FULL STACK DEVELOPMENT WITH MERN

Project Report

FLIGHT BOOKING APP

Project overview:

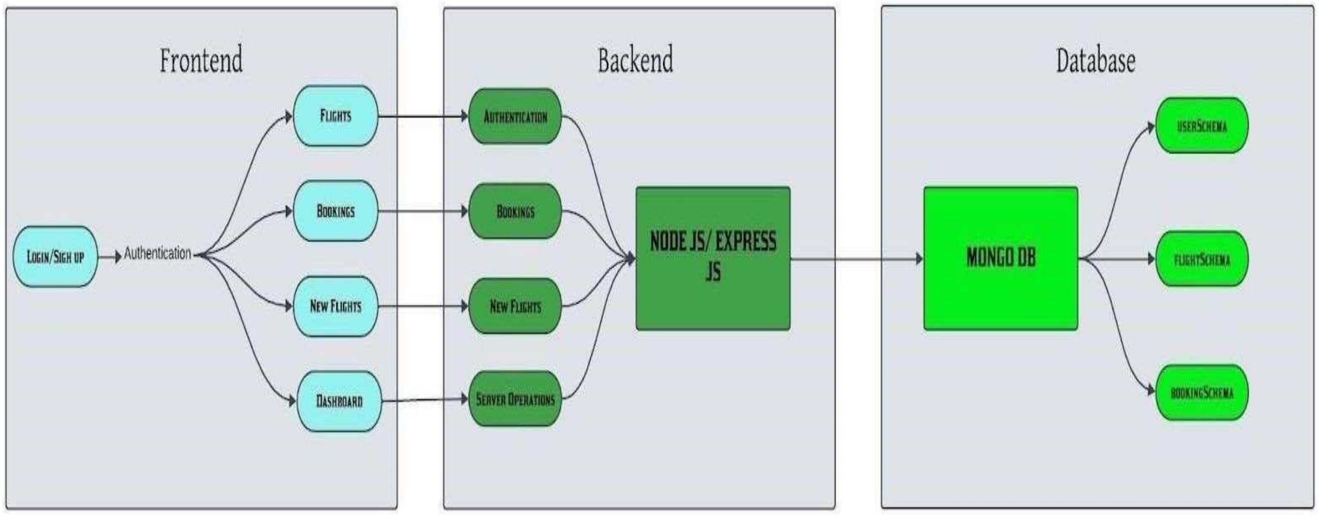
Purpose:

The flight booking app provides users with a seamless, efficient, and user-friendly platform to search, compare, book, and manage flights. It simplifies the flight booking process by offering a wide range of features and integrations for a hassle-free travel experience. The app ensures accurate and up-to-date information, secures transactions, and provides personalized recommendations based on user preferences. Additionally, it enhances the user experience with real-time notifications, robust customer support, making it an indispensable tool for modern travelers. The app also includes secure user authentication to protect personal data, ensuring only authorized users access their bookings. Additionally, an admin panel provides tools for managing flights and overseeing bookings, giving administrators control over available inventory. The app's ultimate goal is to offer a hassle-free, all-in-one flight booking experience for travelers.

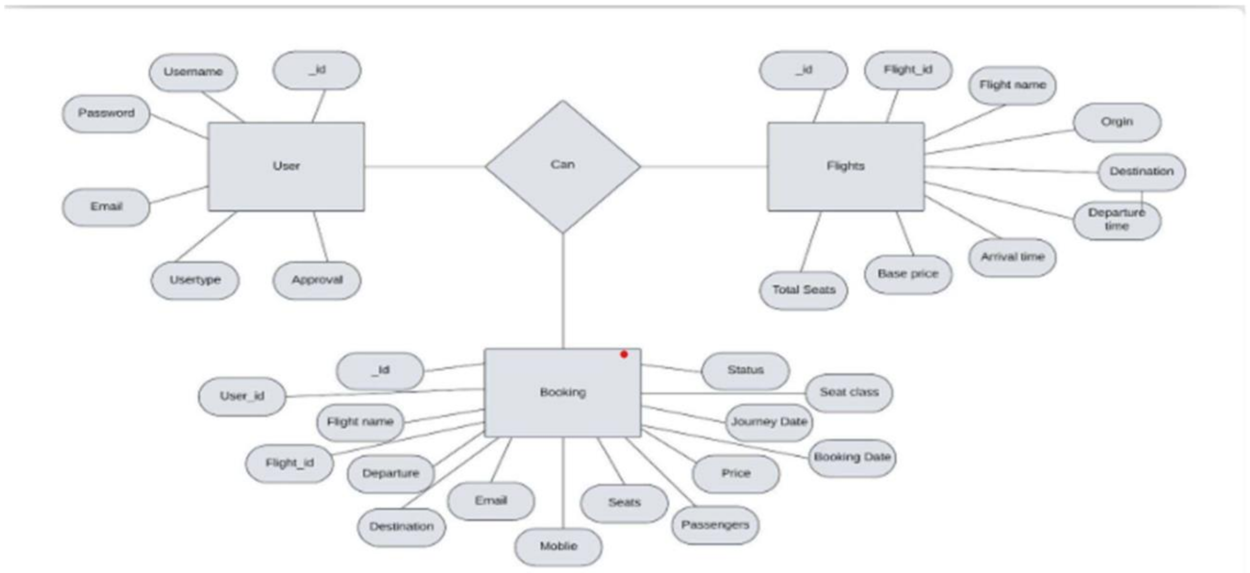
Features:

- Flight Search: Users can search for available flights by specifying criteria like origin, destination, and travel dates.
- Booking Management: After booking, users can view, update, or cancel their reservations.
- User Authentication: Secure login and registration for new users, as well as role-based access for admins.
- Admin Panel: Admins can add, update, and delete flights and manage bookings.
- Payment Integration: Secure payment processing for flight bookings.

Architecture:


Frontend
Backend
Database
Login/Sign up
Authentication
Flights
Bookings
New Flights
Dashboard
Authentication
Bookings
New Flights
Server Operations
NODE JS/ EXPRESS JS
MONGO DB
userSchema
flightSchema
bookingSchema

ER Diagram


User
Username
_id
Password
Email
Usertype
Approval
Can
Flights
_id
Flight_id
Flight name
Orgin
Destination
Departure time
Arrival time
Base price
Total Seats
Booking
_Id
User_id
Flight name
Flight_id
Departure
Destination
Email
Moblie
Seats
Passengers
Price
Booking Date
Journey Date
Seat class
Status

Frontend Components

React.js:

- Component-Based Architecture: React allows the development of interactive UIs by breaking down the UI into reusable components. This makes the code more modular and easier to maintain.
- Virtual DOM: React uses a virtual DOM to optimize rendering performance. It updates only the parts of the UI that have changed, making the app faster and more efficient.

Redux:

- State Management: Redux helps manage the state of the application in a predictable way. It centralizes the application's state, allowing for better debugging and tracing of state changes.
- Middleware Integration: Redux supports middleware like Redux Thunk or Redux Saga, which allows handling of asynchronous actions, making it easier to manage complex state logic.

Backend

Node.js:

- Event-Driven Architecture: Node.js uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient. It's well-suited for building scalable applications.
- Large Ecosystem: Node.js has a vast ecosystem of libraries and frameworks available through npm, which speeds up development by providing pre-built modules.

Express.js:

- Routing: Express provides a robust set of features to manage routing, allowing the development of complex web applications and APIs.
- Middleware: Express supports middleware functions to handle requests, responses, and error handling, making it highly extensible and customizable.

