

TABLE OF CONTENTS:

TITLE	PAGE NO
1. Introduction	4
2. Project Overview	5
3. Architecture	7
4. Setup Instructions	10
5. Folder Structure	12
6. Running the Application	15
7. API Documentation	15
8. Authentication	18
9. User Interface	18
10. Testing	19
11. Screenshots or Demo	20
12. Known Issues	22
13. Future Enhancements	22

NM1042 - MERN Stack powered by MongoDB - project.

Flight Booking App - Project Report

1. Introduction:

Project Title: Flight Booking App

Team Members:

1. Haresh. V - Frontend Developer
2. Dhamodharan. V. C - Backend Developer
3. Arvinth. N. D - Database Administrator
4. Logeshwaran. M - Quality Assurance

Roles and Responsibilities:

- **Haresh. V:** Developed the frontend user interface to provide a seamless experience for users with the help of React and Express.js.
- **Dhamodharan. V. C:** Created functionality to each and every component in the frontend with the help of Node.js.
- **Logeshwaran. M:** Administered a database for storing transactional data within the application with the help of MongoDB.
- **Arvinth. N. D:** Performed API testing to test each and every endpoint of the app with the help of PostmanAPI.

2. Project Overview:

Purpose:

The purpose of the **Flight Booking App** is to provide a seamless and user-friendly platform for booking flights. The app is designed to help users easily find flights based on their travel preferences, view available flight options, and complete bookings securely. The goal is to make flight booking simple, accessible, and efficient for everyone. The project leverages the MERN stack, which ensures a modern and scalable architecture. The app also aims to support both regular users and administrators, giving them distinct interfaces with varying levels of control.

Key goals include:

