

POLITECHNIKA BIAŁOSTOCKA

Wydział Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

TEMAT: Aplikacja mobilna wspierająca odkrywanie lokalnych doświadczeń podróžniczych z wykorzystaniem sztucznej inteligencji

WYKONAWCA: Juliusz Mandrosz

OPIEKUN PRACY DYPLOMOWEJ : dr inż. Tomasz Grzes

BIAŁYSTOK 2025 ROK

Subject of diploma thesis

A mobile application supporting the discovery of local travel experiences using artificial intelligence

Summary

The main goal of this project is to develop a mobile application that uses Artificial Intelligence to help users discover local and authentic travel experiences.

An analysis of existing applications has shown that most of them are designed for a broad audience of tourists, which results in recommendations that focus mainly on the most popular and mainstream attractions.

The technologies used in this project include Flutter and Dart for building the mobile application, Firebase for database management, user authentication, and infrastructure, as well as Node.js and TypeScript for developing and running serverless cloud functions. Additionally, the OpenAI API is used to generate personalized recommendations for users.

In the fourth chapter, I describe the implementation of the project, starting with Firestore database modeling. Then, I present the application's architecture, state management, integration with third-party services, and testing.

The fifth chapter focuses on the complete user flow of the application, beginning with onboarding for new users and the option to generate recommendations based on selected preferences.

Spis treści

1. Wstęp.....	1
2. Analiza wymagań	2
2.1. Przedstawienie problemu	2
2.2. Analiza istniejących rozwiązań	4
2.3. Wymagania funkcjonalne.....	8
2.4. Wymagania niefunkcjonalne.....	8
3. Technologie wykorzystane w projekcie.....	9
3.1. Flutter	9
3.2. Dart	10
3.3. Firebase.....	11
3.4. Node.js.....	12
3.5. TypeScript	13
3.6. OpenAI API.....	14
4. Implementacja aplikacji	15
4.1. Model bazy danych	15
4.2. Moduły aplikacji.....	18
4.3. Struktura modułów.....	20
4.4. Zarządzanie stanem aplikacji	21
4.5. Integracja z Firebase Firestore	25
4.6. Integracja z Firebase Authentication.....	27
4.7. Integracja z Cloud Functions i OpenAI API	30
4.8. Testowanie aplikacji	35
5. Prezentacja aplikacji.....	38
5.1. Onboarding użytkownika	39
5.2. Logowanie do aplikacji.....	46
5.3. Wyszukiwanie rekomendacji zakwaterowania.....	48
5.4. Przeglądanie zapisanych rekomendacji.....	52
6. Podsumowanie	54
Literatura.....	56
Spis rysunków.....	56

1. Wstęp

Podejście do tworzenia oprogramowania zmienia się z każdym rokiem. Jeszcze 25 lat temu większość stron internetowych składała się z samego HTML, były one statyczne i pozbawione interakcji z użytkownikiem, a także wymagały odświeżania w celu załadowania nowej treści. Interfejsy były surowe, składały się głównie się z tekstu i obrazków, nie były responsywne, zazwyczaj dostosowane do jednego rozmiaru ekranu.

Z biegiem lat, wraz z pojawiением się CSS i JavaScriptu oraz rosnącej świadomości znaczenia UX (*user experience*), a także rozwojem technologii frontendowych i rozpowszechnieniem frameworków takich jak chociażby jQuery, strony internetowe stały się bardziej przyjazne dla użytkowników. Statyczne treści przekształciły się w dynamiczne aplikacje webowe, które umożliwiły płynne korzystanie ze strony bez potrzeby przeładowania, a prosty i monotonny interfejs ewoluował w atrakcyjne i intuicyjne środowisko, dostosowane do potrzeb odwiedzających.

Wraz z rozwojem sztucznej inteligencji, uczenia maszynowego oraz analizy danych, współczesne aplikacje definiują doświadczenia użytkownika na nowo i wynoszą je na poziom, który jeszcze jakiś czas temu był nie do osiągnięcia lub wymagał dużego budżetu. Tworzenie spersonalizowanych aplikacji, które odpowiadają na potrzeby użytkowników w czasie rzeczywistym i przewidują ich aktualne potrzeby jest prostsze niż kiedykolwiek.

Celem mojej pracy jest opracowanie aplikacji mobilnej wspierającej odkrywanie lokalnych doświadczeń podróżniczych, wykorzystującej sztuczną inteligencję. Praca ta ma również na celu zaprezentowanie, w jaki sposób sztuczna inteligencja może zostać wykorzystana do zapewnienia wysokiego poziomu doświadczenia użytkownika poprzez odpowiadanie na jego aktualne potrzeby.

2. Analiza wymagań

W procesie tworzenia oprogramowania kluczowym etapem jest zebranie wymagań przed rozpoczęciem implementacji. W tym rozdziale skupię się na problemie, który rozwiązuje aplikacja LocAdvisor, oraz przeanalizuję istniejące rozwiązania rynkowe, które docierają do podobnej niszy. Ponadto przedstawię wymagania funkcjonalne i niefunkcjonalne, jakie powinna spełniać aplikacja.

2.1. Przedstawienie problemu

Podczas podróżowania do nowego miejsca wiele osób nie wie, co robić ani gdzie się udać. Wyszukiwanie atrakcji często prowadzi do najbardziej popularnych lokalizacji oraz pułapek turystycznych, gdzie ceny są nieproporcjonalnie wysokie w stosunku do jakości. Turyści często trafiają do zatłoczonych miejsc, które nie odzwierciedlają autentycznego charakteru danego regionu. Tymczasem wielu podróżnych woli unikać takich obszarów i odkrywać miejsca uczęszczane przez mieszkańców, aby poczuć prawdziwy klimat lokalizacji. Chcą doświadczać tego, czego przeciętny turysta nigdy nie zobaczy.