Database

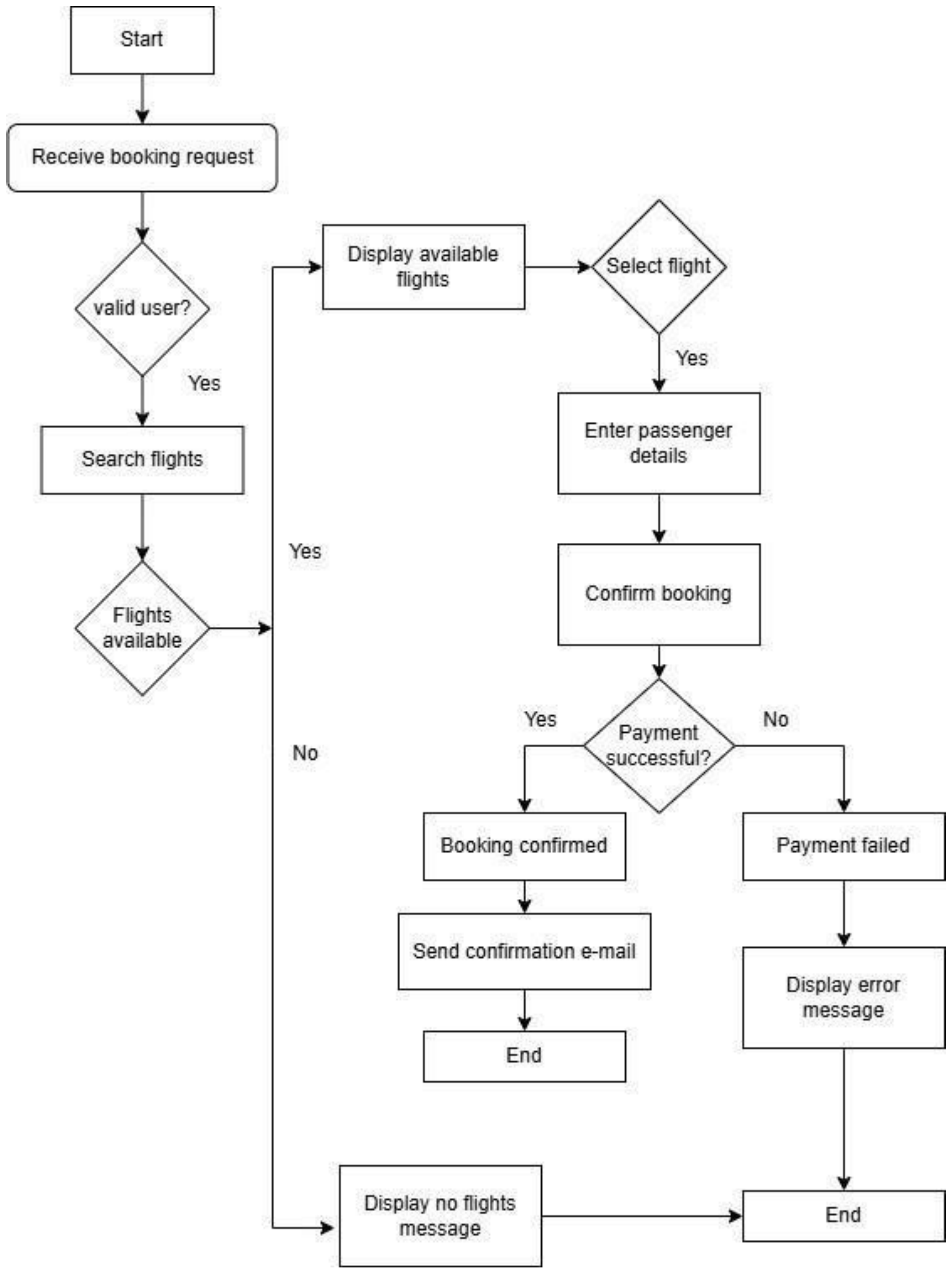
MongoDB:

- Document-Oriented: MongoDB stores data in flexible, JSON-like documents, which allows for schema flexibility. This is especially useful for applications where the data structure can evolve over time.
- Scalability: MongoDB is designed to scale horizontally by sharding, which distributes data across multiple servers, ensuring high availability and load balancing.

Mongoose:

- Schema-Based Data Modeling: Mongoose provides a schema-based solution to model application data. It enforces schemas at the application layer, providing structure and validation.
- Middleware Hooks: Mongoose supports middleware hooks for performing actions before and after database operations, which helps in maintaining data integrity and implementing business logic.

Flow Chart:


Start
Receive booking request
valid user?
Yes
Search flights
Flights available
Yes
No
Display available flights
Select flight
Yes
Enter passenger details
Confirm booking
Payment successful?
Yes
No
Booking confirmed
Send confirmation e-mail
End
Payment failed
Display error message
Display no flights message
End

Prerequisites:

Node.js and npm:

- Server Environment: Node.jsprovides the runtime environment needed to run JavaScript on the server side. npm, the Node package manager, is used to install and manage dependencies.
- Cross-Platform Support: Node.jsand npm work across different operating systems, including Windows, macOS, and Linux, making them versatile for development environments.

MongoDB:

- Database Server: MongoDB must be installed and running as a database server. It handles the data storage, retrieval, and management for the application.
- Configuration: Proper configuration of MongoDB, including setting up access controls and backups, is essential for ensuring data security and integrity.
- Environment Variables: Create a .env file in the root directory to store sensitive information such as:

    DATABASE_URL: MongoDB connection URL

    JWT_SECRET: Secret key for JSON Web Token generation

    API_KEY: Required for third-party services (e.g., payment

    gateway)

Installation:

- Clone the Repository: Open a terminal and clone the repository to your local machine using the following command: git clone https://github.com/harsha-vardhan-reddy-07/Flight-Booking-App-MERN.git

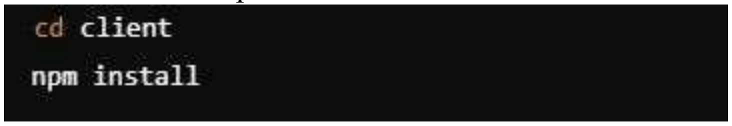
- Navigate to the project's root directory:

  cd Flight-Booking-App-MERN Install

  root-level dependencies:

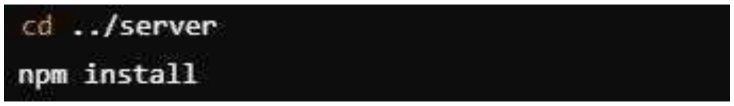
  npm install

- Install frontend dependencies:



```
cd client
npm install
```

- Install backend dependencies:



```
cd ../server
npm install
```

- Configure Environment Variable:

Ensure the .env file is set up in the root directory with the necessary secrets and configurations. This file should be in the same directory as your package.json files for the backend. Do not include sensitive information in version control.

Database Setup

Start MongoDB:

- Local Setup: Ensure MongoDB is running locally on your system. This can typically be done using the mongod command.
- MongoDB Atlas: If you're using MongoDB Atlas, make sure your connection string is correctly configured in the .env file. This will usually look something like mongodb+srv://username:password@cluster.mongodb.net/databaseName.

Create Database:

- New Database: If required, create a new database for your application. This can be done through MongoDB Atlas or using MongoDB Shell commands.
- DATABASE_URL Configuration: Ensure the DATABASE_URL in your .env file is updated to reflect the connection string for your new database.

Seed Database:

- Initial Data: Optionally, you can seed your database with initial flight data. If there's a provided script (e.g., npm run seed), execute this command to populate your database with the necessary data for testing and development.

Frontend and Backend Configuration Backend
Server Configuration:

- Port Setup: Ensure the backend server is configured to listen on a specified port by setting the PORT variable in your .env file (e.g., PORT=5000).

- API Endpoints: Define the base URL for the backend server in the frontend configuration file (e.g., client/src/config.js) to ensure that API requests are routed correctly.

Frontend API Configuration:

- Update API URLs: Make sure any API endpoint URLs in the frontend application are updated to match the backend server's base URL. This ensures seamless communication between the frontend and backend.