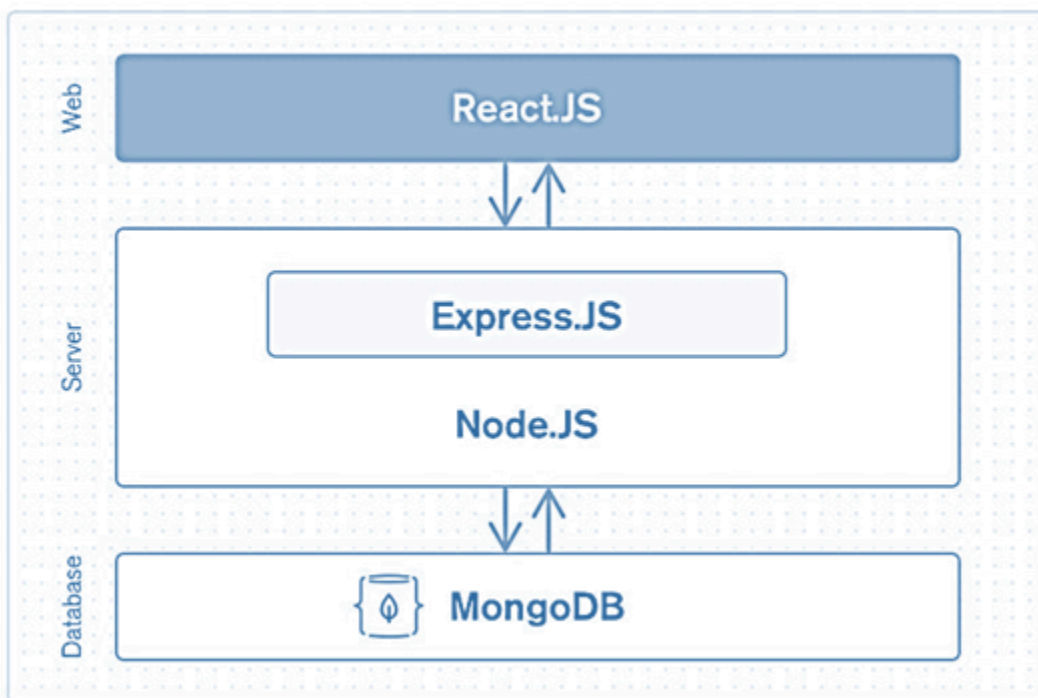
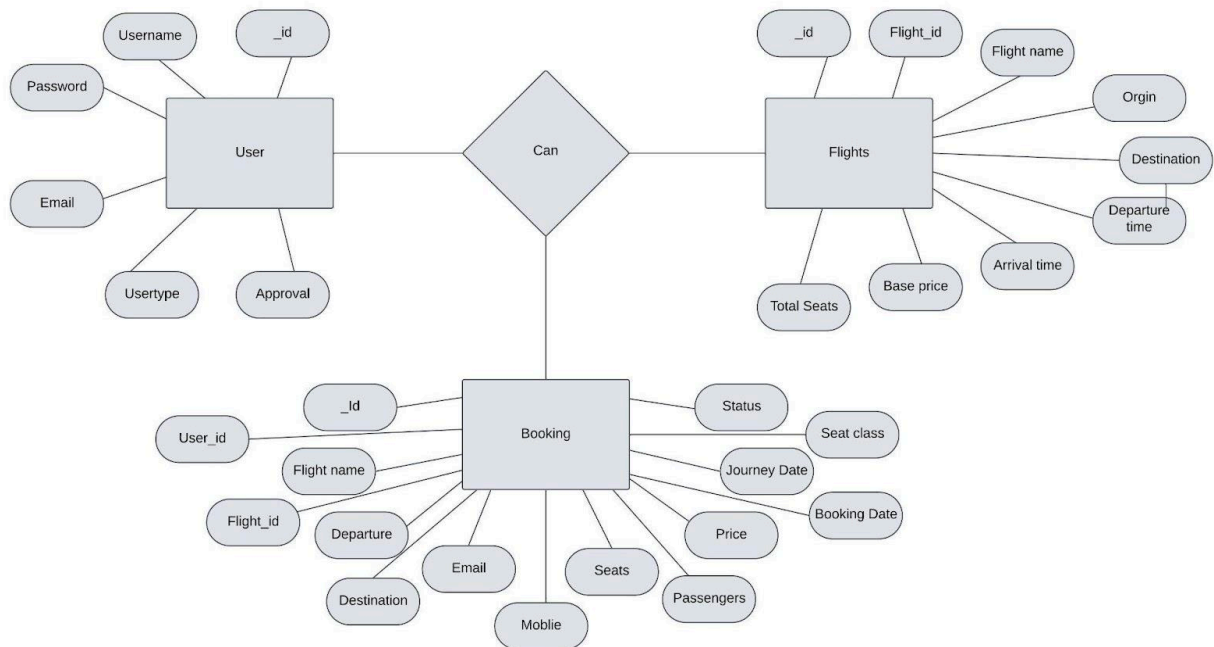
- Enabling users to search for flights based on departure and destination cities, travel dates, and class preferences.
- Displaying detailed flight information, including flight duration, seat availability, and pricing.
- Allowing users to book flights, providing a smooth and secure booking process.
- Providing an admin panel for flight management, including adding, editing, and deleting flights.
- Integrating a payment system for secure transaction processing.
- Ensuring that the application is scalable, secure, and responsive on various devices.

Features:

- **Flight Search:** Users can search for flights based on origin, destination, date of travel, and class (economy, business, first class). Filters like price range and airline preferences are also available.

- **Flight Details:** Each flight listing shows detailed information such as the departure and arrival times, flight duration, price, available seats, and any additional baggage information.
- **User Authentication:** Registered users can sign in and out securely using JWT tokens. Authentication is handled with strong encryption and ensures user privacy.
- **Flight Booking:** Users can book a flight by selecting it from the search results. They need to fill out their personal information and payment details. Once booked, users receive a confirmation email with the booking details.
- **Booking History:** Users can view all their past bookings, including flight details, dates, and payment information. Users can cancel or modify bookings if applicable.
- **Admin Panel:** Admin users have the ability to add new flights, update flight details, and delete flights from the database. They can also view all bookings and manage user profiles.
- **Responsive Design:** The application is fully responsive and works seamlessly across different devices, including desktops, tablets, and smartphones.
- **Payment Gateway Integration:** A payment gateway is integrated for processing flight bookings securely. Payment details are handled with encryption to ensure that sensitive information remains protected.

3. Architecture:



Frontend:

The frontend of the **Flight Booking App** is built using **React**. React is a component-based JavaScript library that allows for efficient rendering of dynamic user interfaces. It is perfect for creating interactive UIs that update in response to user interactions. The architecture follows a **component-based** design, where the UI is broken down into reusable components that handle specific tasks.