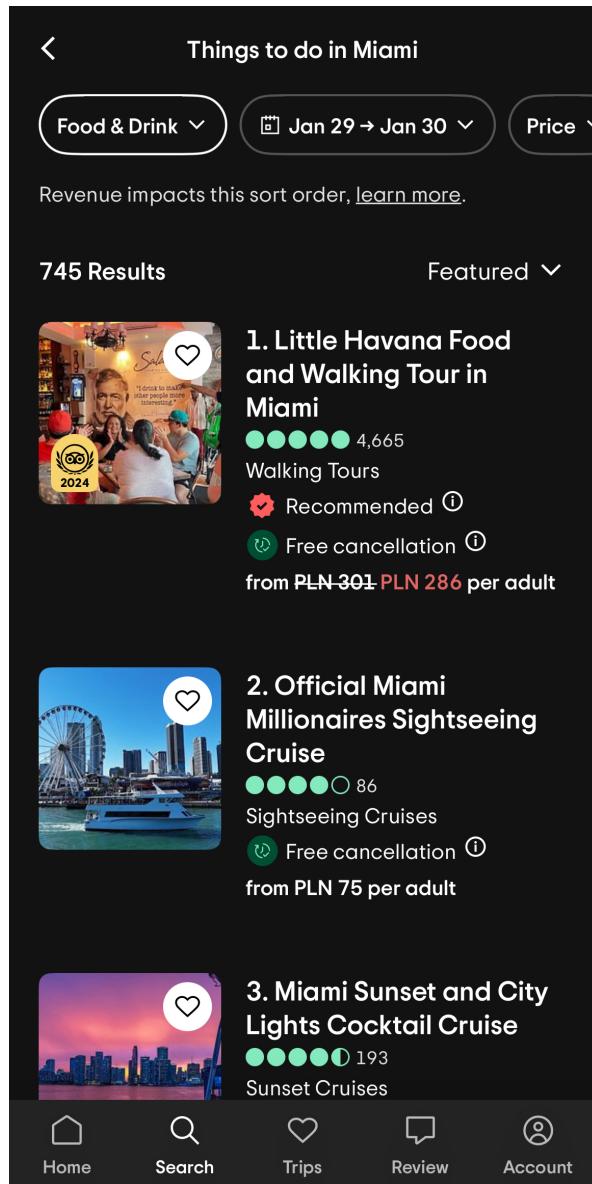
Podobne podejście dotyczy zakwaterowania, zamiast zatrzymywać się w dużych, turystycznych resortach, ludzie coraz częściej wybierają mniej popularne dzielnice, by poczuć się, jakby naprawdę tam mieszkali. Niestety, wiąże się to z kilkoma wyzwaniami. Popularne aplikacje turystyczne są skierowane do szerokiej grupy odbiorców, przez co polecają głównie popularne atrakcje i miejsca, pomijając mniej oczywiste, ale warte odwiedzenia lokalizacje. W efekcie podróżni często muszą szukać informacji na własną rękę, na przykład pytając przypadkowych przechodniów czy przeglądając fora internetowe. Takie podejście bywa nieoptimalne, ponieważ osoby udzielające porad mogą mieć zupełnie inne gusta i potrzeby niż poszukujący.

Eksplorowanie na własną rękę wiąże się z ryzykiem, zwłaszcza w mniej bezpiecznych krajach, gdzie brak odpowiedniej wiedzy może skutkować nieprzyjemnymi sytuacjami. Przykładowo, ktoś wynajmie mieszkanie na Airbnb w San Juan w Portoryko, na osiedlu Ocean Park w pobliżu plaży, nie zdając sobie sprawy, że tuż obok znajduje się osiedle Luis Llorens Torres, którego należy unikać, jeśli nie zna się tamtejszych mieszkańców. Podobnie zwiedzanie malowniczego osiedla La Perla nocą może być

niebezpieczne. Choć to miejsce, w którym nagrano teledysk do utworu „Despacito” Luisa Fonsiego i tamtejsi mieszkańcy są z reguły otwarci na turystów, nieświadome robienie zdjęć na głównej ulicy może skończyć się poważnymi problemami.

W związku z tym pojawia się potrzeba narzędzi, które umożliwią podróżnym odkrywanie mniej popularnych, autentycznych miejsc w bezpieczny sposób, dostosowanych do ich indywidualnych preferencji i potrzeb.

