmindashboard,wewillupdatetheflight detailsincaseifwewant to make any edits to it

- **Frontend:**



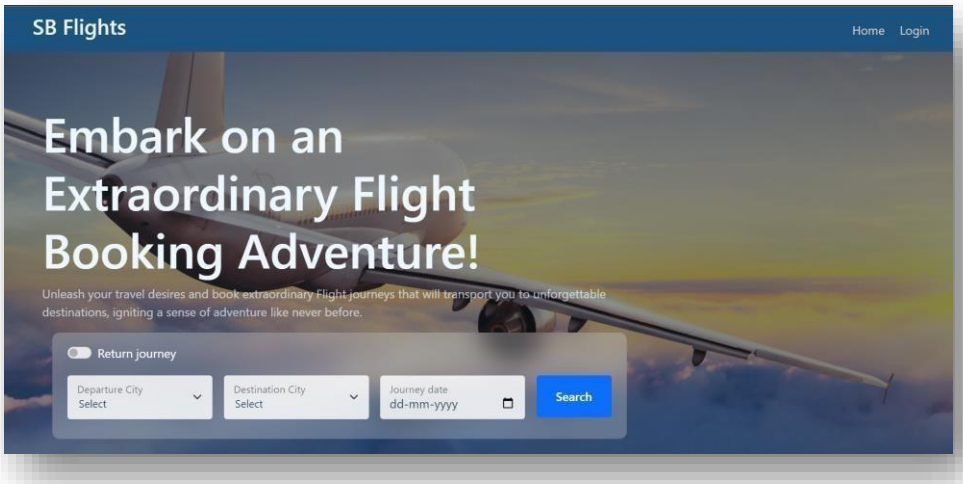
- **Backend:**



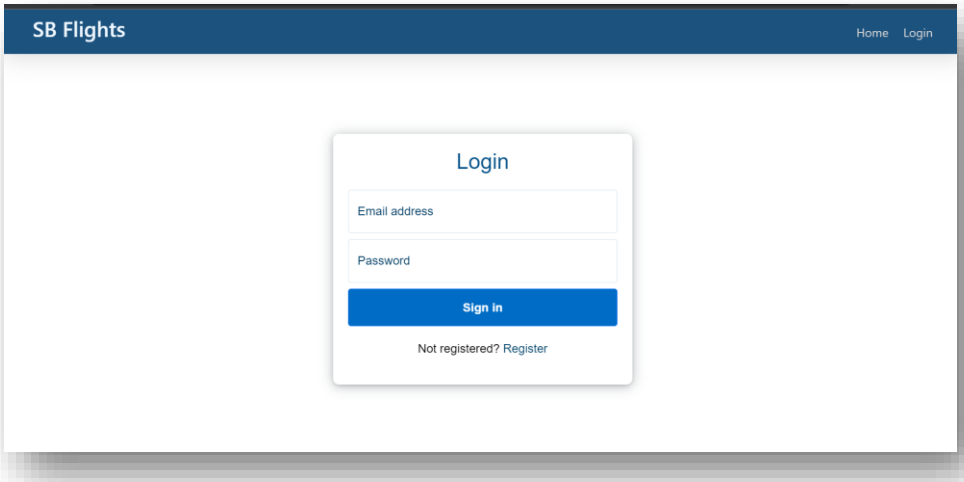
Alongwiththis,implement additionalfeaturestoviewallflights,bookings,andusersin admin dashboard.

DemoUI images:

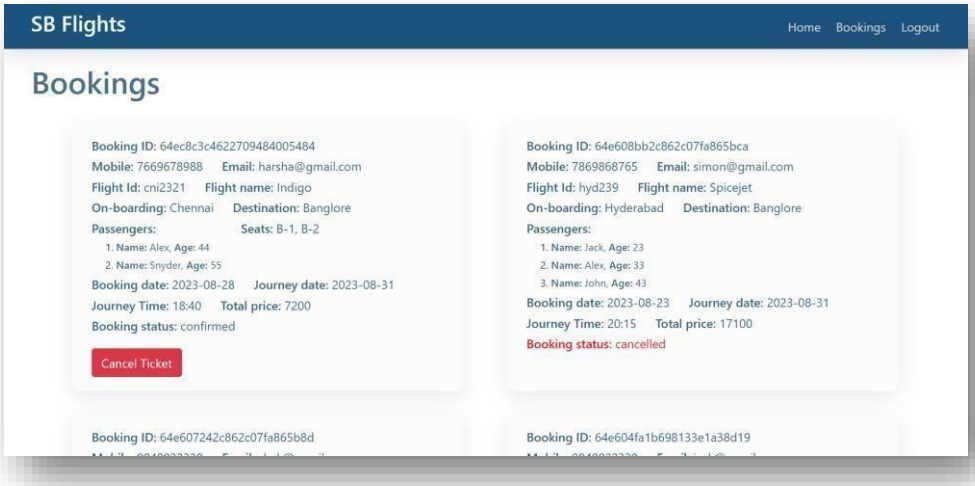
- **Landingpage**



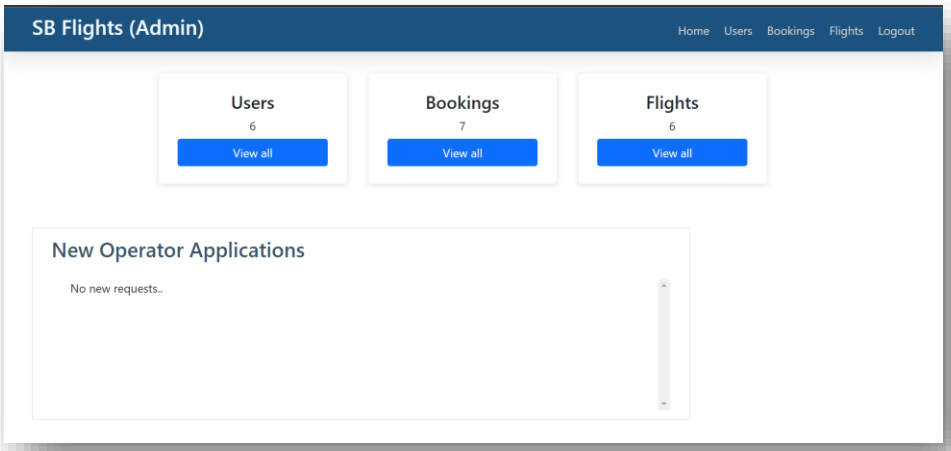
- **Authentication**



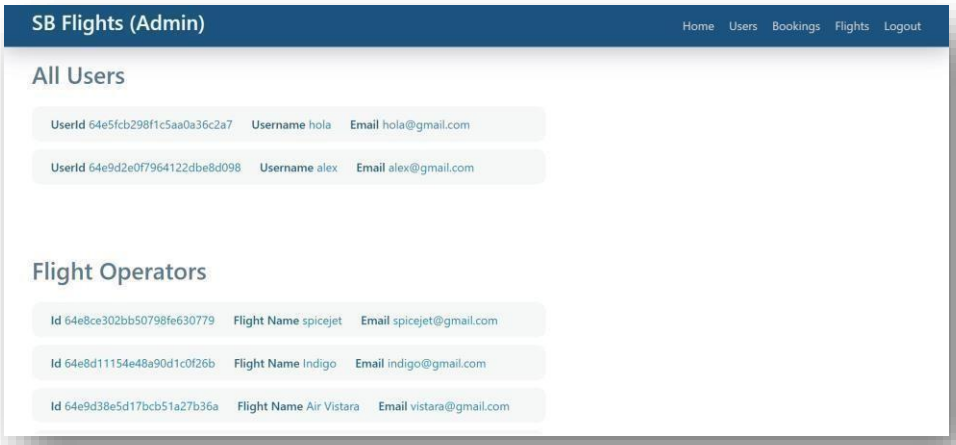
- **Userbookings**



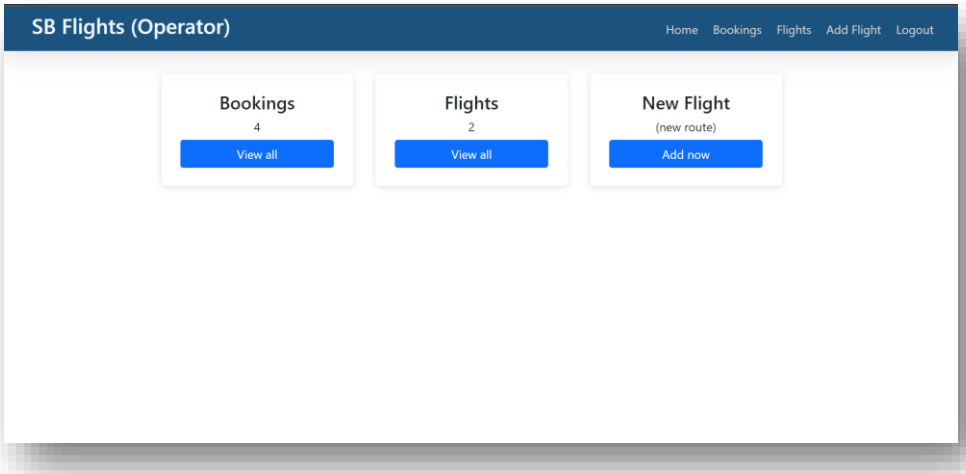
- **AdminDashboard**



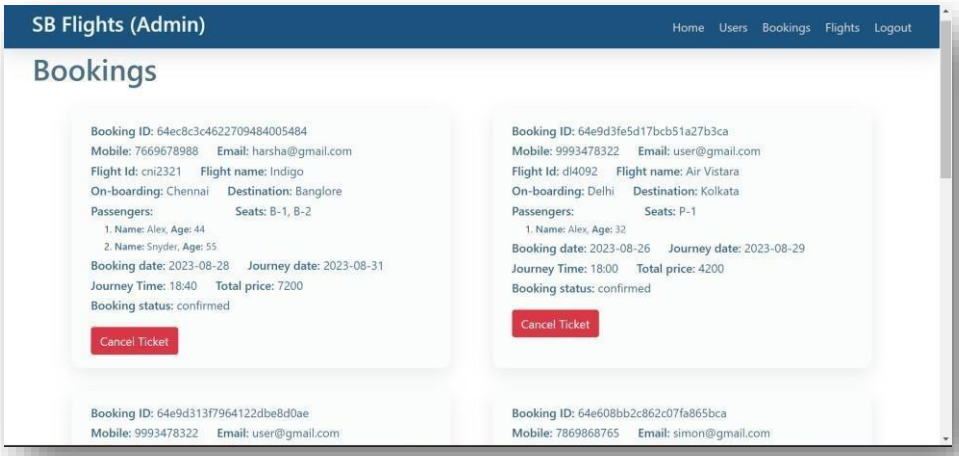
- **Allusers**



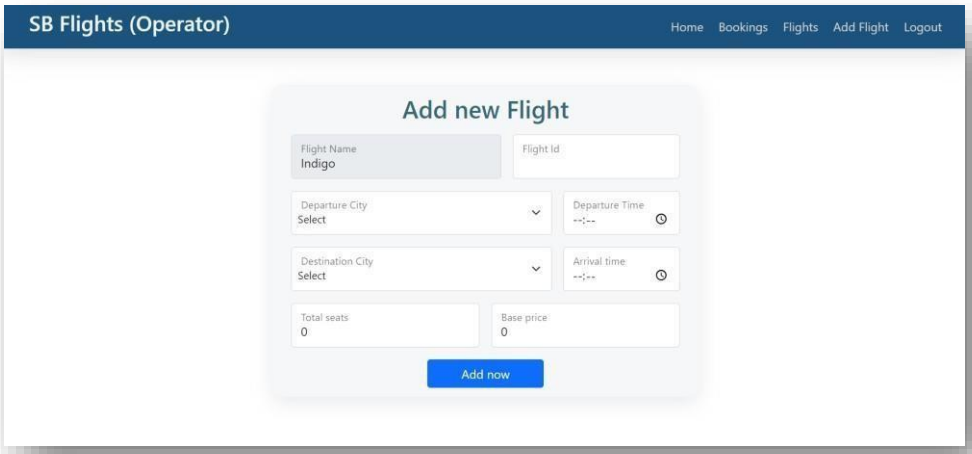
- **FlightOperator**



- **All Bookings**



- **New Flight**



For any further doubts or help,please consider the GitHub repo:
https://github.com/Dhanushmuddana/flightfinder.git

The demo of the app is available at:
https://drive.google.com/file/d/1Q0XwKtAz7EkaKNJv3_gbo6mZE9nfuBTK/view?usp=drive_link

HappyCoding