- **Component Structure:** The frontend app is built around reusable components such as `Header`, `FlightCard`, `BookingForm`, `SearchResults`, etc. These components are structured hierarchically to maintain a clean separation of concerns.
- **State Management:** We use **Redux** for state management. Redux helps in managing the global state across the application, such as user authentication status, flight search results, and booking status. Redux also simplifies passing data between components and ensures that the app is responsive to user interactions.
- **Routing:** The app uses **React Router** to handle page navigation. This allows for smooth transitions between different pages like the home page, search results, flight details, and user profile pages without reloading the page.
- **Styling:** For styling, we use **CSS Modules** in combination with **SASS** to create modular, maintainable, and scalable styles. The UI is designed to be responsive using **Flexbox** and **Grid** systems to accommodate various screen sizes.

Backend:

The backend is developed using **Node.js** and **Express.js**, which together form a powerful, scalable, and fast API layer. Express.js is a minimal web application framework for Node.js, which makes routing and middleware management simple

and efficient. The server handles requests from the frontend, processes them, and returns appropriate responses.

- **RESTful API:** The backend is designed around REST principles. The API endpoints are structured to handle CRUD operations for flights, bookings, and users.

- **Authentication:** The backend handles user authentication and authorization using **JWT** tokens. The server generates a JWT token when the user logs in or registers, and this token is used to authenticate subsequent requests.

- **Data Validation:** Data validation is performed both on the client side (to improve UX) and on the server side (for security). **Joi** is used for schema validation of incoming requests.

- **Security:** The backend implements **bcryptjs** for hashing user passwords and **Helmet** middleware to secure HTTP headers. We also use **CORS** to enable cross-origin requests between the frontend and backend.

Database:

The database used for this project is **MongoDB**, a NoSQL database known for its flexibility and scalability. The app stores various entities like users, flights, and bookings.

- MongoDB Schema:

- **User Schema:** Contains fields such as ``name``, ``email``, ``password`` (hashed), and ``role`` (user or admin).

- **Flight Schema:** Stores flight details like ``flightNumber``, ``origin``, ``destination``, ``departureTime``, ``arrivalTime``, ``price``, and ``availableSeats``.

- **Booking Schema:** Contains fields like `userId`, `flightId`, `passengerDetails`, and `bookingStatus`.
- **Mongoose:** We use **Mongoose** to define and interact with MongoDB schemas in a more structured and reliable way. Mongoose allows us to enforce validation rules, define model methods, and perform complex queries.

4. Setup Instructions:

Prerequisites:

Before setting up the application, you need to ensure the following tools and technologies are installed:

- **Node.js:** Download and install the latest version of Node.js from <https://nodejs.org/>
- **MongoDB:** You can use a local MongoDB installation or a cloud-based instance via <https://www.mongodb.com/cloud/atlas>
- **Git:** Install Git from <https://git-scm.com/>

Installation:

To set up the app on your local machine, follow these steps:

A. Clone the repository:

```
git clone https://github.com/Haresh-V-2003/flight-booking-app.git
cd flight-booking-app
```

B. Backend Setup:

- Navigate to the `server` directory:

```
cd server
```


- Install backend dependencies:
npm install
- Create an `.env` file in the `server` directory to configure environment variables:
touch .env
- Add the following environment variables to the `.env` file:

MONGO_URI=your_mongodb_connection_string

JWT_SECRET=your_jwt_secret_key

C. Frontend Setup:

- Navigate to the `client` directory:
cd ../client
- Install frontend dependencies:
npm install

D. Running the Application:

- Start the backend server:
cd server
npm start
- Start the frontend server:
cd client
npm start

By default, the backend will run on `http://localhost:5000`, and the frontend will run on `http://localhost:3000`.

