Dokumentacja Projektu Flutter – **Fly Finder**(Aplikacja Wyszukiwania Lotów)

1. Wstęp

Projekt został zaprojektowany i zaimplementowany jako aplikacja mobilna do wyszukiwania lotów, korzystając z API Amadeus. Aplikacja pozwala użytkownikom na wyszukiwanie lotów na podstawie kryteriów takich jak miejsce wylotu, miejsce przylotu, data wylotu, data powrotu i liczba dorosłych pasażerów.

Wybrane problemy, z którymi się zmierzyłem podczas rozwoju projektu

1. Zarządzanie tokenem dostępu

Podczas tworzenia aplikacji napotkałem problem z szybko wygasającymi tokenami dostępu, co powodowało błędy autoryzacji przy próbach wykonania żądań do API. Aby zaradzić temu problemowi, zaimplementowałem mechanizm odświeżania tokena, który automatycznie odnawia token przed jego wygaśnięciem, zapewniając ciągłość dostępu do API.

2. Rozdzielenie lotów bezpośrednich i pośrednich

W odpowiedziach z API loty były przedstawiane razem, bez wyraźnego rozróżnienia między lotami bezpośrednimi i pośrednimi. Zdecydowałem się wdrożyć funkcję w flight_search_page.dart, która segreguje loty na podstawie liczby przesiadek. Dzięki temu użytkownicy mogą łatwiej znaleźć loty dostosowane do swoich preferencji.

3. Eliminacja duplikatów lotów

Zauważyłem, że API czasami zwraca duplikaty tej samej oferty lotu. Wprowadziłem funkcję filtrującą w module AmadeusApi, która porównuje loty i eliminuje powtarzające się wpisy przed ich wyświetleniem użytkownikowi. To usprawniło prezentację ofert i poprawiło doświadczenia użytkowników.

4. Optymalizacja żądań podczas wpisywania danych przez użytkownika

Kluczowym wyzwaniem było zbyt częste wysyłanie żądań do API podczas wpisywania nazw miast w formularzu wyszukiwania. Aby zminimalizować to ryzyko, stworzyłem metodę debounce, która opóźnia wysyłanie żądania do momentu, gdy użytkownik przestanie wpisywać tekst przez określony czas. Dzięki temu uniknąłem niepotrzebnego obciążenia serwera oraz możliwych opłat za przekroczenie limitu żądań do API.

Wdrożenie tych rozwiązań pozwoliło mi na poprawę funkcjonalności i wydajności aplikacji, jednocześnie zapewniając lepszą obsługę błędów i bardziej intuicyjne doświadczenie użytkownika.

1. Architektura:

Aplikacja składa się z kilku kluczowych komponentów:

Main.dart: Plik główny, który inicjuje aplikację i ładuje główną stronę.

flight_search_page.dart: Strona, która zawiera logikę interfejsu użytkownika dla formularza wyszukiwania lotów oraz wyświetlania wyników.

flight_card.dart: Widget reprezentujący wizualną reprezentację lotu w wynikach wyszukiwania.

amadeus_api.dart: Moduł odpowiedzialny za komunikację z API Amadeus, w tym autentykację i odbieranie danych o lotach.

2. Szczegóły Techniczne

main.dart

Inicjalizuje aplikację Flutter poprzez runApp, ustawiając MyApp jako widget root. MyApp tworzy MaterialApp, który z kolei ładuje **FlightSearchPage** jako domyślny ekran startowy.

flight_search_page.dart

Zawiera formularz do wprowadzania danych przez użytkownika i przycisk do wyszukiwania. Po wprowadzeniu danych i zatwierdzeniu formularza, dane są przekazywane do AmadeusApi do wyszukiwania lotów, a wyniki są wyświetlane przy użyciu FlightCard.

flight_card.dart

Prezentuje kluczowe informacje o locie, takie jak linie lotnicze, godzina odlotu/przylotu, czas trwania lotu, itp. Dane są sformatowane i wyświetlane w czytelny sposób.

amadeus_api.dart

Odpowiada za zarządzanie wszystkimi żądaniami sieciowymi do API Amadeus. W tym module generowany jest token dostępu i przeprowadzane są wyszukiwania lotów na podstawie parametrów przekazanych przez użytkownika.

