Full Stack Development with MERN Project Documentation format

## Introduction

* + **Project Title:** Flight booking
  + **Team Members:**

Nivetha V- 413021104024 (Team leader)

Harini B- 413021104010

Jencybhel R- 413021104013

Yogeshwaran V- 413021104047

## Project Overview

* **Purpose:** The purpose of the flight booking application is to provide users with a

convenient, efficient, and user-friendly platform to search for, compare, and book flights. It aims to simplify the travel planning process by offering real-time flight availability, pricing, and customization options to meet diverse customer needs.

## Goals:

### Enhanced User Experience:

* + Deliver a seamless and intuitive interface for searching and booking flights.
  + Minimize the time and effort required for flight selection and payment.

### Cost Efficiency and Transparency:

* + Provide transparent pricing, including all taxes and fees.
  + Offer promotional deals, discounts, and loyalty program integration.

### Accessibility and Convenience:

* + Ensure the platform is accessible via web and mobile devices.
  + Support multiple languages and currencies for a global audience.

### Secure Transactions:

* + Implement secure payment gateways for a variety of payment methods.
  + Protect user data with robust security protocols.

### Customer Support:

* + Provide 24/7 assistance for queries, changes, or cancellations.
  + Offer real-time updates on flight statuses and booking confirmations.

### Features:

1. **Search Flights**: Users can search for one-way, round-trip, or multi-city flights by entering their origin, destination, travel dates, and passenger details.
2. **Advanced Filters**: Filter results by:
   * Price range
   * Airlines
   * Departure and arrival times
   * Layovers and direct flights
   * Class (economy, premium economy, business, first-class)

## Architecture

A robust and scalable architecture for a flight booking application must efficiently handle complex operations like real-time search, booking, secure transactions, and integrations with external systems. Below is an overview of a layered architecture:

## Front-End

This layer interacts directly with the users.

* **Web Application**: Built using frameworks like React, Angular, or Vue.js for a responsive and interactive user interface.

#### Features:

* + - User-friendly interface for flight search, booking, and payment.
    - Localization for multi-language and multi-currency support.
    - Notifications for booking updates, flight status, and offers.

## Back-End

The core business logic and functionality reside in this layer.

#### Microservices:

* 1. **Flight Search Service**: Manages search requests and retrieves flight data from external APIs or databases.
  2. **Booking Service**: Handles seat selection, reservation, and payment integration.
  3. **User Management Service**: Manages user profiles, preferences, and travel history.
  4. **Notifications Service**: Sends emails, SMS, and push notifications.

#### Real-Time Services:

* 1. Manages live updates on seat availability, prices, and flight status.
  2. Technologies like WebSockets or Server-Sent Events (SSE).

1. **Database :**
   * **Relational Database (RDBMS)**: For structured data like user profiles, bookings, and payment records. (e.g., MySQL, PostgreSQL)
   * **NoSQL Database**: For unstructured data like search logs and real-time analytics. (e.g., MongoDB, DynamoDB)

## Diagram Overview:

#### Clients:

* + Users interact via web or mobile applications.

#### API Gateway:

* + Directs requests to respective microservices.

#### Databases:

* + Stores structured and unstructured data.

#### External Integrations:

* + Communicates with GDS, payment gateways, and third-party APIs.

#### Infrastructure:

* + Deployed on a cloud platform with monitoring and scaling capabilities.

This architecture ensures modularity, scalability, and fault tolerance, making it ideal for a flight booking application.

## Setup Instructions

* 1. **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
  2. **Installation of MongoDB**: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.
  3. **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.

* 1. **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.

## 5 Folder Structure :

Static files like index.html, favicon, and manifest files.

# Src:

The primary source folder containing all application logic and resources.

# Assets :

1. **Purpose**: Store static resources.

#### Subfolders:

* + /images: Icons, logos, or general images.
  + /icons: SVGs or other icons.
  + /styles: Global CSS, SCSS, or theme file

# Components:

1. **Purpose**: Reusable UI components, further subdivided by type or functionality.

#### Subfolders:

* + /common: Generic components like buttons, modals, dropdowns.
  + /layout: Navigation bars, footers, and header components.
  + /search: Components for flight search and filters.
  + /booking: Components for seat selection, summary, and booking details.
  + /user: User profile and account-related components.
  + /notifications: Toasts, alerts, or messages.

# Features :

1. **Purpose**: Modular organization of app features.

#### Subfolders:

* + /auth: Login, signup, and authentication logic.
  + /flightSearch: Components and logic for flight search and filters.
  + /flightDetails: Handles flight details and itineraries.
  + /payment: Payment gateway integration and confirmation.
  + /profile: User profile management.

#### Pages :

1. **Purpose**: Full-page views rendered via routing.

#### Subfolders:

* + /home: Landing page.
  + /searchResults: Results displayed after flight search.
  + /booking: Flight booking page.
  + /profile: User profile and history.
  + /error: Custom error pages like 404 or server errors.

1. **Running the Application**
   * Provide commands to start the frontend and backend servers locally.
     + **Frontend:** npm start in the client directory.
     + **Backend:** npm start in the server directory.

## API Documentation

This document provides an overview of the endpoints exposed by the backend. Each endpoint includes the HTTP method, parameters, and example responses.