5. Folder Structure:

Client (Frontend):

client/

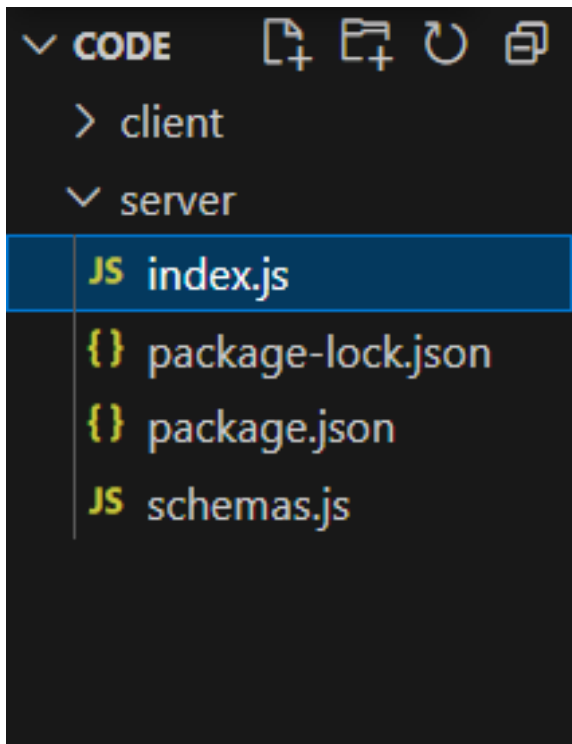
- |— public/ # Static assets (index.html, images)
 - | |— index.html # Main HTML file
 - | |— assets/ # Images, logos, icons
- |— src/
 - | |— components/ # UI components (Header, FlightCard, etc.)
 - | |— pages/ # Pages (Home, SearchResults, etc.)
 - | |— redux/ # Redux store (actions, reducers, etc.)
 - | |— App.js # Main component for app routing
 - | |— index.js # Entry point to React app
 - | |— utils/ # Helper functions (API calls)
- |— .env # Environment variables





Server (Backend):

server/

- |— config/ # Configuration files (database, server)
 - | |— database.js # MongoDB connection setup
 - | |— server.js # Express server setup
- |— controllers/ # Request handlers (flight, user, booking)
 - | |— flightController.js
 - | |— userController.js
 - | |— bookingController.js
- |— models/ # Mongoose models (User, Flight, Booking)

- | | — User.js
- | | — Flight.js
- | | — Booking.js
- | — routes/ # Express route definitions
- | | — authRoutes.js
- | | — flightRoutes.js
- | | — bookingRoutes.js
- | — middleware/ # Custom middleware (authentication, error handling)
- | — authMiddleware.js
- | — errorMiddleware.js



✓ CODE    

✓ client

> public

✓ src

> assets

> components

> context

> pages

> RouteProtectors

> styles


App.css

JS App.js

JS App.test.js


index.css

JS index.js

 logo.svg


JS reportWebVitals.js

JS setupTests.js

 .gitignore

{ } package-lock.json

{ } package.json

 README.md

> server

6. Running the Application:

Frontend:

1. Navigate to the `client` directory and run:

```
npm start
```

Backend:

1. Navigate to the `server` directory and run:

```
npm start
```

Both servers should now be running locally on `**http://localhost:5000**` for the backend and `**http://localhost:3000**` for the frontend.

7. API Documentation:

Flight Search:

- GET /api/flights

- Parameters:

`origin` (string) - Departure city.

`destination` (string) - Arrival city.

`date` (string) - Departure date (ISO 8601 format).

`class` (string) - Class of travel (economy, business, first).

- Example Response:

```
[  
  {  
    "flightNumber": "AB123",  
    "origin": "New York",  
    "destination": "London",  
    "date": "2024-12-15",
```

```
"price": 500,  
  "availableSeats": 100  
}  
]
```

User Authentication:

- POST /api/auth/register

- Body:

```
{  
  "name": "John Doe",  
  "email": "john@example.com",  
  "password": "password123"  
}
```

- Example Response:

```
{  
  "message": "User registered successfully",  
  "token": "jwt_token_here"  
}
```

- POST /api/auth/login

- Body:

```
{  
  "email": "john@example.com",  
  "password": "password123"  
}
```

- Example Response:

```
{  
  "message": "Login successful",  
  "token": "jwt_token_here"  
}
```

Flight Booking:

- POST /api/bookings

- Body:

```
{  
  "flightId": "12345",  
  "userId": "67890",  
  "passengerDetails": {  
    "name": "John Doe",  
    "passportNumber": "A1234567"  
  }  
}
```

- Example Response:

```
{  
  "message": "Booking successful",  
  "bookingId": "booking123"  
}
```

8. Authentication:

- Authentication is managed using **JWT (JSON Web Tokens)**. When a user logs in or registers, the backend generates a signed JWT token that is sent to the client. The client stores this token (typically in `localStorage` or `sessionStorage`), and for every subsequent API request, the client sends this token in the Authorization header. The backend validates the token to ensure the user is authenticated.