2.2. Analiza istniejących rozwiązań



Rysunek 2.1 Aplikacja TripAdvisor

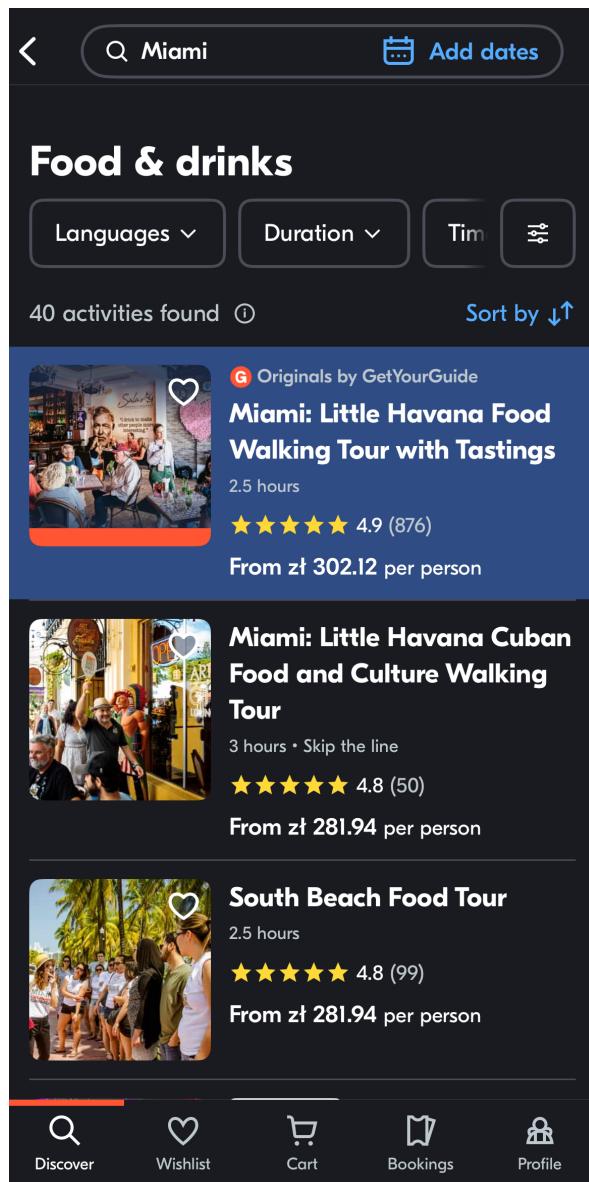
Źródło: [TripAdvisor](#)

Aplikacja TripAdvisor (Rysunek 2.1) umożliwia wyszukiwanie atrakcji w wybranym mieście, oferując różne kategorie, które pomagają użytkownikowi w szybkim znalezieniu interesujących miejsc. Interfejs użytkownika jest przejrzysty i intuicyjny, a dostęp do recenzji innych użytkowników ułatwia podejmowanie decyzji.

Pomimo że aplikacja świetnie spełnia swoje funkcje biznesowe i jest popularnym wyborem na rynku, jest skierowana do bardzo szerokiej grupy odbiorców. W efekcie

skupia się głównie na promowaniu popularnych atrakcji turystycznych, które zazwyczaj nie odzwierciedlają autentycznego klimatu danego miejsca i są rzadko odwiedzane przez lokalnych mieszkańców. W przeciwieństwie do tego, LocAdvisor oferuje rekomendacje, które pozwalają użytkownikowi zanurzyć się w lokalnym charakterze odwiedzanej lokalizacji.

Dodatkowo, personalizacja w TripAdvisor jest ograniczona i opiera się głównie na predefiniowanych filtrach. Użytkownik nie ma możliwości dodania własnych preferencji, co ogranicza szansę na dopasowanie rekomendacji do jego unikalnych potrzeb i stylu podrózowania.

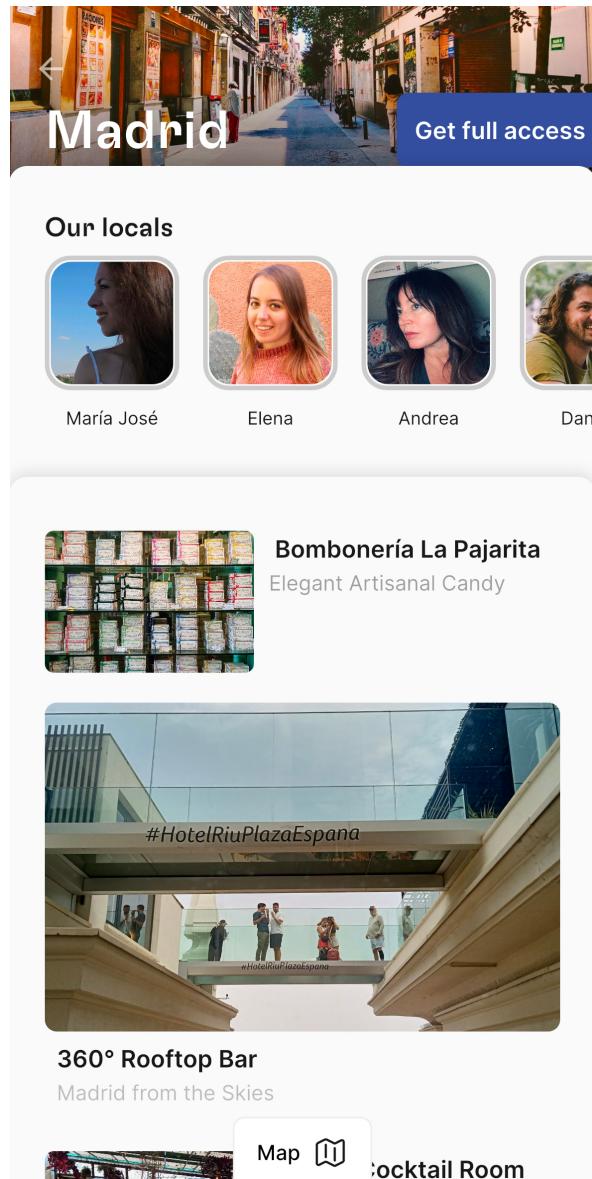


Rysunek 2.2 Aplikacja GetYourGuide

Źródło: [GetYourGuide](#)

Kolejną popularną aplikacją jest GetYourGuide (Rysunek 2.2), która koncentruje się przede wszystkim na sprzedaży biletów na popularne atrakcje turystyczne. Aplikacja oferuje przejrzysty interfejs oraz szeroki wybór atrakcji, co sprawia, że jest łatwa w obsłudze i przyjazna dla użytkownika. Jednak, podobnie jak w przypadku innych aplikacji skierowanych do szerokiej grupy odbiorców, promowane są głównie najpopularniejsze i najbardziej turystyczne miejsca. W rezultacie brakuje w niej propozycji bardziej

autentycznych i lokalnych doświadczeń, które są kluczowym elementem wyróżniającym LocAdvisor.



