

FLIGHT BOOKING APP USING MERN NAANMUDHALVAN PROJECT REPORT

SUBMITTED BY

NM TEAM ID : NM2024TMID16174

NAME	REG NO
NADHIYA D	312521104312
RAJESH KANNAN S	312521104315
SANJAYKUMAR C	312521104318
SACHINKUMAR R	312521104316
THIYAGARAJAN V	312521104320

IN PARTIAL FULFILMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



T.J. INSTITUTE OF TECHNOLOGY

KARAPAKKAM

CHENNAI 600097

ANNA UNIVERSITY

CHENNAI 600025



TJ INSTITUTE OF TECHNOLOGY

APPROVED BY AICTE AND AFFILIATED TO ANNA UNIVERSITY, CHENNAI

RAJIV GANDHI SALAI, OMR KARAPAKKAM, CHENNAI-600 097

ANNA UNIVERSITY CHENNAI 600025

BONAFIDE CERTIFICATE

**Certified that this project FLIGHT BOOKING APP
USING MERN STACK is the bonafide work of NADHIYA & TEAM
MEMBERS (RAJESHKANNAN , SANJAYKUMAR, SACHINKUMAR,
THIYAGARAJAN**

SIGNATURE

MS.D.EVANGELINE NESA PRIYA

HEAD OF THE DEPARTMENT

DEPARTMENT OF COMPUTER SCIENCE

AND ENGINEERING

TJ INSTITUTE OF TECHNOLOGY

SIGNATURE

MR.MOHAMMAD ARASATH

CLASS INCHARGE

DEPARTMENT OF COMPUTER SCIENCE

AND ENGINEERING

TJ INSTITUTE OF TECHNOLOGY

**Submitted for University Practical Examinations held on at
T.J.Institute of Technology, Chennai-6000097.**

INTERNALEXAMINER

EXTERNALEXAMINER

ACKNOWLEDGEMENT

The success and final outcome of learning machine learning required a lot of guidance and assistance from many people and I am extremely privileged to have until now got this all along the completion of my course and few of the projects. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and am thankful to and fortunate enough to get constant encouragement, support and guidance from all teaching staffs which helped me in successfully completing my internship and project work.

Project is done by

NADHIYA D(Team Leader)

RAJESH KANNAN S

SANJAYKUMAR C

SACHINKUMAR R

THIYAGARAJAN V

Date: _____

S.NO:	TABLE OF CONTENT
1	Introduction
2	Project Overview
3	Architecture – Backend , Frontend , Database
4	Setup Instruction – prerequisites , Installation
5	Folder Structure – Client , Server
6	Running The App – Frontend , Backend
7	API Documentation
8	Authentication & Authorization
9	User Interface
10	Testing and Tools
11	Known Issues
12	Future Enhancements
13	Screenshots

PROJECT REPORT OF FLIGHT BOOKING APP USING MERN

INTRODUCTION:

- Project Title: Flight booking app using MERN.
- Team members:
 1. Nadhiya (Project Team Leader)
Role:
 - Design and implement the server-side logic using Node.js and Express.js.
 - Create RESTful APIs to handle requests for flight data, user management, and bookings.
 - Integrate with a database (MongoDB) to manage and store data related to flights, users, and bookings.
 - Ensure security best practices are followed for API endpoints.
 - Collaborate with the frontend developer to ensure seamless integration of APIs.
 2. Rajeshkannan
Role:
 - Coordinate tasks among team members and ensure deadlines are met.
 - Facilitate communication between team members.
 - Create progress reports.
 - Manage resources, risks, and any issues that arise during the project.
 3. Sachin Kumar
Roles:

- Develop components for the flight search, booking, and user profile functionalities.
- Implement state management (e.g., using Redux or Context API) to manage the application's state.
- Ensure the application is responsive and provides a good user experience.
- Work closely with the backend developer to consume APIs and handle data.

4. Sanjay Kumar

Role:

- Develop and execute test plans to ensure the application functions as intended.
- Perform manual and automated testing of the frontend and backend.
- Identify and document bugs and issues, and work with developers to resolve them.

5. Thiagarajan

Role:

- Create progress reports.
- Support for Project design to Team Members
- Handling a Software data to upload
- Handling a frontend and backend Support

The Flight ticket booking app is composed of the following Features:

Front-End

- Sign-In & Sign-Up Pages.
- Uses Token based system, so only registered users can access the website passport js.
- Password hashing using passport js.

- Has a profile page, which will display all information about the signed in user.
- List of cities for users to choose from (starting city & destination city).
- Getting list of flight's of different airlines with various details.
- Seat selection page has a very user friendly environment, which also generates dynamic forms for storing data's of passengers.
- Has a Confirmation page, which gets a debit card data using react-credit-cards. This version of the application does not include handling the payment process.
- Final page has a boarding pass displaying component, it displays all passenger data and also generates a random number as a transaction ID.
- Ticket Cancellation page will cancel the ticket which was booked.
- Also has an integrated ai chatbot

Back-End :

- Uses Express js based application for the backend process.
- Uses MongoDB atlas for storing the collections.
- Uses passport js for authenticating user and token based system.
- Uses passport js for hashing the password before sending the data to the cloud.
- This version does not support dynamic seat data being stored from cloud.

Overview:

Purpose of the Flight Booking Project

1. User-Friendly Experience:

- To provide travelers with an intuitive and seamless interface for searching and booking flights.
- To simplify the process of comparing flight options and prices.

2. Real-Time Flight Data:

- To enable users to access up-to-date flight information, including availability, schedules, and prices.
- To integrate with flight APIs for real-time data retrieval.

3. Efficient Booking Management:

- To facilitate users in booking flights, managing their reservations, and modifying bookings as needed.
- To implement user accounts for tracking past and upcoming flights.

4. Learning and Skill Development:

- To allow developers to apply their skills in a full-stack environment, utilizing JavaScript for both client-side and server-side development.
- To gain practical experience with the MERN stack and modern web development practices.

Goals of the Flight Booking Project

1. Functional Requirements:

- **User Registration and Authentication:** Implement secure user registration and login functionalities, enabling users to create accounts and log in to manage bookings.
- **Flight Search Functionality:** Allow users to search for flights based on various criteria such as departure and arrival locations, dates, and passenger details.
- **Booking Process:** Develop a straightforward booking process where users can select flights, enter passenger details, and complete the payment process.
- **Admin Dashboard:** Create an admin interface to manage flights, view bookings, and analyze user activity.

2. Non-Functional Requirements:

- **Performance:** Ensure the application loads quickly and handles a large number of concurrent users without significant delays.
- **Scalability:** Design the system architecture to accommodate growth in user numbers and data.
- **Security:** Implement security measures to protect user data and ensure secure transactions.

3. User Experience Goals:

- **Responsive Design:** Ensure the application is fully responsive and works well on various devices, including desktops, tablets, and mobile phones.
- **Intuitive Navigation:** Create a clear and intuitive navigation structure, making it easy for users to find what they need.

4. Integration Goals:

- **API Integration:** Integrate with third-party APIs for flight data (e.g., Skyscanner, Amadeus) to provide accurate and up-to-date information.
- **Payment Gateway:** Implement a secure payment gateway (e.g., Stripe or PayPal) for processing transactions.

5. Testing and Quality Assurance:

- **Comprehensive Testing:** Conduct thorough testing of all functionalities to ensure the application is bug-free and performs as expected.
- **User Feedback:** Gather user feedback through surveys or usability tests to identify areas for improvement.

