FLIGHT BOOKING WEB APP USING MERN STACK

FlightGo Documentation

Table of Contents

1.Introduction

FlightGo is a flight booking application designed to simplify the process of searching, booking, and managing flights. The app provides a user-friendly interface for customers to plan their travel and make flight reservations in just a few steps.

Purpose
The purpose of the FlightGo app is to help users:

- Search for flights from multiple airlines.
- Compare prices and availability.
- Securely book and manage reservations.
- Receive notifications and reminders about bookings.

Target Audience
FlightGo is designed for travelers who are looking for a hassle-free way to book flights for their personal or business trips.

2.Features

Core Features

- Flight Search: Search for flights based on source, destination, travel dates, and number of passengers.
- Price Comparison: Compare flight prices from multiple airlines.
- Booking: Select flight options and complete bookings securely.
- User Profile: Create, update, and manage user profiles.
- Booking History: View and manage past bookings.
- Notifications: Get alerts for flight confirmations, cancellations, or changes.
- Payment Integration: Support for credit card, debit card, and third-party payment options (e.g., PayPal).
- Flight Status: Check real-time flight status and updates.
- Reviews and Ratings: Rate airlines and share feedback.

Additional Features

- Multilingual Support: Available in various languages to cater to global users.
- Multiple Currencies: Supports various currencies for payment.
- Seat Selection: Choose seats based on airline availability.

---

3. Architecture

FlightGo follows a client-server architecture with the following key components:

- Frontend (Client-side): The user interface is built with Flutter for cross-platform support (Android and iOS).
- Backend (Server-side): The backend is built with Node.js and Express, providing RESTful APIs for handling flight searches, bookings, user authentication, and payment processing.
- Database: PostgreSQL for storing user data, flight details, bookings, payment transactions, and reviews.
- Payment Gateway: Integration with Stripe/PayPal for processing payments.

---

4. Tech Stack

- Frontend:
  - Flutter (for cross-platform mobile app development)
- Backend:
  - Node.js with Express (for API development)
- Database:
  - PostgreSQL (for data storage)
- Payment Gateway:
  - Stripe / PayPal
- Authentication:

- JWT (JSON Web Tokens) for secure authentication
- Hosting/Deployment:
    - AWS (Amazon Web Services) or Heroku for deployment
- Third-party Services:
    - Flight API providers (e.g., Skyscanner API) for retrieving flight data

---

5. User Stories

As a User, I want to:

1. Search for flights: Enter departure and arrival locations, select travel dates, and search for available flights.
2. Compare prices: View flight options from multiple airlines and compare prices.
3. Book a flight: Select a flight and complete the booking process, including seat selection and payment.
4. View my booking: See my flight details, status, and past bookings.
5. Cancel or modify bookings: Modify or cancel a booking if needed, with clear instructions on refund policies.
6. Leave reviews: Rate the airline and leave feedback based on the flight experience.

---

6. UI/UX Design

The app follows a minimalistic and intuitive design:

- Home Screen: Prominent flight search bar, recent bookings, and promotional banners.
- Flight Search Results: List of flights with filters for airlines, price, and flight duration.
- Booking Screen: Detailed flight options, seat selection, and payment process.
- Profile Screen: User profile information and booking history.
- Notifications: Alerts for flight changes, promotions, and booking confirmations.

---

7. API Endpoints

Authentication

- `POST /api/auth/signup`: User registration
- `POST /api/auth/login`: User login
- `POST /api/auth/logout`: User logout

Flights

- `GET /api/flights`: Search for flights based on query parameters (source, destination, dates, etc.)
- `GET /api/flights/{id}`: Retrieve details of a specific flight
- `POST /api/flights/book`: Create a booking for a selected flight

User

- `GET /api/user/{id}`: Retrieve user profile
- `PUT /api/user/{id}`: Update user profile
- `GET /api/user/bookings`: Get user's booking history

Payment

- `POST /api/payment`: Process payment for flight booking

---

8. Authentication & Security

- JWT Authentication: Users are authenticated using JWT tokens. The token is sent with each API request to ensure secure access.
- Password Hashing: User passwords are hashed using bcrypt for security.
- HTTPS: All communication between the client and server is encrypted using HTTPS.

---

9. Error Handling

- Client-Side:
  - Inform users of validation errors (e.g., invalid flight search input or payment issues).
  - Display loading indicators and handle timeouts gracefully.
- Server-Side:
  - Return appropriate HTTP status codes (e.g., 400 for bad requests, 404 for not found, 500 for server errors).
  - Use try-catch blocks to handle unexpected errors and return user-friendly error messages.

---

10. Testing

Unit Testing

- Unit tests will be written using Jest for both the frontend and backend components to ensure individual functions work as expected.

Integration Testing

- End-to-end tests will be conducted to verify that flight searches, bookings, and payments work seamlessly.

UI/UX Testing

- A/B testing and user feedback will be gathered to ensure that the app is intuitive and user-friendly.

---

11. Future Enhancements

- AI-based Flight Recommendations: Use machine learning to recommend personalized flights based on past travel history.
- Chatbot Integration: Implement a chatbot for customer service and flight queries.
- Loyalty Programs: Integrate a rewards system for frequent flyers.