Introduction

Project Title: FLIGHT BOOKING WEBSITE

Team Members: MUGESH G -

Project Overview:

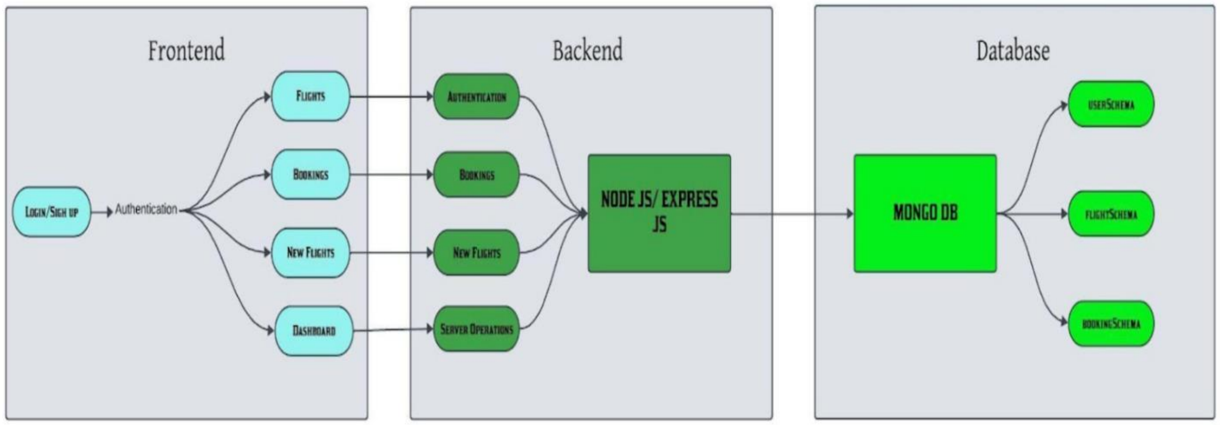
The purpose of this project is to demonstrate the construction of a comprehensive, modern, and scalable web application for flight reservations using the MERN stack.

Key functionalities include **Flight Search & Booking**, allowing users to search for flights by destination, dates, and preferences, and book tickets securely. Users can also **View & Manage Bookings**, modifying or canceling reservations, and manage their **User Profile**, including personal details and frequent flyer information. For **Admins/Operators**, features include **Flight Management** (adding, editing, or removing schedules and pricing), **Booking Management** (monitoring user reservations), and **User Management** (accessing profiles and history).

The defined roles and responsibilities are: **Customer** (signs up, manages profile, searches and books flights, views bookings, contacts support) and **Admin / Operator** (manages flights and pricing, handles incoming bookings and availability, manages operations schedules, analyzes business performance, and communicates with users).

Architecture:

The system follows a three-tier MERN architecture.



Level Architecture Diagram - Frontend, Backend, Database:

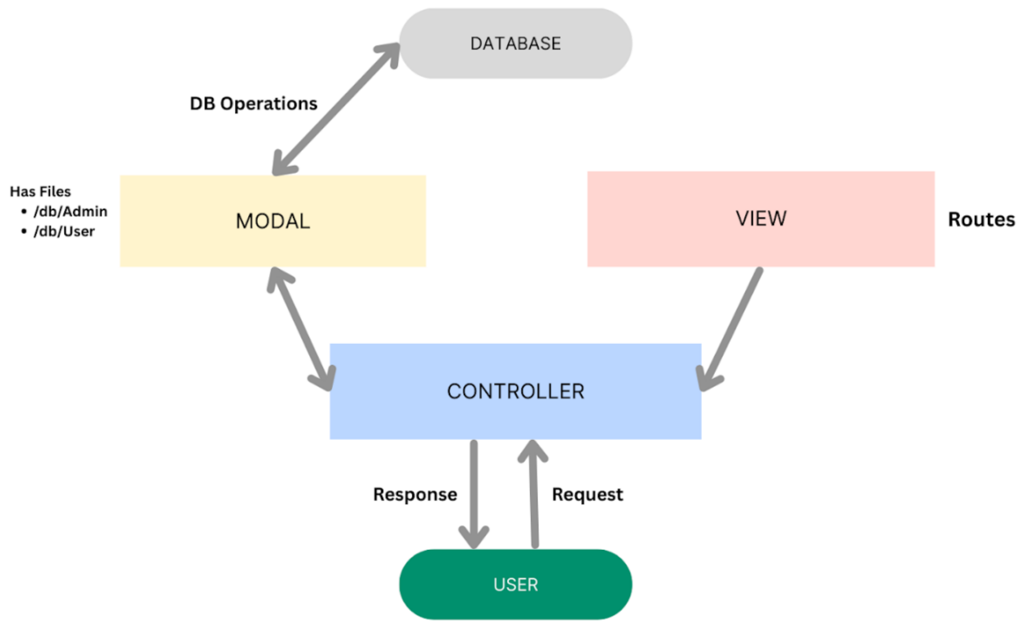
The **Frontend** is built using **React** to handle components like Login/Sign Up, Flights (search and view), Bookings (management), New Flights (admin/operator function), and the Dashboard. Authentication ensures only verified users interact with the app.