6. Deployment and Maintenance:

- **Deployment:** Deploy the application on a cloud platform (e.g., AWS, Heroku) for public access.
- **Ongoing Maintenance:** Establish a plan for regular updates, bug fixes, and feature enhancements based on user feedback and changing market needs.

Key Features of a Flight Booking App

1. User Authentication and Profiles

- **User Registration and Login:** Allow users to create accounts and log in securely using email and password or social media accounts.
- **User Profiles:** Enable users to manage their profiles, view booking history, and save preferences (e.g., frequent flyer information, payment methods).

2. Flight Search and Filters

- **Search Functionality:** Users can search for flights based on departure and arrival locations, dates, number of passengers, and cabin class (e.g., economy, business).
- **Filtering Options:** Implement filters for sorting flights by price, duration, airline, and layover times to help users find the best options.

3. Flight Details and Comparison

- **Flight Information:** Display detailed information for each flight, including airline, departure and arrival times, duration, layovers, and prices.
- **Compare Flights:** Allow users to compare multiple flights side-by-side based on their criteria.

4. Booking Management

- **Booking Process:** Streamline the booking process with a clear step-by-step interface for selecting flights, entering passenger details, and confirming payment.
- **Manage Bookings:** Allow users to view, modify, or cancel their bookings through their profiles.

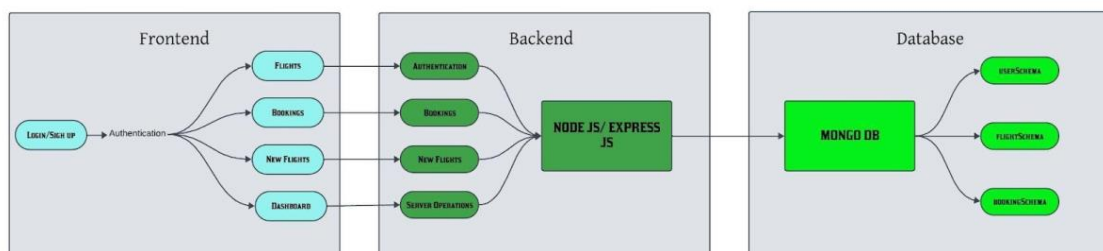
5. Payment Integration

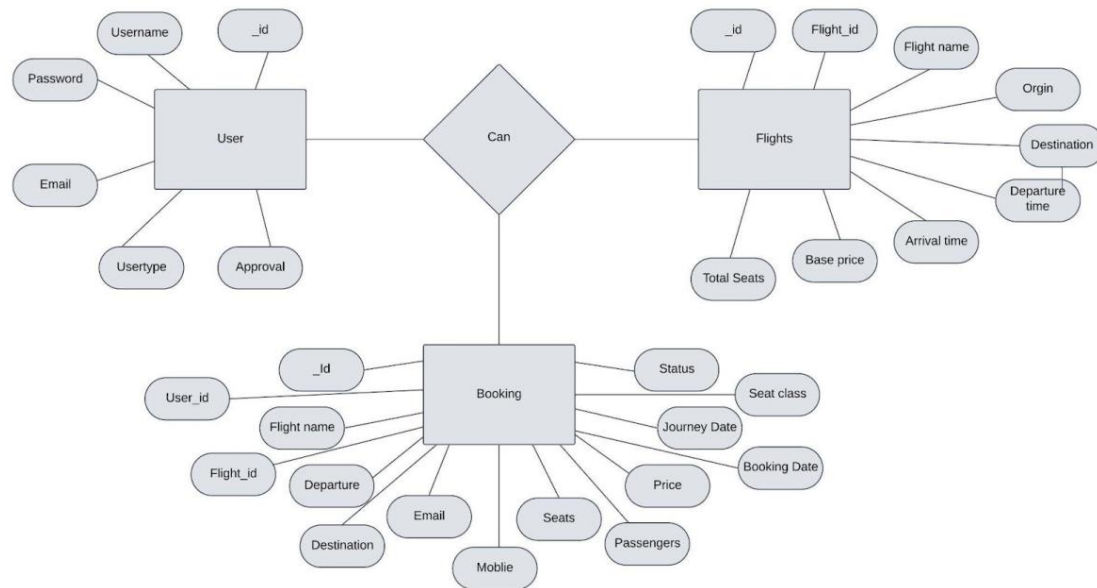
- **Secure Payment Processing:** Integrate with popular payment gateways (e.g., Stripe, PayPal) to handle transactions securely.

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.





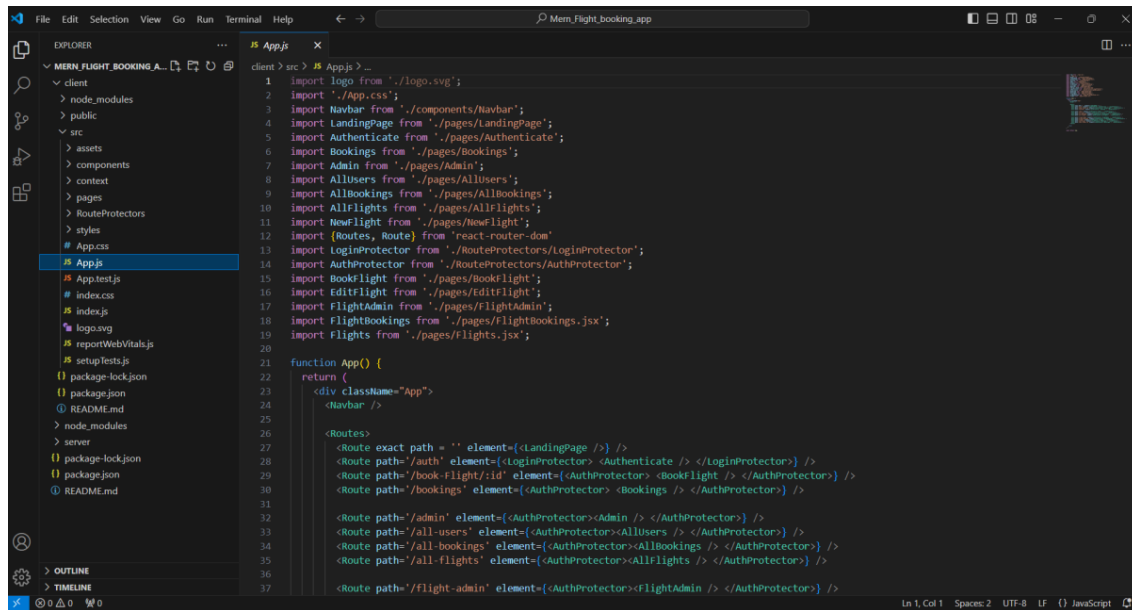
USER: Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.

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.

