FLIGH BOOKING APP

Team members

- Mahalashmi.M
- Pavan Kumar.A.k
- Priyadharshini .D
- Punithavathy.S

Purpose

The purpose of a flight booking app is to enable users to search, compare, and book flights easily and efficiently, providing a seamless experience for planning and managing air travel. It simplifies the process of booking, tracking, and managing flight itineraries while offering added services like price comparison, payment options, and travel updates.

Features

- Flight Search and Comparison: Allows users to search for flights based on various criteria (destination, dates, budget) and compare prices across different airlines.
- Booking and Payment: Users can book flights directly through the app, enter passenger details, and securely process
- payments.Itinerary Management: Provides a clear view of flight details, booking confirmation, and allows users to modify or cancel bookings if needed.
- Notifications and Alerts: Sends real-time updates about flight status, delays, cancellations, or gate changes to keep travelers informed.

Architecture

The architecture of a flight booking app can be divided into Frontend and Backend components, each responsible for different aspects of the application.

Frontend Architecture

The frontend is the part of the app that interacts directly with the user. It includes everything users see and interact with on their devices.

User Interface (UI):

- Screens: Home page, flight search page, booking page, user profile, payment screen, etc.
- Frameworks/Technologies: React Native, Flutter, or native Android (Java/Kotlin) and iOS (Swift/Objective-C).

Components:

- Flight Search: Input fields for destination, dates, number of passengers.
- Results Display: A list of available flights with price, timing, and airline details.
- Booking Form: User enters details such as name, passport info, and payment method.
- Payment Gateway Integration: Secure integration with payment processors like Stripe, PayPal, etc.

State Management:

- State Management: Redux, Context API (for React Native/React apps) or Provider/Bloc (for Flutter apps) to manage app state, such as user details, search results, and booking status.

API Calls:

Communicates with the backend to fetch flight data, prices, availability, etc.

Uses REST or GraphQL API for real-time data fetching.

Backend Architecture

The backend handles data storage, business logic, authentication, and communication with external services like airline APIs or payment gateways.

Server:

- Web Server: A web server (e.g., Apache, Nginx) to manage client requests and responses.
- API Layer: RESTful or GraphQL APIs to interact with the frontend and other services.

Services:

- Flight Data Provider: Integration with third-party flight data providers (e.g., Amadeus, Skyscanner, or directly with airlines) to get real-time flight information.
- User Management: Authentication, registration, and profile management (via OAuth, JWT tokens, or Firebase Authentication).
- Payment Gateway Integration: Integration with services like Stripe or PayPal for secure payment processing.

Database:

- SQL/NoSQL Database: Store user data, booking history, preferences, payment transactions, etc. (e.g., MySQL, PostgreSQL, MongoDB).

Booking Engine:

- Booking Logic: Manages availability, booking confirmations, cancellations, seat selection, etc.
- Price Calculation: Algorithms that calculate ticket prices, including taxes, discounts, and promotions.

Notification Service:

- Real-Time Updates: Send push notifications or emails for booking confirmations, flight updates, or promotions using services like Firebase, Twilio, or AWS SNS.

Cloud Infrastructure:

- Cloud Hosting: Servers and databases hosted on platforms like AWS, Google Cloud, or Microsoft Azure for scalability and reliability.

Caching and Load Balancing:

- Caching: Use tools like Redis or Memcached to cache frequently accessed flight data and reduce database load.
- Load Balancing: Ensure app performance under high traffic conditions.

Interaction Between Frontend and Backend

- The frontend makes HTTP requests (GET, POST) to the backend to fetch flight data, book tickets, and manage user profiles.

APIs send and receive data in JSON format between frontend and backend.

- The backend processes requests, communicates with external flight data providers and payment systems, and responds with necessary data, which is displayed in the app frontend.
- This layered architecture helps ensure scalability, security, and efficiency in delivering a smooth flight booking experience.

DATABASE AND PREREQUISITE

For a flight booking app, several key database components and prerequisites are required to ensure efficient operation. Below is a detailed explanation of the database structure and prerequisites, formatted for a 10-mark answer:

Database Structure

The primary goal of the database is to manage and store all data related to flights, customers, bookings, payments, and more. The following tables and relationships are commonly used:

Flight Information Table

Columns: flight_id, airline_name, departure_airport, arrival_airport, departure_time, arrival_time, available_seats, price, etc.