Steps:

A. User Registration: The user's password is hashed using **bcryptjs** before saving it in the database.

B. Login: During login, the password is verified against the stored hash. If the credentials are correct, the server issues a JWT token.

C. Authorization: For any restricted routes (e.g., flight booking), the JWT token is checked to verify the user's identity. If the token is invalid or expired, the request is rejected.

9. User Interface:

The user interface is designed to be simple, modern, and easy to navigate. Below are some key UI features:

- **Landing Page:** The homepage allows users to input flight search parameters like departure city, destination, and dates.

- **Flight Search Results:** Displays available flights with options to filter and sort.

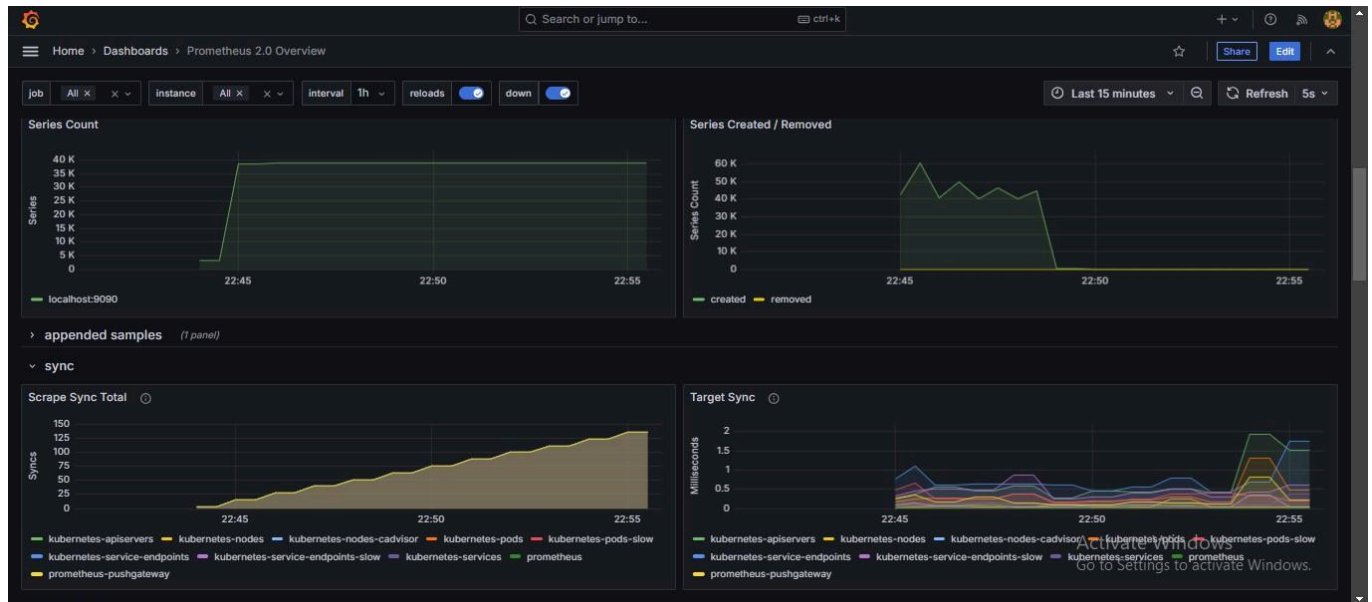
- **Weather Reports:** Displays the expected weather conditions for all days of the week.
- **Recommended Hotels:** Provides a smart recommendation for the nearby hotels.
- **Tourist Spots:** Displays the famous places to visit in each and every country.

These recommendations provide a seamless user experience and makes the application more user-friendly.