Rysunek 2.3 Aplikacja Spotted by Locals

Źródło: [Spotted by Locals](#)

Aplikacją, która stara się rozwiązać problem oferowania jedynie najbardziej popularnych atrakcji turystycznych, jest Spotted by Locals (Rysunek 2.3). Oferuje ona rekomendacje sprawdzonych miejsc przygotowanych przez zweryfikowanych

mieszkańców, dzięki czemu użytkownik może odkrywać atrakcje i miejsca oddające autentyczny klimat danego miasta.

Jednakże aplikacja posiada istotne wady, personalizacja jest bardzo ograniczona, co sprawia, że użytkownik nie może dopasować rekomendacji do swoich indywidualnych preferencji. Dodatkowo baza danych jest stosunkowo uboga, co skutkuje niewielką liczbą dostępnych miast, co ogranicza jej zastosowanie.

2.3. Wymagania funkcjonalne

Wymagania funkcjonalne, jakie powinna spełniać aplikacja LocAdvisor, obejmują:

- In
- Generowanie rekomendacji miejsc zakwaterowania zgodnie z podanymi wymaganiami.
- Logowanie do aplikacji przy użyciu adresu e-mail i hasła.
- Rejestracja nowego użytkownika za pomocą adresu e-mail i hasła.
- Możliwość resetowania hasła poprzez wysłanie linku na podany adres e-mail.
- Przeglądanie zapisanych rekomendacji w aplikacji.
- Filtrowanie zapisanych rekomendacji według nazwy docelowej lokalizacji.
- Usuwanie zapisanych rekomendacji.

2.4. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne, jakie powinna spełniać aplikacja LocAdvisor, obejmują:

- System operacyjny - Android 6.0+ lub iOS 15.0+.
- Internet - wymagane połączenie.
- Bezpieczeństwo - bezpieczne przechowywanie danych użytkowników.
- Dostępność - aplikacja powinna być dostępna przez większość czasu.
- Wydajność - płynne działanie na wspieranych urządzeniach.
- Responsywność - dostosowanie do różnych ekranów.
- Dostępność - interfejs aplikacji powinien być intuicyjny.

3. Technologie wykorzystane w projekcie

W tym rozdziale opiszę technologie użyte w projekcie oraz ich zastosowanie.

3.1. Flutter



Rysunek 3.1 Logo frameworka Flutter

Źródło: <https://flutter.dev/brand>

Flutter to framework umożliwiający tworzenie aplikacji mobilnych, webowych i desktopowych z wykorzystaniem jednej bazy kodu. Dzięki funkcji *hot reload* pozwala na szybkie iterowanie i testowanie zmian podczas pisania kodu. Aplikacje we Flutterze są napisane w języku Dart, co zapewnia wysoką wydajność i elastyczność.

W odróżnieniu od innych rozwiązań cross-platformowych, takich jak React Native, Flutter korzysta z własnego silnika renderowania, co eliminuje potrzebę używania natywnych komponentów systemowych. Dzięki temu osiąga wydajność zbliżoną do natywnej oraz pełną kontrolę nad wyglądem interfejsu[10].

Ze względu na te zalety, Flutter został wykorzystany do stworzenia aplikacji mobilnej LocAdvisor.

3.2. Dart



Rysunek 3.2 Logo języka Dart

Źródło: <https://dart.dev/brand>

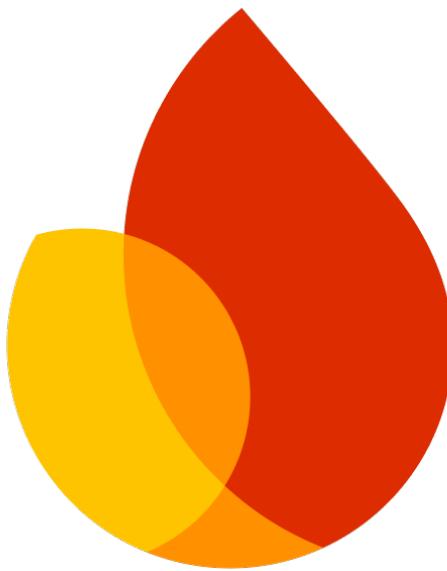
Dart to silnie typowany, zorientowany obiektowo język programowania. Posiada automatyczne zarządzanie pamięcią. Umożliwia tworzenie szybkich aplikacji działających na dowolnej platformie. Jest wykorzystywany m.in. we frameworku Flutter. Technologia kompilatora Darta pozwala na uruchomienie kodu na różne sposoby:

- Platforma natywna - dla aplikacji mobilnych i desktopowych, Dart oferuje zarówno maszynę wirtualną (Dart VM) z komplikacją *just-in-time* (JIT), jak i kompilator *ahead-of-time* (AOT) do generowania kodu maszynowego.
- Platforma webowa - dla aplikacji webowych kompilatory tłumaczą kod na JavaScript lub WebAssembly.