Folder Structure

Client src/components:

- Flight Card: A reusable component to display individual flight details.
- Booking Summary: Summarizes the booking details before final confirmation. Login Form: Handles user authentication and login functionality.

src/pages:

- Home: The main landing page of the application, featuring search and quick access to major functionalities.
- Flights: Displays available flights based on user search criteria.
- MyBookings: Allows users to view and manage their bookings.
- AdminDashboard: For administrative tasks such as managing flights and users.

src/services:

- API Service Functions: Functions to interact with the backend APIs. These typically include methods for fetching, creating, updating, and deleting resources.

Server

- API Routes: Define endpoints like /api/flights, /api/bookings, and /api/users. Each route specifies the HTTP methods (GET, POST, PUT, DELETE) supported and the corresponding controller functions.

server/controllers:

- Business Logic: Contains logic for handling requests. For example, creating bookings, fetching flight details, updating user information, etc.

server/models:

- Mongoose Schemas: Define the structure of the MongoDB documents. For example, the Flight schema might include fields like flight number, departure time, arrival time, and price.

Running the Application:
API Documentation:

- Flight Search Endpoint:

- URL: /api/flights

- Method:

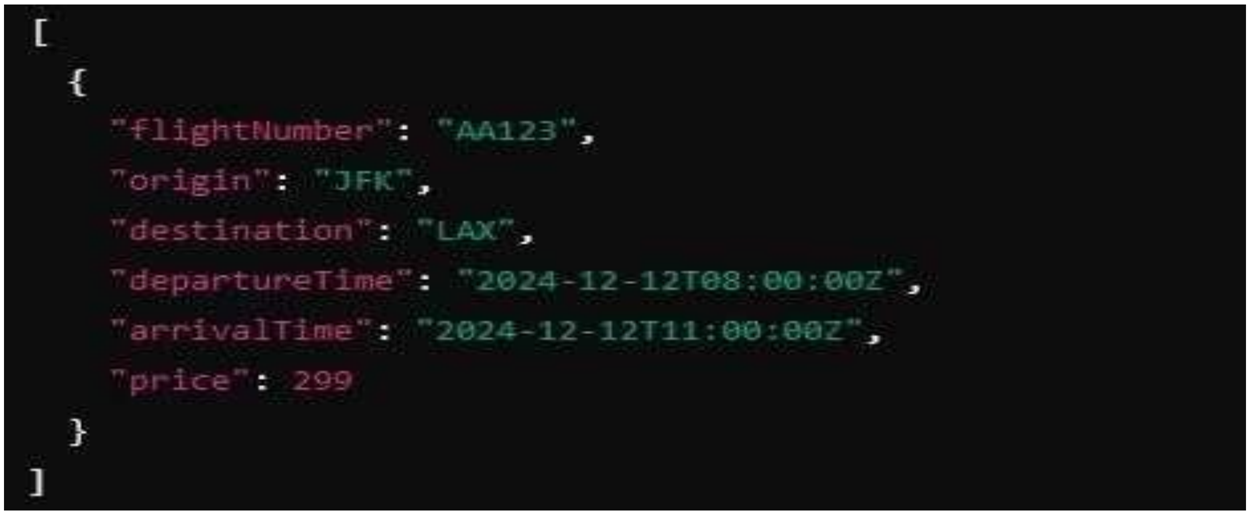
1. GET Description: Retrieves available flights based on search criteria.

2. Parameters:

   origin (required): String, airport code of departure location.

   destination (required): String, airport code of arrival location. o date (optional): Date string for the travel date.

Example Response:



```
[
  {
    "flightNumber": "AA123",
    "origin": "JFK",
    "destination": "LAX",
    "departureTime": "2024-12-12T08:00:00Z",
    "arrivalTime": "2024-12-12T11:00:00Z",
    "price": 299
  }
]
```

JWT Authentication

Working:

User Login:

- When a user logs in with valid credentials, the server generates a JWT token.The login endpoint verifies the user’s credentials against the database.

Token Generation:

- The JWT contains the user's unique identifier (e.g., userID) and optionally, role information (admin or user).
- The token is signed with a secret key stored on the server (specified in the JWT_SECRET environment variable).

Token Structure:

- Header: Contains the type of token (JWT) and the signing algorithm (e.g., HS256).

- Payload: Contains claims, which are statements about an entity (typically, the user) and additional data. It includes:

1. sub: Subject of the token (userID).
2. iat: Issued at time.
3. exp: Expiration time.

 Signature: The header and payload are encoded and signed with a secret key to create the signature.

Token Storage:

- The generated JWT token is sent to the client, which stores it in localStorage or cookies for future requests.

Token Usage:

- On subsequent requests to protected routes (e.g., viewing bookings, updating profile information), the client includes the JWT in the HTTP headers (usually in the Authorization header as Bearer <token>).

Token Verification:

- The server verifies the token's validity on each request. This involves checking the signature and ensuring the token has not expired.

Token Expiration:

- JWTs include an expiration time to enhance security. Once the token expires, the user must log in again to receive a new token, reducing the risk of unauthorized access.

Authorization (Role-Based Access Control) Roles Defined:

User:

- Regular users who can search for flights, book tickets, and manage their bookings.

Admin:

- Users with special privileges, including managing flights, updating flight details, and viewing all user bookings.

Working:

Role Assignment:

- Each user is assigned a role upon registration or by an admin. The role information is stored in the user's JWT token payload.

Access Control Middleware:

- Middleware functions check user roles. For example:
  1. authMiddleware: Ensures the request contains a valid JWT.
  2. adminMiddleware: Verifies if the user has an admin role.

Protected Routes

User Routes:
- Accessible to authenticated users. A user with a valid JWT can view and update their bookings.

Admin Routes:
- Restricted to admins. Only users with an admin role can access these endpoints. If a non-admin user tries to access these routes, the server returns a "Forbidden" error (HTTP status code 403).

Flow Example
User Booking a Flight:

- When a user tries to book a flight, the server checks for a valid JWT token to ensure the user is authenticated. If valid, the booking process proceeds.

Admin Managing Flights:

- When an admin attempts to add or modify flight data, the server checks both the validity of the JWT token and the role encoded within it. If the token is valid and the user's role is admin, the action is permitted.

Security Measures
Token Expiration and Refresh:

- To enhance security, JWT tokens have an expiration time. This limits the validity of the token, ensuring users must reauthenticate after a period.

- Refresh tokens can be implemented to issue new tokens without requiring the user to log in again, ensuring a smoother user experience.

HTTPS:

- For additional security, it is recommended to run the app over HTTPS to prevent token interception during transmission.

Revoking Tokens:

- If a user logs out, their token is considered invalid. Implementing a token blacklist or token versioning can further enhance security by invalidating tokens when needed.

Secure Storage:

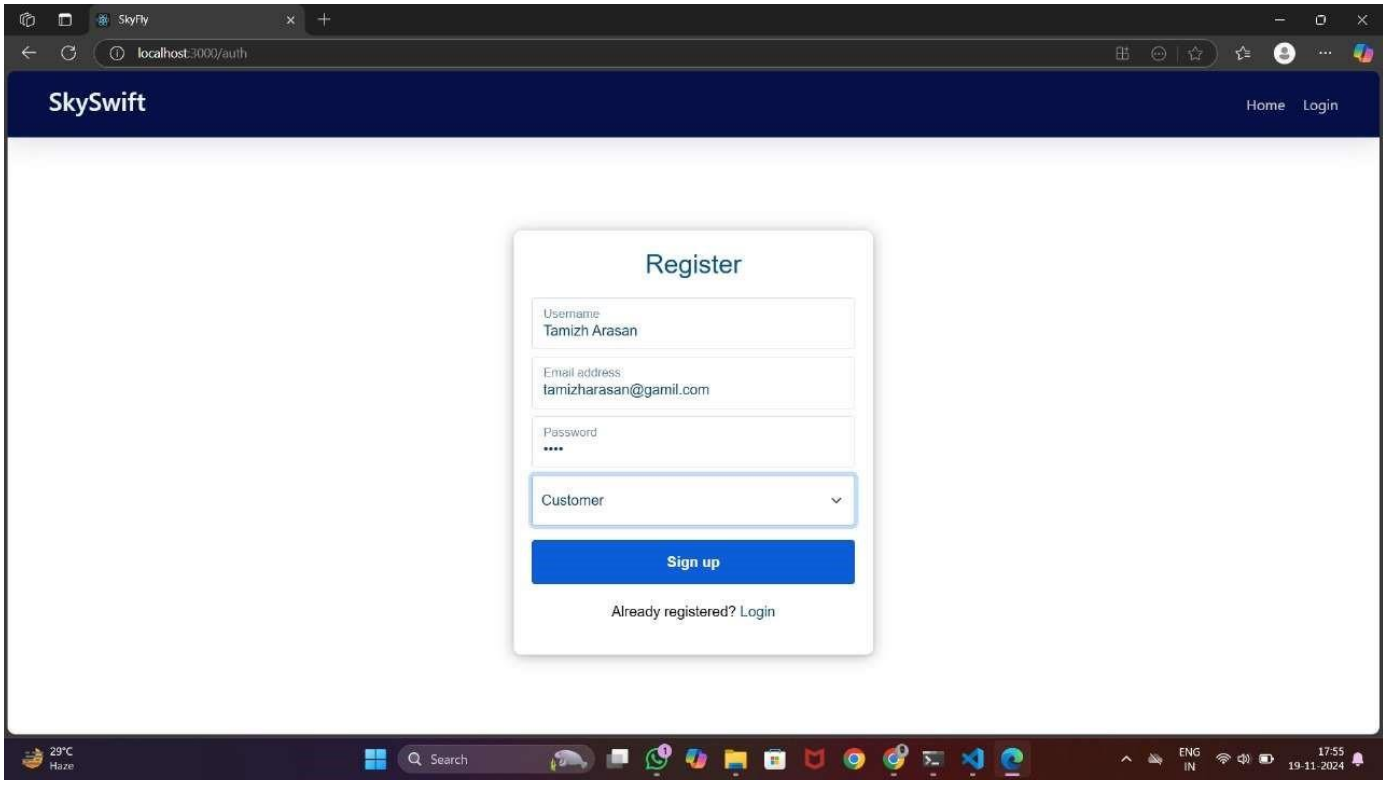
- Store tokens in secure, HttpOnly cookies to protect them from XSS (Cross-Site Scripting) attacks.
- Use SameSite cookie attribute to prevent CSRF (Cross-Site Request Forgery) attacks.

Token Rotation:

- Regularly rotate secret keys used to sign JWTs and implement a mechanism to handle key rotation without disrupting user session.

User Interface:

Customer Registration:


SkySwift

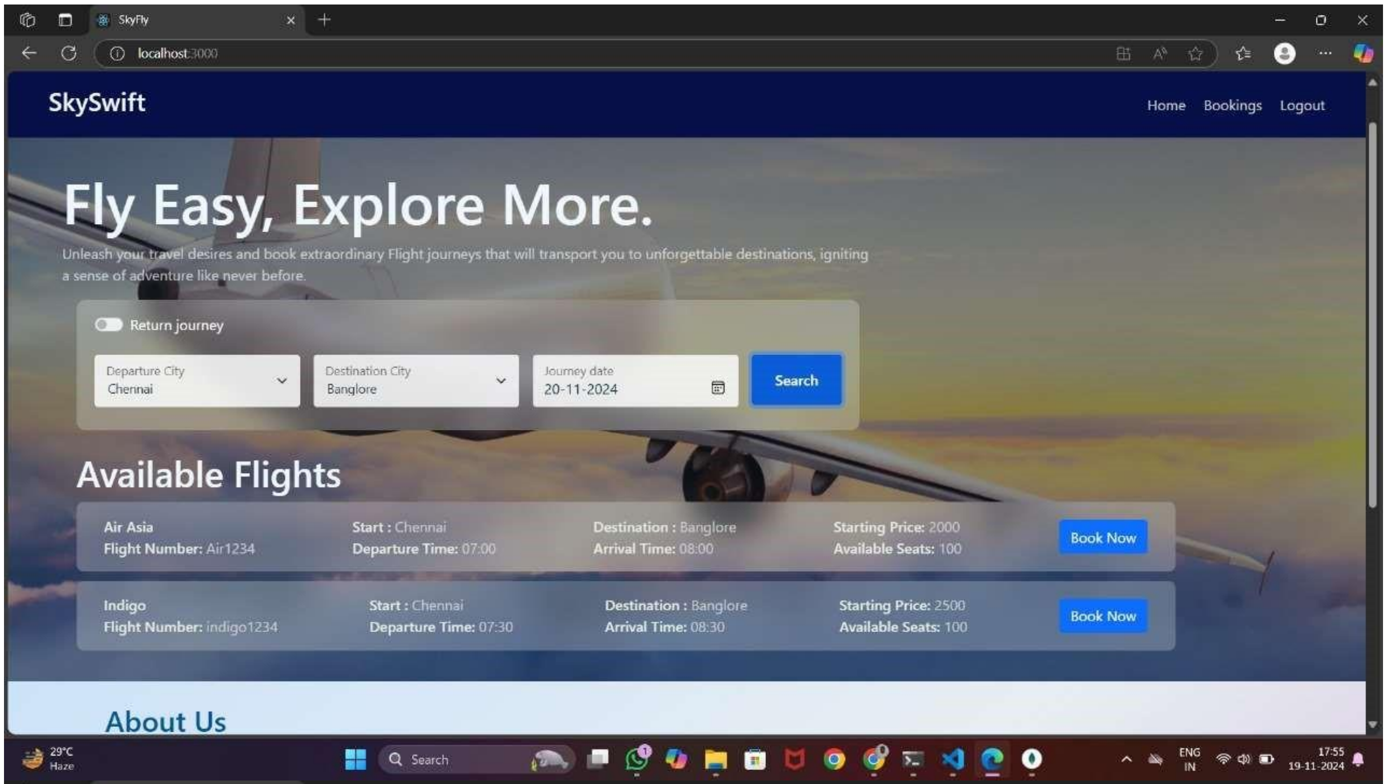
Register

Username
Tamizh Arasan

Email address
tamizharasan@gamil.com

Password
••••

Customer



Fly Easy, Explore More.

Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.