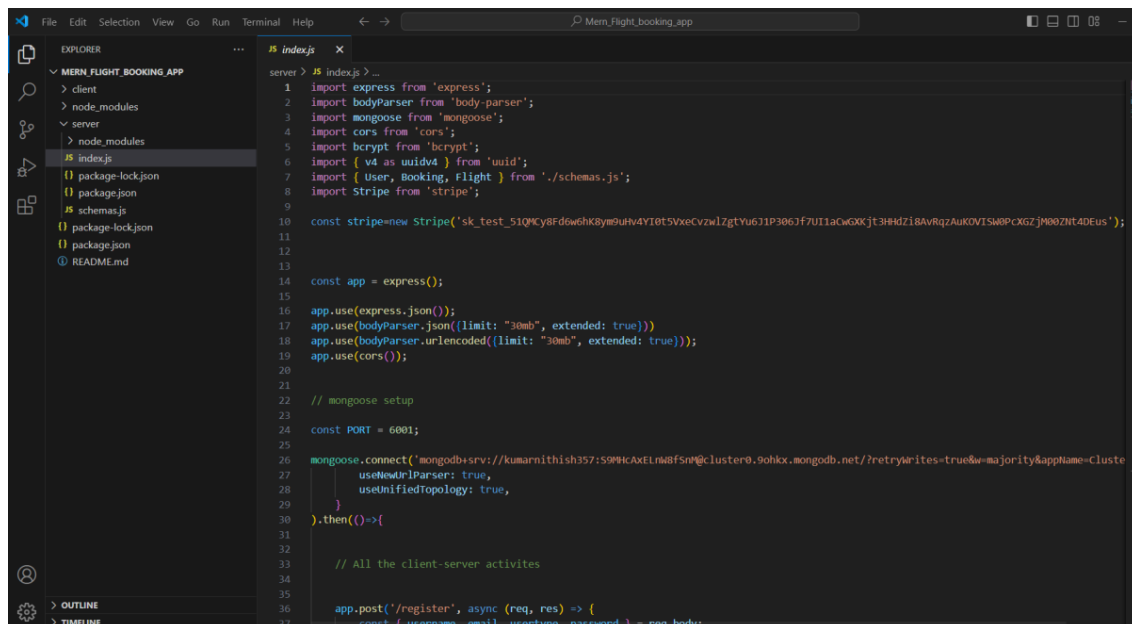
Frontend:



```
client > src > App.js >  
1 import logo from './logo.svg';  
2 import './App.css';  
3 import Navbar from './components/Navbar';  
4 import LandingPage from './pages/LandingPage';  
5 import Authenticate from './pages/Authenticate';  
6 import Bookings from './pages/Bookings';  
7 import Admin from './pages/Admin';  
8 import AllUsers from './pages/AllUsers';  
9 import AllBookings from './pages/AllBookings';  
10 import AllFlights from './pages/AllFlights';  
11 import NewFlight from './pages/NewFlight';  
12 import {Routes, Route} from 'react-router-dom';  
13 import LoginProtector from './RouteProtectors/LoginProtector';  
14 import AuthProtector from './RouteProtectors/AuthProtector';  
15 import BookFlight from './pages/BookFlight';  
16 import EditFlight from './pages/EditFlight';  
17 import FlightAdmin from './pages/FlightAdmin';  
18 import FlightBookings from './pages/flightBookings.jsx';  
19 import Flights from './pages/flights.jsx';  
20  
21 function App() {  
22   return (  
23     <div className="App">  
24       <Navbar />  
25  
26       <Routes>  
27         <Route exact path = '/' element={<LandingPage />} />  
28         <Route path="/auth" element={<LoginProtector> <Authenticate /> </LoginProtector>} />  
29         <Route path="/book-flight/:id" element={<AuthProtector> <BookFlight /> </AuthProtector>} />  
30         <Route path="/bookings" element={<AuthProtector> <Bookings /> </AuthProtector>} />  
31  
32         <Route path="/admin" element={<AuthProtector><Admin /> </AuthProtector>} />  
33         <Route path="/all-users" element={<AuthProtector><AllUsers /> </AuthProtector>} />  
34         <Route path="/all-bookings" element={<AuthProtector><AllBookings /> </AuthProtector>} />  
35         <Route path="/all-flights" element={<AuthProtector><AllFlights /> </AuthProtector>} />  
36  
37         <Route path="/flight-admin" element={<AuthProtector><FlightAdmin /> </AuthProtector>} />  
38       </Routes>  
39     </div>  
40   );  
41 }  
42  
43 export default App;
```

Designing the frontend architecture of a flight booking app using React in the MERN (MongoDB, Express.js, React.js, Node.js) stack involves creating a well-structured, maintainable, and scalable application. Here's a detailed breakdown of how you can structure the frontend architecture, covering key components, state management, routing, and styling.

Backend:



```
server > index.js >  
1 import express from 'express';  
2 import bodyParser from 'body-parser';  
3 import mongoose from 'mongoose';  
4 import cors from 'cors';  
5 import bcrypt from 'bcrypt';  
6 import { v4 as uuidv4 } from 'uuid';  
7 import { User, Booking, Flight } from './schemas.js';  
8 import Stripe from 'stripe';  
9  
10 const stripe = new Stripe('sk_test_51QCy8Fdw6hK8ym9u4V10t5VxcCvzwlZgtYU631P306Jf7U11aCwGKJt3H4Z18AVRqZAU0KV1SW8PCXGZJH00ZHT4DEus');  
11  
12  
13  
14 const app = express();  
15  
16 app.use(express.json());  
17 app.use(bodyParser.json({limit: "30mb", extended: true}));  
18 app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));  
19 app.use(cors());  
20  
21  
22 // mongoose setup  
23  
24 const PORT = 6001;  
25  
26 mongoose.connect('mongodb://kumarnithish357:59PHKAXELnW8fSn@cluster0.9ohkx.mongodb.net/?retryWrites=true&majorityAppname=Cluster0', {  
27   useNewUrlParser: true,  
28   useUnifiedTopology: true,  
29 }).then(() => {  
30  
31  
32  
33   // All the client-server activities  
34  
35  
36   app.post('/register', async (req, res) => {  
37     const { username, email, usertype, password } = req.body;
```

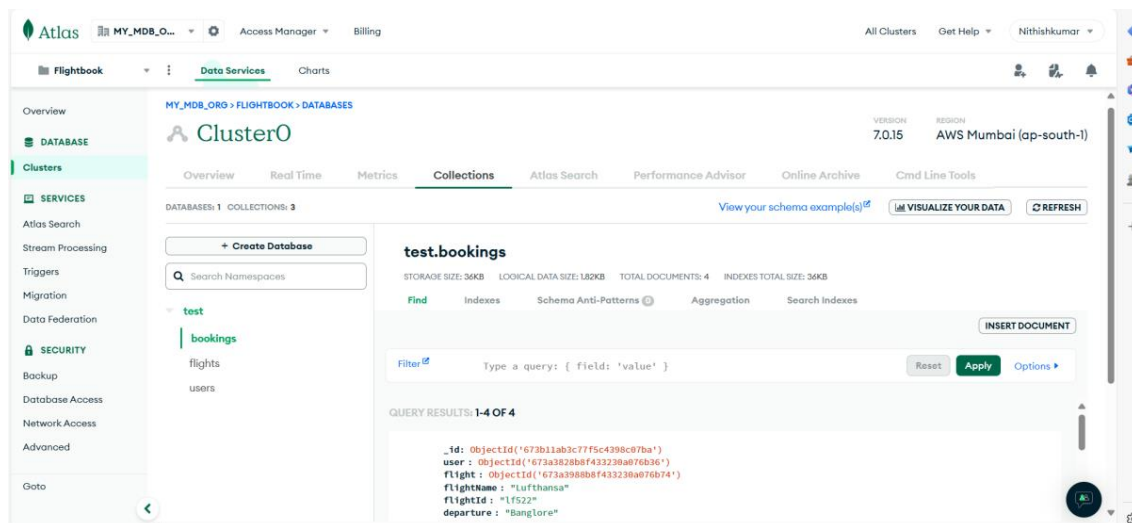
The backend architecture of a flight booking project built with the MERN stack (MongoDB, Express.js, React.js, Node.js) is crucial for ensuring efficient data management, secure transactions, and a seamless user experience. Below is a detailed breakdown of the backend architecture, including components, folder structure, API design, and key functionalities.

