



**LOYOLA INSTITUTE OF TECHNOLOGY,**

**ANNA UNIVERSITY: CHENNAI 600 025**

# FLIGHT BOOKING APPLICATION USING MERN STACK

**A Project Report**

***Submitted by***

**DINESH S - 210921205013**

**MURUGAN A - 210921205032**

**GOKUL PRASATH H– 210921205017**

**VISWHA NATHAN S- 210921205058**

**In partial fulfillment for the award of the degree**

**Of**

**BACHELOR OF TECHNOLOGY**

**In**

**INFORMATION TECHNOLOGY**

# PROJECT OVERVIEW

# PURPOSE:

The primary goal of the flight booking application is to provide users with a seamless and efficient platform for searching, comparing, and booking flights. The application aims to enhance user convenience by offering features such as:

1. **User-Friendly Interface**: Simplifying the flight booking process with an intuitive design that caters to all user demographics.
2. **Comprehensive Search Options**: Allowing users to search flights based on various parameters such as destination, date, price, airline, and class of service.
3. **Transparent Pricing**: Displaying detailed fare breakdowns and ensuring no hidden charges.
4. **Customizable Features**: Offering filters for user preferences like layovers, baggage allowance, and in-flight services.
5. **Real-Time Updates**: Providing up-to-date flight availability and pricing through integration with airline databases.
6. **Secure Transactions**: Ensuring user data and payment security with robust encryption and fraud prevention mechanisms.
7. **Value-Added Services**: Including features like booking management, e-tickets, notifications, and travel insurance options.

# Features and Functionalities of the Flight Booking Application

### 1. User Account Management

- User registration and login options (email, social media, or guest login).
- Profile management to save preferences (frequent destinations, airlines, etc.).
- Booking history and saved trips for easy reference.

### 2. Flight Search and Filters

- Search flights by destination, date, price range, and preferred airlines.
- Advanced filters for stops (non-stop, 1-stop), departure/arrival times, and ticket class.
- Multi-city search functionality for complex itineraries.

### 3. Flight Comparison

- Side-by-side comparison of flight options based on price, duration, and amenities.
- Highlighting best deals and promotions.

### 4. Booking and Payment

- Detailed flight selection with fare breakdown and baggage information.
- Multiple payment options (credit/debit cards, wallets, net banking, etc.).
- Secure payment gateway with encryption and fraud detection.

**5. Real-Time Updates**

- Live flight availability and price updates.
- Notifications for price changes, flight delays, and cancellations.

**6. Ticket Management**

- E-ticket generation and email confirmation.
- Options for seat selection and meal preferences during booking.
- Easy cancellations and refund tracking.

**7. Customer Support**

- 24/7 customer support via chat, email, or phone.
- Help centre with FAQs and troubleshooting guides.

**8. Additional Services**

- Travel insurance options during checkout.
- Add-ons like priority boarding, extra baggage, and airport transfers.
- Integration with hotel booking and car rental services.

**9. Accessibility and User Experience**

- Multi-language support for global users.
- Mobile-friendly design with dedicated Android/iOS apps.
- Personalized recommendations based on user behaviour.

**10. Security and Privacy**

- Data encryption and GDPR compliance for user privacy.
- Two-factor authentication for enhanced security.

# SETUP INSTRUCTIONS

# PREREQUISITES

To set up and run the Flight Booking Application project using the MERN stack, the following software dependencies and tools are required:

**Software Dependencies:**

**1.Node.js**

- Required to run the backend server and manage dependencies.
- Download Node.js

**2.MongoDB**

- Database to store user details, project data, and transactions.
- Install a local MongoDB instance or use a cloud solution like MongoDB Compass.

**3.React.js**

- Frontend framework for building the user interface.
- Runs in the browser, typically managed with npm.

**4.Express.js**

- Web framework for the backend to handle API requests.

**5.npm (Node Package Manager):**

- Comes with Node.js and is required to install project dependencies.

**Development Tools:**

**1.Git**

- For version control and managing code repositories.
- Download Git

**2.Code Editor**

- Recommended: Visual Studio Code (VS Code).
- Download VS Code

**3.Postman**

- For testing APIs during backend development.
- Download Postman

**4.Browser**

- A modern web browser for testing the frontend, such as Google Chrome or Firefox.

**Installed Node.js Dependencies:**

**1.Backend Dependencies:**

- express: Web server framework.
- mongoose: MongoDB object modeling tool.
- jsonwebtoken: For user authentication using JWT.
- bcryptjs: For hashing passwords securely.
- dotenv: For managing environment variables.

**Install them using:**

Code:

```
npm install express mongoose jsonwebtoken bcryptjs dotenv
```

**2.Frontend Dependencies**:

- react: Core React library for building UI.
- react-router-dom: For handling navigation between pages.
- axios: For making HTTP requests.
- redux (optional): For state management (if needed).

**Install them using:**

Code:

```
npm install react react-router-dom axios
```

**Environment Variables:**

Set up a .env file in the root directory to manage configuration:

Code:

```
MONGO_URI=<Your MongoDB connection string>
JWT_SECRET=<Your secret key for JWT>
PORT=5000
```

# Installation: Step-by-Step Guide for Flight Booking Application Project

Here's a detailed step-by-step guide to install and set up your flight booking application project. The guide assumes you are using a typical development stack (e.g., front-end, back-end, and database).

### 1. Clone the Repository

To get the project files on your local machine:

Code:

```
git clone <repository_url>
cd <project_directory>
```

### 2. Install Dependencies

1. **Backend Dependencies:**
   - Navigate to the backend folder (e.g., server/) if it's structured separately:

   Code:

   ```
   cd server
   npm install
   ```

   - This installs the following dependencies:
     - express
     - mongoose
     - jsonwebtoken
     - bcryptjs

2. **Frontend Dependencies:**
    - Navigate to the frontend folder (e.g., client/) if the frontend is in a separate directory:

    Code:

    ```
    cd client
    npm install
    ```

    - This installs the following dependencies:
        - react
        - react-router-dom
        - axios

**3. Set Up Environment Variables**

- Create a .env file in the **backend directory** (or wherever the backend code is located).
- Add the following environment variables:

Code:

```
MONGO_URI=<Your MongoDB connection string>
JWT_SECRET=<Your secret key>
PORT=5000
```

- **MONGO_URI**: Replace with your MongoDB connection string (local or cloud).
- **JWT_SECRET**: A strong secret key for generating and verifying JWT tokens.
- **PORT**: Port number to run your backend server.

**4. Start the Development Server**

1. **Start Backend Server:**
    - Navigate to the backend directory:

      Code:

      ```
      cd server
      npm start
      ```

    - The server should start running on the defined PORT (e.g., http://localhost:5000).
2. **Start Frontend Development Server:**
    - Navigate to the frontend directory:

      Code:

      ```
      cd client
      npm start
      ```

    - The React development server will typically run at http://localhost:3000.

**5. Test the Application**

- Open your browser and test the frontend at http://localhost:3000.
- Verify the backend by making API requests using tools like Postman or directly through the frontend.

**Software Requirements:** Ensure **Node.js**, **MongoDB**, and **Git** are installed before starting.

**Dependencies:** All mentioned backend (express, mongoose, etc.) and frontend (react, axios, etc.) dependencies are installed during the npm install steps.