Maszyna wirtualna Dart VM oferuje kompilator *just-in-time*, co pozwala na szybkie przeładowanie (*hot-reload*).

Gdy aplikacje są gotowe do wdrożenia produkcyjnego, kompilator *ahead-of-time* może skompilować kod do natywnego kodu maszynowego ARM lub x64. Aplikacja skompilowana za pomocą AOT uruchamia się szybko, zapewniając krótszy czas startu[11].

3.3. Firebase



Rysunek 3.3 Logo Firebase

Źródło: <https://firebase.google.com/brand-guidelines>

Firebase to platforma typu *Backend as a Service* (BaaS), będąca częścią Google Cloud, która upraszcza zarządzanie infrastrukturą. Umożliwia szybkie prototypowanie i wdrażanie aplikacji.

Jednymi z kluczowych komponentów Firebase są:

- Cloud Firestore - nierelacyjna baza danych obsługująca *real-time data*, umożliwiająca natychmiastową synchronizację danych z aplikacją.
- Firebase Authentication - gotowe rozwiązanie do zarządzania użytkownikami, umożliwiające logowanie za pomocą maila, Google oraz wielu innych opcji.
- Cloud Functions - bezserwerowe, skalowalne funkcje, pozwalające na wykonywanie operacji w chmurze.

- Firebase Storage - usługa do przechowywania plików, np. zdjęć.
- Firebase Hosting - szybki hosting dla aplikacji webowych.

Firebase posiada świetną integrację z Flutterem, dzięki oficjalnym bibliotekom, które dostarczają kompletne SDK (*software development kit*). Te narzędzia znaczco ułatwiają pisanie kodu, redukując ilość kodu potrzebnego do implementacji kluczowych funkcji aplikacji[7].

3.4. Node.js



Rysunek 3.4 Logo Node.js

Źródło: <https://nodejs.org/en/about/branding>

Node.js to środowisko uruchomieniowe JavaScript, które pozwala na uruchamianie kodu poza przeglądarką. Jest powszechnie stosowane do tworzenia aplikacji backendowych, zapewniając wysoką wydajność i asynchroniczne przetwarzanie żądań.

3.5. TypeScript



Rysunek 3.5 Logo języka TypeScript

Źródło: [https://www.typescriptlang.org/
branding/](https://www.typescriptlang.org/branding/)

TypeScript to nadzbiór języka JavaScript, który wprowadza system typów, pozwalając na wykrywanie i unikanie wielu błędów już na etapie pisania kodu. Jest wszechstronny i znajduje zastosowanie w tworzeniu aplikacji webowych, mobilnych oraz backendowych, dzięki środowiskom uruchomieniowym takim jak Node.js czy Deno. TypeScript umożliwia transpilację kodu do JavaScript, co zapewnia lepsze doświadczenie programistyczne, a jednocześnie pozwala na jego uruchamianie w przeglądarkach lub innych środowiskach wspierających JavaScript[6].

3.6. OpenAI API



Rysunek 3.6 Logo OpenAI

Źródło: <https://openai.com/brand/>

OpenAI to firma zajmująca się rozwojem sztucznej inteligencji, znana z tworzenia zaawansowanych modeli językowych (LLM), takich jak GPT-4. Dzięki udostępnionemu API możliwe jest budowanie aplikacji o wysokim poziomie personalizacji. Pozwala to na przesyłanie do modelu sparametryzowanych zapytań, a w odpowiedzi uzyskiwanie szczegółowych i dopasowanych wyników[9].

4. Implementacja aplikacji

W tym rozdziale przedstawię implementację aplikacji, uwzględniając modelowanie bazy danych, projektowanie architektury, zarządzanie stanem, integrację z zewnętrznymi usługami, a także testowanie.

4.1. Model bazy danych

Baza danych wykorzystana w projekcie to nierelacyjna baza Firebase Firestore, zaprojektowana zgodnie z wymaganiami interfejsu oraz założeniami biznesowymi. Można ją przedstawić w formie obiektów JSON.

```
▼ activity_requests: [] 1 item
  ▼ 0: {} 9 keys
    id: "8LItgaCu62GSVsGdd4Fm"
    userId: "Y4XZgoXCs8f4F4U2o0eflT3iJ3K3"
    destination: "Miami"
    ▼ activityPreferences: [] 2 items
      0: "Jedzenie"
      1: "Życie nocne"
      additionalNotes: ""
      atmosphereOption: "Tętniące życiem"
      budgetOption: "Średnie"
      dateOption: "Na dzisiaj"
      createdAt: "January 19, 2025 at 5:42:47 PM UTC+1"
```

Rysunek 4.1 Model *activity_requests*

Kolekcja *activity_requests* (Rysunek 4.1) przechowuje historię wprowadzonych parametrów wyszukiwania aktywności przez użytkowników (Rysunek 5.4). Nie ma bezpośredniego zastosowania w aplikacji, jednak może być przydatna do powiązania konkretnych rekomendacji z parametrami wyszukiwania, zwłaszcza gdy okaże się, że nie są one do końca trafne. Może również służyć do celów analitycznych np. identyfikacji najczęściej wyszukiwanych aktywności, a także do tworzenia lepszych personalizacji dla

