FlightFinder:Navigating Your Air Travel Options

TEAM ID: LTVIP2025TMID52738

Team Members: Karimisetty Srikala,B Lalitha, Katta Mounika,Mabunni,Gundam Sandya Rani

Santhiram Engineering College(Autonomous),

Nandyal,Andhra Pradesh, India

INTRODUCTION :

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

KEY FEATURES

✈️ User Authentication & Onboarding

- Login/register screens for user accounts (via email or social login)

🔍 Search Flights

- Input fields for origin, destination, and flight dates (one-way or round-trip)
- Calendar UI to select dates and show availability across multiple days

📋 Flight Results & Route Display

- List of available flight options after search
- Map view illustrating the flight route (with polylines)

💺 Seat Selection Interface

- Interactive seat map interface for picking seats on the flight

📲 Smooth Navigation and Animations

- UI animations for transitioning between screens (search → results → seat booking)
- Bottom-tab or drawer navigation for app sections (search, bookings, profile)
  awesome.ecosyste.ms

🧭 State Management

- Efficient state handling using Flutter solutions (e.g., GetX) or React Native approaches

🌐 API Integration

- REST API calls to fetch flights, schedule, booking data
- Integration with mapping APIs (e.g. Google Maps) for route visualization

DESCRIPTION :

This Flight Booking APP is the ultimate digital platform designed to revolutionize the way you book flight tickets. With this app your flight travel experience will be elevated to new heights of convenience and efficiency. Our user-friendly web app empowers travelers to effortlessly discover, explore, and reserve flight tickets based on their unique

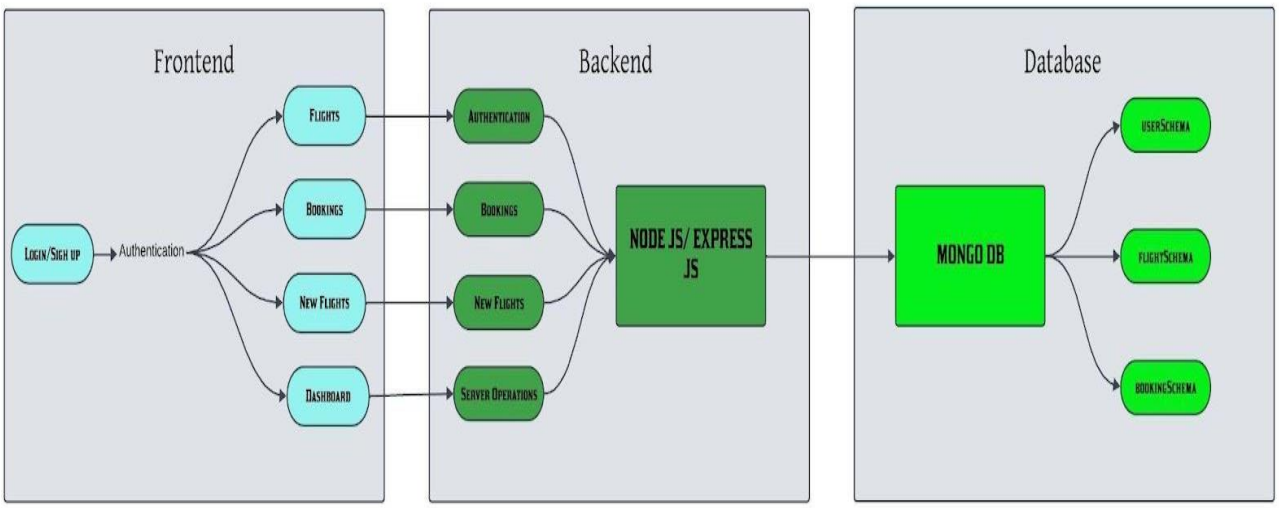
preferences. Whether you're a frequent commuter or an occasional traveler, finding the perfect flight journey has never been easier.

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

SCENARIO-BASED CASE STUDY :

- John, a frequent traveler and business professional, needs to book a flight for an upcoming conference in Paris. He prefers using a flight booking app for its convenience and features.
- John opens the flight booking app on his smartphone and enters his travel details for Departure as New York City, Destination as Paris, Date of Departure on April 10th and return on April 15th and Class as Business class, Number of passengers as 1
- The app quickly retrieves available flight options based on John's preferences. He sees a range of choices from different airlines, including direct flights and those with layovers. The results show details such as price, airline, duration, and departure times.
- Using the app's filters, John narrows down the options to show only direct flights with convenient departure times. He also selects his preferred airline based on past experiences and loyalty programs.
- After choosing a flight, John proceeds to select his seat in the business class cabin. The app provides a seat map with available seats highlighted, allowing John to pick a window seat with extra legroom.
- John securely enters his payment information using the app's integrated payment gateway. The app processes the payment and generates a booking confirmation with his e-ticket and itinerary details.
- This scenario demonstrates how a flight booking app streamlines the entire travel process for users like John, offering convenience, customization, and real-time assistance throughout their journey

TECHNICAL ARCHITECTURE :



In this architecture diagram:

• The frontend is represented by the "Frontend" section, including user interfacecomponents such as User Authentication, Flight Search, and Booking.

• The backend is represented by the "Backend" section, consisting of API endpointsfor Users, Flights, Admin and Bookings. It also includes Admin Authentication andan Admin Dashboard.

• The Database section represents the database that stores collections for Users,Flights, and Flight Bookings.

FRONTEND TECHNOLOGIES :

- **React.js –** component-based, large ecosystem, ideal for dynamic flight search and results UI
- **Angular –** full-fledged TypeScript framework, structured (incl. DI, services) for large apps
- **Next.js (React) or Nuxt.js (Vue) –** for SEO, SSR, fast-first-load, helpful in travel search scenarios
- **HTML, CSS, and JavaScript –** the essential foundation for any responsive and interactive UI

BACKEND FRAMEWORK :

- **Express.js** – Minimalist, flexible REST API engine with huge middleware ecosystem. Great if you want full control and simplicity

DATABASE AND AUTHENTICATION:

- **MongoDB**: A NoSQL database used for flexible and scalable storage of user data, flight metadata profiles, and appointment records. It supports fast querying and large data volumes.
- **JWT (JSON Web Tokens):** Used for secure, stateless authentication, allowing users to remain logged in without requiring session storage on the server.
- **Bcrypt**: A library for hashing passwords, ensuring that sensitive data is securely stored in the database.

SCALABILITY AND PERFORMANCE :

- **MongoDB:** Scales horizontally, supporting increased data storage and high user traffic as the platform grows.

Load Balancing: Ensures traffic is evenly distributed across servers to optimise performance, especially during high traffic periods.

- **Caching:** Reduces database load by storing frequently requested data temporarily, speeding up response times and improving user experience.

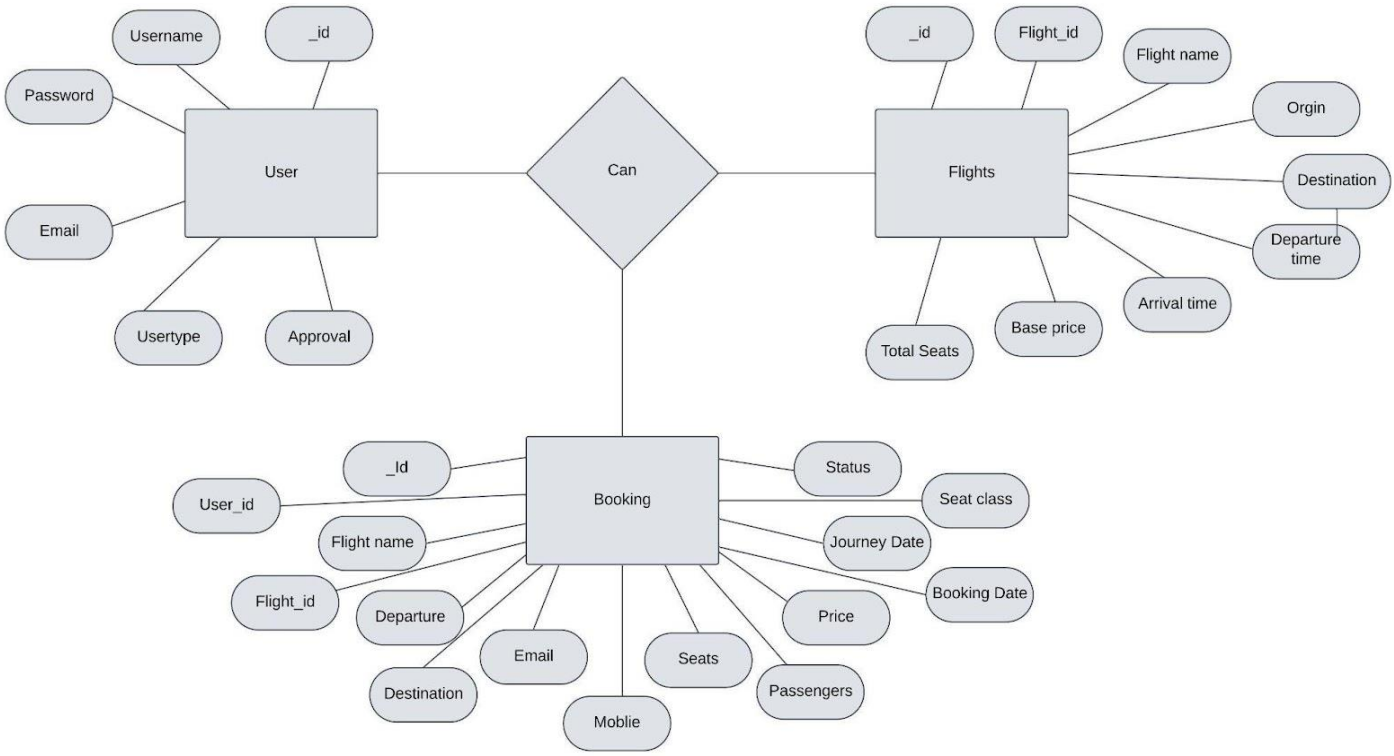
SECURITY FEATURES :