{

"name": "John Doe",

"email": "[johndoe@example.com](mailto:johndoe@example.com)", "password": "securepassword"

}

#### Response:

{

"message": "Signup successful", "userId": "12345"

}

## Authentication

Authentication verifies the identity of users. In this project, it is implemented using **JSON Web Tokens (JWT)**.

1. **User Signup**:
   * Endpoint: /api/auth/signup
   * Users provide their details (e.g., name, email, and password).
   * Passwords are securely hashed using a library like **bcrypt** and stored in the database.
   * After successful registration, the user is notified (but no token is issued yet).

### User Login:

* + Endpoint: /api/auth/login
  + Users provide their credentials (email and password).
  + The password is verified against the hashed version stored in the database.
  + Upon successful authentication:
    - A **JWT** is generated and returned to the user.
    - This token contains the user’s ID, email, and other claims (e.g., roles) in its payload.
    - The token is signed using a secret key to ensure its integrity.

### Token Generation:

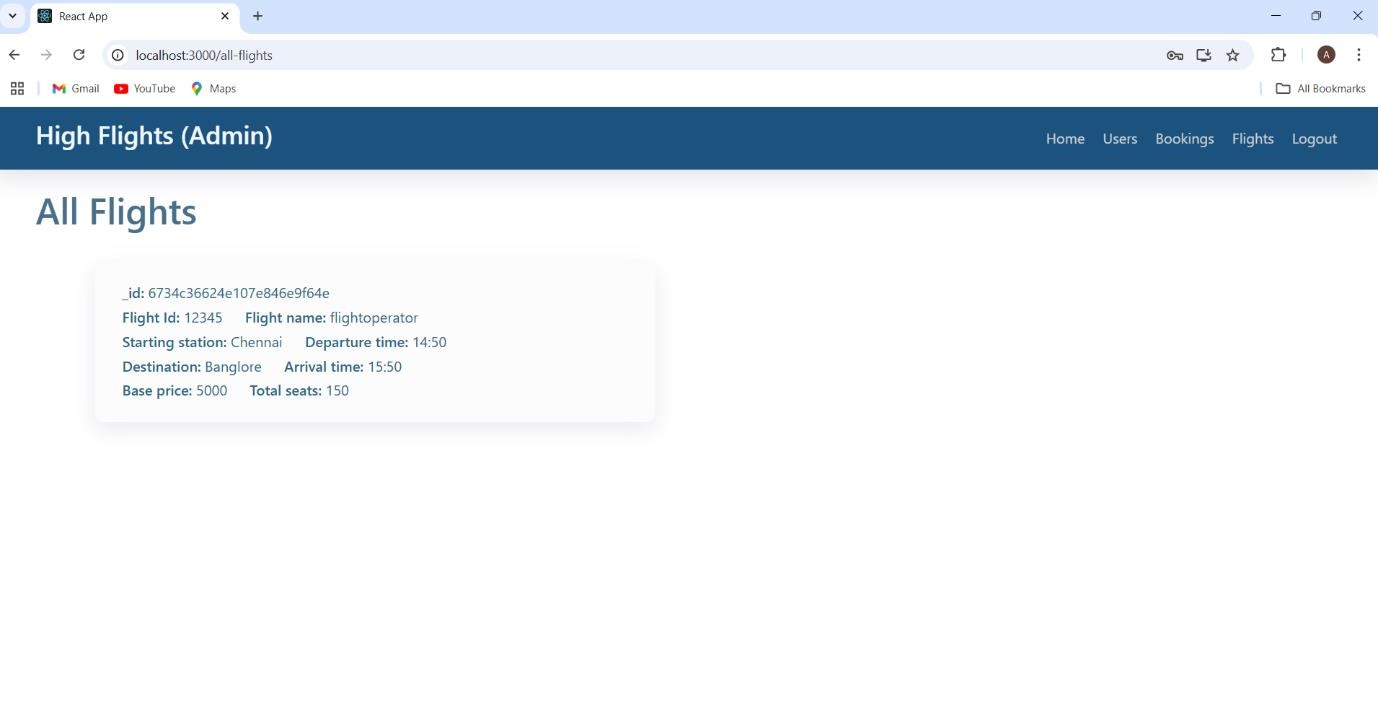
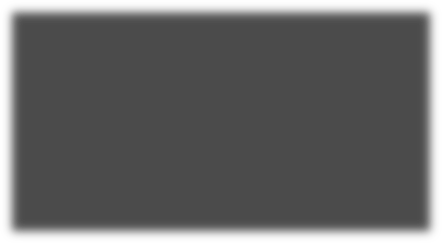
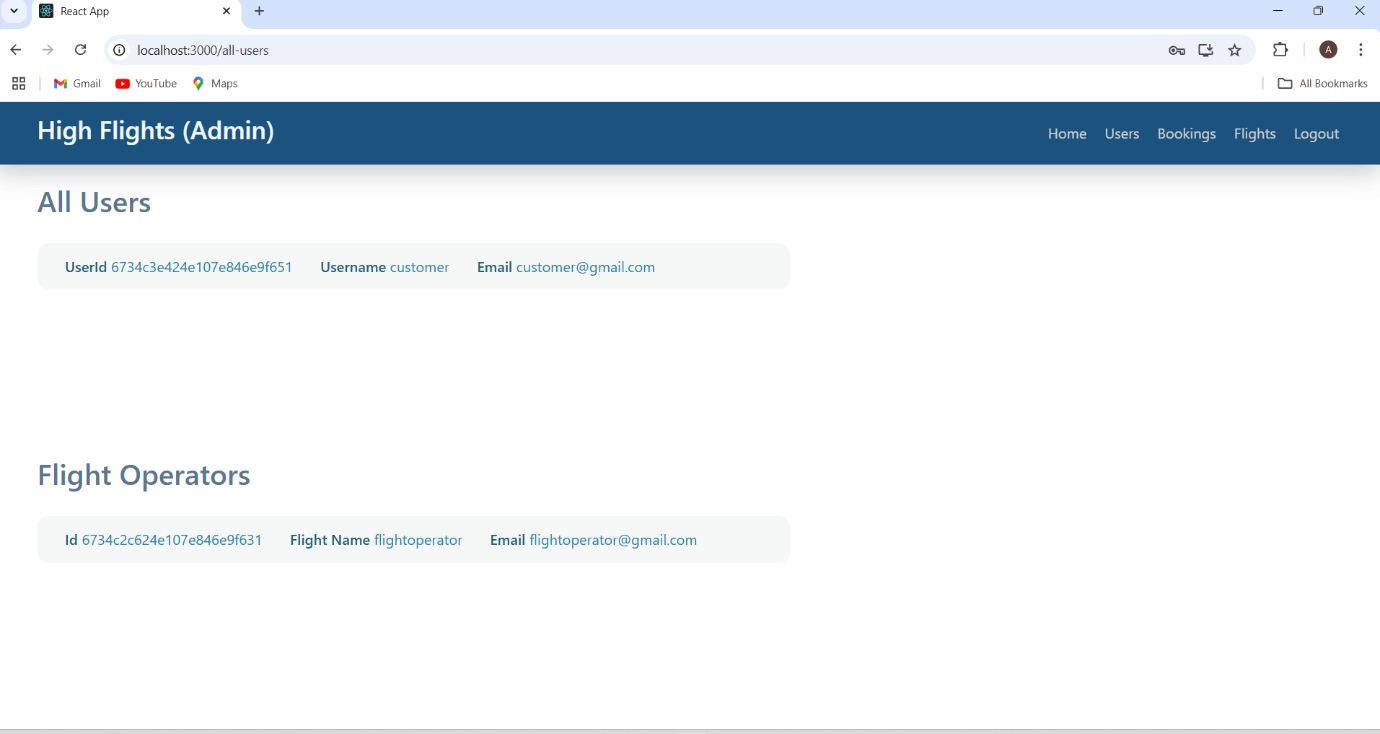
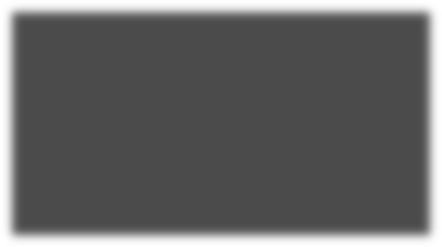
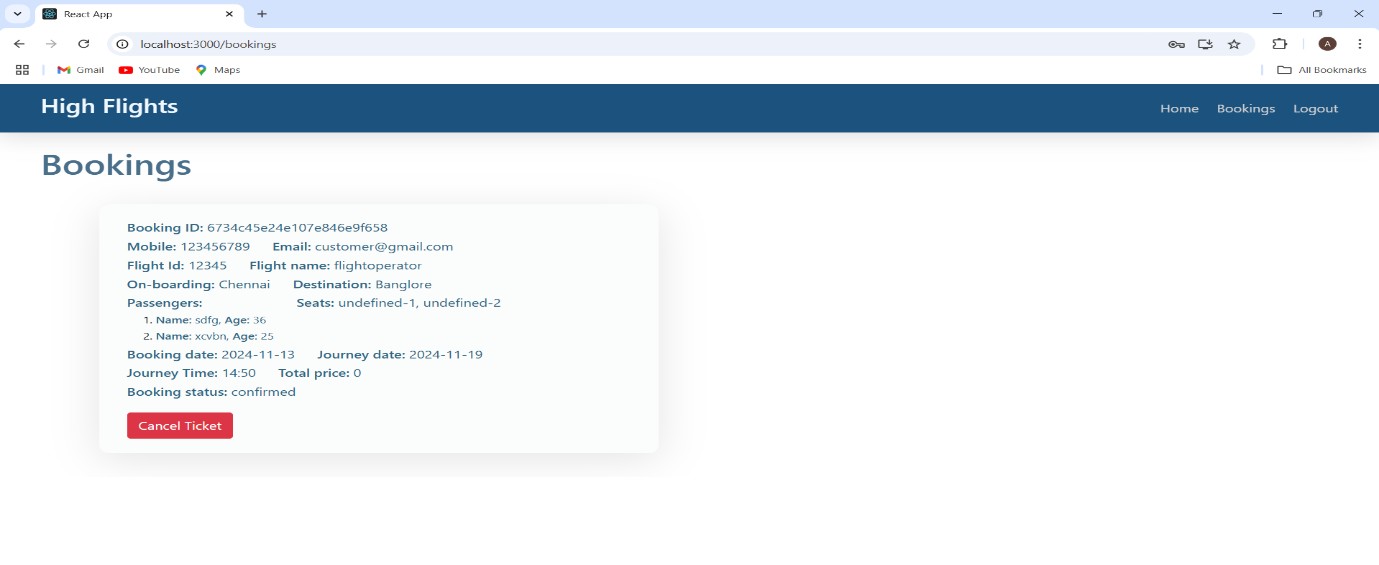
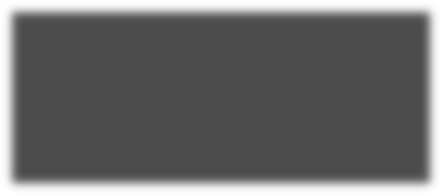