1. Overview of Backend Architecture

The backend of the flight booking project will consist of several key components:

- Node.js: The JavaScript runtime for executing server-side code.
- Express.js: A web application framework for Node.js to build RESTful APIs.
- MongoDB: A NoSQL database to store user data, flight details, and booking information.
- Mongoose: An Object Data Modeling (ODM) library for MongoDB to simplify data modeling and interactions.

Mongodb Database:



Interacting with MongoDB in a flight booking project involves a series of operations for managing data related to flights, users, bookings, and other related entities. Below is a detailed explanation of how you can set up these interactions, including CRUD (Create, Read, Update, Delete) operations, schema design, and examples of using Mongoose (an ODM for MongoDB) in a Node.js/Express environment.

Setup Instructions:

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://git-scm.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>

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

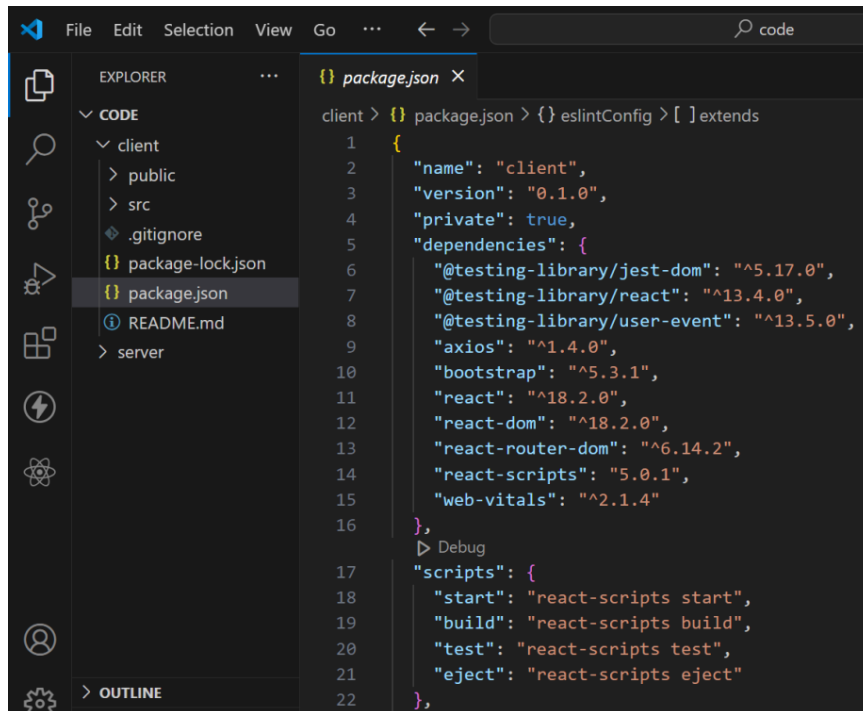
Installation of required tools:

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.

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project structure with a 'client' folder containing 'public', 'src', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The 'package.json' file is selected and its content is displayed in the main editor. The code is a JSON object for a client application, including dependencies like jest-dom, react, user-event, axios, bootstrap, react, react-dom, react-router-dom, react-scripts, and web-vitals, as well as scripts for start, build, test, and eject.

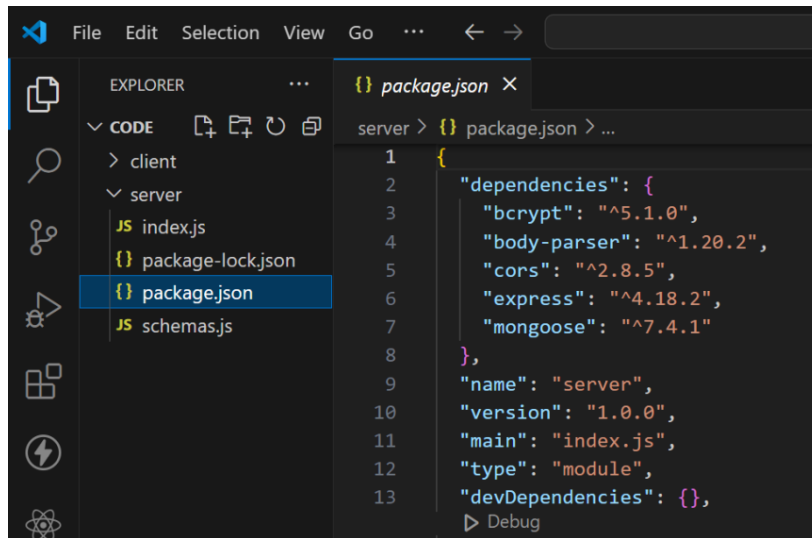
```
1 {
2   "name": "client",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^5.17.0",
7     "@testing-library/react": "^13.4.0",
8     "@testing-library/user-event": "^13.5.0",
9     "axios": "^1.4.0",
10    "bootstrap": "^5.3.1",
11    "react": "^18.2.0",
12    "react-dom": "^18.2.0",
13    "react-router-dom": "^6.14.2",
14    "react-scripts": "5.0.1",
15    "web-vitals": "^2.1.4"
16  },
17  "scripts": {
18    "start": "react-scripts start",
19    "build": "react-scripts build",
20    "test": "react-scripts test",
21    "eject": "react-scripts eject"
22  },
```

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

For backend, we use:

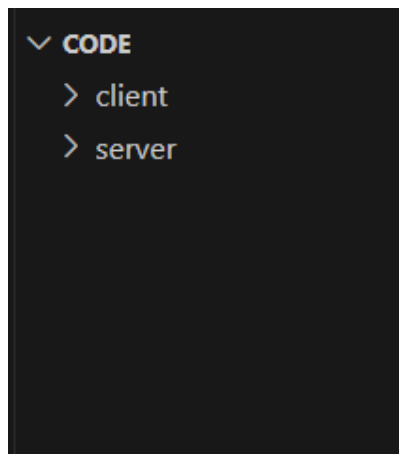
- bcrypt
- body-parser
- cors
- express
- mongoose

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



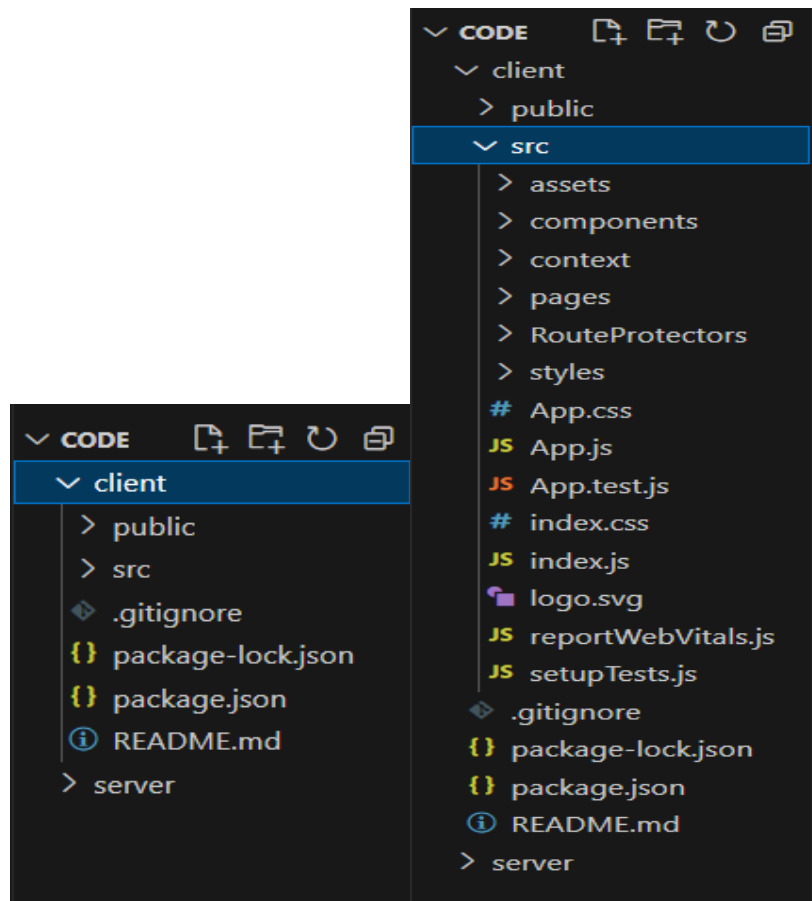
FOLDER STRUCTURE:

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



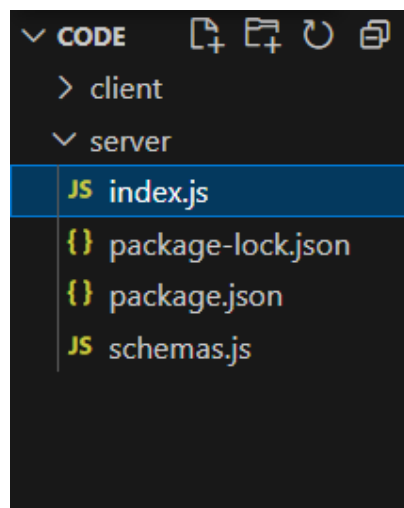
- **Client directory:**

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



- **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.



RUNNING THE APPLICATION:

SERVER:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\arunp\Downloads\Flight-Booking-App-MERN-main\Mern_Flight_booking_app\server> node .\index.js
(node:2128) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Running @ 6001
█
```

CLIENT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\arunp\Downloads\Flight-Booking-App-MERN-main\Mern_Flight_booking_app\client> npm start

Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.109:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
█
```

API DOCUMENTATION:

Here's a comprehensive list of all the endpoints exposed by the backend of a flight booking application built using the MERN stack (MongoDB, Express.js, React.js, Node.js). This documentation includes HTTP methods, paths, descriptions, request parameters, and example responses for each endpoint.

API Endpoints Documentation

1. User Endpoints

1.1 Create User

- **Endpoint:** POST /api/users
- **Description:** Create a new user account.
- **Request Body:**

Json

Copy code

```
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

- **Responses:**

- **201 Created:** User successfully created.

json

Copy code

```
{
  "_id": "user_id",
  "username": "string",
  "email": "string",
  "bookings": [],
  "createdAt": "timestamp",
  "updatedAt": "timestamp"
}
```

- **400 Bad Request:** Validation error or duplicate entry.

json

Copy code

```
{
  "message": "Error creating user",
  "error": "error details"
}
```

1.2 Get User by ID

- **Endpoint:** GET /api/users/:id

- **Description:** Retrieve a user's details by ID.

- **Parameters:**

- id (path): User ID.

- **Responses:**

- **200 OK:** User details retrieved successfully.

json

Copy code

```
{
  "_id": "user_id",
  "username": "string",
```

```
"email": "string",  
"bookings": ["booking_id"],  
"createdAt": "timestamp",  
"updatedAt": "timestamp"  
}
```

- **404 Not Found:** User not found.

json

Copy code

```
{  
  "message": "User not found"  
}
```

1.3 Update User

- **Endpoint:** PUT /api/users/:id
- **Description:** Update user details.
- **Parameters:**
 - id (path): User ID.
- **Request Body:**

json

Copy code

```
{  
  "username": "new_username",  
  "email": "new_email",  
  "password": "new_password"  
}
```

- **Responses:**
 - **200 OK:** User details updated successfully.

json

Copy code

```
{  
  "_id": "user_id",  
  "username": "string",  
}
```

```
"email": "string",
"bookings": [],
"createdAt": "timestamp",
"updatedAt": "timestamp"
}
```

- **400 Bad Request:** Validation error.

json

Copy code

```
{
  "message": "Error updating user",
  "error": "error details"
}
```

1.4 Delete User

- **Endpoint:** DELETE /api/users/:id
- **Description:** Delete a user account.
- **Parameters:**
 - id (path): User ID.
- **Responses:**
 - **204 No Content:** User successfully deleted.
 - **404 Not Found:** User not found.

json

Copy code

```
{
  "message": "User not found"
}
```

2. Flight Endpoints

2.1 Create Flight

- **Endpoint:** POST /api/flights
- **Description:** Create a new flight entry.
- **Request Body:**

json

Copy code

```
{
  "airline": "string",
  "flightNumber": "string",
  "departure": "string",
  "arrival": "string",
  "departureTime": "date",
  "arrivalTime": "date",
  "price": number,
  "seatsAvailable": number
}
```

- **Responses:**

- **201 Created:** Flight successfully created.

json

Copy code

```
{
  "_id": "flight_id",
  "airline": "string",
  "flightNumber": "string",
  "departure": "string",
  "arrival": "string",
  "departureTime": "date",
  "arrivalTime": "date",
  "price": number,
  "seatsAvailable": number,
  "createdAt": "timestamp",
  "updatedAt": "timestamp"
}
```

- **400 Bad Request:** Validation error.

json

Copy code

```
{  
  "message": "Error creating flight",  
  "error": "error details"  
}
```

2.2 Get All Flights

- **Endpoint:** GET /api/flights
- **Description:** Retrieve a list of all available flights.
- **Responses:**
 - **200 OK:** List of flights.

json

Copy code

```
[  
  {  
    "_id": "flight_id",  
    "airline": "string",  
    "flightNumber": "string",  
    "departure": "string",  
    "arrival": "string",  
    "departureTime": "date",  
    "arrivalTime": "date",  
    "price": number,  
    "seatsAvailable": number  
  }  
]
```

2.3 Get Flight by ID

- **Endpoint:** GET /api/flights/:id
- **Description:** Retrieve a flight's details by ID.
- **Parameters:**
 - id (path): Flight ID.
- **Responses:**

- **200 OK:** Flight details retrieved successfully.

json

Copy code

```
{
  "_id": "flight_id",
  "airline": "string",
  "flightNumber": "string",
  "departure": "string",
  "arrival": "string",
  "departureTime": "date",
  "arrivalTime": "date",
  "price": number,
  "seatsAvailable": number
}
```

- **404 Not Found:** Flight not found.

json

Copy code

```
{
  "message": "Flight not found"
}
```

2.4 Update Flight

- **Endpoint:** PUT /api/flights/:id
- **Description:** Update flight information.
- **Parameters:**
 - id (path): Flight ID.
- **Request Body:**

json

Copy code

```
{
  "airline": "new_airline",
  "flightNumber": "new_flightNumber",
}
```

```
"departure": "new_departure",
"arrival": "new_arrival",
"departureTime": "new_date",
"arrivalTime": "new_date",
"price": number,
"seatsAvailable": number
}
```

- **Responses:**

- **200 OK:** Flight details updated successfully.

json

Copy code

```
{
  "_id": "flight_id",
  "airline": "string",
  "flightNumber": "string",
  "departure": "string",
  "arrival": "string",
  "departureTime": "date",
  "arrivalTime": "date",
  "price": number,
  "seatsAvailable": number
}
```

- **404 Not Found:** Flight not found.

json

Copy code