poszczególnych użytkowników. Każdy dokument w tej kolekcji zawiera referencję do użytkownika poprzez *userId*, co pozwala powiązać historię wyszukiwania z konkretną osobą.

```
▼ activity_recommendations: [] 1 item
  ▼ 0: {} 8 keys
    id: "HLwI8XgKyN8oFZx4B43m"
    requestId: "8LItgaCu62GSVsGdd4Fm"
    userId: "Y4XZgoXC8f4F4U2o0efLT3iJ3K3"
    destination: "Miami"
    destinationLowerCase: "miami"
    additionalNotes: "Zacznij wieczór od kolacji w Batch Gastropub, następnie wybierz się na Rosa Sky na before, a potem baw się w Space Club."
    createdAt: "January 19, 2025 at 5:42:47 PM UTC+1"
  ▼ activities: [] 1 item
    ▼ 0: {} 6 keys
      placeName: "Space Club"
      description: "Legendarny klub w Miami z całonocnymi imprezami, występami topowych DJ-ów i niepowtarzalnym klimatem."
      bestTimeToVisit: "Najlepiej po godzinie 1:00, kiedy klub jest pełen energii."
      safetyTips: "Korzystaj z Ubera lub Lyft po zakończeniu imprezy."
      combinationTips: "Po klubie warto odwiedzić food trucki w pobliżu na późnonocną przekąskę"
      budgetTips: "Średni – wejście od $40, drinki od $15. Możesz pobrać aplikację Dice i sprawdzić dostępne imprezy oraz ceny biletów."
```

Rysunek 4.2 Model *activity_recommenations*

Kolekcja *activity_recommendations* (Rysunek 4.2) przechowuje wygenerowane rekomendacje aktywności. Służą one do wyświetlania użytkownikowi dopasowanych propozycji miejsc i atrakcji, zgodnych z jego preferencjami. Przykładem wykorzystania tej kolekcji jest ekran listy rekomendacji aktywności (Rysunek 5.18 i 5.19), zawierający szczegółowe informacje o sugerowanych aktywnościach, takie jak opis miejsca, najlepszy czas na wizytę, wskazówki dotyczące bezpieczeństwa oraz orientacyjne koszty. Każda rekomendacja zawiera odniesienie do *activity_requests*, na podstawie którego została wygenerowana poprzez *requestId*, co umożliwia analizę zależności między danymi wejściowymi a otrzymanymi wynikami. Dodatkowo każda rekomendacja posiada referencję do użytkownika poprzez *userId*, co pozwala wyświetlać sugestie dla konkretnego użytkownika.

```
▼ accommodation_requests: [] 1 item
  ▼ 0: {} 8 keys
    id: "1vU0agZEKwBIkRFQpg9p"
    userId: "Y4XZgoXCs8f4F4U2o0eflT3iJ3K3"
    destination: "Miami"
    ▼ locationPreferences: [] 3 items
      0: "Blisko plaży lub natury"
      1: "Blisko centrum"
      2: "Blisko knajpekk i kawiarni"
    additionalNotes: ""
    atmosphereOption: "Spokojnie"
    budgetOption: "Średnie"
    createdAt: "January 19, 2025 at 5:46:47 PM UTC+1"
```

Rysunek 4.3 Model accommodation_requests

Kolekcja *accommodation_requests* (Rysunek 4.3) przechowuje historię wprowadzonych parametrów wyszukiwania rekomendacji przez użytkowników (rysunek 5.15). Tak jak w przypadku *activity_requests* nie ma bezpośredniego zastosowania w aplikacji, jednak może być wykorzystana w celach analitycznych. Każdy dokument w tej kolekcji zawiera referencję do użytkownika poprzez *userId*, co pozwala powiązać historię wyszukiwania z konkretną osobą.

```

▼ accommodation_recommendations: [] 1 item
  ▼ 0: {} 8 keys
    id: "C29g5P8mxTVDoWSVvaTS"
    requestId: "1vU0agZEKwBIkRFQpg9p"
    userId: "Y4XZgoXCs8f4F4U2o0eflt3iJ3K3"
    destination: "Miami"
    destinationLowerCase: "miami"
    createdAt: "January 19, 2025 at 5:46:47 PM UTC+1"
    additionalNotes: "Miami oferuje wiele autentycznych miejsc, ale warto unikać turystycznych pułapek, takich jak Ocean Drive na South Beach."
    ▼ locations: [] 1 item
      ▼ 0: {} 6 keys
        placeName: "North Beach"
        description: "North Beach to spokojna okolica w Miami, idealna dla osób szukających relaksu blisko plaży. Znajdują się tu lokalne restauracje i kawiarnie oferujące autentyczne smaki."
        localVibe: "Autentyczny klimat Miami, z dala od tłumów South Beach. Popularne wśród lokalnych mieszkańców."
        safetyTips: "Bezpieczna okolica, ale unikaj spacerów w mniej oświetlonych miejscach w nocy."
        transportTips: "Najlepiej poruszać się rowerem lub korzystać z usług Uber/Lyft. W okolicy kursują również autobusy miejskie, które łączą North Beach z centrum Miami."
        budgetTips: "Ceny zakwaterowania są średnie, z dużą liczbą opcji Airbnb. Lokalne restauracje oferują przystępne ceny w porównaniu do turystycznych miejsc w South Beach."