**Environment Variables:** Properly setting up the .env file aligns with the prerequisites for MongoDB, JWT, and server configuration.

**Database Setup**

- **Local MongoDB Setup:**
  - If running MongoDB locally, ensure MongoDB is installed and the service is running.
  - Create a new database (e.g., freelancer_website) using the MongoDB shell or GUI tools like **MongoDB Compass**.
  - Update the MONGO_URI in the .env file to point to your local database:

Code:

```
MONGO_URI=mongodb://localhost:27017/freelancer_website
```

**Cloud MongoDB Setup (MongoDB Atlas):**

- Create a cluster on MongoDB Atlas.
- Whitelist your IP address.
- Get the connection string and replace it in your .env file.

**Install Development Tools**

**Nodemon for Backend Development:**

Use **Nodemon** to automatically restart the server during development:

**Initialize Git and Push to a Repository**

1. Initialize a Git repository:

   Code:

   ```
   git init
   git add .
   git commit -m "Initial commit"
   git branch -M main
   git remote add origin <your-repository-url>
   git push -u origin main
   ```

2. Ensure your backend and frontend are in separate folders (if applicable) and push them both.

**Configure CORS (Cross-Origin Resource Sharing)**

If the frontend and backend are hosted on different servers or ports, configure CORS to allow them to communicate:

- Install cors in your backend:

  Code:

  ```
  npm install cors
  ```

- Add it to your backend code (e.g., app.js):

  **javascript**

  code:

  ```
  const cors = require('cors');
  app.use(cors());
  ```

**Testing**

1. **Backend Testing:**

- Use Postman to test API endpoints and ensure all routes are functioning as expected.
- Example test: Test user login and registration APIs with valid/invalid inputs.

2. **Frontend Testing:**
   - Test the user interface manually to ensure all pages are responsive and functional.
   - Optionally, use testing libraries like **Jest** or **React Testing Library**:

Code:

```
npm install --save-dev jest react-testing-library
```

**Linting and Code Formatting**

- Install **ESLint** and **Prettier** to ensure consistent code quality:

Code:

```
npm install --save-dev eslint prettier
```

- Configure .eslintrc.json for your project:

Code:

```
{
 "env": {
  "browser": true,
  "es2021": true
 },
 "extends": "eslint:recommended",
 "parserOptions": {
  "ecmaVersion": 12,
```

```
    "sourceType": "module"
  },
  "rules": {}
}
```

- Add a Prettier configuration (.prettierrc):

**json**

code:

```
{
  "singleQuote": true,
  "semi": false
}
```

**Deployment**

1. **Deploy Backend:**
    - Host the backend on a platform like Heroku, Railway, or [AWS EC2].
    - Set environment variables (e.g., MONGO_URI, JWT_SECRET) in the platform's environment settings.
2. **Deploy Frontend:**
    - Build the React app:

      Code:

      ```
      npm run build
      ```

    - Deploy the build/ directory to a hosting platform like Netlify, Vercel, or [AWS S3].
3. **Configure Proxy (Optional):**

- If hosting the frontend and backend separately, configure a proxy in your React app's package.json:

Code:

```
"proxy": "http://localhost:5000"
```

**Monitor and Log Errors**

1. Add error-handling middleware in the backend:

**javascript**

code:

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

2. Use tools like **Sentry** for monitoring:

Code:

```
npm install @sentry/node
```

**Database Backup**

- If using MongoDB, ensure regular backups of your database:
  - Use **MongoDB Atlas backup options** or tools like mongodump.

**Future Enhancements (Optional)**

- **Add Unit Tests:** Use testing frameworks like Mocha, Chai, or Jest.

- **Optimize SEO:** Improve metadata, sitemaps, and social media previews for better search rankings.
- **Enable SSL:** Use HTTPS for secure communication when deploying.

# FOLDER STRUCTURE

## Client: React Frontend Structure

```
client/
├── public/
│   ├── index.html
│   ├── favicon.ico
│   └── assets/
├── src/
│   ├── components/
│   │   ├── Header.jsx
│   │   ├── Footer.jsx
│   │   └── FlightCard.jsx
│   ├── pages/
│   │   ├── Home.jsx
│   │   ├── SearchResults.jsx
│   │   ├── BookingForm.jsx
│   │   └── NotFound.jsx
│   ├── services/
│   │   ├── api.js
│   │   └── authService.js
│   ├── context/
│   │   └── AuthContext.js
│   ├── utils/
```

```
│   │   ├── formatDate.js
│   │   └── validateForm.js
│   ├── App.jsx
│   ├── index.js
│   ├── routes.js
│   └── styles/
│       ├── App.css
│       ├── variables.css
│       └── components/
└── package.json
```

**Description**:

- **Public Folder**: Contains static files like index.html and assets.
- **Components**: Houses reusable UI components for modularity.
- **Pages**: Organizes the main views or screens of the application.
- **Services**: Manages API calls and external data integration.
- **Context**: Handles global state management using Context API.
- **Utils**: Utility functions to keep the code clean and reusable.
- **Styles**: Maintains consistent styling across the application.

# 2. Server: Node.js Backend Structure

server/

```
├── config/
│   ├── db.js
│   ├── jwt.js
│   └── env.js
├── controllers/
│   ├── flightController.js
│   ├── bookingController.js
│   └── userController.js
├── models/
│   ├── Flight.js
│   ├── Booking.js
│   └── User.js
├── routes/
│   ├── flightRoutes.js
│   ├── bookingRoutes.js
│   └── userRoutes.js
├── middleware/
│   ├── authMiddleware.js
│   ├── errorHandler.js
│   └── logger.js
├── utils/
│   ├── sendEmail.js
│   └── formatResponse.js
├── tests/
│   ├── flight.test.js
│   ├── booking.test.js
│   └── user.test.js
├── index.js
```

├── package.json

├── .env

└── README.md

**Description**:

- **Config**: Contains configuration files for the database, authentication, and environment variables.
- **Controllers**: Includes business logic for handling specific routes or APIs.
- **Models**: Defines database schemas using ORM tools like Mongoose or Sequelize.
- **Routes**: Defines endpoint routes and links them to controllers.
- **Middleware**: Holds custom middleware for authentication, logging, and error handling.
- **Utils**: Provides helper functions for common tasks like sending emails or formatting data.
- **Tests**: Contains test cases for unit and integration testing.

# RUNNING THE APPLICATION

To run the flight booking application locally, follow these steps to start the frontend and backend servers.

**Step 1: Starting the Frontend Server**

1. Navigate to the client directory:

   bash

   Code: cd client

2. Install the dependencies (if not done already):

   bash

   Code: npm install

3. Start the React development server:

   bash

   Code: npm start

4. The frontend server will run at http://localhost:3000 by default. Open this URL in your browser to access the application.

**Step 2: Starting the Backend Server**

1. Navigate to the server directory:

   bash

   Code: cd server

2. Install the dependencies (if not done already):

   bash

   Code: npm install

3. Start the Node.js backend server:

   bash

   Code: npm start

4. The backend server will run at http://localhost:5000 by default (or the port defined in your .env file). This will serve as the API endpoint for the frontend.