Purpose: Stores details about individual flights.

User (Customer) Table

Columns: user_id, first_name, last_name, email, phone_number, address, password_hash, etc.

Purpose: Stores personal and account information for registered users.

Booking Table

Columns: booking_id, user_id, flight_id, booking_date, status, seat_number, payment_status, etc.

Purpose: Records customer bookings for specific flights.

Payment Table

Columns: payment_id, booking_id, payment_date, amount, payment_method, payment_status, etc.

Purpose: Stores payment transactions related to bookings.

Airplane Table (Optional)

Columns: airplane_id, airline_id, capacity, model, etc.

Purpose: Contains details of the airplanes used for flights, linked to airlines.

Airport Table

Columns: airport_id, airport_name, location, country, IATA_code, etc.

Purpose: Contains information about the airports.

Customer Reviews Table (Optional)

Columns: review_id, user_id, flight_id, rating, review_text, etc.

Purpose: Allows customers to leave reviews and ratings for flights.

1. Relationships Between Tables

Users ↔ Bookings: A one-to-many relationship where a user can have multiple bookings.

Bookings ↔ Flights: A many-to-one relationship where many bookings can correspond to a single flight.

Bookings ↔ Payments: A one-to-one relationship where each booking is associated with a specific payment.

Airports ↔ Flights: A many-to-many relationship, as a flight can involve multiple airports (departure and arrival).

3. Prerequisites for Developing the Flight Booking App

Technologies and Tools

Backend Framework: Use frameworks such as Node.js, Django, or Spring Boot for API development.

Database Management System (DBMS): Choose a relational database like MySQL, PostgreSQL, or a NoSQL database like MongoDB for certain non-relational data storage.

Payment Gateway: Integrate a third-party payment service such as Stripe or PayPal for secure online payments.

Authentication: Implement user authentication using technologies like OAuth, JWT, or Firebase Authentication.

User Interface Design

Frontend Framework: Use React Native or Flutter for building cross-platform mobile apps, or ReactJS/VueJS for web-based interfaces.

Search Functionality: Implement a flight search system that queries flight details based on user input (date, origin, destination, etc.).

Security Considerations

Data Encryption: Ensure sensitive user data like passwords and payment information are encrypted using protocols like SSL/TLS.

GDPR/Privacy Compliance: Adhere to data privacy laws such as GDPR for handling user data, particularly payment and personal information. Testing and Deployment

- Unit Testing: Implement testing for API endpoints and core functionality.
- CI/CD: Set up Continuous Integration and Continuous Deployment pipelines for efficient development and deployment.

Additional Features

- Push Notifications: For booking confirmations, reminders, or special offers.
- Flight Status Tracking: Integration with real-time flight status APIs to inform users of delays or cancellations.

INSTALATION

Download the App: Go to your device's app store (Google Play for Android or App Store for iOS). Search for the specific flight booking app (e.g., Expedia, Kayak, Skyscanner).

Install the App: Click on the "Install" or "Get" button to download and install the app onto your device. Once the download is complete, the app will appear on your home screen or app menu.

After installation, you can open the app, create an account or log in, and start booking flights.

FLODER STRUCTURE

The folder structure of a flight booking app typically follows a modular design for maintainability and scalability. Here's a basic example:

```
/flight-booking-app

│

├── /assets          # Images, icons, fonts, and other static resources

│

├── /components      # Reusable UI components (e.g., buttons, cards)

│

├── /config          # Configuration files (e.g., API endpoints, environment settings)

│

├── /models          # Data models (e.g., Flight, User, Booking)

│

├── /pages           # App pages/screens (e.g., Home, Search, Booking, Profile)

│

├── /services        # API services (e.g., Flight API, User API)

│

├── /store           # State management (e.g., Redux, Context API)

│

├── /styles          # CSS/SCSS files or style-related components

│

└── /utils           # Utility functions (e.g., date formatting, validation)
```

API DOCUMENTATION

Search Flights

Endpoint:
`GET /api/flights/search`

Description:
Search for available flights based on criteria such as departure, destination, date, and passengers.