```
{
  "message": "Flight not found"
}
```

2.5 Delete Flight

- **Endpoint:** DELETE /api/flights/:id
- **Description:** Delete a flight entry.

- **Parameters:**
 - id (path): Flight ID.
- **Responses:**
 - **204 No Content:** Flight successfully deleted.
 - **404 Not Found:** Flight not found.

json

Copy code

```
{
  "message": "Flight not found"
}
```

3. Booking Endpoints

3.1 Create Booking

- **Endpoint:** POST /api/bookings
- **Description:** Create a new booking for a flight.
- **Request Body:**

json

Copy code

```
{
  "userId": "user_id",
  "flightId": "flight_id",
  "passengerDetails": [
    {
      "name": "string",
      "age": number,
      "gender": "string"
    }
  ],
  "totalPrice": number
}
```

- **Responses:**

- **201 Created:** Booking successfully created.

json

Copy code

```
{
  "_id": "booking_id",
  "user": "user_id",
  "flight": "flight_id",
  "passengerDetails": [
    {
      "name": "string",
      "age": number,
      "gender": "string"
    }
  ],
  "totalPrice": number,
  "status": "confirmed",
  "createdAt": "timestamp",
  "updatedAt": "timestamp"
}
```

- **400 Bad Request:** Validation error or flight/user not found.

json

Copy code

```
{
  "message": "Error creating booking",
  "error": "error details"
}
```

AUTHNTICATION:

1. **Registration Endpoint:** A POST endpoint (e.g., /users/register) is created for user registration.
2. **Request Handling:** When a user registers, their credentials (username, email, and password) are sent to this endpoint.

Integrating Stripe for payment processing in a flight booking project using the MERN stack involves setting up the Stripe API, creating a payment flow, and securely handling sensitive information such as API keys. Below are the detailed steps to implement Stripe payment processing, including how to send keys securely and manage transactions.

Steps to Integrate Stripe Payment in a Flight Booking Project

1. Create a Stripe Account

1. Sign up for a Stripe account at [Stripe](https://stripe.com).
2. Obtain your **API keys** from the Stripe Dashboard:
 - **Publishable Key**: Used in the frontend (public).
 - **Secret Key**: Used in the backend (private).

2. Install Stripe SDKs

In your project, install the Stripe SDK for both the backend and frontend.

- **Backend** (Node.js):

bash

Copy code