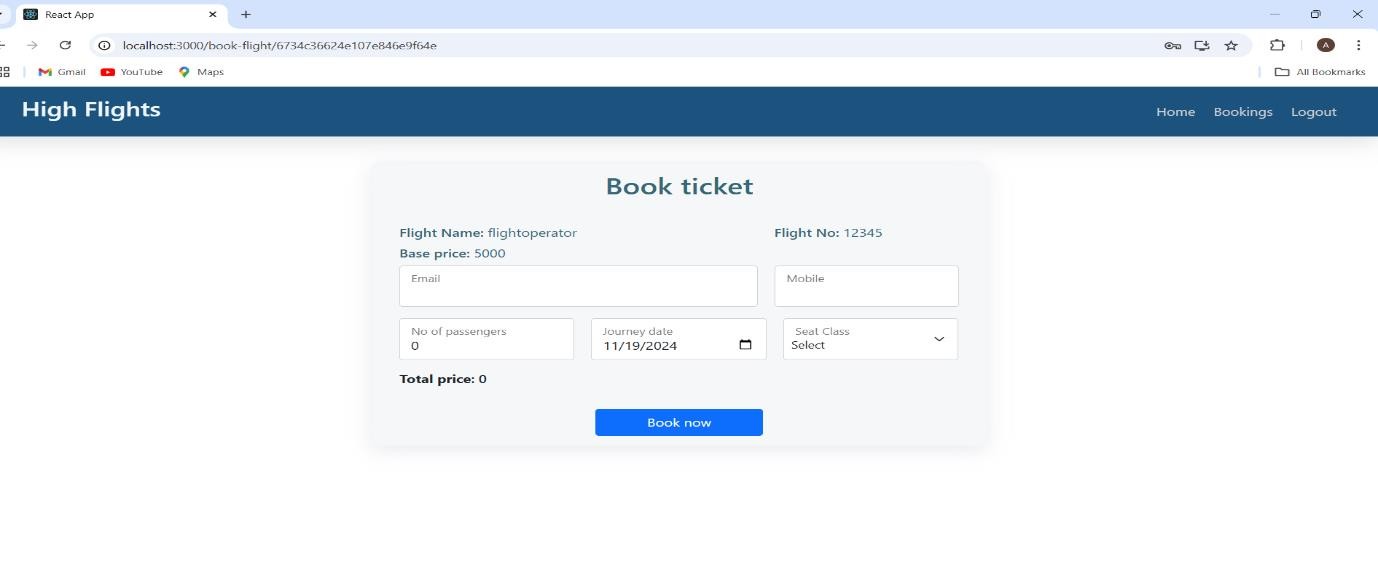
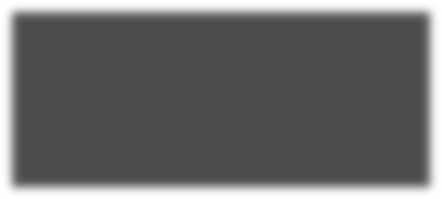
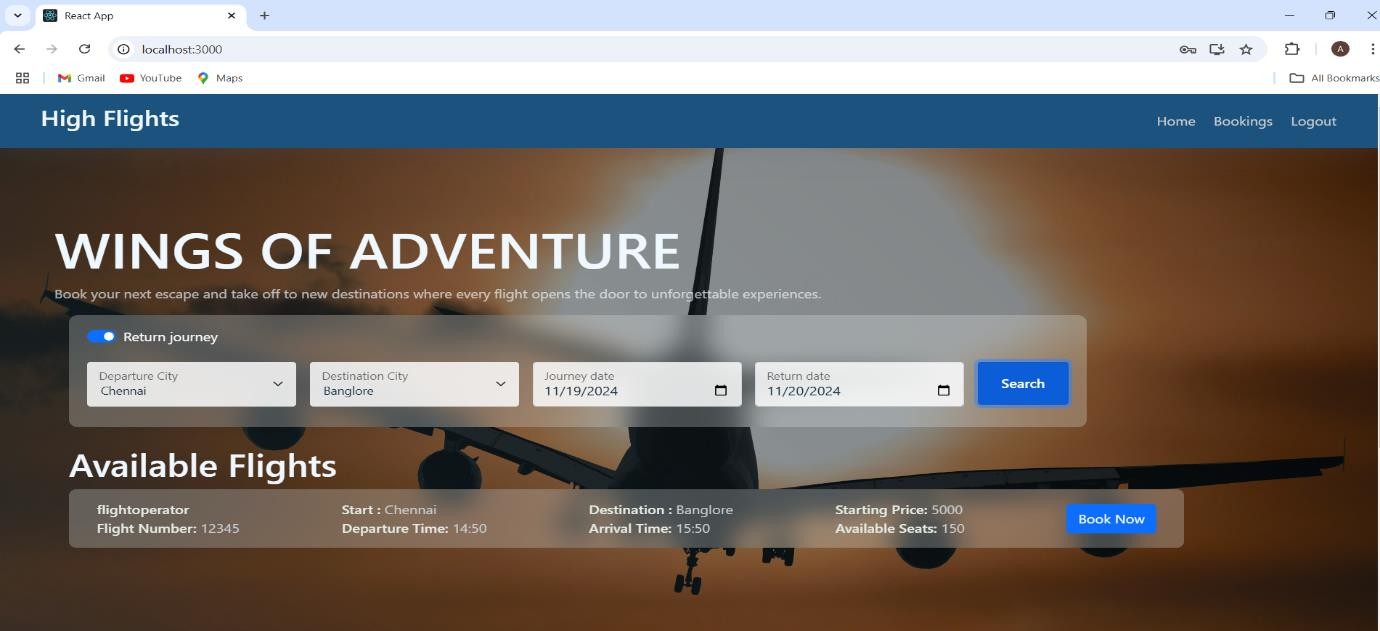
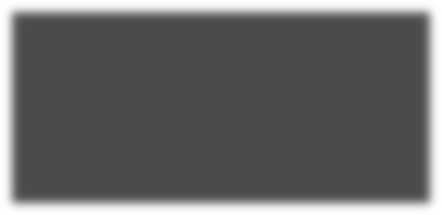
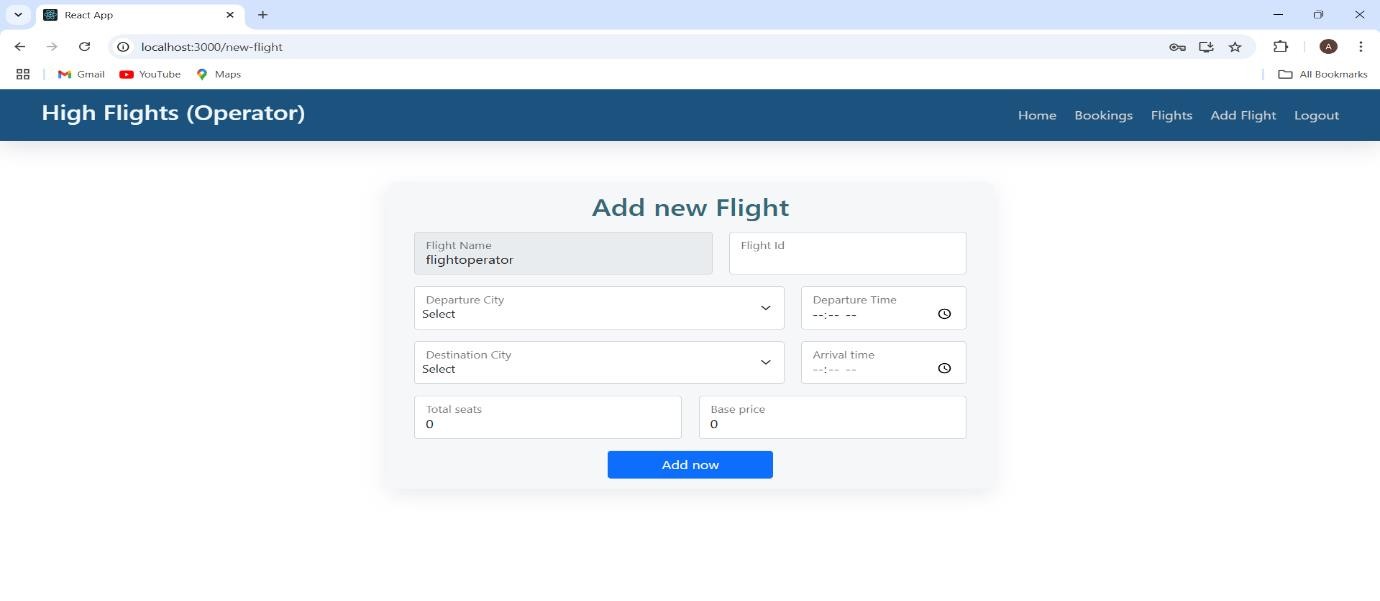
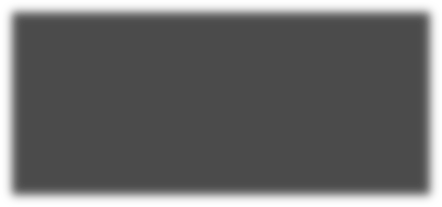
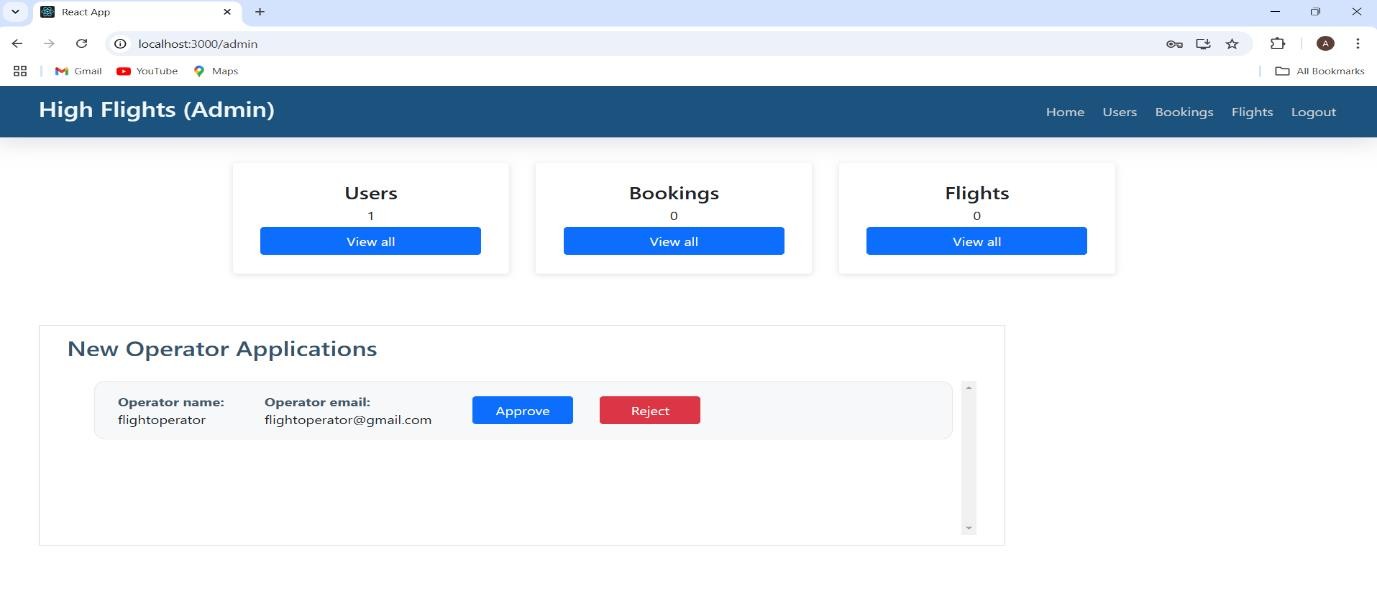
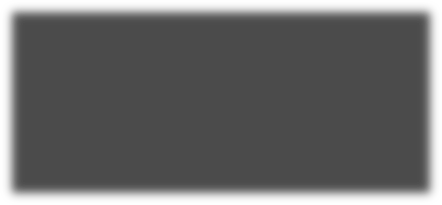
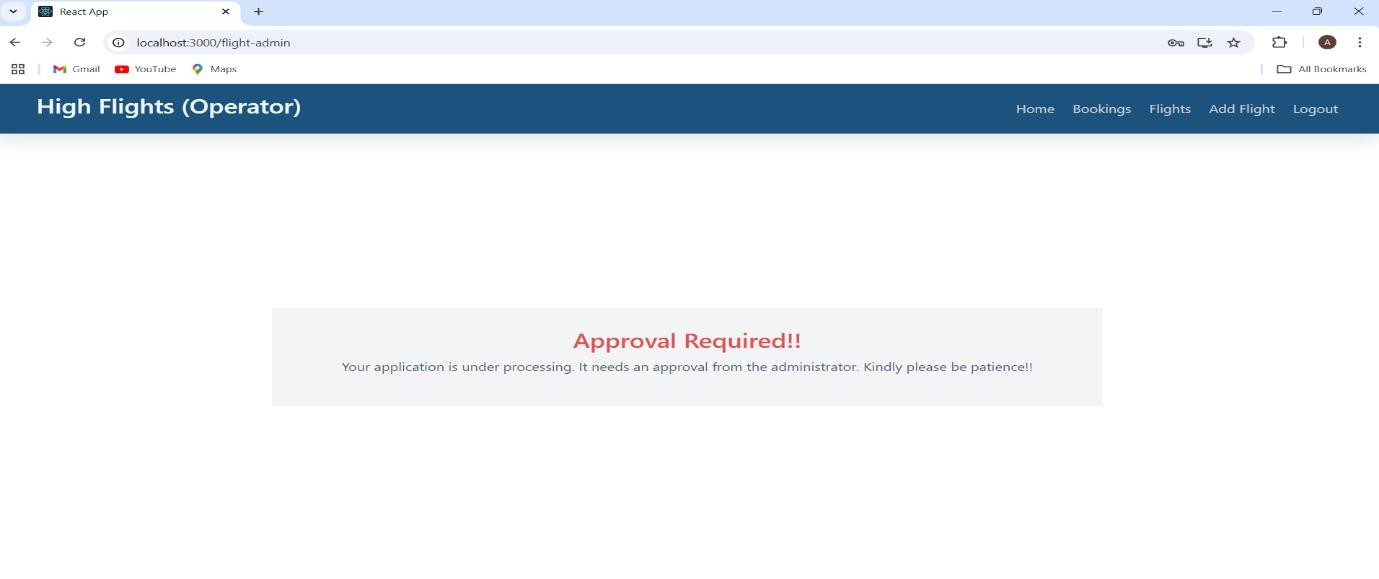
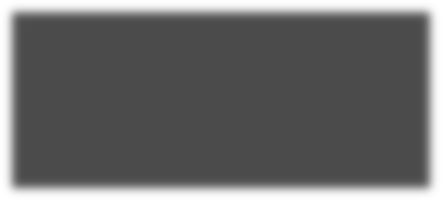
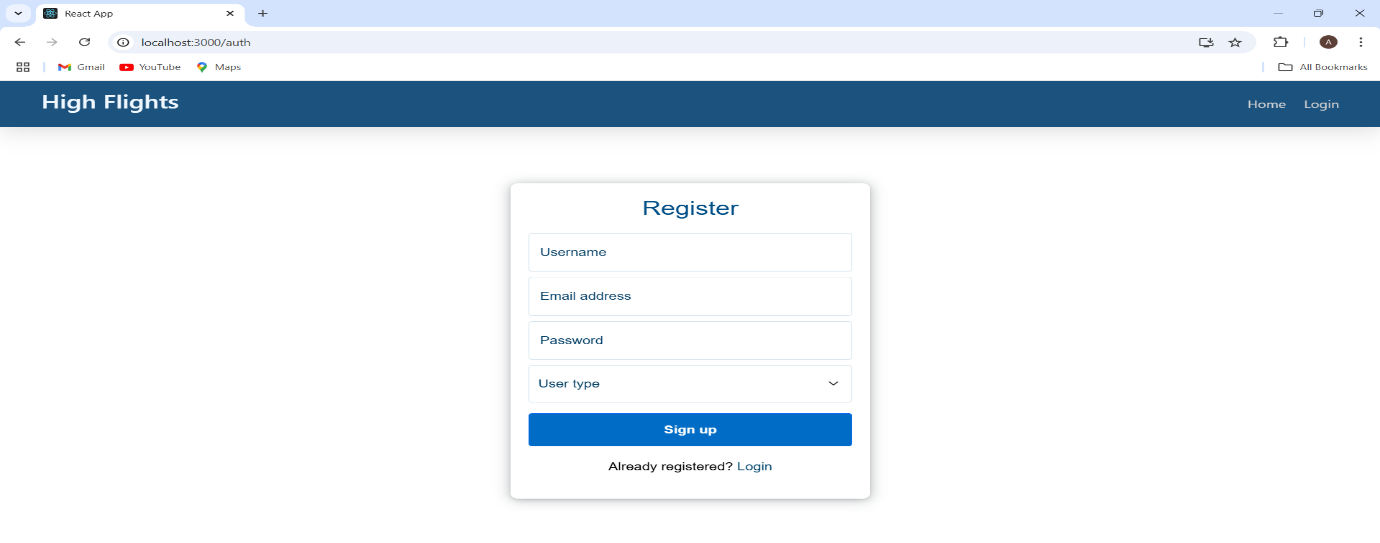