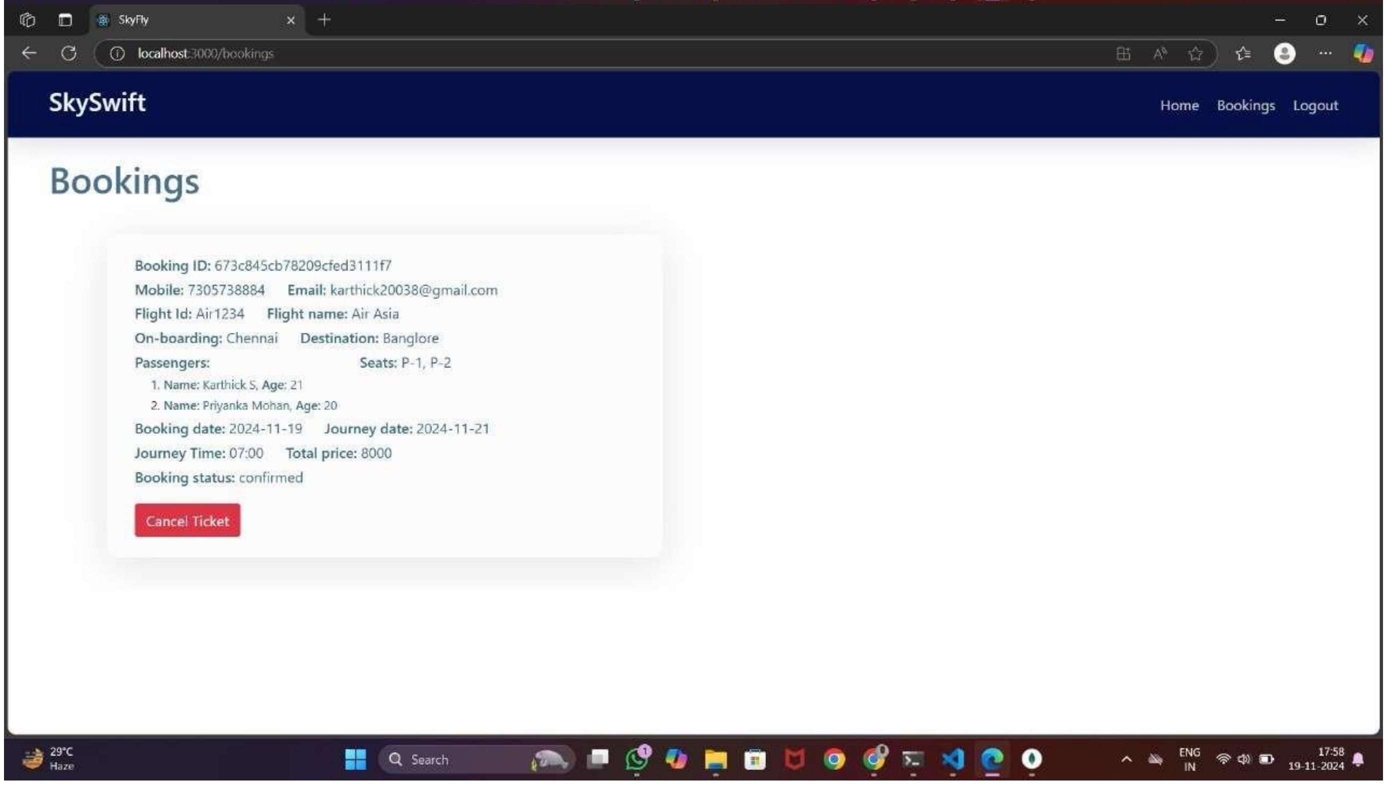
Return journey

Departure City: Chennai
Destination City: Banglore
Journey date: 20-11-2024

Available Flights

Air Asia
Flight Number: Air1234
Start : Chennai
Departure Time: 07:00
Destination : Banglore
Arrival Time: 08:00
Starting Price: 2000
Available Seats: 100

Indigo
Flight Number: indigo1234
Start : Chennai
Departure Time: 07:30
Destination : Banglore
Arrival Time: 08:30
Starting Price: 2500
Available Seats: 100



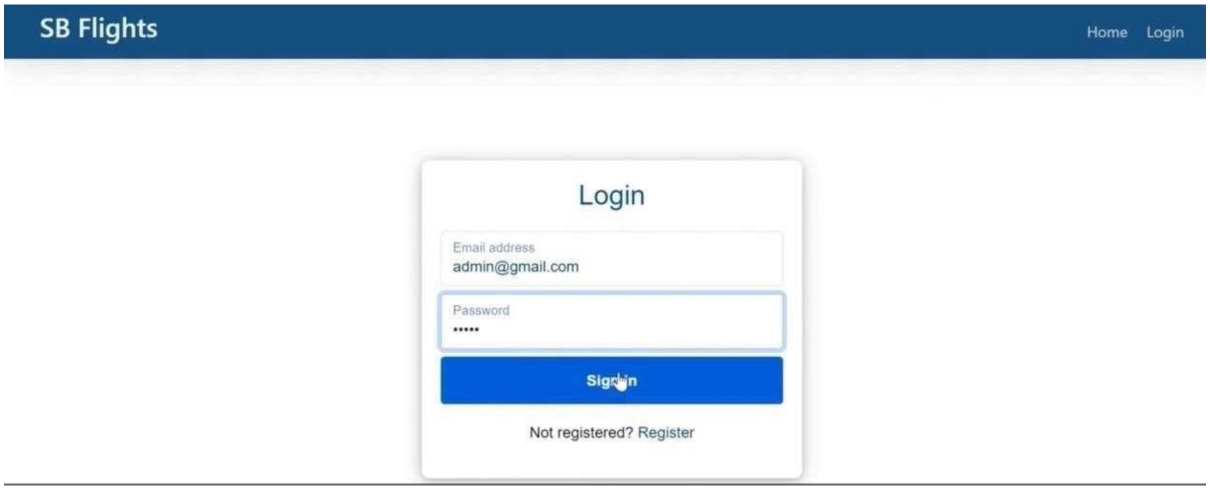
Bookings

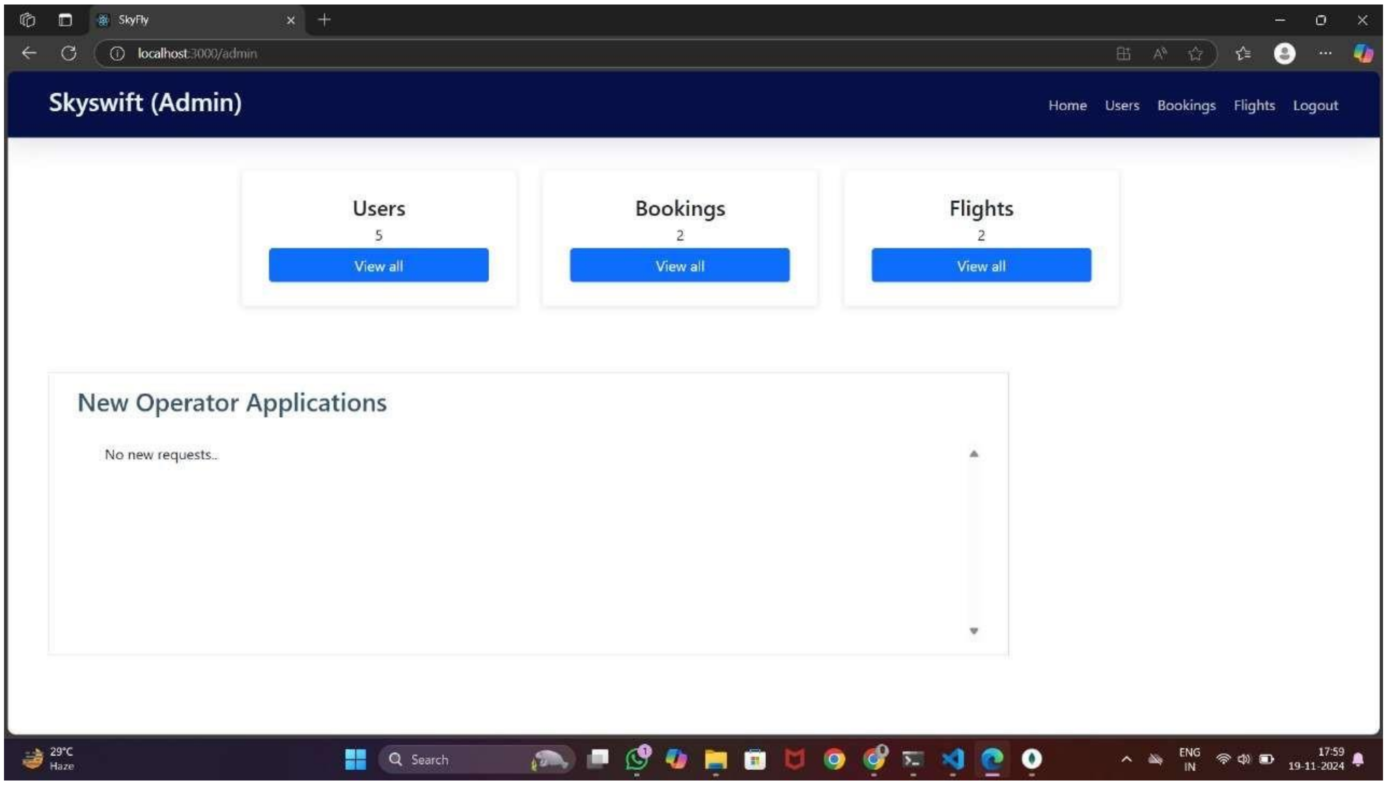
Booking ID: 673c845cb78209cfed3111f7
Mobile: 7305738884 **Email:** karthick20038@gmail.com
Flight Id: Air1234 **Flight name:** Air Asia
On-boarding: Chennai **Destination:** Banglore
Passengers: **Seats:** P-1, P-2