```
npm install stripe
```

- **Frontend** (React):

bash

Copy code

```
npm install @stripe/stripe-js
```

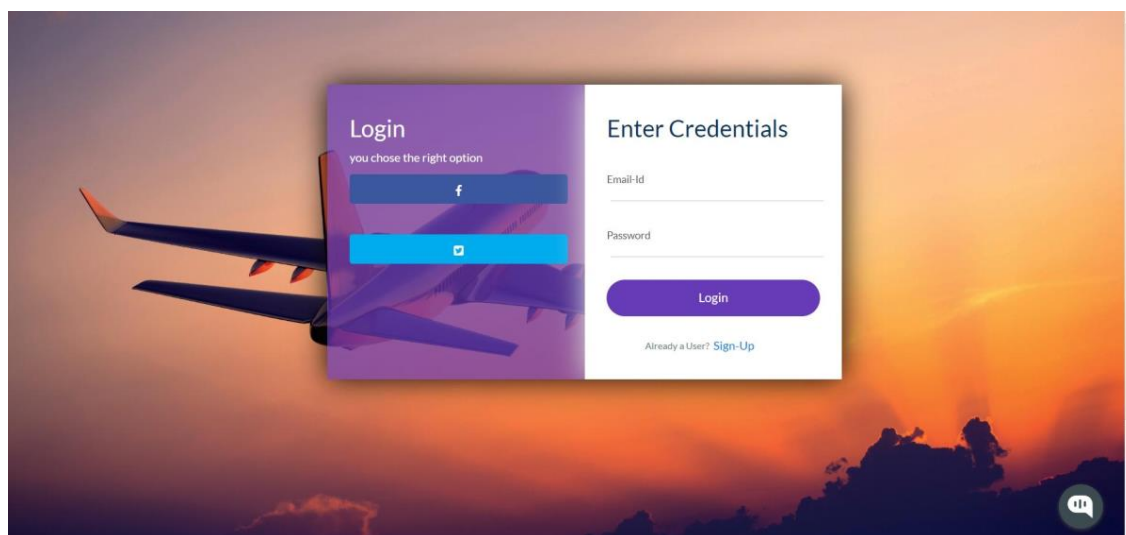
- Setting up a Stripe account and obtaining API keys.
- Creating a backend endpoint to handle payment intent creation securely.
- Using the Stripe.js library in the frontend to handle payments while keeping sensitive keys secure.
- Implementing a flow to confirm payments and handle bookings accordingly.
- Ensuring security through the use of environment variables and best practices for handling sensitive data.

USER INTERFACE:

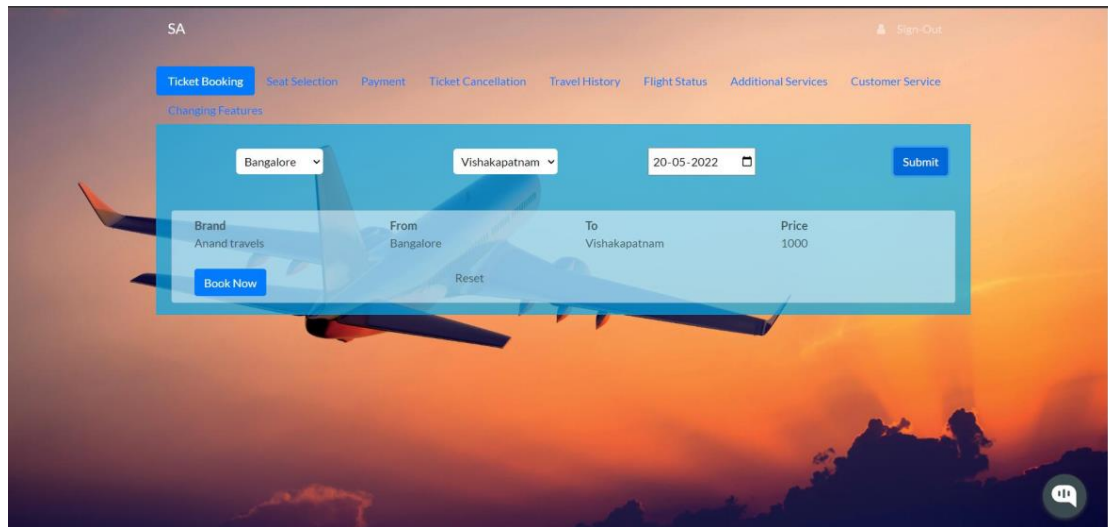
- **Landing page UI**



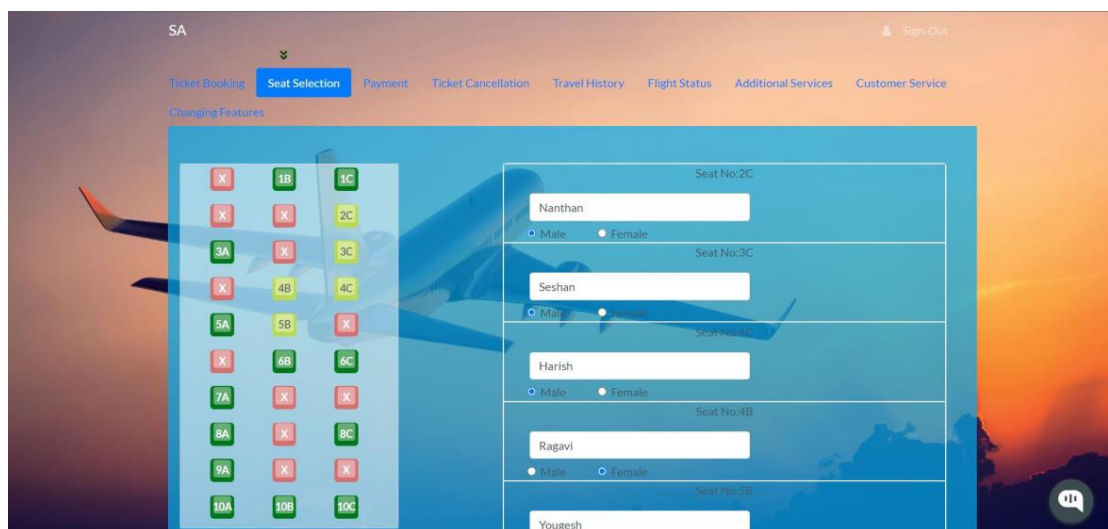
- **Authentication(Signing In Page)**



- **User bookings(Flight Selection Page)**



- **Seat Selection Page:**



- **All users**

SA Sign Out

[Ticket Booking](#)
[Seat Selection](#)
[Payment](#)
[Ticket Cancellation](#)
[Travel History](#)
[Flight Status](#)
[Additional Services](#)
[Customer Service](#)

Changing Features

X	1B	1C
X	X	2C
3A	X	3C
X	4B	4C
5A	5B	X
X	6B	6C
7A	X	X
8A	X	9C
9A	X	X
10A	10B	10C

Seat No:2C

Nanthan

☒ Male ☐ Female

Seat No:3C

Seshan

☒ Male ☐ Female

Seat No:4C

Harish

☐ Male ☒ Female

Seat No:4B

Ragavi

☐ Male ☒ Female

Seat No:5B


Yougesh

- **Payment & Confirmation Page:**

[Ticket Booking](#)
[Seat Selection](#)
[Payment](#)
[Ticket Cancellation](#)
[Travel History](#)
[Flight Status](#)
[Additional Services](#)
[Customer Service](#)

Changing Features

ENTER CREDIT CARD DETAILS



**** * * * *

YOUR NAME HERE valid thru **/**

Card Number

Name

Valid Thru

CVC

PAY

Swadeshi Airlines

BOOKING DETAILS

Username

Date 2022-05-20

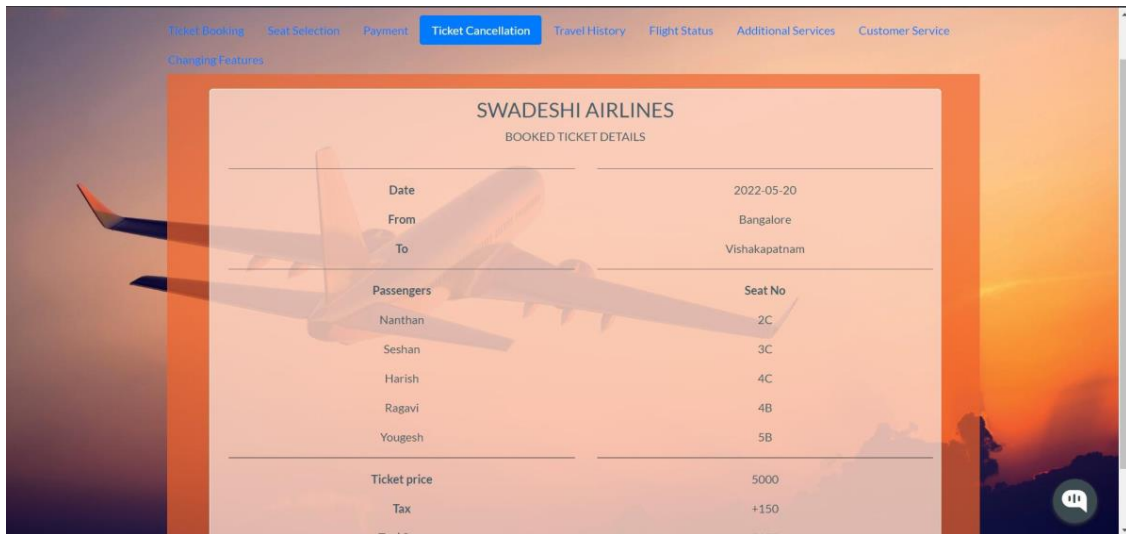
From Bangalore

To Vishakapatnam

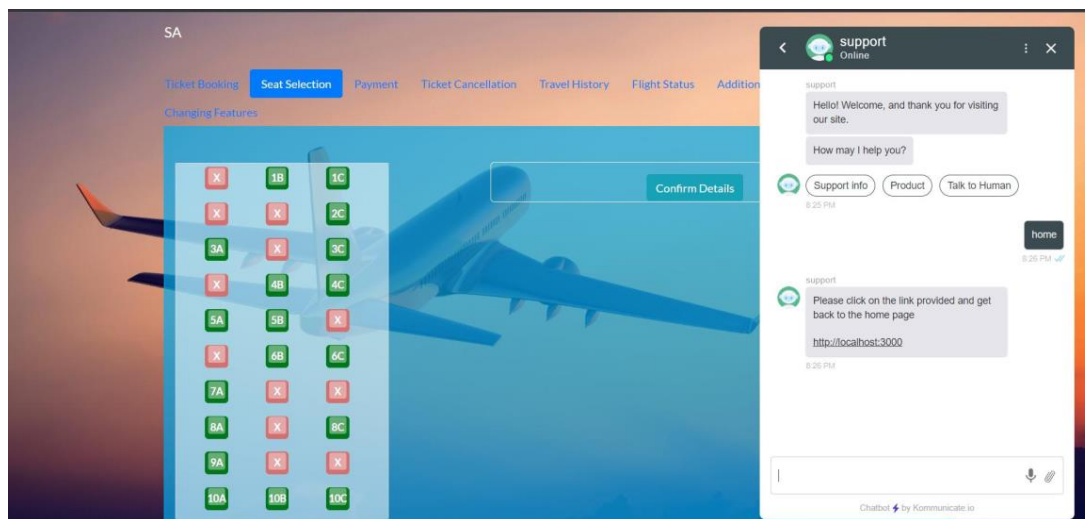
Passengers	Seat No
Nanthan	2C
Seshan	3C
Harish	4C
Ragavi	4B
Yougesh	5B

Ticket price 5000

- **Ticket Cancellation Page**



- Integrated AI Chatbot :



TESTING STRATEGIES:

a. Unit Testing

- **Definition:** Unit testing involves testing individual components or functions in isolation to ensure they work as intended. This helps catch bugs early in the development process.
- **Purpose:** Verifies the correctness of individual functions or methods, making debugging easier when issues arise.

- **Example:** Testing a function that calculates the total price of flight bookings or validating user input.

b. Integration Testing

- **Definition:** Integration testing checks how different modules or components interact with each other. This is crucial in a MERN stack where the frontend and backend components must work seamlessly together.
- **Purpose:** Ensures that the integrated parts of the application (like APIs, database connections, and UI components) work as expected when combined.
- **Example:** Testing API endpoints that interact with the database or checking if the React components correctly display data retrieved from the backend.

c. End-to-End (E2E) Testing

- **Definition:** E2E testing simulates real user scenarios to test the application from start to finish. This involves testing the entire application flow, including UI, API, and database interactions.
- **Purpose:** Validates the user experience and ensures that all components of the application work together correctly.
- **Example:** Testing the entire flight booking process, from searching for flights, selecting a flight, entering payment information, to confirming a booking.

d. Performance Testing

- **Definition:** Performance testing assesses how the application behaves under load. This includes measuring response times, throughput, and resource usage.
- **Purpose:** Ensures that the application can handle the expected load and performs well under stress.
- **Example:** Simulating multiple users booking flights simultaneously to test how the system performs under high traffic.

DEMO VIDEO LINK:

https://drive.google.com/file/d/1A_1-MyuiLxtbrD4Dcjc3xIQCRDEkisVV/view?usp=drivesdk

KNOWN ISSUES:

1. User Interface (UI) Issues

- **Responsiveness:** Certain pages may not be fully responsive on all screen sizes, particularly on older mobile devices or small tablets. Users may experience layout issues or difficulty navigating.
- **Loading States:** There might be inconsistent loading indicators when fetching flight data or processing payments, which could confuse users regarding whether their action was successful.

2. Functionality Issues

- **Search Filtering:** The flight search functionality may not always provide accurate results when filters (such as date, price range, and number of stops) are applied. This can result in flights not being displayed as expected.
- **Payment Processing:** Occasionally, users may experience delays or errors when processing payments due to network issues or Stripe API downtime. Users are advised to check their payment status if they encounter errors.
- **Booking Confirmation Emails:** Some users may not receive booking confirmation emails due to spam filters or issues with the email service provider. Users should check their spam/junk folder if they do not receive confirmation.

3. Performance Issues

- **Slow Loading Times:** Under heavy load, especially during peak travel seasons, the application may exhibit slower response times. This can impact the overall user experience when searching for flights or completing bookings.
- **Database Queries:** Some database queries may not be optimized, leading to longer load times when retrieving flight data. Developers are encouraged to review and optimize queries to improve performance.

4. Authentication and Authorization Issues

- **Token Expiry:** Users may experience unexpected logouts if their session token expires while they are using the application. Users are encouraged to log in again if they notice being redirected to the login page unexpectedly.
- **Role-Based Access Control:** There may be instances where users with insufficient permissions can access certain admin functionalities due to incomplete role checks in the authorization middleware. This can lead to unauthorized actions being performed.

5. Cross-Browser Compatibility

- **Browser-Specific Bugs:** The application may work optimally in modern browsers like Chrome and Firefox but may exhibit bugs in older versions of Internet Explorer or Safari. Users are encouraged to use the latest version of their preferred browser for the best experience.