```

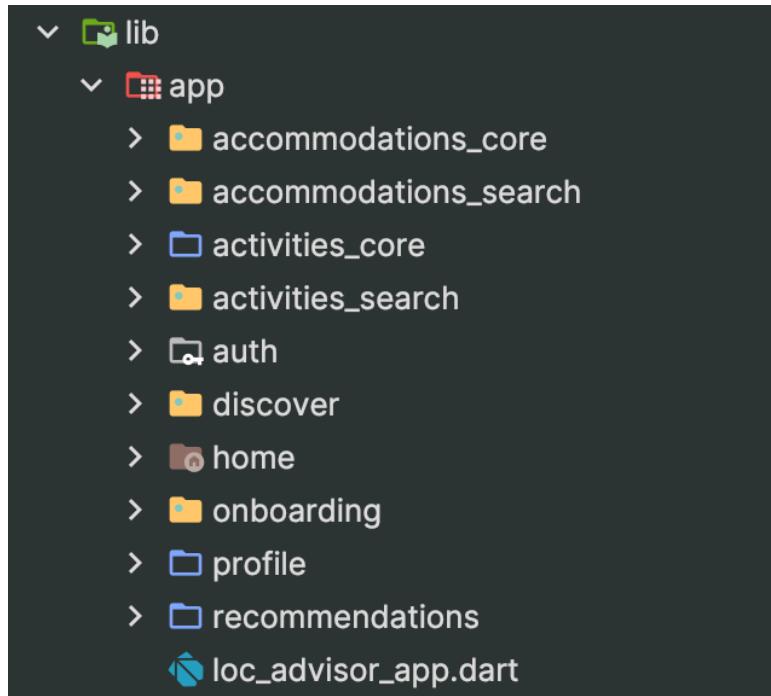
Rysunek 4.4 Model *accommodation_recommendations*

Kolekcja *accommodation_recommendations* (Rysunek 4.4) przechowuje wygenerowane rekomendacje zakwaterowania. Służą one do wyświetlania użytkownikowi dopasowanych propozycji miejsc, zgodnych z jego preferencjami. Przykładem wykorzystania tej kolekcji jest ekran wygenerowanych rekomendacji zakwaterowania (rysunek 5.16 i 5.17), zawierający szczegółowe informacje o sugerowanych zakwaterowaniach, takie jak opis miejsca, lokalny klimat, wskazówki dotyczące bezpieczeństwa oraz orientacyjne koszty. Każda rekomendacja zawiera odniesienie do *accommodation_requests*, na podstawie którego została wygenerowana poprzez *requestId*, co umożliwia analizę zależności między danymi wejściowymi a otrzymanymi wynikami. Dodatkowo każda rekomendacja posiada referencję do użytkownika poprzez *userId*, co pozwala wyświetlać sugestie dla konkretnego użytkownika.

4.2. Moduły aplikacji

“If you think good architecture is expensive, try bad architecture” - Brian Foote i Joseph Roder[1].

Aplikacja LocAdvisor została zaprojektowana zgodnie z podejściem Domain-Driven Design[5] oraz Clean Architecture[1][4]. Dzięki temu kod jest dobrze zorganizowany i łatwy do rozbudowy. Cała struktura została podzielona na moduły i warstwy, co pozwala na lepsze zarządzanie zależnościami i separację odpowiedzialności.

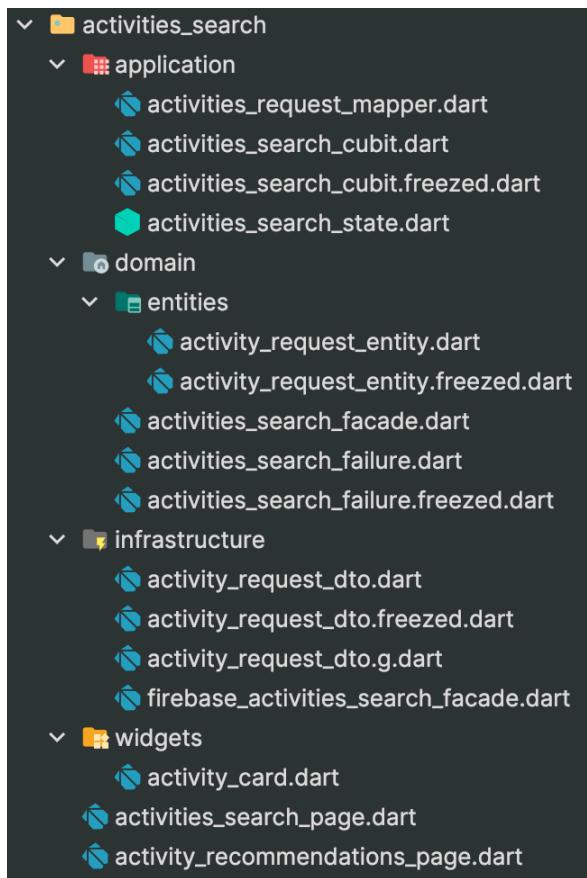


W aplikacji znajdują się następujące moduły (Rysunek 4.5):

- *auth* - obsługuje rejestrację i logowanie użytkowników, a także sprawdza, czy użytkownik jest zalogowany oraz umożliwia resetowanie hasła.
- *activities_core* - przechowuje wspólne encje i DTO (*data transfer object*) dla modułów *activities_search* i *recommendations*, ponieważ w jednym przypadku użytkownik otrzymuje wygenerowane rekomendacje aktywności, a w drugim może je przeglądać na liście.
- *accommodations_core* - przechowuje wspólne encje i DTO (*data transfer object*) dla modułów *accommodations_search* i *recommendations*, ponieważ w jednym przypadku użytkownik otrzymuje wygenerowane rekomendacje zakwaterowania, a w drugim może je przeglądać na liście.
- *activities_search* - moduł wyszukiwania aktywności
- *accommodations_search* - moduł wyszukiwania miejsc zakwaterowania.
- *recommendations* - odpowiada za przeglądanie zapisanych rekomendacji.
- *profile* - profil użytkownika.

Każdy moduł zawiera logikę specyfczną dla swojej funkcjonalności i jest podzielony na warstwy, co ułatwia utrzymanie kodu i jego testowanie.