- **HTTPS**: The platform uses SSL/TLS encryption to secure data transmission between the client and server.
- **Data Encryption:** Sensitive user information, such as medical records, is encrypted both in transit and at rest, ensuring privacy and compliance with data protection regulations.
-

NOTIFICATIONS AND REMINDERS :

- **Email/SMS Integration:** Notifications for appointment confirmations, reminders, cancellations, and updates are sent to users via email or SMS, ensuring timely communication.

ER DIAGRAM :



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

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

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

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

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

PRE REQUISITES :

NODE.JS AND NPM:

- Node.js is a JavaScript runtime that allows you to run JavaScript code on the server-side. It provides a scalable platform for network applications.
- npm (Node Package Manager) is required to install libraries and manage dependencies.
- Download Node.js: Node.js Download
- Installation instructions: Installation Guide
- Run npm init to set up the project and create a package.json file.

EXPRESS.JS:

- Express.js is a web application framework for Node.js that helps you build APIs and web applications with features like routing and middleware.

- Install Express.js to manage backend routing and API endpoints.
- Install Express:
- Run npm install express

MONGODB:

- MongoDB is a NoSQL database that stores data in a JSON-like format, making it suitable for storing data like user profiles, doctor details, and appointments.
- Set up a MongoDB database for your application to store data.
- Download MongoDB: MongoDB Download
- Installation instructions: MongoDB Installation Guide

MOMENT.JS:

- Moment.js is a JavaScript package for handling date and time operations, allowing easy manipulation and formatting.
- Install Moment.js for managing date-related tasks, such as appointment scheduling.
- Moment.js Website: Moment.js Documentation

REACT.JS:

- React.js is a popular JavaScript library for building interactive and reusable user interfaces. It enables the development of dynamic web applications.
- Install React.js to build the frontend for your application.
- React.js Documentation: Create a New React App

ANTD (ANT DESIGN):

- Ant Design is a UI library for React.js, providing a set of reusable components to create user-friendly and visually appealing interfaces.
- Install Ant Design for UI components such as forms, tables, and modals.
- Ant Design Documentation: Ant Design React

HTML, CSS, AND JAVASCRIPT:

- Basic knowledge of HTML, CSS, and JavaScript is essential to structure, style, and add interactivity to the user interface.

DATABASE CONNECTIVITY (MONGOOSE):

- Use Mongoose, an Object-Document Mapping (ODM) library, to connect your Node.js backend to MongoDB for managing CRUD operations.
- Learn Database Connectivity: Node.js + Mongoose + MongoDB

FRONT-END FRAMEWORKS AND LIBRARIES:

- React.js will handle the client-side interface for managing doctor bookings, viewing appointment statuses, and providing an admin dashboard.
- You may use Material UI and Bootstrap to enhance the look and feel of the application.

SETUP AND INSTALLATION INSTRUCTIONS :

CLONE THE PROJECT REPOSITORY:

- Download the project files from GitHub or clone the repository using Git.

INSTALL DEPENDENCIES:

- Navigate to the frontend and backend directories and install all required dependencies for both parts of the application.
- Frontend:
- Navigate to the frontend directory and run npm install.
- Backend:
- Navigate to the backend directory and run npm install.

START THE DEVELOPMENT SERVER:

- After installing the dependencies, start the development server for both frontend and backend.
- Frontend will run on http://localhost:3000.
- Backend will run on http://localhost:8001 or the specified port.

ACCESS THE APPLICATION:

- After running the servers, access the Doctor Appointment Webpage in your browser at http://localhost:3000 for the frontend interface and http://localhost:8001 for backend API services.

PROJECT STRUCTURE :

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

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:

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.

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

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

Clone the repository:

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

Git clone: https://github.com/Srikala-25/Fight-Booking-App#
Install Dependencies:

- Navigate into the cloned repository directory:

cd Flight-Booking-App-MERN

- Install the required dependencies by running the following command:

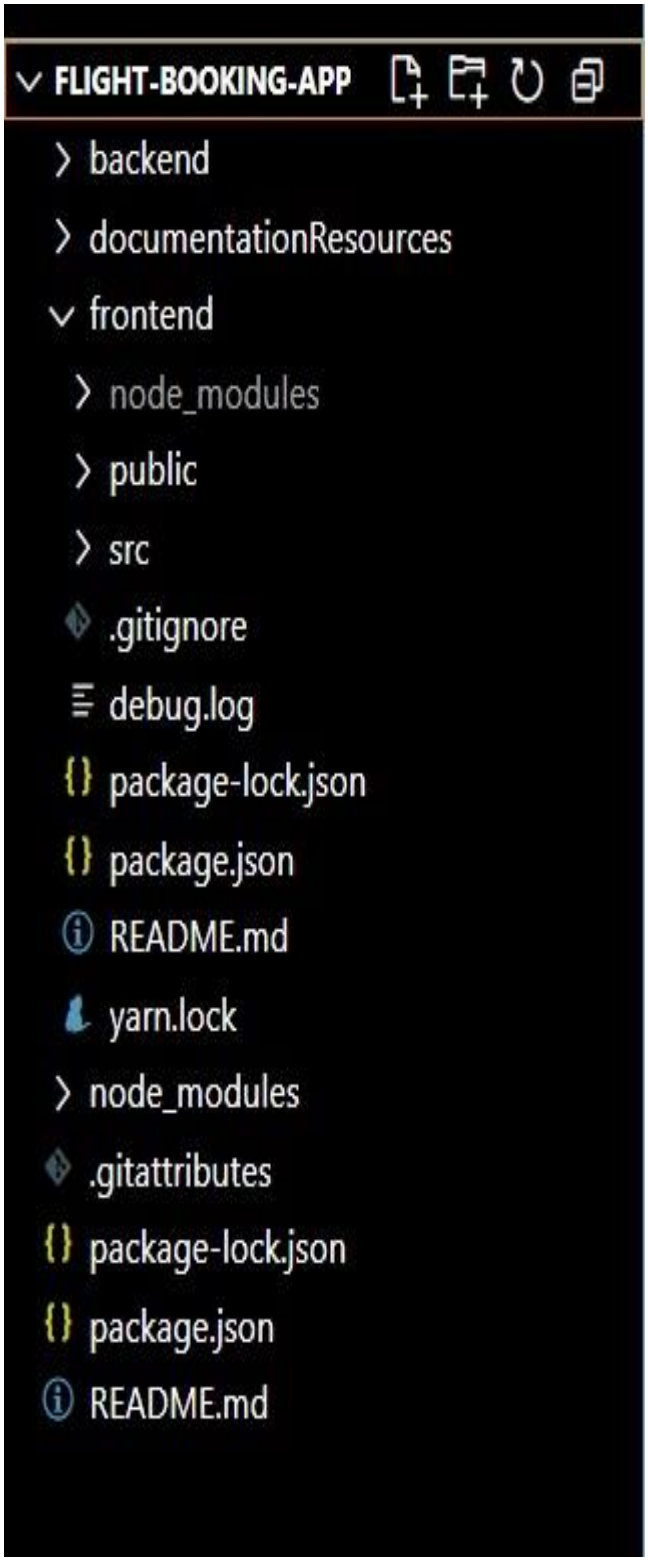
npm install
Start the Development Server:

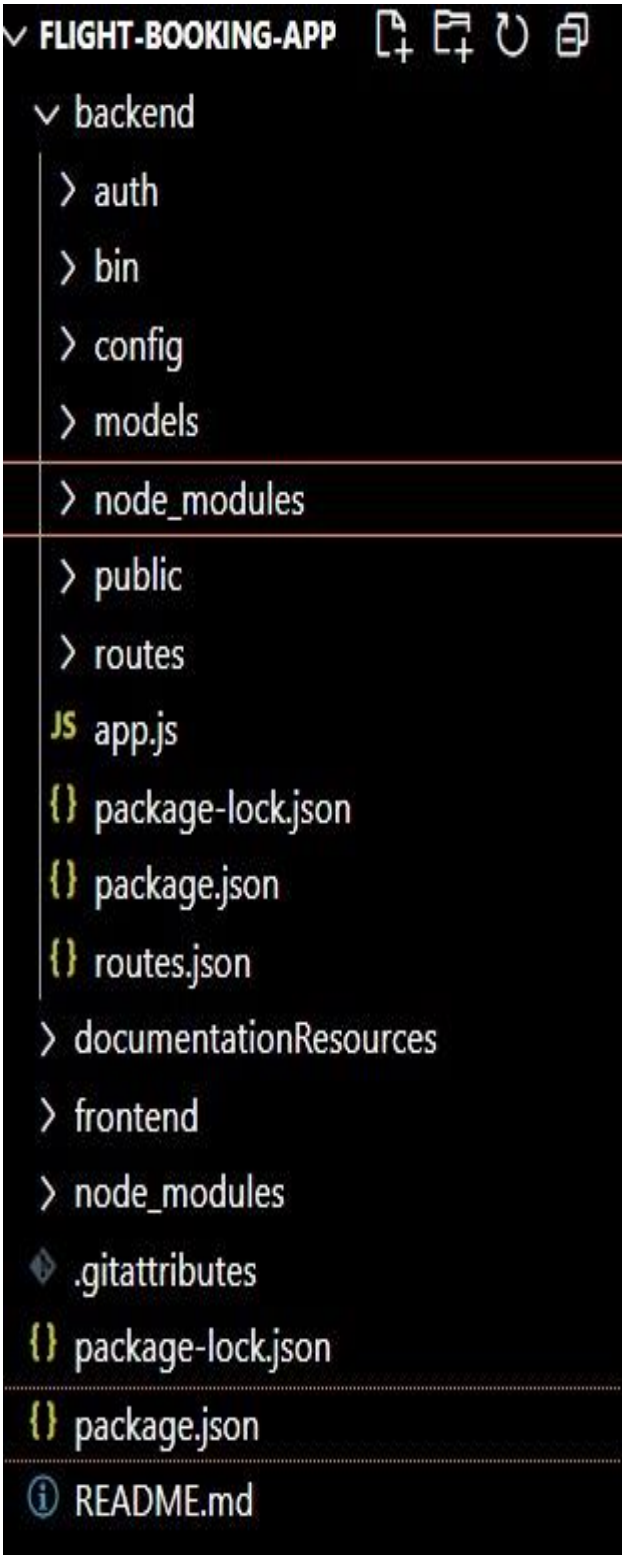
- To start the development server, execute the following command:

npm run dev or npm run start

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

Access the App:
• Open your web browser and navigate to http://localhost:3000
• You should see the flight booking app's homepage, indicating that the installation and the setup was successful.
You have successfully installed and set up the flight booking app on your local machine. You can now proceed with further customization, development, and testing as needed.





The project is structured to include both Frontend and Backend components, each responsible for distinct tasks in the development of the flight booking Webpage.

FRONTEND PART :

- REACT COMPONENTS – Each component is designed for user interactions such as displaying number of seats booking airline, and viewing notifications.
- ROUTING – Handles navigation between pages like customer dashboard, booking form, history, etc.
- STATE MANAGEMENT – Keeps track of the logged-in user's session, flight details, and booking statuses.
- STYLING – Uses CSS and UI libraries (e.g., Ant Design) to style the components.

BACKEND PART :

The backend handles the server-side operations, including user authentication, data handling, and API responses. It contains the following files and folders:

- API ENDPOINTS – Defines the routes for handling passengers, functionalities, such as booking, updating statuses, etc.
- DATABASE MODELS – Defines schemas for Users, flight booking using MongoDB and Mongoose.
- AUTHENTICATION & AUTHORIZATION – Manages login, registration, and access control for different user roles.
- NOTIFICATION SYSTEM – Sends notifications to users about booking status updates.

APPLICATION FLOW:

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

SETUP & CONFIGURATION :

Setting up the flight booking Webpage involves configuring both the Frontend (React.js) and Backend (Node.js, Express.js, MongoDB) to ensure the application runs smoothly. Below are the steps to set up and configure the environment for your project.

FRONTEND CONFIGURATION :

INSTALLATION :

Clone the Repository:

- Clone the project from GitHub to your local machine:
- bash
- Copy code
- git clone <your-repository-url>
- Replace <your-repository-url> with the URL of your project repository.

Navigate to Frontend Directory:

- After cloning, navigate to the frontend folder where the React.js app is located:
- bash
- Copy code
- cd frontend

Install Dependencies:

- Use npm (Node Package Manager) to install the necessary dependencies:

- bash
- Copy code
- npm install
- This will install all the required libraries, such as React, React Router, Ant Design, and others.

Run the React Development Server:

- To start the frontend server and run the React application:
- bash
- Copy code
- npm start
- The application will be available at http://localhost:3000 in your browser.

BACKEND CONFIGURATION :

INSTALLATION :
Navigate to Backend Directory:

- Move to the backend folder of your project:
- bash
- cd backend
- Install Dependencies:
- Install the necessary backend dependencies using npm:
- bash
- Copy code
- npm install
- This will install all required libraries for Node.js and Express.js, such as express, mongoose, bcryptjs, jsonwebtoken, etc.

Configure MongoDB :

- Ensure you have MongoDB installed on your local machine or use a cloud-based MongoDB service like MongoDB Atlas.
- In the backend, open the configuration file (e.g., config/db.js or .env) and configure the connection URL for MongoDB:
- If you are using MongoDB Atlas, replace the localhost part with your MongoDB Atlas connection string.

Set Up Environment Variables:

- Create a .env file in the backend directory to store environment-specific variables such as:
- bash
- Copy code
- PORT=5000
- JWT_SECRET=your_jwt_secret
- Make sure to replace your_jwt_secret with a strong secret key for JWT authentication.

Run the Backend Server:

- Start the backend server by running:
- bash
- Copy code

- npm start
- The backend server will be running at http://localhost:5000.

DATABASE CONFIGURATION (MONGODB) :

Install MongoDB (Local Installation):

- If you are using a local MongoDB instance, download and install it from the official MongoDB website: Download MongoDB

Set Up MongoDB Database:

- After installation, start the MongoDB service:
- bash
- Copy code
- mongod
- This will run MongoDB on the default port 27017.

MongoDB Atlas (Cloud-based MongoDB):

- If you prefer to use MongoDB Atlas, create an account on MongoDB Atlas and create a cluster.
- Once the cluster is set up, get the connection string and replace it in the backend's .env file:
- bash
- Copy code
- MONGO_URI=<your-mongodb-atlas-connection-string>

FINAL CONFIGURATION & RUNNING THE APP

Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function correctly.
- You can open two terminal windows or use tools concurrently to run both servers together.
- To install concurrently, run:
- bash
- Copy code
- npm install concurrently --save-dev
-

In your package.json file, add a script to run both servers:

json
Copy code

```
  "dependencies": {
    "@kommunicate/kommunicate-chatbot-plugin": "^0.0.5",
    "mongoose": "^6.3.3"
  },
  "devDependencies": {
    "gh-pages": "^4.0.0"
  }
}
```

Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash

- Copy code
- npm start
- The backend will be available at http://localhost:5000, and the frontend at http://localhost:3000.

VERIFYING THE APP :

Check Frontend:

- Open your browser and go to http://localhost:3000. The React.js application should load with the list of doctors, booking forms, and status updates.

Check Backend:

- Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as user login, doctor registration, and appointment creation.

ADDITIONAL SETUP :

- Version Control:
- If you haven't already done so, initialise a Git repository in the root of your project:
- bash
- Copy code
- git init

FOLDER SETUP :

The folder structure for your Doctor Appointment Webpage project will include separate folders for the frontend and backend components to keep the code organised and modular. Here's how to set it up:

PROJECT ROOT STRUCTURE :

Create the Main Folders:

- In your project's root directory, create two main folders: frontend and backend.
- plaintext
- Copy code
- project-root/
  ├── frontend/
  └── backend/

BACKEND SETUP :

- Install Necessary Packages in the Backend Folder:
- Navigate to the backend folder and install the following essential packages:
- plaintext
- Copy code
- backend/
  ├── config/
  ├── node_modules/
  ├── models/
  ├── routes/
  ├── public/
  └── server.js

Packages to Install :

- cors: To enable cross-origin requests.
- bcryptjs: For securely hashing user passwords.
- express: A lightweight framework to handle server-side routing and API management.
- dotenv: For loading environment variables.
- mongoose: To connect and interact with MongoDB.
- multer: To handle file uploads.
- nodemon: A utility to auto-restart the server upon code changes (for development).
- jsonwebtoken: To manage secure, stateless user authentication.
- Installation Commands

Run these commands in the backend folder:

- bash
- Copy code
- npm init -y
- npm i
- npm start

FRONTEND SETUP :