The **Backend** powered by **Node.js/Express.js**, processing logic and acting as the RESTful API server. It includes modules for **Authentication**, **Bookings**, **New Flights**, and **Server Operations** (routing/error handling).

The **Database** utilizes **MongoDB**, a NoSQL solution, with three primary schemas: **userSchema** (for user info and credentials), **flightSchema** (for routes, timings, and seat availability), and **bookingSchema** (for linking users, flights, and transaction histories).

Backend MVC Pattern:

The backend application follows the **Model-View-Controller (MVC)** architectural pattern, separating concerns for modularity, maintenance, and scalability.

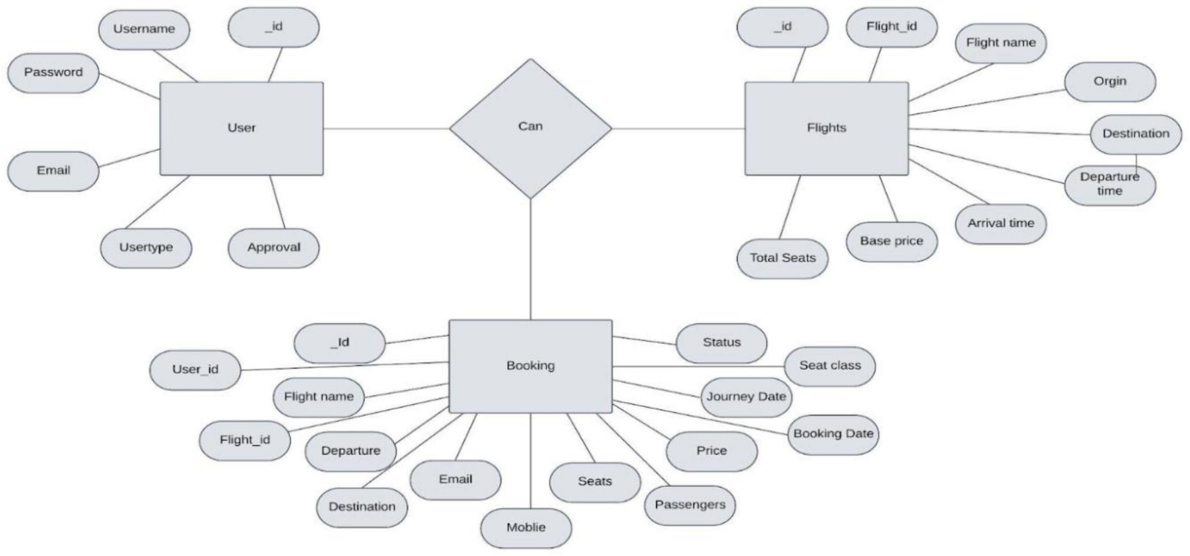


MVC Pattern Diagram - Model, View, Controller:

The **Model Layer** handles all data logic and Mongoose schemas for MongoDB operations. The **Controller Layer** receives requests, processes input, calls model methods, and returns the response. The **View Layer** implemented as the **routing layer**, defining API endpoints and invoking the appropriate controller functions.

Database Model (ER Diagram):

The MongoDB schema configuration based on the following entity relationships:



Entity Relationship Diagram - User, Flights, and Booking Schemas

Setup Instructions:

Prerequisites include having **Node.js** and **MongoDB** installed or accessible. The installation involves cloning the repository, navigating to the client and server directories, running `npm install` in each, and setting up the environment variables (`.env` file) for the MongoDB URI, server port, and JWT secret.

Folder Structure:

The **Client** (`/client`) organizes the React frontend, containing folders like `public`, `src/assets`, `src/components`, `context`, `pages`, `RouteProtectors`, and `styles`.