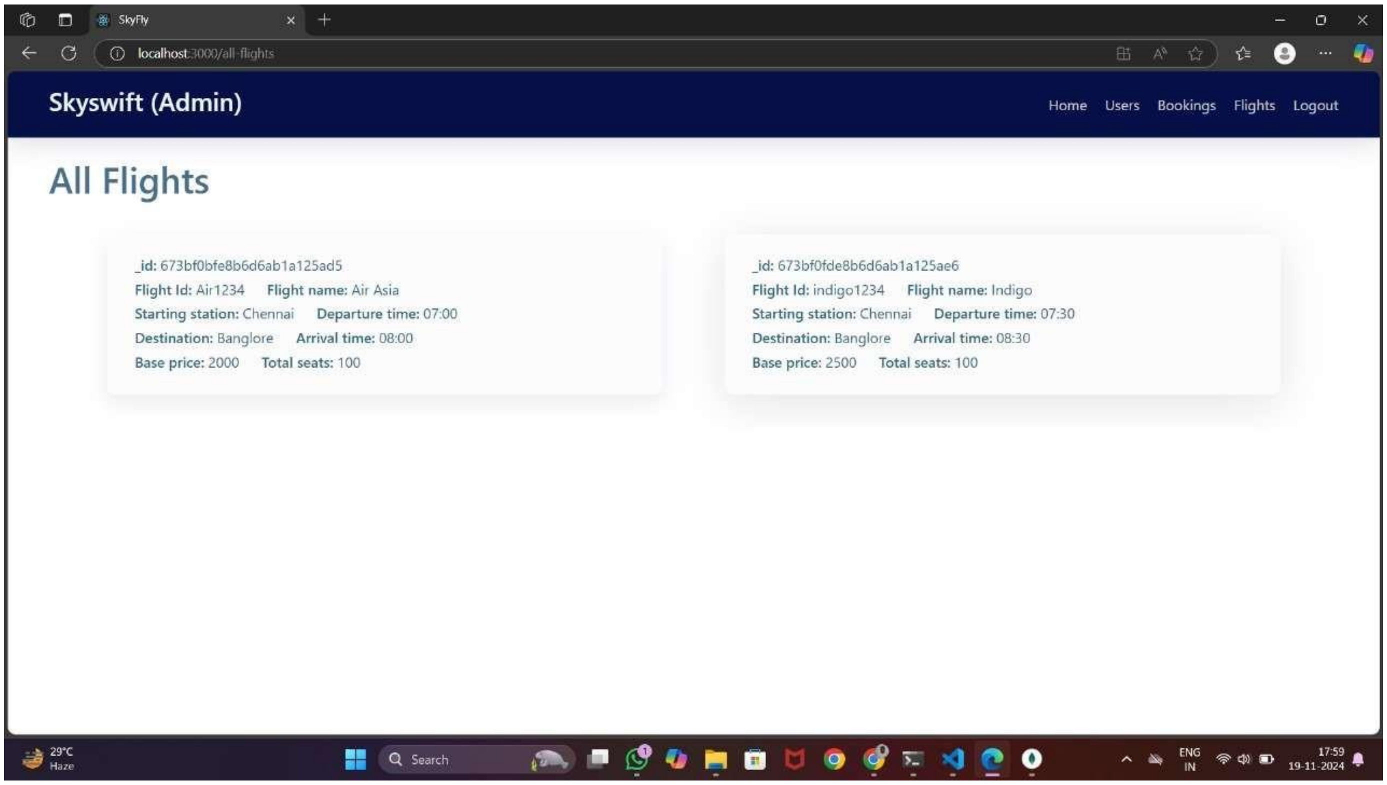
1. **Name:** Karthick S, **Age:** 21
2. **Name:** Priyanka Mohan, **Age:** 20

Booking date: 2024-11-19 **Journey date:** 2024-11-21
Journey Time: 07:00 **Total price:** 8000
Booking status: confirmed

Admin Login:


SB Flights
Home Login
Login
Email address
admin@gmail.com
Password
•••••
Sign in
Not registered? Register


SkyFly
localhost:3000/admin
Skyswift (Admin)
Home Users Bookings Flights Logout
Users
5
View all
Bookings
2
View all
Flights
2
View all
New Operator Applications
No new requests..
29°C Haze
Search
ENG IN
17:59
19-11-2024

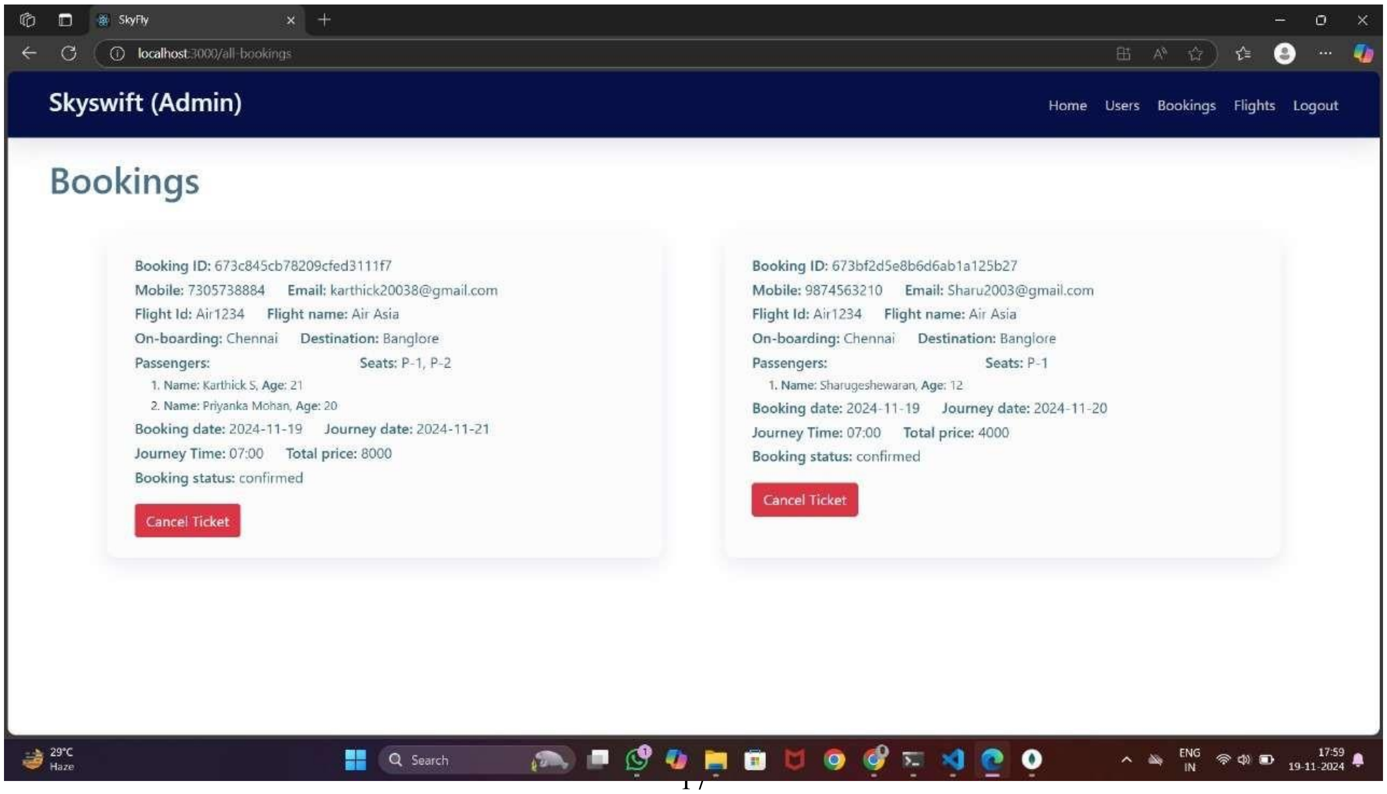


Skyswift (Admin)