- React Project Initialization:
- Navigate to the frontend folder and initialise a new React application:
- plaintext
- Copy code
- frontend/

```
├── public/
├── src/
│   ├── components/
│   ├── pages/
│   ├── services/
│   └── App.js
└── package.json
```

- Setting Up the Frontend Project
- In the frontend folder, run the following commands to set up and install any initial dependencies:
- bash

PROJECT FLOW :

Project Demo :

- Before diving into development, you can view a demo of the project to understand its functionality and user interactions.
- Project FlowDemo Video : https://drive.google.com/drive/folders/1pteT8STdObONWwELNDHRK9biItLuiJ-1?usp=sharing
- Project Code Repository
- The source code for this project can be accessed and cloned from GitHub, providing a base structure and example code for further customization and understanding.
- GitHub Repository: Source Code

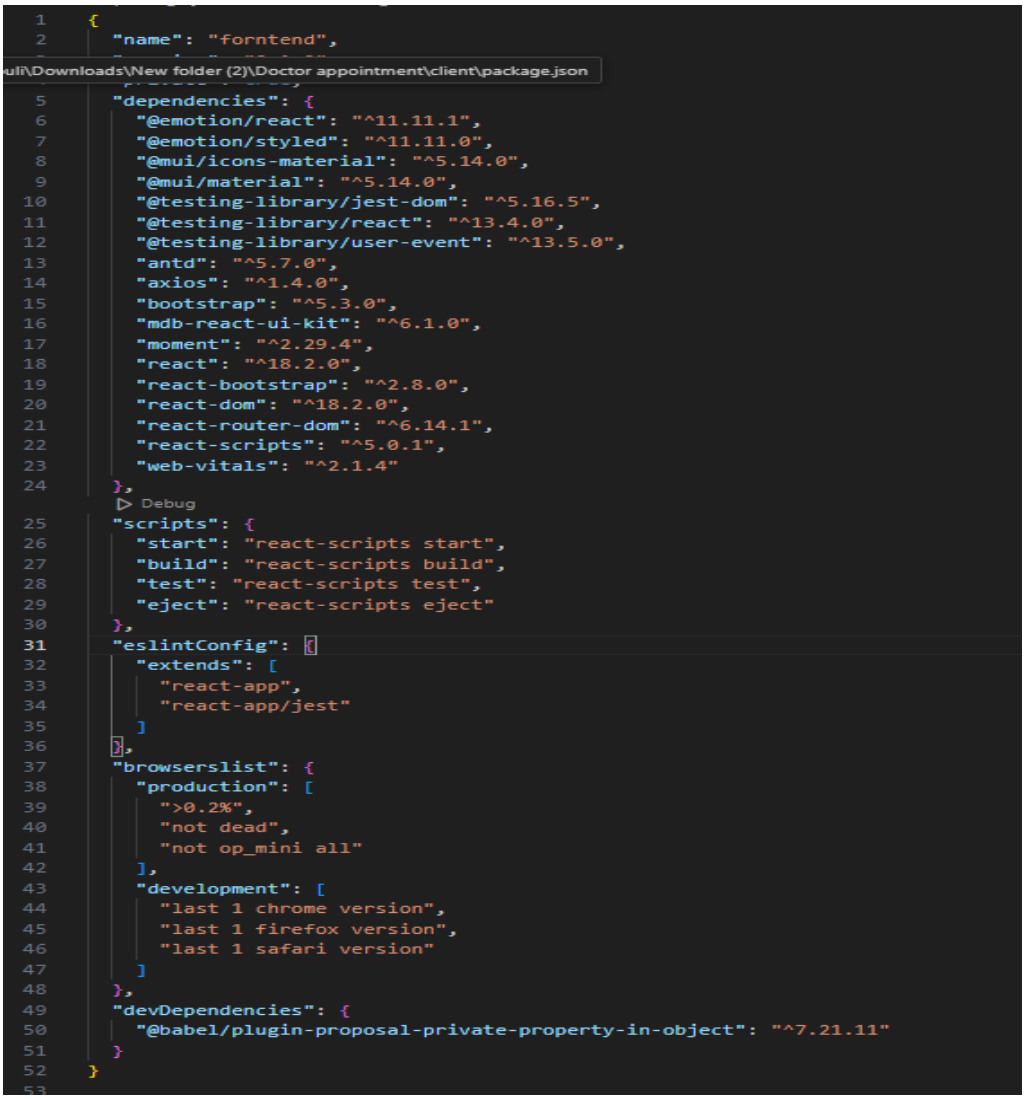
Video Tutorials :

- For a step-by-step guide on setting up and working with this project, follow the video tutorials below:

- Video Guide 1: Setting Up the Backend
- Video Guide 2: Configuring Frontend and API Endpoints
- Video Guide 3: Testing and Deployment
- These resources should give you a solid foundation and clear understanding of the project structure and workflow before development begins.

MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

- Setting up a structured environment is crucial for any application. By organising the project into separate folders for the frontend and backend, we ensure that the codebase is manageable and scalable.



```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.1",
    "mongoose": "^7.3.2",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.0.1"
  }
}
```

PROJECT FOLDERS:

- Frontend Folder: Contains all code related to the user interface, written in JavaScript using frameworks and libraries like React, Material UI, and Bootstrap. This setup helps maintain a clear boundary between UI logic and server logic.
- Backend Folder: Manages the server, API routes, and database interactions, typically handled through Node.js and Express.js. Using separate folders enables a modular structure, allowing changes in one area without affecting the other.

LIBRARY AND TOOL INSTALLATION:

Backend Libraries:

- Node.js: Provides a runtime environment to run JavaScript code on the server side.
- MongoDB: A NoSQL database, perfect for flexible and schema-less data storage, ideal for applications needing frequent updates and various data types.
- Bcrypt: Encrypts passwords for secure authentication, helping protect user data from potential breaches.
- Body-parser: Parses incoming request bodies, making it easy to access data in various formats like JSON.
- Frontend Libraries:
- React.js: Manages component-based UI creation, providing the flexibility to build reusable UI components.
- Material UI & Bootstrap: Provides styling frameworks, ensuring a consistent, responsive, and visually appealing design.
- Axios: Facilitates easy HTTP requests, allowing the frontend to communicate with the backend effectively.

MILESTONE 2: BACKEND DEVELOPMENT :

The backend forms the core logic and data management for the project. A well-structured backend ensures efficient data handling, security, and scalability.

EXPRESS.JS SERVER SETUP:

- Express Server: Acts as a hub for all requests and responses, routing them to appropriate endpoints. It's Essential for managing incoming requests from the frontend, processing them, and sending responses back.
- Middleware Configuration: Middleware like body-parser parses JSON data in requests, while cors enables cross-origin communication between the frontend and backend. Middleware makes it easy to add additional functionality, like error handling or data validation, without interfering with core application logic.

API ROUTE DEFINITION:

- Route Organization: Organizing routes by functionality (e.g., authentication, appointments, complaints) keeps the codebase readable and easy to maintain. For instance, all authentication-related routes can reside in an auth.js file, ensuring each file has a single purpose.
- Express Route Handlers: Route handlers manage the flow of data between client and server, such as fetching, creating, updating, and deleting records.

DATA MODELS (SCHEMAS) WITH MONGOOSE:

- User Schema: Defines structure for user data and includes fields for personal information and role-based access (e.g., customer, admin). Using a schema ensures consistent data storage.
- Appointment and Complaint Models: Models like Appointment and Complaint manage complex data interactions, including relationships between users and appointments, allowing efficient querying.
- CRUD Operations: CRUD functionalities (Create, Read, Update, Delete) provide a standard interface for handling data operations. They simplify data manipulation in a structured, predictable way.

USER AUTHENTICATION:

- JWT Authentication: JSON Web Tokens securely handle session management, ensuring that only verified users access protected routes. JWT tokens are embedded in request headers, verifying each request without storing session data on the server.

ADMIN AND TRANSACTION HANDLING:

- Admin Privileges: Administrators oversee user registrations, approve appointments, and manage doctor applications. Admin-specific routes ensure that these actions are isolated and secure.
- Transaction Management: This functionality allows customers to interact with appointments, booking history, and cancellations in real-time.

ERROR HANDLING:

- Middleware for Error Handling: Error-handling middleware catches issues and sends meaningful error responses with HTTP status codes (like 404 for Not Found or 500 for Server Error), enabling better debugging and user feedback.
-

MILESTONE 3: DATABASE DEVELOPMENT

- Using MongoDB as the database provides a flexible, schema-less structure, perfect for handling different types of user and appointment data.

SCHEMAS FOR DATABASE COLLECTIONS:

- User Schema: Defines fields for user information like name, email, password, and userType. This schema allows fine-grained control over user data and easy retrieval of information.
- Complaint and Assigned Complaint Schemas: These schemas manage complaint data, with fields linking complaints to users and statuses. They allow efficient tracking of complaints and status updates by linking agents to users.
- Chat Window Schema: This schema organises messages between users and agents, storing them by complaint ID for a streamlined user-agent communication flow.