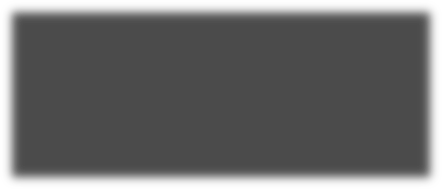
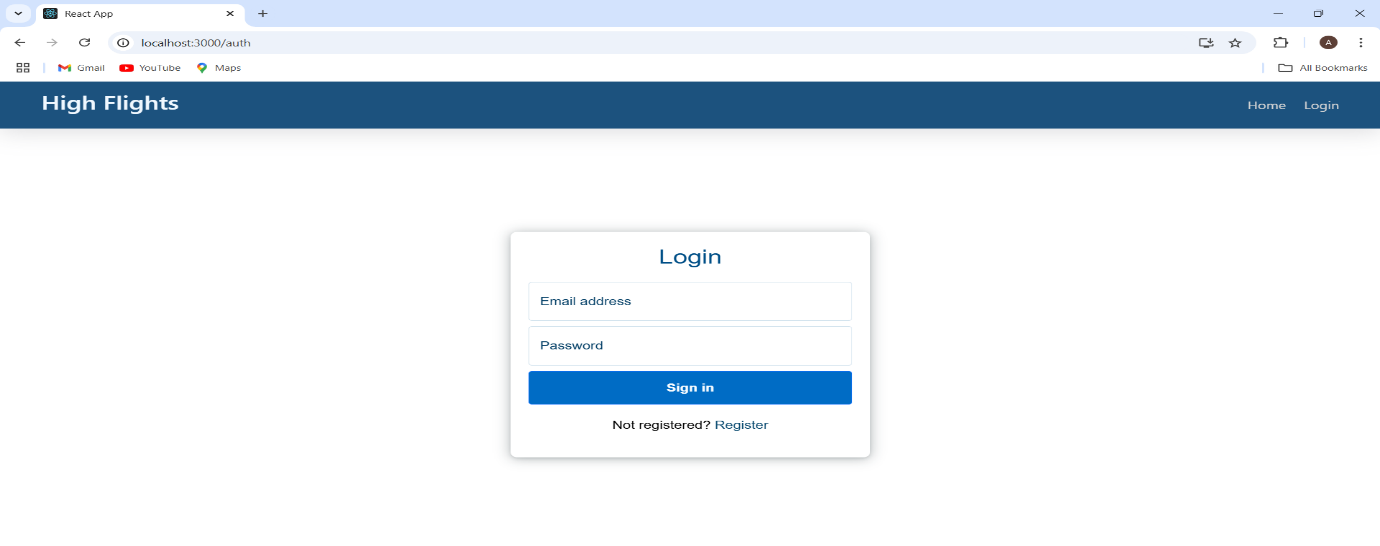
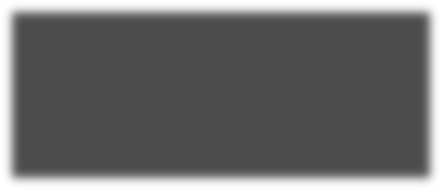
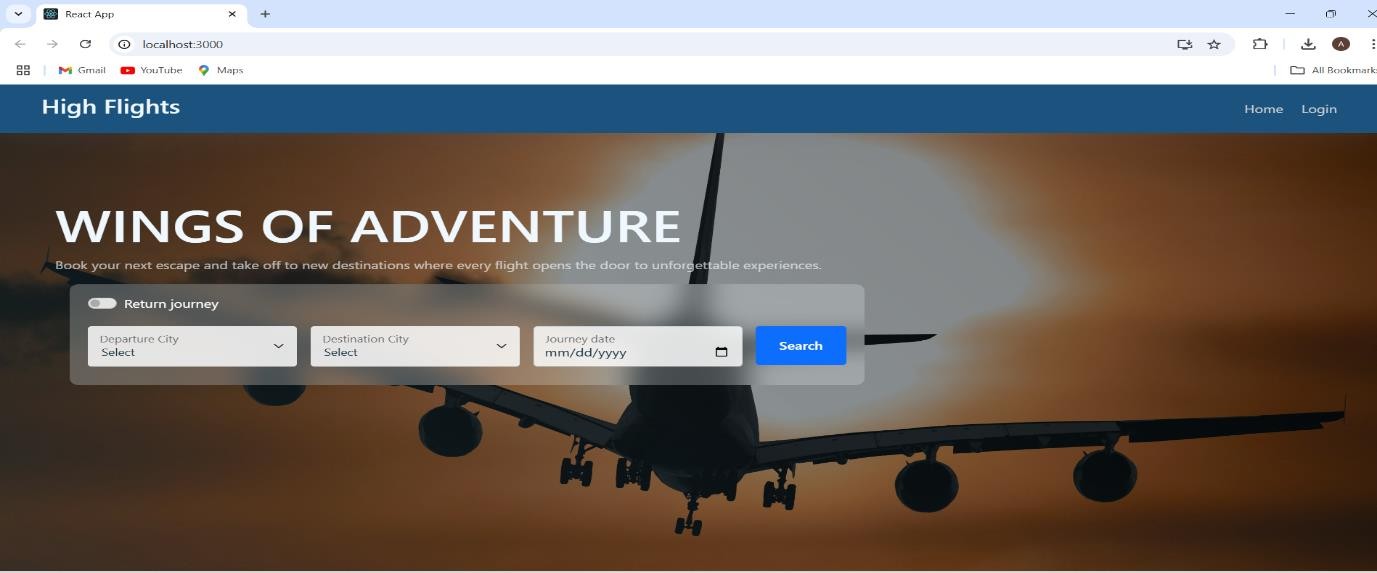
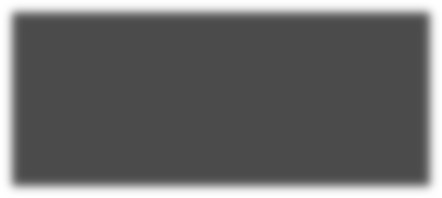
* + The JWT is generated using libraries like **jsonwebtoken**.
  + Claims include:
    - sub (subject): User ID.
    - iat (issued at): Timestamp of token issuance.
    - exp (expiration): Token expiry time .

### Storing the Token:

* + The client stores the JWT in **localStorage** or **cookies** (with HttpOnly for security).
  + For mobile apps, secure storage mechanisms like **Keychain** (iOS) or **Keystore**

(Android) are used

## User Interface:



1. **Testing**
   * Manual Testing

## Screenshots or Demo

* + Screenshot and demo video link:

https://github.com/ezhilmahi/NM2024TMID00136\_ezhilarasi.git

## Known Issues

* + Initially, if we book a certain number of seats, it will be treated as an unlimited number of seats.
  + The page is not fully responsive on some mobile devices, causing some form fields to overlap or appear off-screen.

## Future Enhancements

* + Expand the application to support multiple currencies and languages to accommodate international users.
  + Expand the app to allow users to book hotels and car rentals along with their flights in one seamless transaction.