The **Server** (`/server`) organizes the Node.js/Express backend according to MVC. This includes the top-level directories for `/controllers` (e.g., `authController.js`), `/models` (e.g., `UserSchema.js`), and `/routes` (e.g., `authRoutes.js`), along with the `.env`, `index.js`, and package files.

Running the Application:

To start the application, ensure the MongoDB server is running. For the **Backend**, navigate to the `/server` directory and run the start command (`npm start`). For the **Frontend**, navigate to the `/client` directory and run the start command (`npm start`).

API Documentation:

The backend exposes RESTful API endpoints:

Authentication Routes (prefixed with `/api/auth`):

- `POST /api/auth/login`: Handles user login.
- `POST /api/auth/register`: Handles user registration.

Admin Routes:

- `POST /approve-operator`: Approves an operator.
- `POST /reject-operator`: Rejects an operator.
- `GET /fetch-users`: Retrieves all users.

Customer Routes:

- `POST /book-ticket`: Handles new flight bookings.
- `PUT /cancel-ticket/:id`: Cancels a specific booking by ID.

Flight Routes:

- `POST /add-flight`: Adds a new flight record.
- `PUT /update-flight`: Updates an existing flight record.
- `GET /fetch-flights`: Retrieves all flights.

Authentication:

Authentication and authorization are handled using **JSON Web Tokens (JWT)**. On successful login, a JWT token is generated and sent back with user data (excluding the password). For registration, the user's password is **securely hashed**. Flight operators have their approval status set to "not-approved" initially, requiring manual approval by an Admin. Authorization is enforced using **Route Protectors** in the frontend, which require authentication and authorization based on the user's `usertype`.

User Interface:

The frontend is a React single-page application.

The **Navbar** component uses `localStorage` to check the logged-in user's role (`usertype`) and displays role-based navigation menus. The **Landing Page** manages flight search inputs and redirects admins/operators to their dashboards on component mount. The **Admin Page** displays a dashboard showing User, Booking, and Flight counts, and a section for reviewing and approving new operator registration requests. The **Login/Signup** components handle form input, secure data transfer via Axios, and manage tokens and redirects.

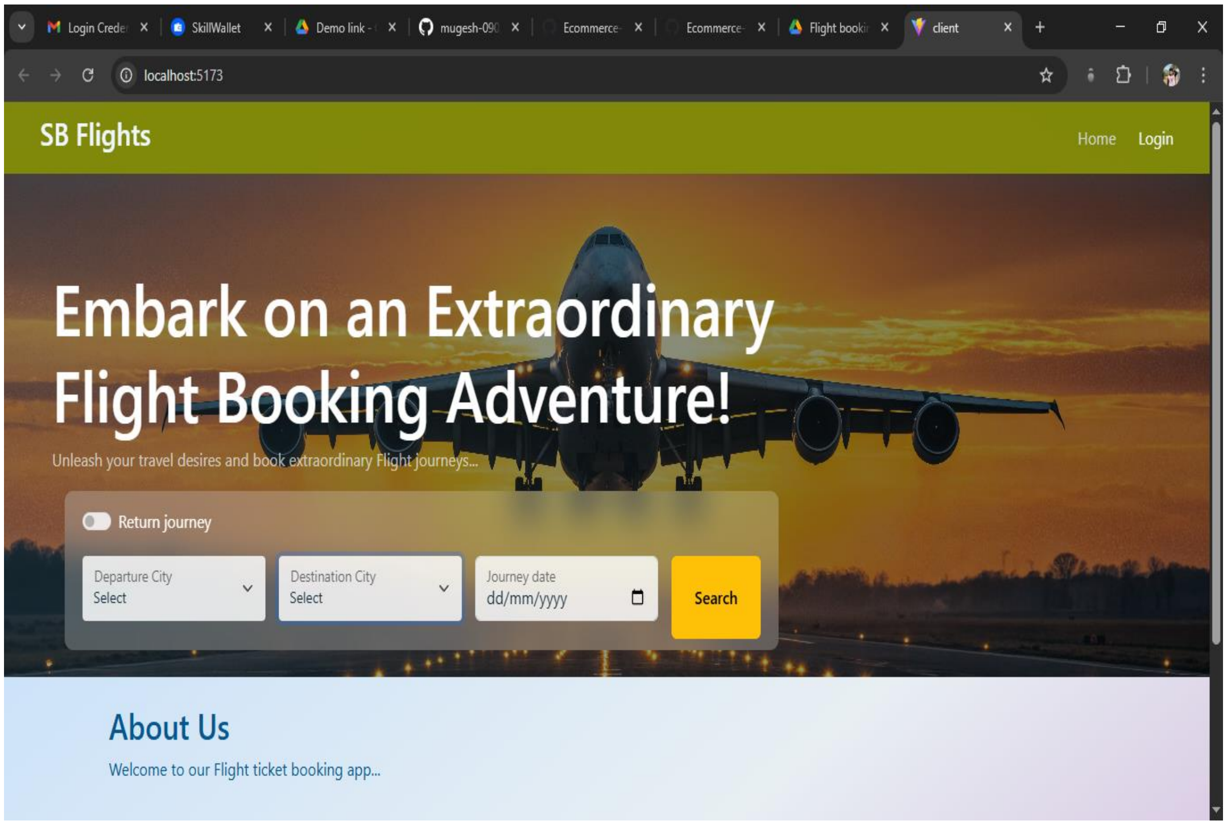
Testing:

The project requires a thorough testing strategy focused on controller logic and API functionality. Tools like **Jest** (frontend) and **Mocha/Chai** (backend) are recommended. Key unit tests focus on:

- **Book Ticket Logic:** Verifying that the controller correctly calculates available seats, retrieves existing bookings for the specified criteria, assigns new seat numbers, and accurately saves the new booking record.
- **Admin Approval Logic:** Testing that the `Approve` function correctly locates a user by ID and sets their `approval` status to "approved", and that the `Reject` function sets the status to "rejected".

Screenshots or Demo:

This section should provide screenshots or a link to a demo showcasing the application.

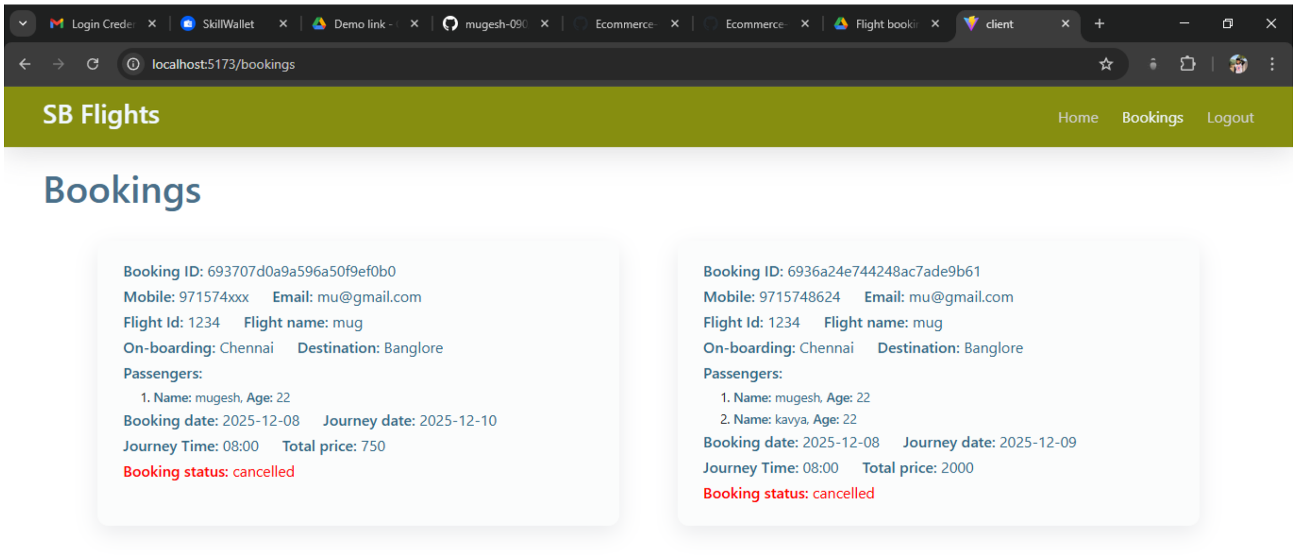


Embark on an Extraordinary Flight Booking Adventure!

Unleash your travel desires and book extraordinary Flight journeys...

About Us

Welcome to our Flight ticket booking app...