**Connecting Frontend and Backend**

- Ensure the frontend is configured to make API requests to the backend server. Update the API base URL in the frontend's services/api.js file or equivalent configuration:

  Javascript

  Code: const API_BASE_URL = ‘ http://localhost:5000 ‘;

Once both servers are running, the application will function locally, with the frontend communicating with the backend APIs for flight booking operations.

# API DOCUMENTATION

**1. Authentication APIs**

**1.1 User Registration**

- **Endpoint**: /api/auth/register
- **Method**: POST
- **Request Parameters**:
  - name (string, required): User's full name.
  - email (string, required): User's email address.
  - password (string, required): User's password.
- **Example Request**:

json

code

{

```
  "name": "John Doe",
  "email": "john.doe@example.com",
  "password": "securePassword123"
}
```

- **Example Response**:

json

code

```
{
  "message": "User registered successfully",
  "user": {
    "id": "abc123",
    "name": "John Doe",
    "email": "john.doe@example.com"
  }
}
```

**1.2 User Login**

- **Endpoint**: /api/auth/login
- **Method**: POST
- **Request Parameters**:
  - email (string, required): User's email.
  - password (string, required): User's password.
- **Example Request**:

json

code

```
{
  "email": "john.doe@example.com",
  "password": "securePassword123"
}
```

- **Example Response**:

```
json
code
{
  "message": "Login successful",
  "token": "jwt-token-string",
  "user": {
    "id": "abc123",
    "name": "John Doe",
    "email": "john.doe@example.com"
  }
}
```

---

**2. Flight APIs**

**2.1 Search Flights**

- **Endpoint**: /api/flights/search
- **Method**: GET
- **Query Parameters**:
    - origin (string, required): Departure city or airport.
    - destination (string, required): Arrival city or airport.

    - date (string, required): Travel date in YYYY-MM-DD format.
    - class (string, optional): Travel class (e.g., economy, business).

- **Example Request**:

sql

code

GET /api/flights/search?origin=NYC&destination=LAX&date=2024-12-01

- **Example Response**:

json

code


```
{
  "flights": [
    {
      "id": "flight123",
      "airline": "Delta Airlines",
      "origin": "NYC",
      "destination": "LAX",
      "departure_time": "2024-12-01T08:00:00Z",
      "arrival_time": "2024-12-01T11:00:00Z",
      "price": 300.00,
      "class": "economy"
    },
    {
```

```
    "id": "flight456",
    "airline": "United Airlines",
    "origin": "NYC",
    "destination": "LAX",
    "departure_time": "2024-12-01T10:00:00Z",
    "arrival_time": "2024-12-01T13:30:00Z",
    "price": 320.00,
    "class": "economy"
   }
 ]
}
```


**2.2 Get Flight Details**

- **Endpoint**: /api/flights/:id
- **Method**: GET
- **Path Parameter**:
  - id (string, required): Unique flight ID.
- **Example Request**:

bash

code

```
GET /api/flights/flight123
```

- **Example Response**:

json

code


```
{
```

```
  "id": "flight123",
  "airline": "Delta Airlines",
  "origin": "NYC",
  "destination": "LAX",
  "departure_time": "2024-12-01T08:00:00Z",
  "arrival_time": "2024-12-01T11:00:00Z",
  "price": 300.00,
  "class": "economy",
  "baggage_allowance": "15kg",
  "cancellation_policy": "Non-refundable"
}
```


---

### 3. Booking APIs

#### 3.1 Create a Booking

- **Endpoint**: /api/bookings
- **Method**: POST
- **Request Body**:
    - flight_id (string, required): ID of the flight to be booked.
    - user_id (string, required): ID of the user making the booking.
    - passenger_details (array, required): List of passenger names and contact details.
- **Example Request**:

json

code

```
{
  "flight_id": "flight123",
  "user_id": "abc123",
  "passenger_details": [
    {
      "name": "Jane Doe",
      "email": "jane.doe@example.com"
    }
  ]
}
```

- **Example Response**:

json

code

```
{
  "message": "Booking successful",
  "booking_id": "booking789",
  "flight_id": "flight123",
  "user_id": "abc123"
}
```

**3.2 View Booking Details**

- **Endpoint**: /api/bookings/:id
- **Method**: GET

- **Path Parameter**:
    - id (string, required): Unique booking ID.
- **Example Request**:

bash

code

GET /api/bookings/booking789

- **Example Response**:

json

code


```
{
  "booking_id": "booking789",
  "user_id": "abc123",
  "flight_id": "flight123",
  "passenger_details": [
    {
      "name": "Jane Doe",
      "email": "jane.doe@example.com"
    }
  ],
  "status": "Confirmed",
  "created_at": "2024-11-17T10:00:00Z"
}
```


**3.3 Cancel a Booking**

- **Endpoint**: /api/bookings/:id/cancel

- **Method**: PUT
- **Path Parameter**:
    - id (string, required): Unique booking ID.
- **Example Response**:

json

code

```
{
  "message": "Booking canceled successfully",
  "status": "Canceled"
}
```

---

**4. Error Handling**

All API responses include appropriate status codes:

- 200 OK: Successful request.
- 400 Bad Request: Missing or invalid parameters.
- 401 Unauthorized: Authentication required.
- 404 Not Found: Resource not found.
- 500 Internal Server Error: Server issues.

This documentation ensures developers can integrate and utilize the backend APIs efficiently.

# Authentication

1. **User Authentication Process:**
    - Login:
        - Users log in using their email/username and password.
        - The system verifies credentials against a secure database.
    - Token Issuance:
        - Upon successful authentication, a JSON Web Token (JWT) is generated.
        - The JWT contains essential claims like:
            - user_id: Unique identifier for the user.
            - role: Defines user access level (e.g., Admin, Customer, Guest).
            - iat and exp: Issued at and expiration times.
        - The JWT is signed with a secret key or asymmetric encryption (e.g., RSA).
2. **Token Management:**
    - Access Tokens:
        - Short-lived tokens (e.g., 15 minutes).
        - Used to authenticate API requests.
    - Refresh Tokens:
        - Longer-lived tokens (e.g., 7 days).
        - Stored securely (e.g., in HTTP-only cookies).
        - Allow users to obtain new access tokens without logging in again.

3. **Social Authentication (if enabled):**
    - Users can log in via third-party providers (e.g., Google, Facebook) using OAuth 2.0.
4. **Multi-Factor Authentication (MFA):**
    - For enhanced security, users can opt for MFA, using:
        - One-Time Passwords (OTPs) via SMS or email.
        - Authenticator apps like Google Authenticator.

---

# Authorization

1. **Role-Based Access Control (RBAC):**
    - Permissions are assigned based on roles:
        - Admin: Access to manage flights, bookings, and user accounts.
        - Customer: Access to search flights, manage bookings, and make payments.
        - Guest: Access to flight search and registration.
    - Role data is embedded in the JWT token.
2. **Resource-Based Authorization**:
    - Middleware checks the user's role and permissions before granting access to specific resources.
    - Example:
        - Admins can access /admin/flights.
        - Customers can access /bookings/{user_id} for their own bookings.