All Flights

_id: 673bf0bfe8b6d6ab1a125ad5
Flight Id: Air1234 Flight name: Air Asia
Starting station: Chennai Departure time: 07:00
Destination: Banglore Arrival time: 08:00
Base price: 2000 Total seats: 100

_id: 673bf0fde8b6d6ab1a125ae6
Flight Id: indigo1234 Flight name: Indigo
Starting station: Chennai Departure time: 07:30
Destination: Banglore Arrival time: 08:30
Base price: 2500 Total seats: 100



Skyswift (Admin)

Bookings

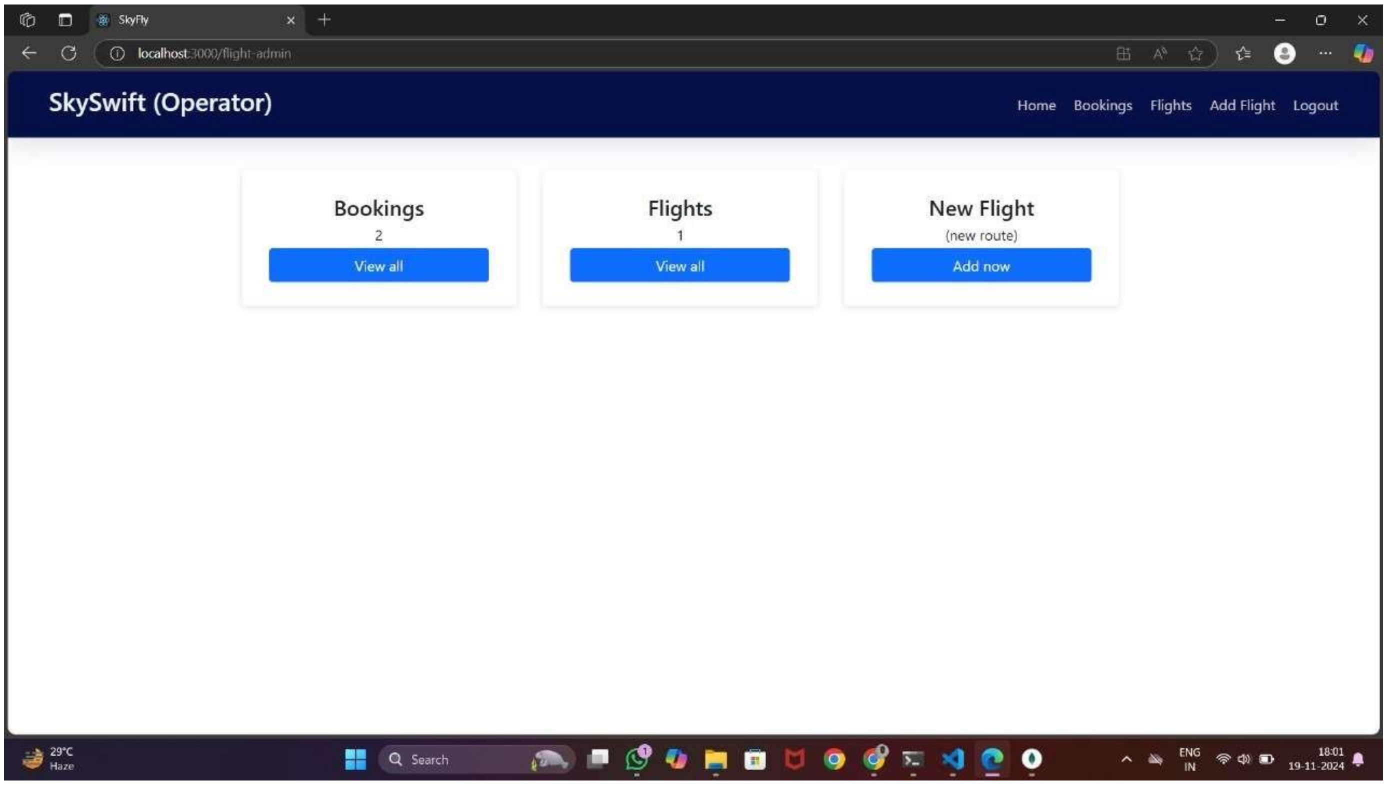
Booking ID: 673c845cb78209cfed3111f7
Mobile: 7305738884 Email: karthick20038@gmail.com
Flight Id: Air1234 Flight name: Air Asia
On-boarding: Chennai Destination: Banglore
Passengers: Seats: P-1, P-2
1. Name: Karthick S, Age: 21
2. Name: Priyanka Mohan, Age: 20
Booking date: 2024-11-19 Journey date: 2024-11-21
Journey Time: 07:00 Total price: 8000
Booking status: confirmed

Cancel Ticket

Booking ID: 673bf2d5e8b6d6ab1a125b27
Mobile: 9874563210 Email: Sharu2003@gmail.com
Flight Id: Air1234 Flight name: Air Asia
On-boarding: Chennai Destination: Banglore
Passengers: Seats: P-1
1. Name: Sharugeshewaran, Age: 12
Booking date: 2024-11-19 Journey date: 2024-11-20
Journey Time: 07:00 Total price: 4000
Booking status: confirmed