DATABASE COLLECTIONS IN MONGODB:

- MongoDB collections, such as users, complaints, and messages, provide a structured, NoSQL approach to data management, making it easy to scale as data grows.
- MILESTONE 4: FRONTEND DEVELOPMENT
- Frontend development focuses on creating an interactive, intuitive user experience through a React-based user interface.

REACT APPLICATION SETUP:

- Folder Structure and Libraries: Setting up the initial React app structure and libraries ensures a smooth development workflow. By organising files into components, services, and pages, the project becomes easy to navigate and maintain.
- UI Component Libraries: Material UI and Bootstrap offer pre-built components, enabling rapid UI development and consistent design across all screens.

UI COMPONENTS FOR REUSABILITY:

- Reusable Components: Each UI element, like forms, dashboards, and buttons, is designed as a reusable component. This modularity allows efficient reuse across the app, reducing development time and ensuring consistency.
- Styling and Layout: Styling and layout components maintain a cohesive look and feel, contributing to the user experience with clean, intuitive visuals.

FRONTEND LOGIC IMPLEMENTATION:

- API Integration: Axios is used to make API calls to the backend, connecting UI components with data from the server.
- Data Binding and State Management: React's state management binds data to the UI, automatically updating it as the user interacts with the app.

MILESTONE 5: PROJECT IMPLEMENTATION AND TESTING:

- After completing development, running a final set of tests is crucial for identifying any bugs or issues.

VERIFY FUNCTIONALITY:

Running the entire application ensures that each part (frontend, backend, database) works cohesively.
Testing various user flows (e.g., booking, cancelling, updating appointments) helps confirm that all processes are functioning as intended.

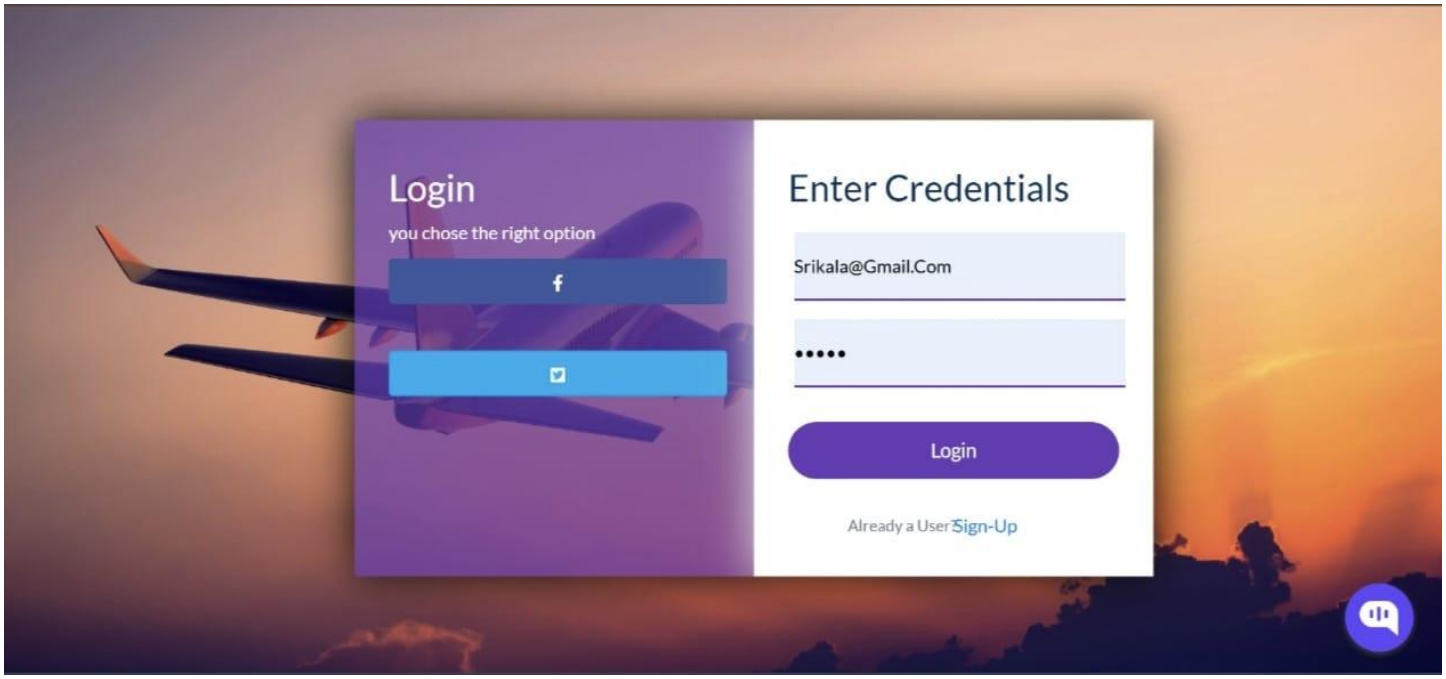
USER INTERFACE ELEMENTS:

Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.
Ensuring responsive design and usability across devices and screen sizes is also essential.
FINAL DEPLOYMENT:

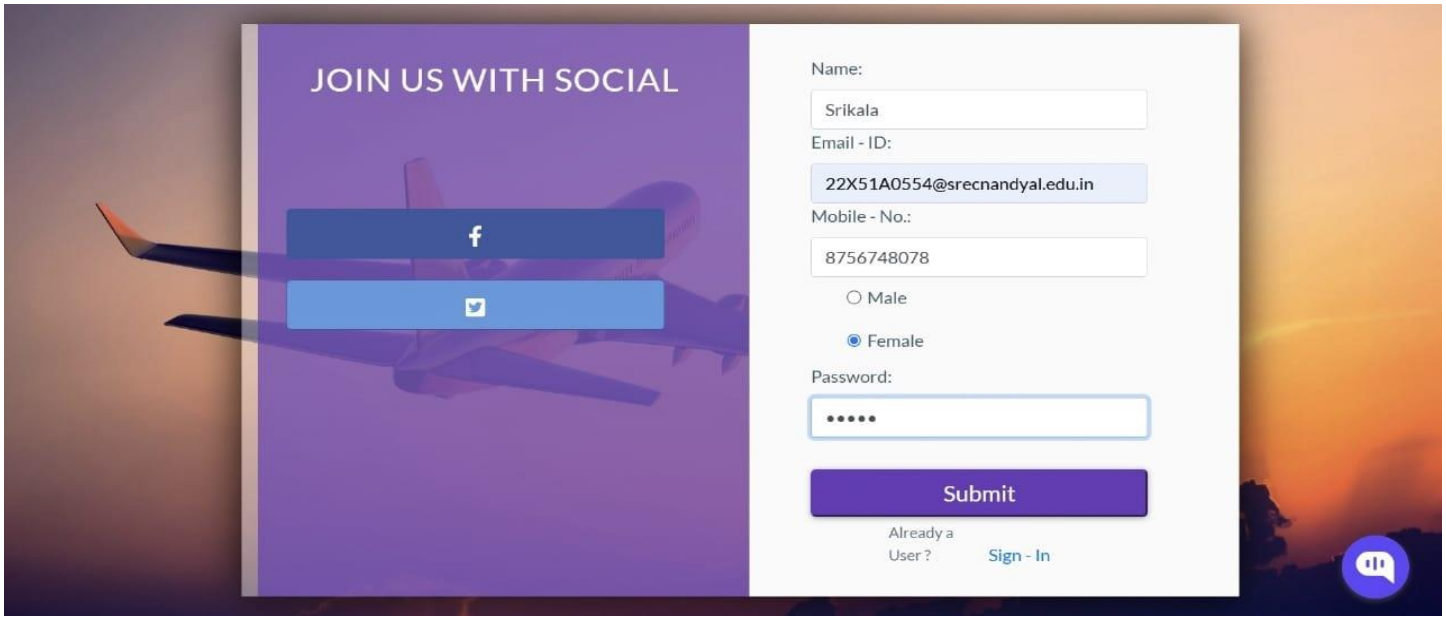
Once testing is complete, the application can be deployed to a production server, making it accessible to end-users.