SB Flights

Home Bookings Logout

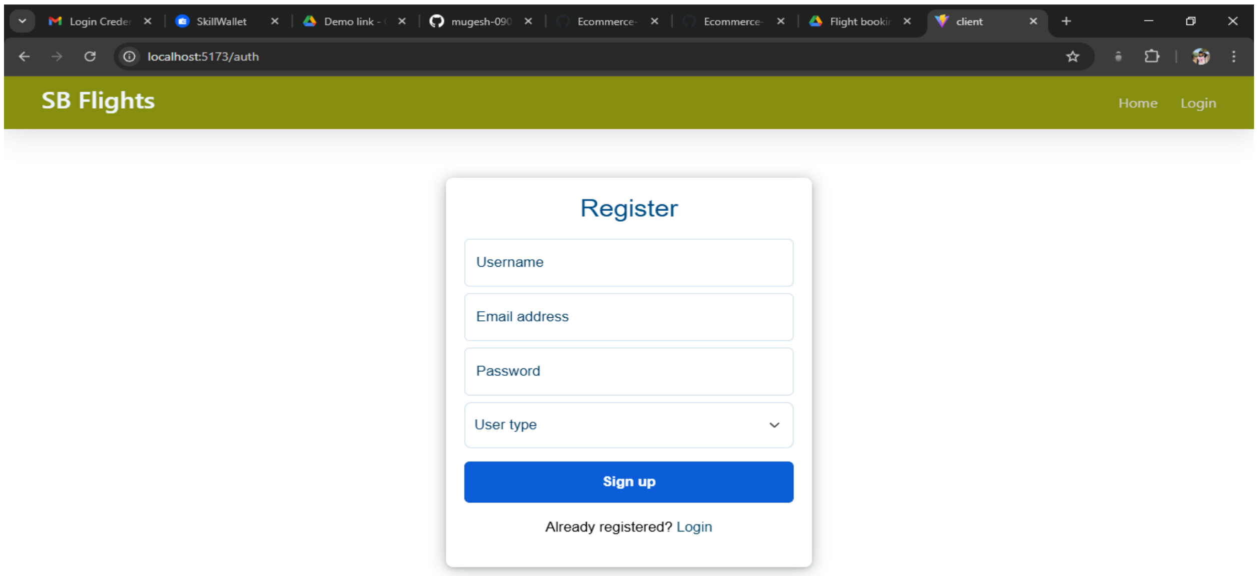
Bookings

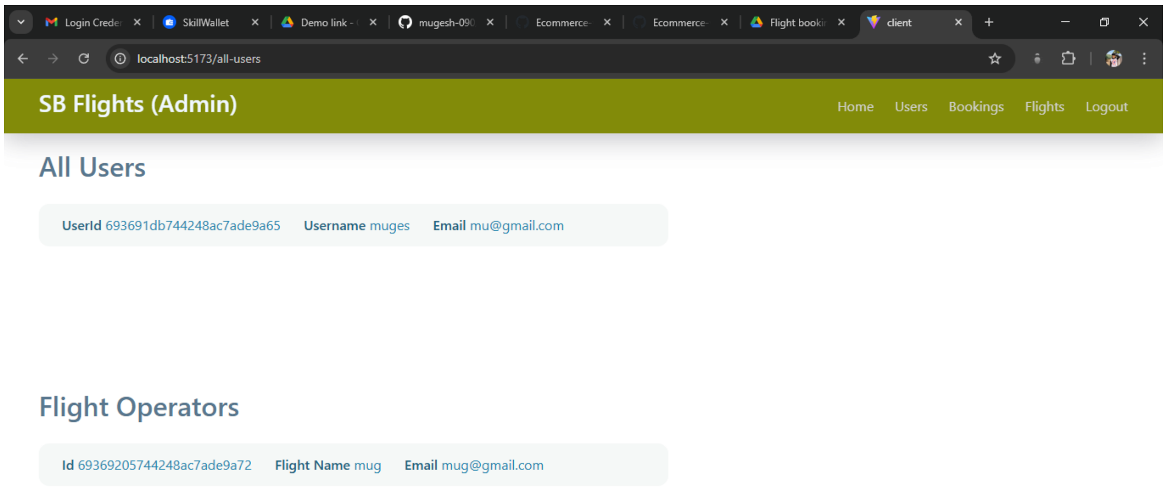
Booking ID: 693707d0a9a596a50f9ef0b0
Mobile: 971574xxx **Email:** mu@gmail.com
Flight Id: 1234 **Flight name:** mug
On-boarding: Chennai **Destination:** Banglore
Passengers:
1. **Name:** mugesh, **Age:** 22