# Session Management

1. **Stateless Sessions**:
    - The application uses JWTs, which are stateless, to avoid server-side session storage.
    - Each request contains the token in the Authorization header (e.g., Bearer <token>).
2. **Stateful Sessions** (if implemented):
    - Optional for critical actions (e.g., payment process).
    - A session ID is stored in a secure database or Redis, tied to the user's current activity.
3. **Session Expiration**:
    - Tokens and sessions have defined lifetimes.
    - Inactivity results in token expiration and logout.

---

# Security Measures

1. **Token Storage**:
    - Access tokens are sent via HTTP headers.
    - Refresh tokens are stored in HTTP-only Secure Cookies to mitigate theft.
2. **Protecting Against Attacks**:
    - CSRF Protection:
        - Anti-CSRF tokens embedded in forms.
    - XSS Prevention:
        - Input sanitization and secure cookie settings.
    - Brute Force Protection:

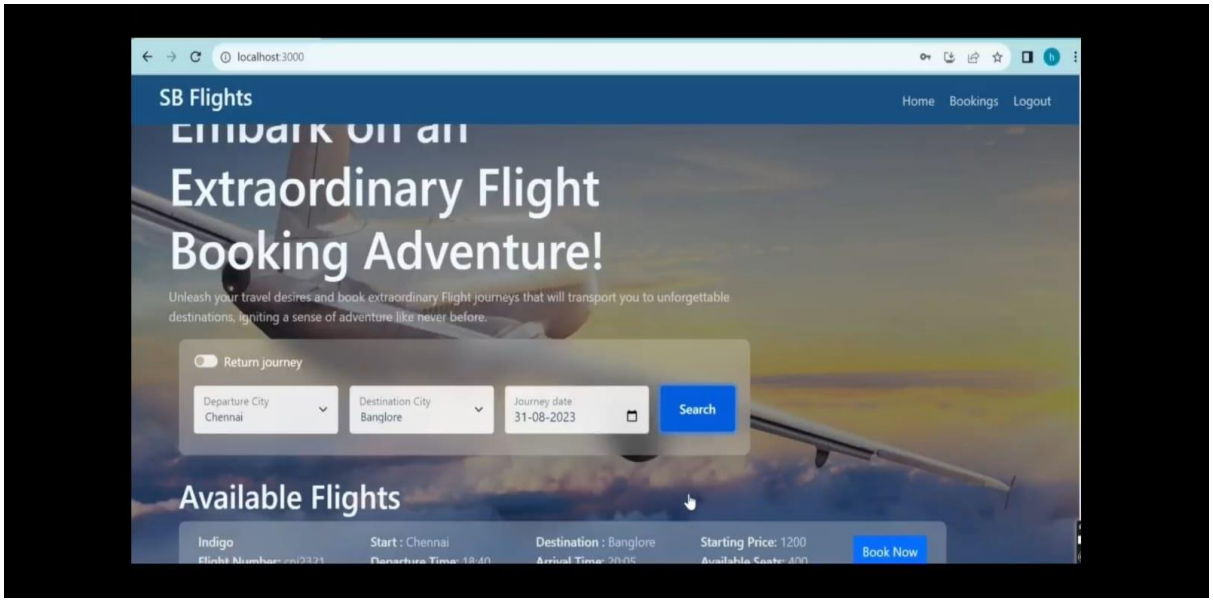
    - Rate-limiting and CAPTCHA during login attempts.
3. **Logout**:
    - Tokens are invalidated upon logout.
    - For stateful implementations, the session ID is deleted.

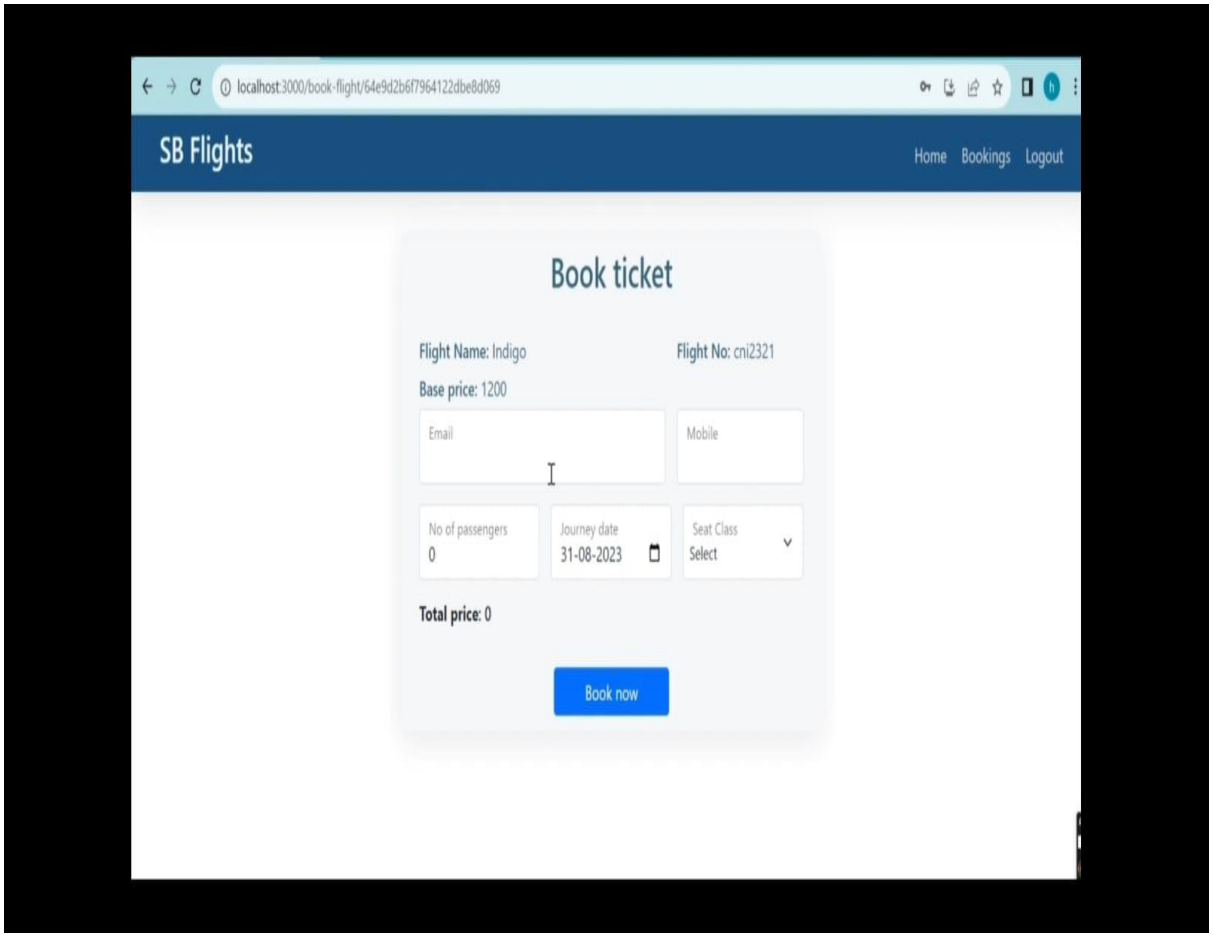
---

# API Integration

1. **Authentication Middleware**:
    - Validates tokens on every request.
    - Ensures expired or tampered tokens are rejected.
2. **API Gateway**:
    - Handles token verification before routing to microservices for flight management, booking, or payment.