6. Security Vulnerabilities

- **Input Validation:** Some forms may lack comprehensive validation, potentially allowing invalid data to be submitted. Users should be cautious when entering personal information.
- **Sensitive Data Exposure:** Although all sensitive data should be handled securely, there could be unintentional exposure of sensitive information in error messages. Developers should ensure that error messages do not leak sensitive information.

7. Documentation and Support

- **Inadequate Documentation:** The user and developer documentation may not cover all aspects of the application, particularly in areas like API usage or troubleshooting common issues. This can lead to confusion for new users or developers.

8. Miscellaneous Issues

- **Flight Data Accuracy:** The flight information displayed is dependent on third-party APIs. Any downtime or inaccuracies from those APIs could lead to incorrect flight data being shown to users.
- **User Session Management:** Users may experience issues with session persistence, where their session does not maintain state across different devices or browser tabs.

Recommendations for Users

- **Use Modern Browsers:** To mitigate cross-browser compatibility issues, users are encouraged to use the latest versions of popular web browsers.

Recommendations for Developers

- **Monitor and Optimize:** Regularly monitor application performance and optimize database queries and API calls to enhance user experience.
- **Enhance Documentation:** Improve the documentation based on user feedback and ensure that all features and known issues are well documented.
- **Implement Comprehensive Testing:** Continue to conduct thorough testing, including regression testing, to catch and resolve any new or existing issues before deploying updates.

FUTURE ENHANCEMENTS:

1. Enhanced Search and Filtering Options

- **Multi-City Search:** Allow users to search for flights across multiple cities in one trip, facilitating complex itineraries.
- **Flexible Dates Search:** Implement a feature that allows users to search for flights within a date range, showing the cheapest options available during that period.

- **Advanced Filtering:** Add additional filters such as airline preference, flight duration, layover times, and specific amenities (Wi-Fi, meals).

2. User Account Management

- **Profile Management:** Enable users to create profiles where they can manage personal information, preferences, and frequent flyer details.
- **Booking History:** Implement a feature that allows users to view their past bookings, making it easier to rebook flights or check travel history.
- **Wishlists:** Allow users to save preferred flights or destinations for future reference.

3. Real-Time Notifications

- **Flight Status Updates:** Provide users with real-time notifications about flight status changes (delays, cancellations) via email or push notifications.
- **Price Alerts:** Enable users to set price alerts for specific routes, notifying them when prices drop or rise.

4. Improved Payment Options

- **Multiple Payment Methods:** Integrate additional payment gateways (e.g., PayPal, Google Pay, Apple Pay) to provide users with more payment options.
- **Installment Payments:** Offer users the option to pay for their flight in installments, making travel more accessible.

5. Mobile App Development

- **Native Mobile Application:** Develop a mobile app for iOS and Android to provide a seamless user experience on mobile devices, allowing for offline access to certain features.
- **Push Notifications:** Integrate push notifications for reminders, offers, and updates related to bookings and promotions.

6. Enhanced Security Features

- **Two-Factor Authentication (2FA):** Implement 2FA for user accounts to enhance security during login.
- **Fraud Detection:** Introduce machine learning algorithms to detect and prevent fraudulent transactions.

7. User Reviews and Ratings

- **Review System:** Allow users to leave reviews and ratings for flights and airlines, providing valuable feedback for future travelers.
- **Feedback on Experience:** Enable users to provide feedback on their booking experience and report issues directly through the platform.

8. Integration with Travel Services

- **Hotel and Car Rental Bookings:** Expand the platform to allow users to book hotels and car rentals alongside their flights for a more comprehensive travel experience.
- **Travel Insurance Options:** Offer users the option to purchase travel insurance during the booking process for added security.

9. Personalization and Recommendations

- **Machine Learning Recommendations:** Use machine learning algorithms to suggest flights and destinations based on user preferences and past behavior.
- **Dynamic Offers:** Provide personalized offers and discounts based on user profiles, search history, and preferences.

10. Analytics and Reporting for Admins

- **Admin Dashboard Enhancements:** Improve the admin dashboard with advanced analytics tools to track user behavior, booking trends, and financial metrics.
- **Reporting Tools:** Implement reporting features that allow admins to generate reports on sales, user engagement, and booking patterns.

11. Accessibility Improvements

- **Accessibility Features:** Ensure that the application is fully accessible to users with disabilities, following WCAG (Web Content Accessibility Guidelines) standards.
- **Language Support:** Add multi-language support to cater to a diverse user base, making the application more accessible globally.

12. Community Features

- **Travel Community Forum:** Create a space for users to share travel tips, experiences, and advice with each other.
- **Social Sharing:** Enable users to share their travel plans and experiences on social media platforms.

Project implementation:

- ◆ After creating the flight ticket booking website run the code for testing bugs
- ◆ Some software tools are used to avoid issues in coding and its is used to build the website

Conclusion :

- ✓ The primary aim of the study is to design an airline booking system for the online Reservation of ticket across different cities
- ✓ Online reservation system will help the users to book their ticket from the comfort Of their home thereby comparing their different website for ensuring the best Possible price could be available