3. Integracja z API Amadeus

1. Autentykacja

API Amadeus używa standardu OAuth 2.0 dla autoryzacji żądań. Do uzyskania tokena dostępu, aplikacja musi najpierw wykonać żądanie POST na endpoint:

Endpoint do uzyskania tokena:

POST https://test.api.amadeus.com/v1/security/oauth2/token

Parametry żądania:

client_id: Identyfikator klienta otrzymany od Amadeus.

client_secret: Sekret klienta otrzymany od Amadeus.

grant_type: Typ grantu.

2 . Wyszukiwanie lotu

Po pomyślnym uzyskaniu tokena, można użyć go do wyszukiwania ofert lotów poprzez endpoint:

Endpoint do wyszukiwania ofert lotów:

GET https://test.api.amadeus.com/v2/shopping/flight-offers

Parametry żądania:

originLocationCode: Kod IATA lotniska wylotu.

destinationLocationCode: Kod IATA lotniska docelowego.

departureDate: Data wylotu (format YYYY-MM-DD).

returnDate: Data powrotu (format YYYY-MM-DD).

adults: Liczba dorosłych pasażerów.

max: Maksymalna liczba ofert lotów do zwrotu.

3. Obsługa odpowiedzi

Odpowiedzi z API są zwracane w formacie JSON. Każda odpowiedź zawiera kluczowe informacje o dostępnych lotach, takie jak:

id: Unikalny identyfikator oferty.

itineraries: Szczegóły dotyczące każdego z segmentów podróży, w tym daty i godziny wylotów/przylotów.

price: Informacje o cenie za całą podróż.

carrierCodes: Kody linii lotniczych operujących na trasie.

4. Obsługa błędów

API Amadeus zwraca różne kody błędów HTTP, które powinny być odpowiednio obsługiwane w aplikacji:

400 Bad Request: Błędne żądanie (np. brakujące wymagane parametry).

401 Unauthorized: Problem z autentykacją lub tokenem dostępu.

500 Internal Server Error: Błąd po stronie serwera Amadeus.

4. **Bezpieczeństwo**

Klucze API są obecnie zahardkodowane w kodzie, co stanowi potencjalne zagrożenie bezpieczeństwa. Zaleca się przeniesienie ich do zmiennych środowiskowych lub bezpiecznego magazynu(pliku).

5. Obsługa Błędów

Każde żądanie sieciowe ma podstawową obsługę błędów, ale zaleca się rozwinięcie tej logiki o bardziej szczegółowe informacje o błędach oraz dodanie logowania błędów dla łatwiejszego debugowania.

6. Przyszłe Rozszerzenia

-Rozszerzenie funkcjonalności o dodatkowe filtry wyszukiwania (np. klasy lotów, przewoźnicy).

-Rozważam użycie podejścia takiego jak Provider, Bloc lub Redux dla lepszego zarządzania stanem, jeśli aplikacja ma rosnąć lub rozszerzać się w przyszłości.

-Dodanie testów jednostkowych i integracyjnych dla zapewnienia jakości kodu.

6. Konkluzja

Podsumowując, projektowanie i rozwój mojej aplikacji do wyszukiwania lotów- Fly Finder przyniosło szereg wyzwań, które wymagały kreatywnego podejścia do rozwiązywania problemów, takich jak zarządzanie tokenami, segregacja i filtrowanie danych oraz optymalizacja żądań API. Dzięki temu doświadczeniu nie tylko poprawiłem funkcjonalność aplikacji, ale także zwiększyłem jej wydajność i użyteczność dla użytkowników. Mam nadzieję, że wnioski i rozwiązania opisane w tej dokumentacji posłużą jako cenny zasób dla wszystkich, którzy będą rozwijać ten projekt w przyszłości.

Autor dokumentacji: Igor Kalinowski
Data ostatniej aktualizacji: 27.04.2024

Dziękuję za korzystanie z tej dokumentacji. W razie pytań lub potrzeby dalszej dyskusji, proszę o kontakt.