# USER INTERFACE






**Components**:

- **Personal Information**: Display and edit personal data (name, email, phone number).
- **Booking History**: List of past and upcoming flight bookings with details.
- **Payment Methods**: Manage saved credit card or PayPal options.
- **Change Password** and **Security Settings** (e.g., MFA setup)

# Testing Strategy and Tools

To ensure the flight booking application is reliable, secure, and user-friendly, a comprehensive testing strategy is implemented. The strategy covers various testing phases, including functional, non-functional, and automated testing, using a mix of manual and automated tools.

---

**Testing Strategy**

**1. Unit Testing**

- **Objective**: Verify individual components (e.g., functions, modules) work as intended.
- **Scope**:
  - Validate core functionalities like flight search, fare calculations, and payment processing.
  - Mock dependencies to isolate testing for each module.
- **Tools**:
  - **JUnit** (Java), **PyTest** (Python), or **Jest** (JavaScript).
  - **Mockito** for mocking in Java.

---

**2. Integration Testing**

- **Objective**: Ensure that different modules or services interact correctly.
- **Scope**:
  - Test APIs between the frontend and backend.
  - Validate third-party integrations (e.g., payment gateways, airline APIs).

- **Tools**:
    - **Postman** or **SOAP UI** for API testing.
    - **Karate** for end-to-end API workflows.

---

### 3. System Testing

- **Objective**: Verify the application functions as a complete system.
- **Scope**:
    - Perform full booking workflow tests, including search, booking, and confirmation.
    - Validate edge cases like flight unavailability or payment failures.
- **Tools**:
    - Manual testing with predefined test cases.
    - Test management tools like **TestRail** or **Zephyr**.

---

### 4. User Interface (UI) Testing

- **Objective**: Ensure the UI is user-friendly, visually appealing, and functional.
- **Scope**:
    - Test the responsiveness of the UI on different devices and browsers.
    - Check alignment, buttons, and forms for proper functioning.
- **Tools**:
    - **Selenium** or **Cypress** for browser automation.

    - **BrowserStack** or **Sauce Labs** for cross-browser compatibility testing.

---

### 5. Performance Testing

- **Objective**: Assess the application's speed, scalability, and reliability under various load conditions.
- **Scope**:
    - Test flight search and booking processes under peak loads.
    - Monitor server response times and database query efficiency.
- **Tools**:
    - **JMeter** or **Gatling** for load testing.
    - **New Relic** or **Dynatrace** for application performance monitoring.

---

### 6. Security Testing

- **Objective**: Identify and mitigate vulnerabilities to ensure secure operations.
- **Scope**:
    - Test for SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
    - Validate token-based authentication and secure payment processes.
- **Tools**:
    - **OWASP ZAP** or **Burp Suite** for vulnerability scanning.
    - **SonarQube** for static code analysis.

---

### 7. Acceptance Testing

- **Objective**: Validate the application meets business requirements.
- **Scope**:
  - Work with stakeholders to confirm the application delivers the expected value.
  - Perform test scenarios covering all user journeys.
- **Tools**:
  - Manual testing with stakeholders.

---

### 8. Regression Testing

- **Objective**: Ensure new features or fixes do not break existing functionality.
- **Scope**:
  - Re-test core workflows like flight booking, payment, and notifications.
- **Tools**:
  - **Selenium**, **TestNG**, or **Cypress** for automated regression testing.

---

### 9. Mobile Testing

- **Objective**: Verify the application works smoothly on mobile devices.
- **Scope**:
  - Test the mobile version of the site and any native mobile apps.

    - Validate gestures, touch interactions, and mobile-specific features.
- **Tools**:
    - **Appium** for mobile automation.
    - **Xcode** (iOS) and **Android Studio** (Android) for device emulation.

---

**10. Usability Testing**

- **Objective**: Assess user satisfaction and ease of use.
- **Scope**:
    - Conduct tests with real users to gather feedback on navigation, clarity, and design.
- **Tools**:
    - **Maze** or **Hotjar** for usability testing and feedback collection.

---

**Testing Tools Summary**

| Category | Tools |
|---|---|
| **Unit Testing** | JUnit, PyTest, Jest, Mockito |
| **API Testing** | Postman, SOAP UI, Karate |
| **UI Testing** | Selenium, Cypress, BrowserStack |
| **Performance Testing** | JMeter, Gatling, New Relic |
| **Security Testing** | OWASP ZAP, Burp Suite, SonarQube |
| **Mobile Testing** | Appium, Xcode, Android Studio |

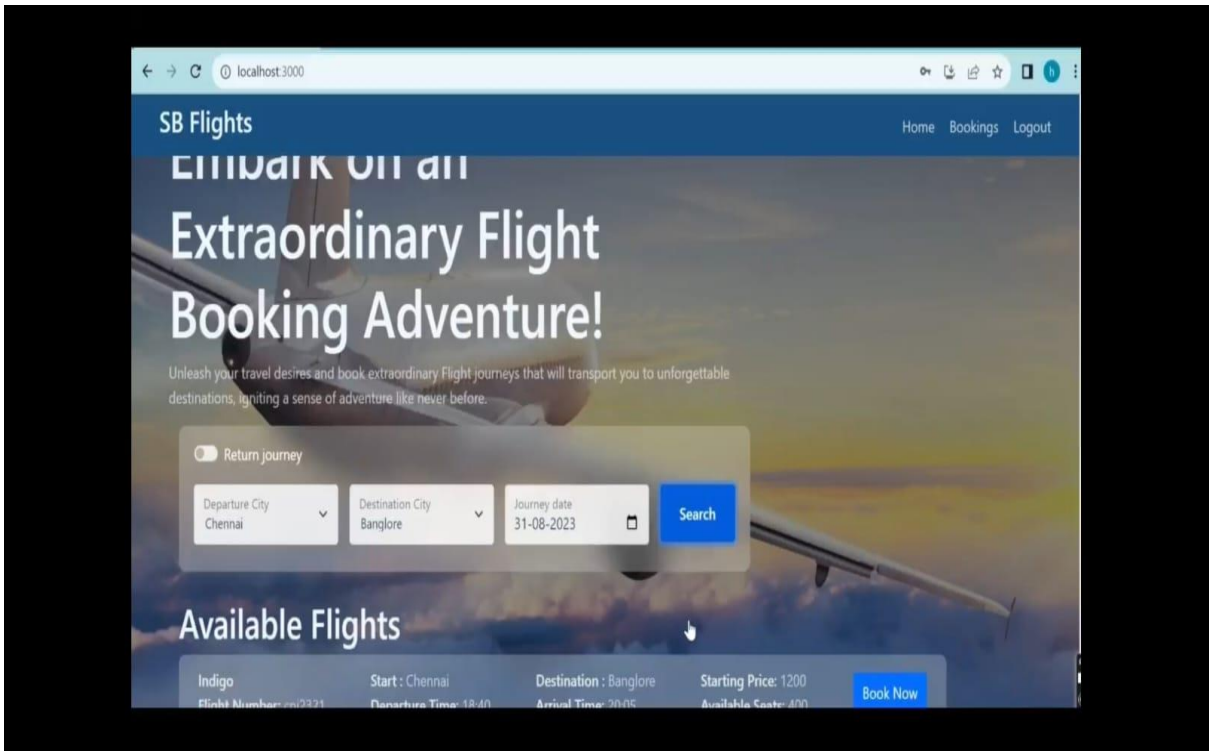
| Category | Tools |
| --- | --- |
| **Test Management** | TestRail, Zephyr |