Booking date: 2025-12-08 **Journey date:** 2025-12-10
Journey Time: 08:00 **Total price:** 750
Booking status: cancelled

Booking ID: 6936a24e744248ac7ade9b61
Mobile: 9715748624 **Email:** mu@gmail.com
Flight Id: 1234 **Flight name:** mug
On-boarding: Chennai **Destination:** Banglore
Passengers:
1. **Name:** mugesh, **Age:** 22
2. **Name:** kavya, **Age:** 22

Booking date: 2025-12-08 **Journey date:** 2025-12-09
Journey Time: 08:00 **Total price:** 2000
Booking status: cancelled





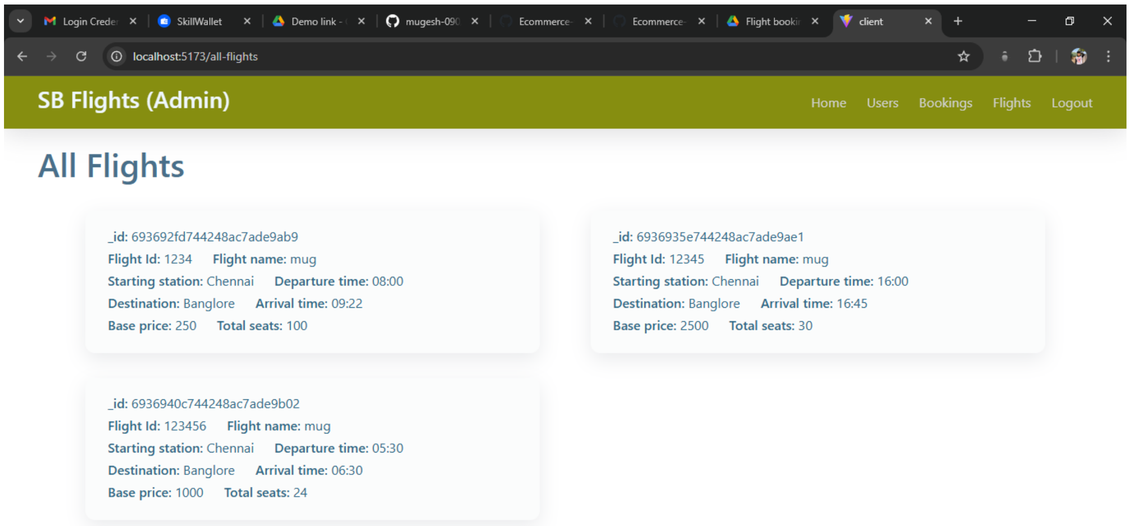
SB Flights (Admin)

All Users

UserId 693691db744248ac7ade9a65 **Username** muges **Email** mu@gmail.com

Flight Operators

Id 69369205744248ac7ade9a72 **Flight Name** mug **Email** mug@gmail.com



SB Flights (Admin)

All Flights

_id: 693692fd744248ac7ade9ab9
Flight Id: 1234 **Flight name:** mug
Starting station: Chennai **Departure time:** 08:00
Destination: Banglore **Arrival time:** 09:22
Base price: 250 **Total seats:** 100

_id: 6936935e744248ac7ade9ae1
Flight Id: 12345 **Flight name:** mug
Starting station: Chennai **Departure time:** 16:00
Destination: Banglore **Arrival time:** 16:45
Base price: 2500 **Total seats:** 30

_id: 6936940c744248ac7ade9b02
Flight Id: 123456 **Flight name:** mug
Starting station: Chennai **Departure time:** 05:30
Destination: Banglore **Arrival time:** 06:30
Base price: 1000 **Total seats:** 24

Known Issues:

- **Operator Approval:** Flight operator registration requires manual approval by an administrator, which can introduce delays in onboarding.
- **CORS Configuration:** Due to the separate frontend (e.g., port 3000) and backend (e.g., port 5000), Cross-Origin Resource Sharing (CORS) setup is critical and may require careful configuration.
- **Payment Gateway:** Initial implementation uses a sandbox environment; full production integration for secure payments is pending.

Future Enhancements:

Potential future features include:

- **GDS Integration:** Integrating with a live Global Distribution System (GDS) API for real-time flight data and availability, replacing static or dummy data.
- **Advanced Seat Selection:** Implementing a more granular and interactive seat selection map, adhering to complex airline seating rules.
- **Email Notifications:** Implementing transactional email services for booking confirmations, cancellations, and flight status updates.