HOME PAGE :

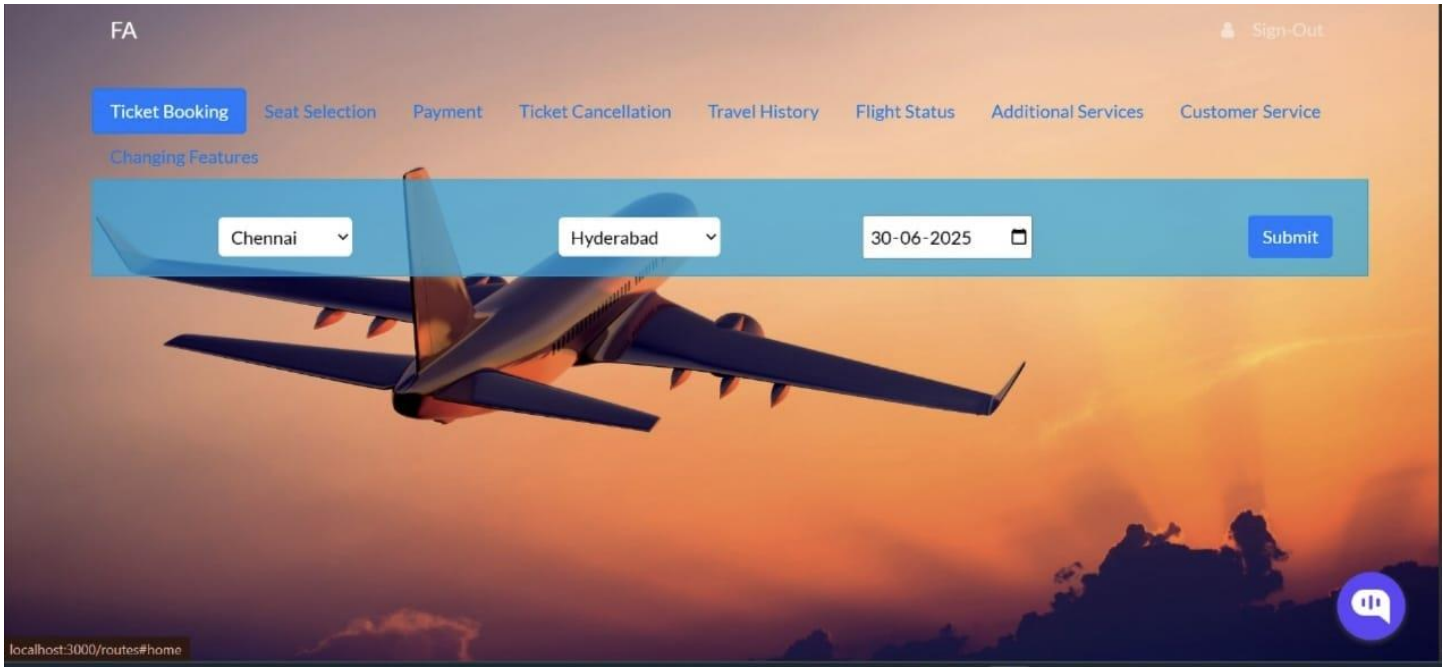
LOGIN PAGE :



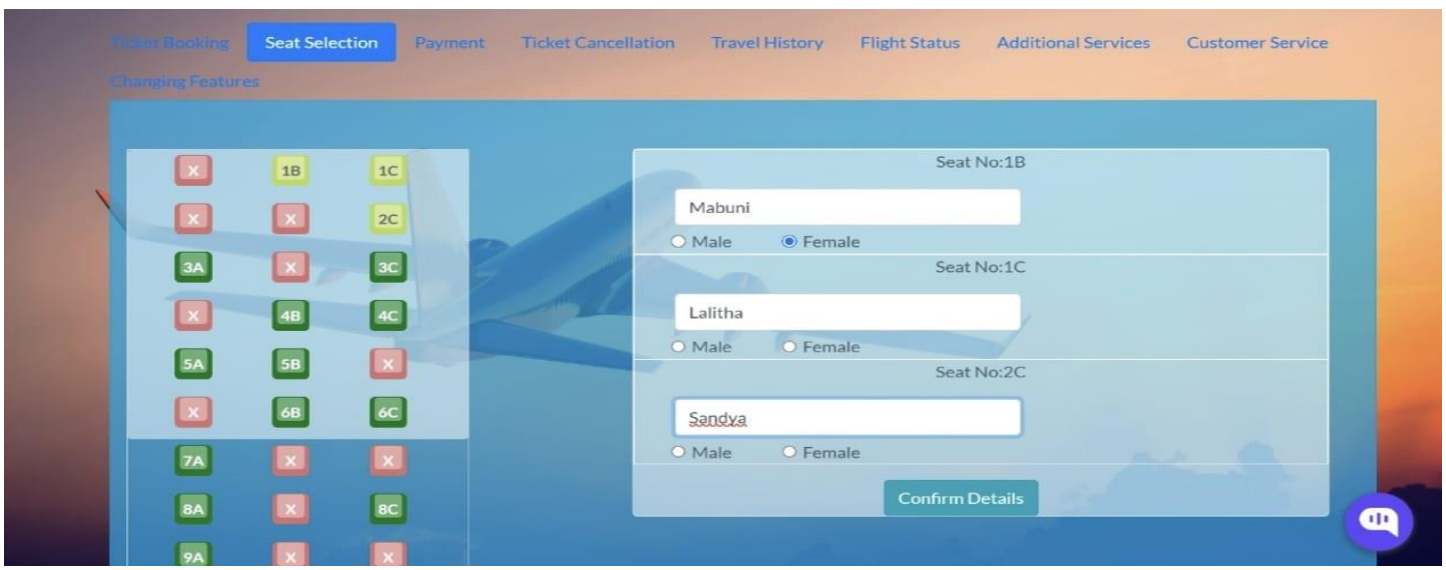
SIGN-UP PAGE :



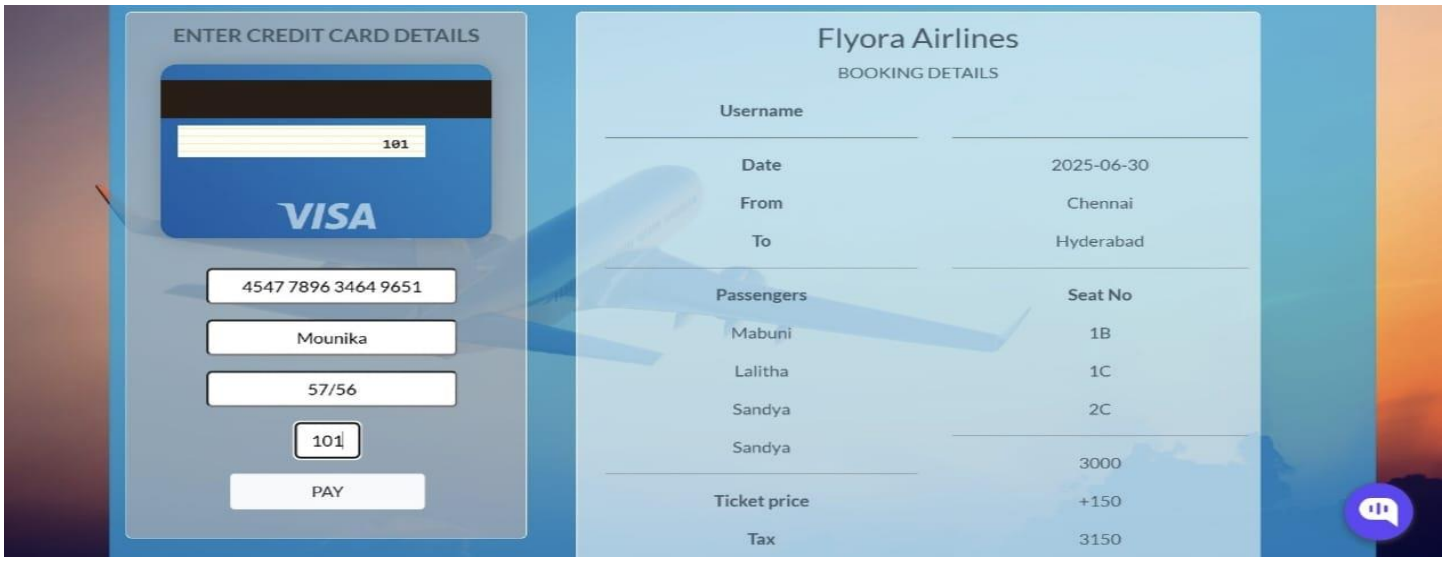
TICKET BOOLING PAGE :