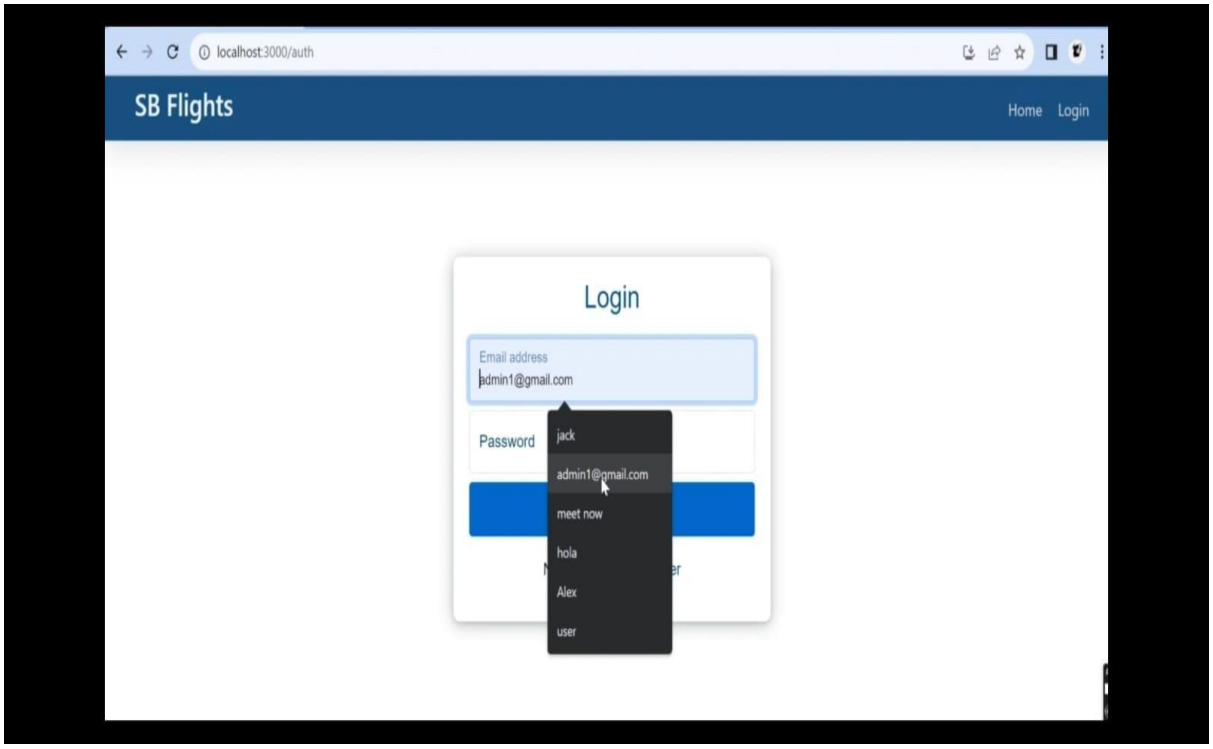
---

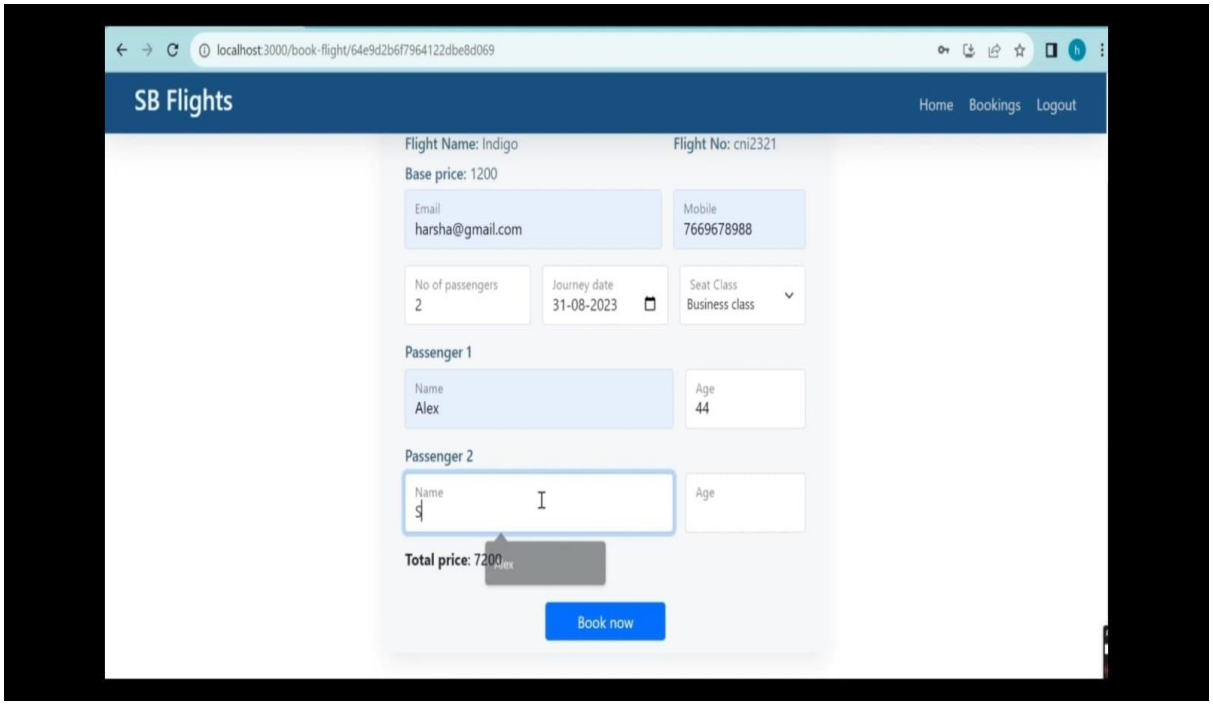
**Testing Workflow**

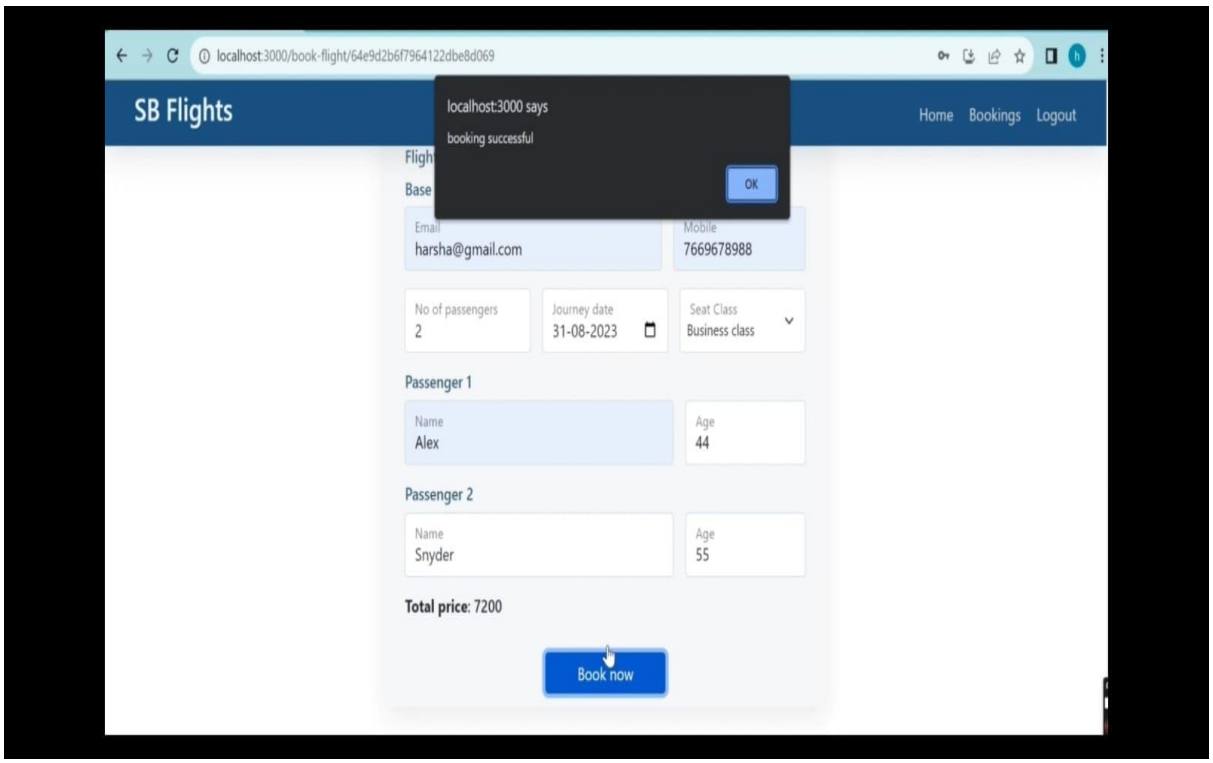
1. **Test Planning**:
    - Define test objectives, scope, and acceptance criteria.
2. **Test Development**:
    - Write test cases for functional and non-functional requirements.
3. **Test Execution**:
    - Execute test cases and record results.
4. **Bug Tracking**:
    - Report and track issues using tools like **JIRA** or **Bugzilla**.
5. **Test Automation**:
    - Automate repetitive tests to increase efficiency.
6. **Continuous Testing**:
    - Integrate tests into CI/CD pipelines using **Jenkins** or **GitLab CI**

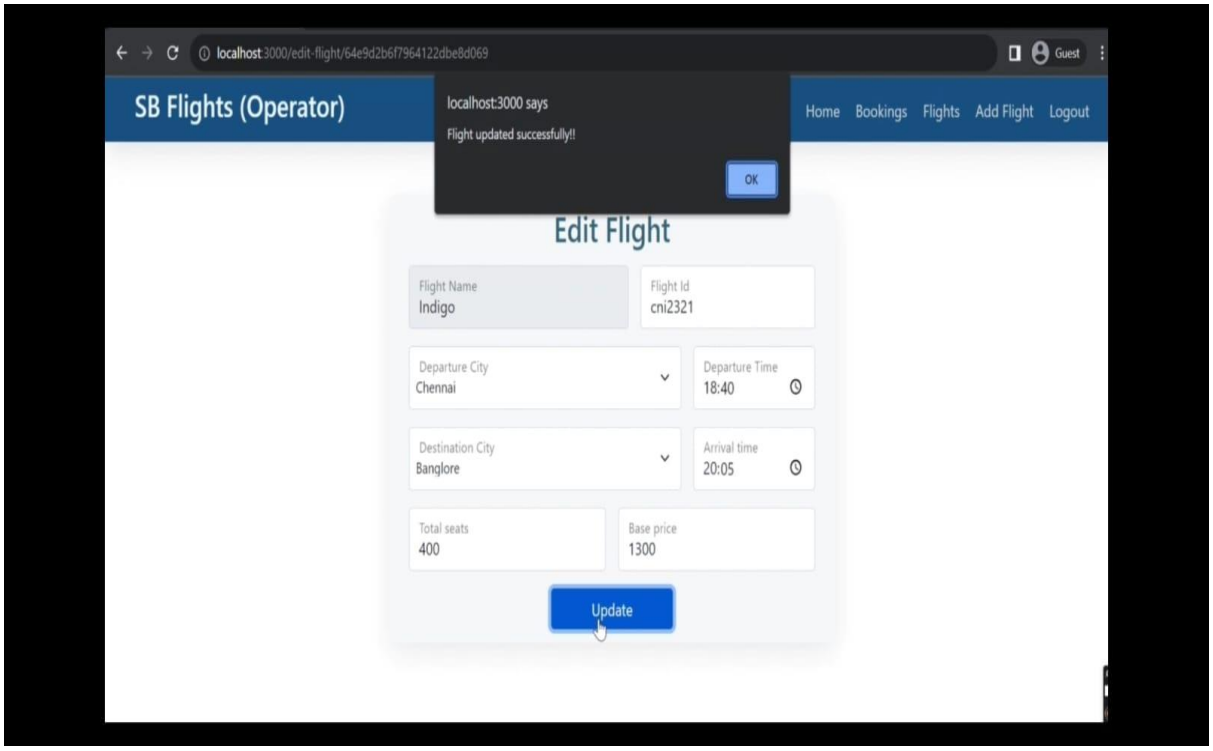
# SCREENSHOTS OR DEMOS




localhost:3000/book-flight/64e9d2b6f796122dbe8d069
SB Flights
Home
Bookings
Logout
Flight Name: Indigo
Flight No: cni2321
Base price: 1200
Email
harsha@gmail.com
Mobile
7669678988
No of passengers
2
Journey date
31-08-2023
Seat Class
Business class
Passenger 1
Name
Alex
Age
44
Passenger 2
Name
S
Age
Total price: 7200
Book now


localhost:3000/book-flight/64e9d2b6f796122dbe8d069
SB Flights
localhost:3000 says
booking successful
OK
Home
Bookings
Logout
Email
harsha@gmail.com
Mobile
7669678988
No of passengers
2
Journey date
31-08-2023
Seat Class
Business class
Passenger 1
Name
Alex
Age
44
Passenger 2
Name
Snyder
Age
55
Total price: 7200
Book now




# KNOWN ISSUES

### 1. Search Functionality

- **Issue**: Inconsistent results for searches with flexible dates.
    - **Details**: When users select a flexible date range, some flights are occasionally omitted from the results.
    - **Impact**: Users might not see all available options.
    - **Status**: Under investigation.

---

### 2. Booking Confirmation Delay

- **Issue**: Booking confirmation email delays.
    - **Details**: Emails with booking details sometimes take longer than expected (up to 30 minutes).

- **Impact**: Users may feel uncertain about the status of their booking.
- **Status**: Scheduled for fix in the next update.

---

### 3. Mobile Responsiveness

- **Issue**: Layout misalignment on smaller screens (e.g., iPhone SE, older Android devices).
  - **Details**: Flight details and buttons may overlap or appear off-screen.
  - **Impact**: Poor user experience on smaller devices.
  - **Status**: Fix in progress.