Query Parameters:

- `departure_city` (string) - City code or name for departure location.
- `arrival_city` (string) - City code or name for arrival location.
- `departure_date` (string) - Date of departure in `YYYY-MM-DD` format.
- `return_date` (optional, string) - Date of return in `YYYY-MM-DD` format.
- `num_passengers` (integer) - Number of passengers.

Response:

```
json
Copy code
{
  "status": "success",
  "data": [
    {
      "flight_id": "123",
      "departure_time": "2024-12-01T08:00:00Z",
      "arrival_time": "2024-12-01T12:00:00Z",
      "price": 299.99,
      "airline": "AirExample",
      "seat_availability": 50
    },
    {
      "flight_id": "124",
      "departure_time": "2024-12-01T15:00:00Z",
      "arrival_time": "2024-12-01T19:00:00Z",
      "price": 320.50,
      "airline": "FlyGlobal",
      "seat_availability": 75
    }
  ]
}
```

2. Book a Flight

Endpoint:
`POST /api/flights/book`

Description:
Book a flight by providing passenger details and selecting a flight.

Request Body:

```
json
Copy code
{
  "flight_id": "123",
  "passenger_details": [
    {
      "first_name": "John",
      "last_name": "Doe",
      "email": "john.doe@example.com",
      "passport_number": "X1234567"
    }
  ],
  "payment_details": {
    "card_number": "4111111111111111",
    "expiration_date": "12/25",
    "cvv": "123"
  }
}
```

Response:

```
json
Copy code
{
  "status": "success",
  "booking_reference": "ABC123XYZ",
  "total_price": 299.99
}
```

3. Get Booking Details

Endpoint:
`GET /api/bookings/{booking_reference}`

Description:
Retrieve details of a booking using the booking reference.

Path Parameters:

- `booking_reference` (string) - Booking reference identifier.

Response:

```
json
Copy code
{
  "status": "success",
  "booking_reference": "ABC123XYZ",
  "flight_id": "123",
  "passenger_details": [
    {
      "first_name": "John",
      "last_name": "Doe",
      "email": "john.doe@example.com",
      "passport_number": "X1234567"
    }
  ],
  "total_price": 299.99,
  "status": "Confirmed"
}
```

4. Cancel a Booking

Endpoint:
`DELETE /api/bookings/cancel/{booking_reference}`

Description:
Cancel an existing booking.

Path Parameters:

- `booking_reference` (string) - Booking reference identifier.

Response:

```
json
Copy code
{
  "status": "success",
  "message": "Booking has been successfully cancelled."
}
```

5. Get Available Seats on a Flight

Endpoint:
`GET /api/flights/{flight_id}/seats`

Description:
Retrieve the available seats for a specific flight.

Path Parameters:

- `flight_id` (string) - Unique identifier of the flight.

Response:

```
json
Copy code
{
  "status": "success",
  "flight_id": "123",
  "available_seats": [
    {
      "seat_number": "12A",
      "class": "Economy",
      "price": 299.99
    },
    {
      "seat_number": "12B",
      "class": "Economy",
      "price": 299.99
    }
  ]
}
```

Authentication

- All endpoints require an **API Key** passed as a header.
- **Authorization:** `Bearer <API_KEY>`

This basic documentation provides a structure for key operations in a flight booking system. Depending on your app's features, you may need to add more specific endpoints such as managing user profiles, payment gateways, or detailed flight status information.

4o mini

Feature enhance

To enhance a flight booking app and make it stand out in the market, here are two key features that could significantly improve user experience and increase engagement:

AI-Powered Personalized Recommendations: Integrate machine learning algorithms to suggest flights based on the user's past travel history, preferences, and budget. This could include destination suggestions, customized offers, and optimal flight times, making the booking process more seamless and tailored to individual needs.

Real-Time Flight Status & Notifications: Provide real-time flight updates on delays, cancellations, gate changes, and weather disruptions. Push notifications and alerts can keep users informed about their flight's status, ensuring a smoother travel experience.

These features would make the app more intuitive and user-centric, improving overall satisfaction and loyalty.