10. Testing:

We use a combination of **Jest**, **Mocha**, **Chai**, and **React Testing Library** for different levels of testing:

- **Unit Testing:** Tests for individual components and backend functions.
- **Integration Testing:** Verifies that the frontend and backend work together as expected.
- **End-to-End Testing:** Simulates real user behavior from searching for flights to booking and viewing booking history.
- **API Testing:** Tests endpoints like **api/auth/login**, **api/auth/register**, **api/flights**, **api/bookings**, etc.



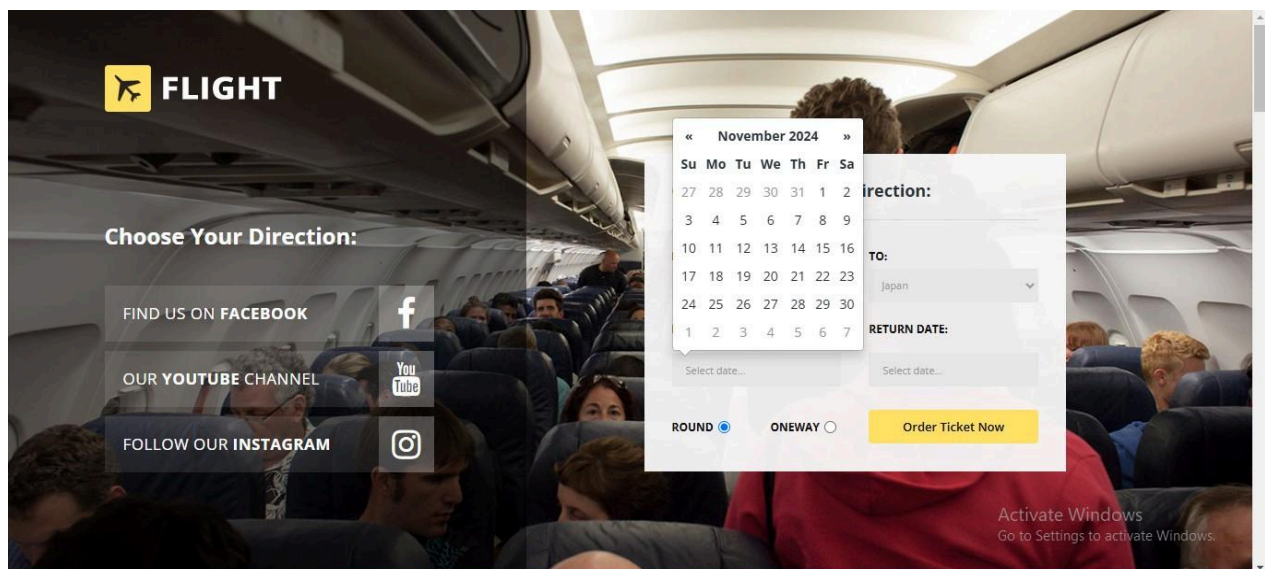
(A Grafana Dashboard describing user traffic pattern for our application)

11. Screenshots or Demo:

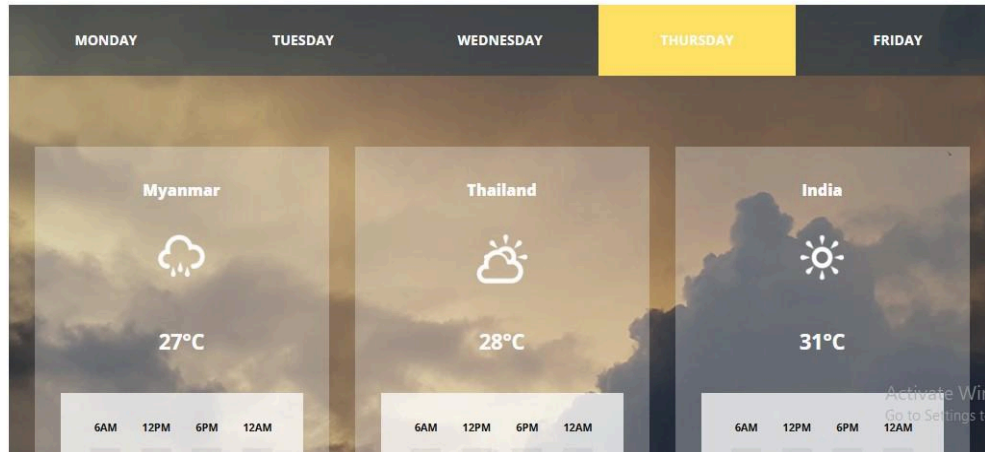
A live demo is available at:

<https://drive.google.com/drive/folders/1nY6tEW3lqKEuw5rigFetVSQs8QTKTc-z>

Screenshots:



CHECK WEATHER FOR 5 NEXT DAYS



RECOMMENDED HOTEL FOR YOU

LIVING ROOM >

SUIT ROOM >

SWIMING POOL >

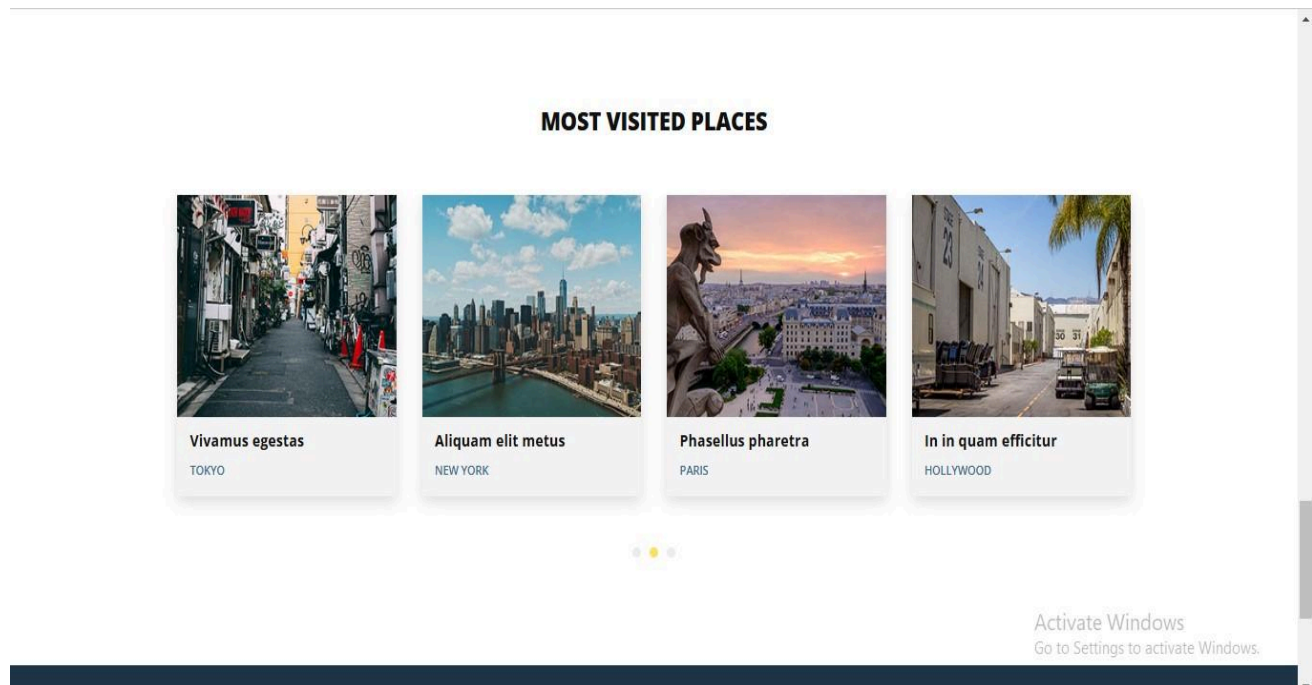
MASSAGE SERVICE >

FITNESS LIFE >

EVENING EVENT >

The Best Sitting
HOTEL GRAND

Activate Windows
Go to Settings to activate Windows.



12. Known Issues:

- **Flight Search Performance:** The search functionality could be slow with a large number of flights.
- **Payment Gateway Integration:** Payment features are not yet fully integrated due to issues with third-party APIs.

13. Future Enhancements:

- **Payment Integration:** Finalize payment integration with a service like Stripe or PayPal.
- **User Reviews:** Add the ability for users to review and rate flights.
- **Mobile App:** Extend the platform to mobile devices using React Native.