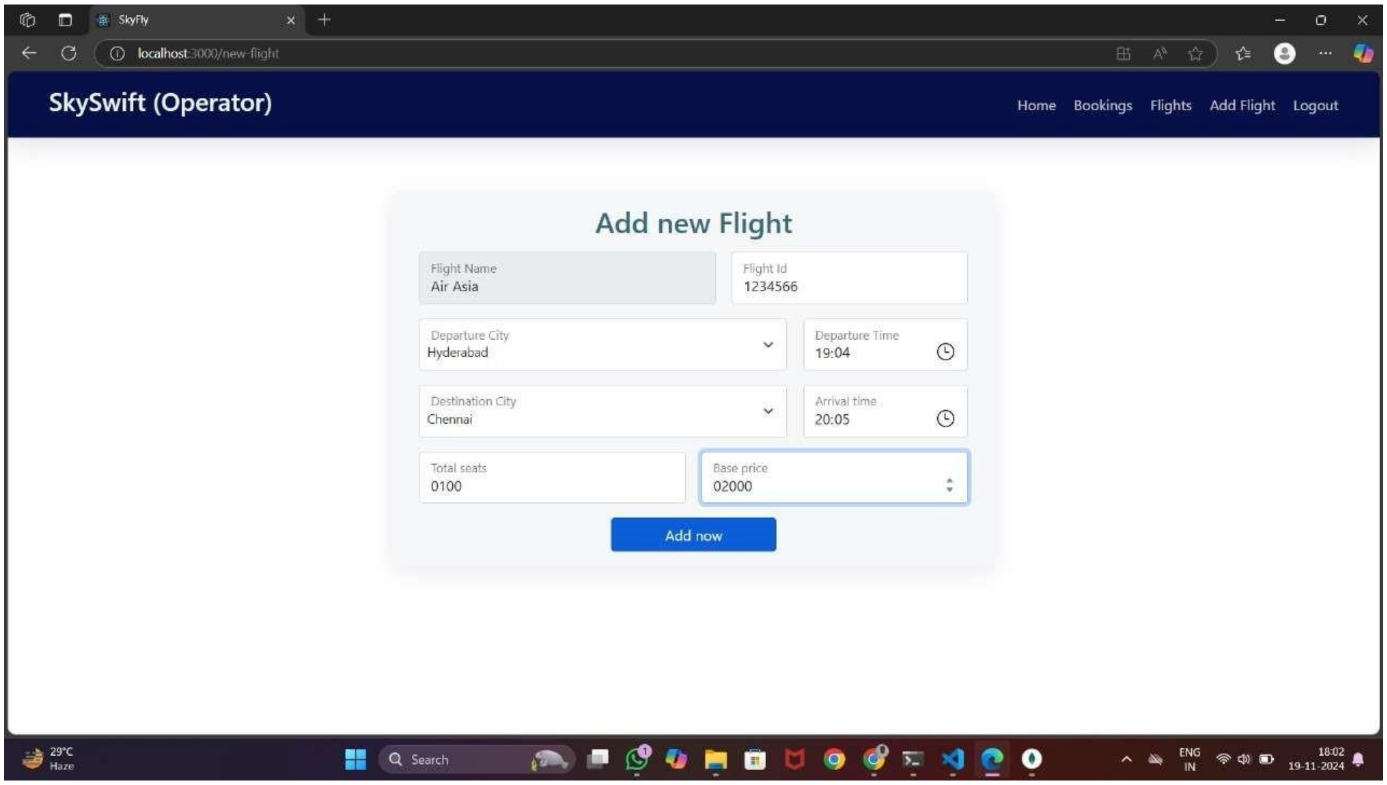
Cancel Ticket

Operator web Page:


SkySwift (Operator)

Home Bookings Flights Add Flight Logout

Bookings	Flights	New Flight
2	1	(new route)
View all	View all	Add now


SkySwift (Operator)

Home Bookings Flights Add Flight Logout

Add new Flight

Flight Name: Air Asia
Flight Id: 1234566
Departure City: Hyderabad
Departure Time: 19:04
Destination City: Chennai
Arrival time: 20:05
Total seats: 0100
Base price: 02000

Add now

Testing

Strategy

1. Unit Tests:
   - o Frontend:
     - Purpose: Tests individual components (e.g., FlightCard) to verify correct UI rendering and behavior.
     - Scope: Ensures that each component works as expected in isolation, including edge cases and different input scenarios.

   o

   Backend:
     - Purpose: Tests isolated functions, like calculatePrice, to ensure accuracy.
     - Scope: Verifies that individual functions and methods return the correct results and handle various inputs correctly.

2.

Integration Tests:

   - o API Routes:
     - Purpose: Tests backend endpoints (e.g., /api/flights) to ensure data flow and accurate responses.
     - Scope: Ensures that different parts of the backend application (routes, controllers, models) work together as expected.

   - o Frontend-Backend Interaction:
     - Purpose: Confirms data received from the backend displays correctly on the frontend.
     - Scope: Verifies that the frontend properly handles data received from the backend and updates the UI accordingly.

3. End-to-End (E2E) Tests:
   - o Purpose: Tests complete user journeys, like booking a flight, to ensure the app functions end-to-end.
   - o Scope: Simulates real user scenarios and interactions across the entire application, from frontend to backend.

Tools

1. Jest:
    - o Usage: Used for both frontend and backend testing, allowing unit and integration tests to run efficiently.
    - o Features: Provides a robust testing framework with features like mocks, spies, and snapshot testing.

2. Supertest:
    - o Usage: Tests HTTP requests in the backend to validate API responses.
    - o Features: Allows writing tests for your Express.js routes, making it easy to test the complete flow of your API.

3. React Testing Library:
    - o Usage: Simulates user actions to verify component rendering and behavior in the frontend.
      o Features: Focuses on testing components from the user's perspective, ensuring that your components work as expected when interacted with.

Running Tests

To run tests using Jest and Supertest, use the following commands:

```
sh npm
test
```

This command will execute all tests defined in your project, including unit, integration, and E2E tests.

Screenshots or Demo:

Demo Link:
https://drive.google.com/file/d/1Cb8bDraKP1GvFWSaNeCMAt1IvTivvsZR/view?usp=drive_link

Knowing Issues:

1. Occasional Delay in Fetching Flights:
   - Issue: There may be delays in retrieving flight data due to network latency or slow database responses, which can impact the user experience.
   - Impact: Users may experience longer waiting times when searching for flights, which can be frustrating and lead to a less seamless experience.

2. Minor UI Bugs on Smaller Screen Sizes:
   - Issue: Certain UI components may not display correctly or align properly on smaller screens, affecting mobile users.
   - Impact: Users accessing the app on mobile devices may encounter misaligned elements or UI glitches, affecting usability and overall user satisfaction.

3.Authentication Timeout:
   - Issue: Users may experience session timeouts if idle for too long, requiring them to log in again. This is due to token expiration for security but can disrupt user flow.
   - Impact: Frequent reauthentication can interrupt the user's interaction with the app, causing inconvenience, especially if the user is in the middle of a booking process.

4. Inconsistent Data Refresh:
   - Issue: After certain actions (e.g., updating or canceling a booking), the app may not immediately refresh to show the updated data without a manual refresh.
   - Impact: Users might not see the most up-to-date information immediately, leading to confusion and the potential for errors, especially in managing bookings.

5. Limited Error Handling on API Failures:
   - Issue: If an API call fails (e.g., due to server downtime), error messages may not display detailed information, leading to a poor user experience.
   - Impact: Users might not understand why an operation failed, reducing their ability to troubleshoot or take corrective action. This can lead to frustration and decreased trust in the app's reliability.

6. Performance Optimization:
   - Issue: The app may experience performance bottlenecks during peak usage times, affecting response times and overall efficiency.
   - Impact: High server load can lead to slower page loads and delayed interactions, impacting user satisfaction and potentially causing users to abandon the app.

7. Cross-Browser Compatibility:

- Issue: Some features might not work as expected across different web browsers, leading to inconsistent user experiences.
- Impact: Users accessing the app through various browsers may encounter different behaviors and potential issues, which can affect the app's accessibility and usability.

FutureEnhancements:

1. Multi-language Support:
   - Implementation: Add support for multiple languages to make the app accessible to a global audience. This includes translating all user interface text, date formatting, and currency conversion based on the user's region.
   - Language Selection: Offer a language selection option in the app settings or detect the user's preferred language automatically based on their location or browser settings.

2. Improved Flight Filtering:
   - Enhanced Search Filters: Include options for flight duration, number of stops, preferred airlines, cabin class, and departure/arrival time ranges.
   - Real-time Filtering: Implement real-time filtering capabilities so users can refine their search criteria without needing to reload the page.

3. Wishlist and Price Alerts:
   - Wishlist: Allow users to add flights to a wishlist and track their favorite options.
   - Price Alerts: Send notifications or emails to users when the price of flights in their wishlist changes or when new flights matching their criteria become available, helping users find the best deals.

4. User Reviews and Ratings:
   - Flight Ratings: Enable users to rate their flight experiences and leave reviews for airlines, providing valuable feedback for other users.

   Verified Reviews: Include star ratings, comments, and verified reviews from users who have completed their bookings to ensure authenticity.

5. Offline Mode:
   - Offline Capabilities: Allow users to view previously searched flights or check their booking history even without an internet connection.
   - Local Storage: Use local storage to maintain data availability and improve the
     app's reliability, especially for users with inconsistent connectivity.

6. Customer Support Chatbot:
   - AI-powered Chatbot: Integrate a chatbot to assist users with common queries, such as booking assistance, itinerary changes, and flight information.
   - 24/7 Support: Provide continuous support, reducing response times and enhancing the overall performance of the application.

Reference Resource:

1. https://www.w3schools.com/nodejs/
2. https://legacy.reactjs.org/tutorial/tutorial.html
3. https://www.mongodb.com/docs/manual/tutorial/
4. https://redux.js.org/tutorials/essentials/part-1-overview-concepts
5. https://mongoosejs.com/docs/
6. https://expressjs.com/en/5x/api.html