---

### 4. Payment Gateway Errors

- **Issue**: Payment failures with specific credit card providers.
  - **Details**: Payments made through certain providers are occasionally rejected due to unhandled API errors.
  - **Impact**: Users need to retry payments, causing inconvenience.
  - **Workaround**: Users can try alternative payment methods.
  - **Status**: High-priority fix in progress.

---

### 5. Session Timeout

- **Issue**: Unexpected session expiration during prolonged inactivity.
  - **Details**: Users filling lengthy forms (e.g., passenger details) are logged out without warning.

    - **Impact**: Loss of entered data; users need to start over.
    - **Workaround**: Save frequently during the process.
    - **Status**: Planned improvement to extend session duration and provide inactivity warnings.

---

### 6. Duplicate Bookings

- **Issue**: Duplicate bookings occur when users refresh the page during payment processing.
    - **Details**: The system occasionally processes duplicate transactions due to a lack of idempotency in API calls.
    - **Impact**: Users may be charged twice.
    - **Workaround**: Avoid refreshing the page during payments.
    - **Status**: High-priority fix under development.

---

### 7. Language Localization

- **Issue**: Incomplete translations in non-English versions.
    - **Details**: Some parts of the UI (e.g., error messages, button labels) remain in English despite selecting another language.
    - **Impact**: Confusion for non-English-speaking users.
    - **Status**: Pending updates to the localization files.

---

### 8. Flight Updates Notifications

- **Issue**: Real-time notifications about flight delays are inconsistent.

- **Details**: Notifications are delayed or not sent in some cases.
- **Impact**: Users may not be informed about critical updates.
- **Status**: Being optimized with the airline APIs.

---

## 9. Account Settings

- **Issue**: Password reset emails occasionally not received.
  - **Details**: Emails are marked as spam or not sent due to mail server misconfigurations.
  - **Impact**: Users unable to reset passwords promptly.
  - **Workaround**: Check spam folder or retry after some time.
  - **Status**: Configuring mail server for reliability.

---

## 10. Frequent Logout on iOS Devices

- **Issue**: Users on Safari browser are logged out frequently.
  - **Details**: Related to strict cookie policies on Safari.
  - **Impact**: Frustrating experience for users.
  - **Workaround**: Use Chrome or another browser until resolved.
  - **Status**: Under analysis

**NOTE:**

***Each issue has been logged in the bug-tracking system (e.g., JIRA) and assigned priority levels based on user impact. Developers are actively addressing these issues in the upcoming sprints, with critical fixes being expedited. Regular updates will be shared in release notes.***

# FUTURE ENHANCEMENT

To keep the flight booking application competitive, user-friendly, and feature-rich, the following enhancements have been identified for future development. These features aim to improve functionality, user experience, and scalability.

---

### 1. Advanced Search Options

- **Details**:
  - Add filters for airline preferences, layover durations, specific airports, and seat types.
  - Introduce flexible fare calendars to help users find cheaper flights.
- **Benefits**:
  - Enhances user control and personalization.
  - Makes the search process more intuitive and efficient.

---

### 2. Loyalty Program Integration

- **Details**:
  - Allow users to register for or link existing airline loyalty programs.
  - Provide reward points for bookings and enable points redemption for discounts or upgrades.
- **Benefits**:
  - Encourages repeat usage and builds customer loyalty.
  - Adds value for frequent flyers.

---

### 3. Multi-City and Open-Jaw Bookings

- **Details**:
  - Enable users to book multi-city itineraries (e.g., A → B → C) and open-jaw trips (e.g., A → B, then C → A).
- **Benefits**:
  - Attracts travelers with complex itineraries.
  - Provides a competitive edge over simpler booking systems.

---

### 4. Personalized Recommendations

- **Details**:
  - Use AI to suggest flights based on past searches, booking history, and preferences.
  - Offer alerts for price drops or promotions on frequently searched routes.
- **Benefits**:
  - Improves user engagement.
  - Helps users discover relevant deals.

---

### 5. Group Booking Support

- **Details**:
  - Introduce options for booking flights for large groups with bulk discounts.
  - Enable features like assigning seats together or splitting payments.
- **Benefits**:

- Attracts corporate clients and family travelers.
- Simplifies the booking process for larger parties.

---

### 6. In-App Travel Insurance

- **Details**:
  - Partner with insurance providers to offer travel insurance as an add-on during booking.
- **Benefits**:
  - Provides users with a one-stop solution for travel needs.
  - Generates additional revenue.

---

### 7. Multi-Language and Multi-Currency Support

- **Details**:
  - Expand the application to support additional languages and currencies.
  - Implement real-time currency conversion for pricing.
- **Benefits**:
  - Attracts global users.
  - Enhances accessibility for non-English speakers.

---

### 8. Enhanced Mobile App Features

- **Details**:
  - Add offline access for booking history and boarding passes.
  - Include push notifications for flight updates, deals, and reminders.

- **Benefits**:
    - Improves the mobile user experience.
    - Keeps users engaged and informed.

---

### 9. Sustainable Travel Options

- **Details**:
    - Highlight eco-friendly airlines or flights with lower carbon footprints.
    - Allow users to contribute to carbon offset programs directly during booking.
- **Benefits**:
    - Appeals to environmentally conscious travelers.
    - Demonstrates corporate responsibility.

---

### 10. Chatbot and Voice Assistance

- **Details**:
    - Implement AI-powered chatbots for 24/7 customer support.
    - Enable voice-based booking and support using assistants like Alexa, Google Assistant, or Siri.
- **Benefits**:
    - Improves accessibility and user convenience.
    - Reduces reliance on human customer support agents.

---

### 11. Dynamic Pricing Insights

- **Details**:

    - Provide users with insights into fare trends and predictions based on historical data.
    - Alert users about optimal times to book flights.
- **Benefits**:
    - Helps users make informed decisions.
    - Builds trust and user loyalty.

---

### 12. Multi-Platform Sync

- **Details**:
    - Enable seamless syncing between the desktop, mobile app, and other devices.
    - Provide features like resuming an incomplete booking on another device.
- **Benefits**:
    - Ensures a consistent experience across platforms.
    - Boosts user retention.

---

### 13. Partner Services Integration

- **Details**:
    - Integrate hotel bookings, car rentals, and local tours into the platform.
    - Offer discounted packages for booking multiple services together.
- **Benefits**:
    - Expands the app into a comprehensive travel platform.
    - Increases cross-selling opportunities.

---

### 14. Real-Time Seat Selection

- **Details**:
  - Allow users to view and choose available seats in real-time during booking.
  - Add options for upgrading seats or purchasing additional legroom.
- **Benefits**:
  - Enhances user satisfaction.
  - Provides additional revenue opportunities.

---

### 15. Improved Security Measures

- **Details**:
  - Implement biometric login (e.g., fingerprint or facial recognition).
  - Enhance fraud detection mechanisms for payments.
- **Benefits**:
  - Boosts user trust in the platform.
  - Reduces risk of fraudulent transactions.

---

### 16. Comprehensive Feedback and Support System

- **Details**:
  - Add a user-friendly feedback form for reporting issues or suggesting improvements.
  - Implement a ticketing system for tracking customer support requests.

- **Benefits**:
    - Improves user satisfaction by addressing concerns efficiently.
    - Helps prioritize future updates based on user feedback.