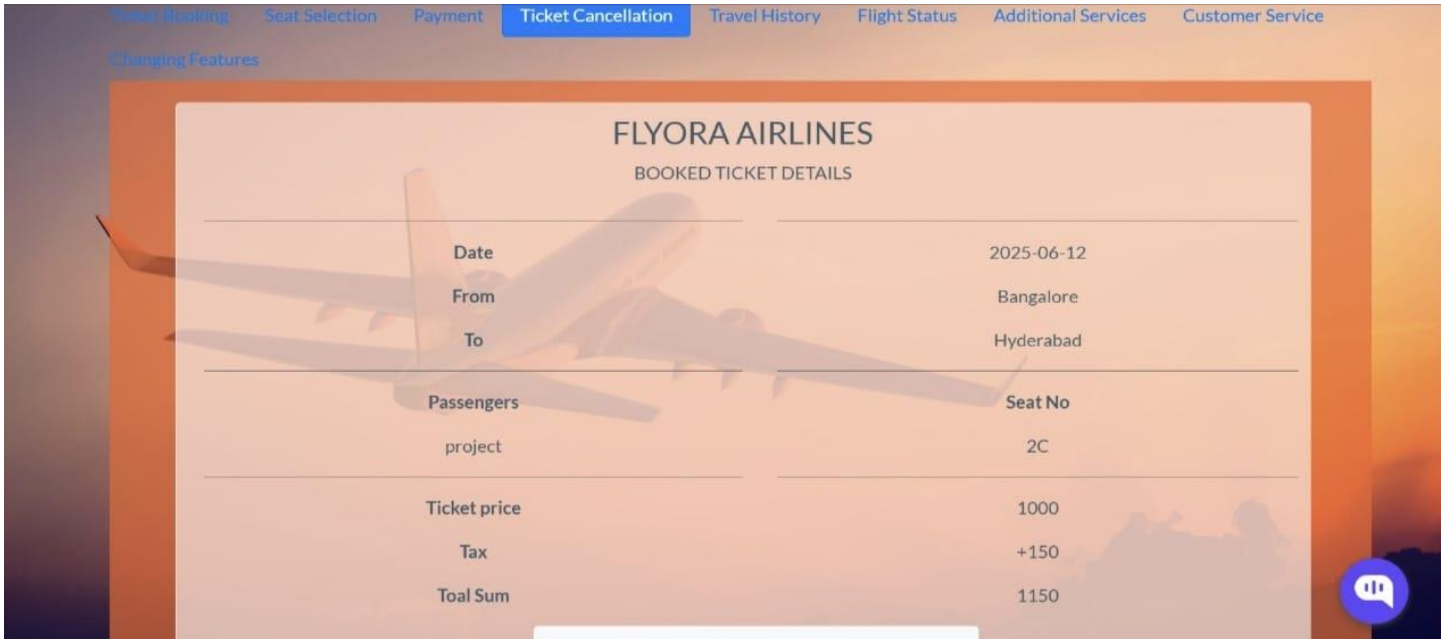
SEAT SELECTION PAGE :



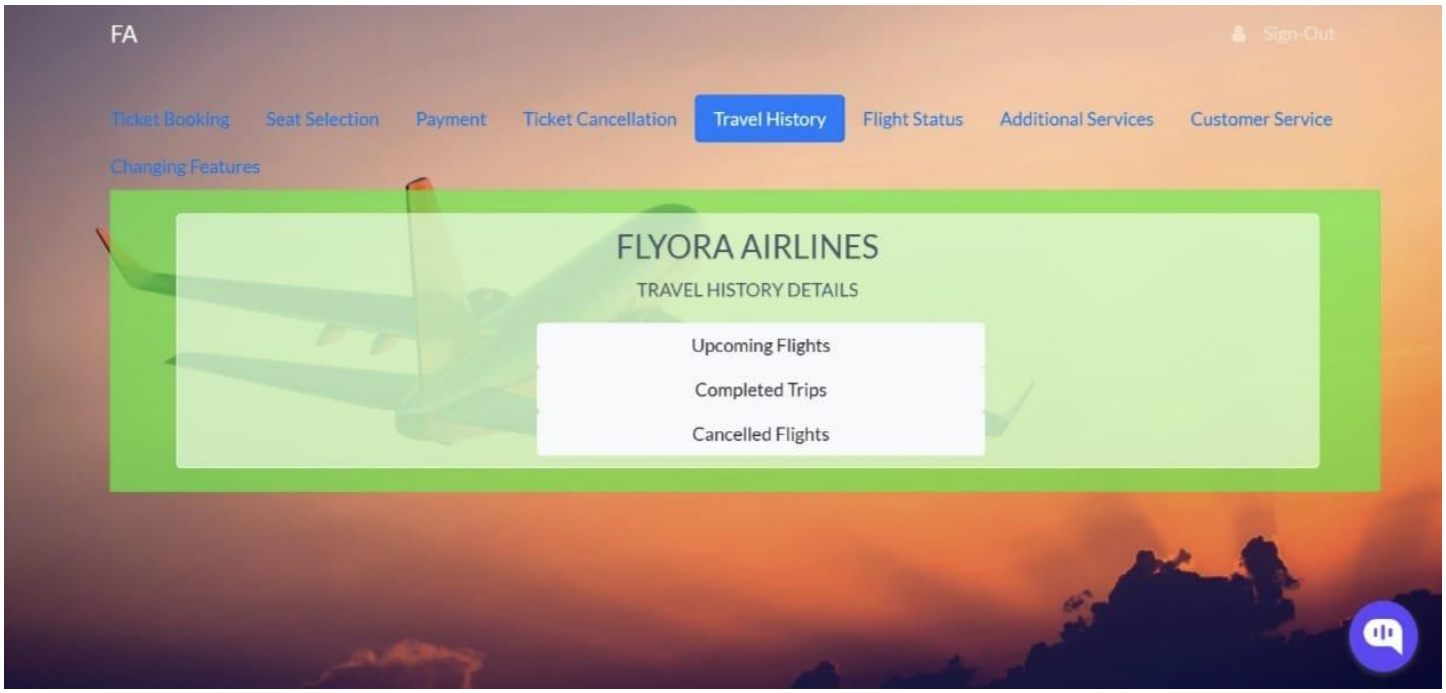
PAYMENT PAGE:



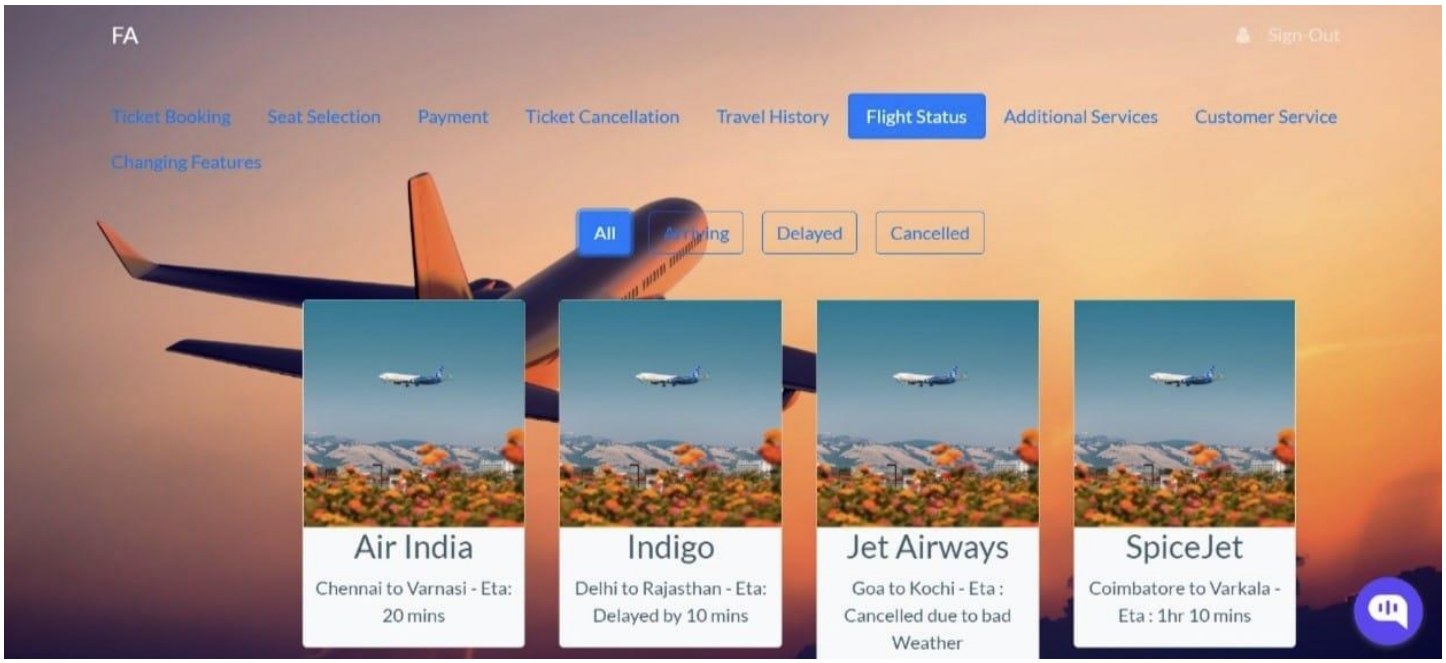
TICKET CANCELLATION PAGE:



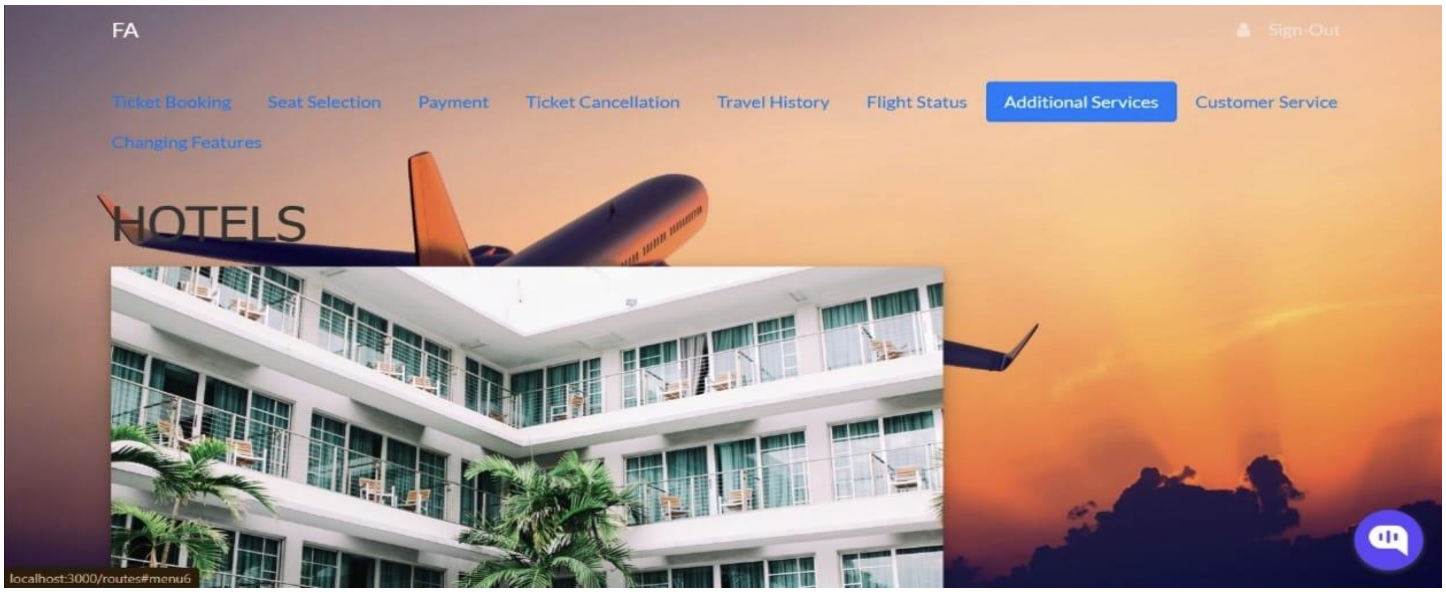
TRAVEL HISTORY PAGE:



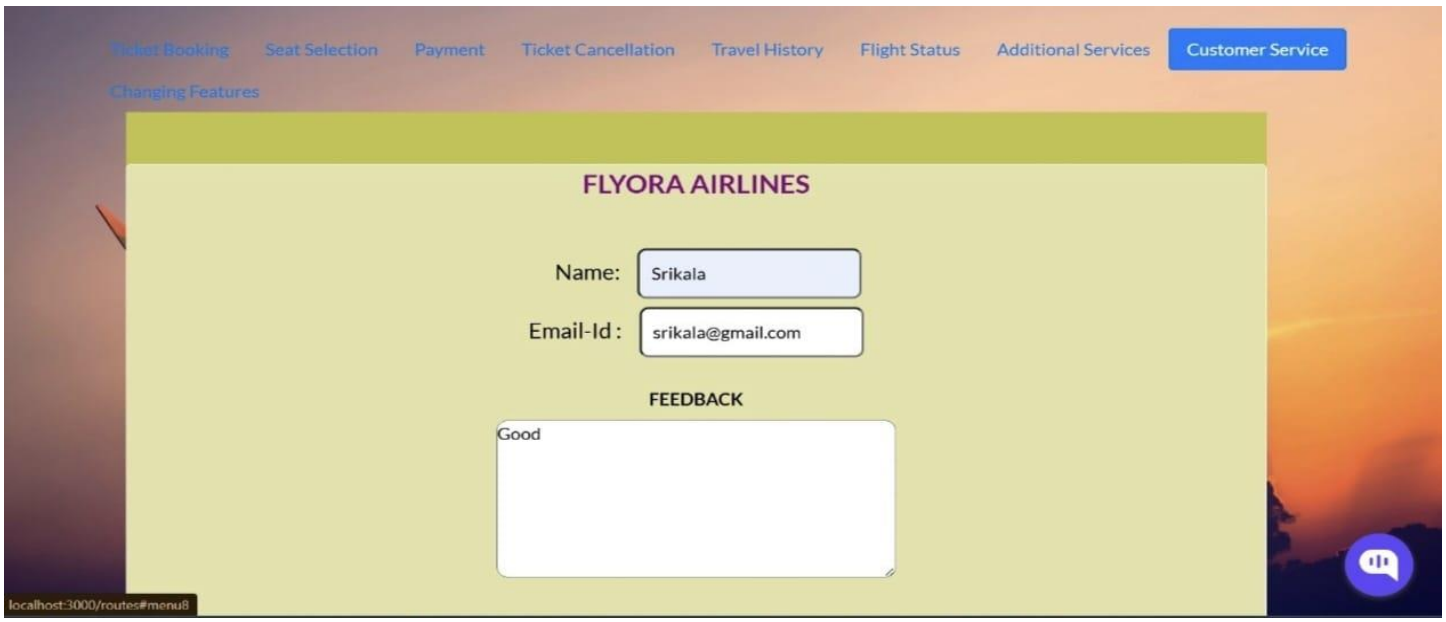
FLIGHT STATUS PAGE:



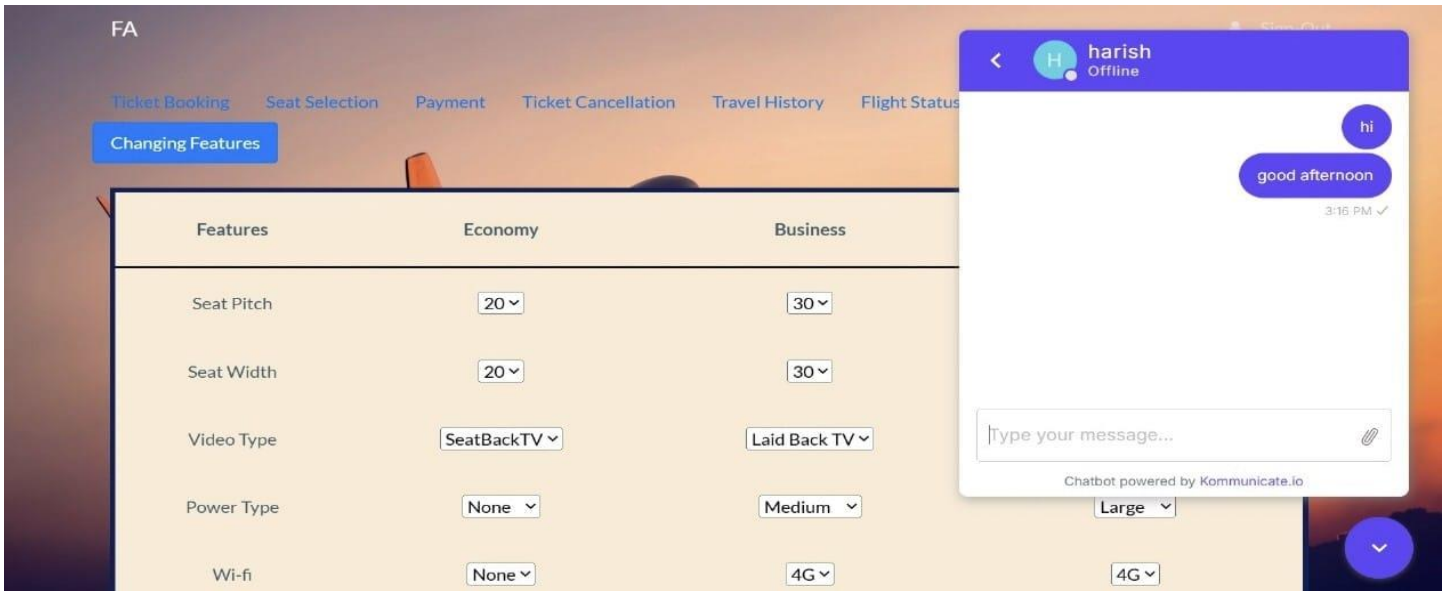
ADDITIONAL SERVICES PAGE:



CUSTOMER SERVICE PAGE:



CHANGING FEATURES PAGE:



CONCLUSION

The Flight Booking Project successfully demonstrates the integration of modern web technologies to streamline the process of searching, booking, and managing flight tickets. Through a user-friendly interface and a robust backend system, the project addresses the core requirements of an airline reservation system, including real-time flight availability, secure user authentication, and efficient booking workflows.

The project also highlights the importance of scalability, data integrity, and responsive design in building reliable travel platforms. Implementing features such as dynamic pricing, ticket cancellations, and email confirmations added significant value and improved the user experience.

In conclusion, this project lays a strong foundation for a fully functional flight reservation system. With further enhancements—such as payment gateway integration, multilingual support, and machine learning-based recommendation engines—it can evolve into a comprehensive solution for real-world deployment in the travel and aviation industry.