4.3. Struktura modułów

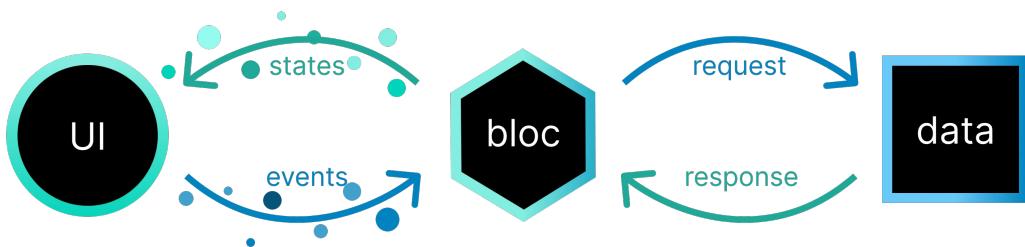


Rysunek 4.6 Struktura modułów

Aplikacja została podzielona na kilka kluczowych warstw:

- *application* - zawiera logikę zarządzania stanem aplikacji.
- *domain* - warstwa biznesowa aplikacji, w której znajdują się przypadki użycia i encje.
- *infrastructure* - obsługuje integracje z zewnętrznymi serwisami takimi jak: Firebase Firestore, Cloud Functions i Firebase Authentication.
- *widgets* - komponenty UI specyficzne dla danego modułu.

4.4. Zarządzanie stanem aplikacji



Rysunek 4.7 Architektura BLoC

Źródło: <https://bloclibrary.dev/architecture/>

Aplikacja wykorzystuje bibliotekę *flutter_bloc*[8], która pozwala na wygodne zarządzanie stanem (Rysunek 4.7). Istnieją dwa podejścia do obsługi stanu:

- Cubit - lekkie rozwiążanie, które eliminuje konieczność definiowania eventów. Zamiast tego UI bezpośrednio wywołuje metody w Cubit, które aktualizują stan. Jest to najczęściej stosowana opcja w sytuacjach, gdzie nie ma potrzeby obsługi zaawansowanych operacji, takich jak paginacja czy obsługa streamów.
- Bloc - bardziej rozbudowane podejście, w którym UI wzywala eventy, a Bloc reaguje na nie, modyfikując stan. Choć wymaga więcej kodu, daje większą kontrolę nad przepływem danych. Bloc sprawdza się w scenariuszach takich jak obsługa streamów w czasie rzeczywistym czy paginacja wyników. Można w nim zastosować transformery, np. *debounce time*, aby ograniczyć częstotliwość wysyłania eventów podczas przewijania listy, co optymalizuje wydajność aplikacji.

```

BlocConsumer<AccommodationListBloc, AccommodationListState>(
  listenWhen: (previous, current) =>
    previous.deleteStatus != current.deleteStatus,
  listener: (context, state) {
    switch (state.deleteStatus) {
      case StateStatus.initial:
      case StateStatus.loading:
        context.unfocus();
        context.loaderOverlay.show();
      case StateStatus.failure:
        context.loaderOverlay.hide();
        context.showSnackBarMessage(
          'Wystąpił błąd, proszę spróbować ponownie',
        );
      case StateStatus.success:
        context.loaderOverlay.hide();
        context.showSnackBarMessage('Usunięto rekomendacje');
    }
  },
  builder: (context, state) {
    switch (state.getAccommodationsStatus) {
      case StateStatus.success:
        return state.recommendations.isEmpty
          ? Expanded(
              child: NoRecommendationsInfo(
                onRefreshed: _refresh,
              ),
            ) // NoRecommendationsInfo
          : Expanded(
              child: RefreshIndicator(
                onRefresh: () async => _refresh(context),
                child: InfiniteList(
                  itemCount: state.recommendations.length,
                  hasError: state.nextPageStatus.isFailure,
                  isLoading: state.nextPageStatus.isLoading,
                  hasReachedMax: state.hasReachedMax,
                  onFetchData: () => context
                    .read<AccommodationListBloc }()
                    .add(const AccommodationListEvent()
                      .nextPageFetched()),
                  separatorBuilder: (_, __) =>
                    const SizedBox(height: 12),
                  itemBuilder: (_, i) => AccommodationListItem(
                    recommendation: state.recommendations[i],
                  ),
                ), // AccommodationListItem
            );
    }
  }
);

```

Rysunek 4.8 Implementacja ekranu zakwaterowań

Powyżej przedstawiono przykład wykorzystania biblioteki `flutter_bloc`. Podczas przewijania listy (Rysunek 4.8), UI wysyła event `nextPageFetched` (Rysunek 4.10). Bloc przechwytuje ten event, a następnie wywołuje metodę w fasadzie, aby pobrać kolejną stronę listy zakwaterowań (Rysunek 4.9). Po otrzymaniu wyników emituje nowy stan zawierający nowe dane (Rysunek 4.9 i 4.11). Aby obsłużyć zmiany w UI, wykorzystano `BlocConsumer` (Rysunek 4.8), który składa się z dwóch elementów:

- Listener - reaguje na zmiany stanu, co pozwala na wykonanie dodatkowych akcji, np. wyświetlenie komunikatu o błędzie.
 - Builder - odpowiada za przebudowę drzewa widgetów w reakcji na zmianę stanu i odświeżenie interfejsu użytkownika, aby zaprezentować nową listę wyników.
- Dzięki temu aplikacja może dynamicznie aktualizować zawartość w odpowiedzi na interakcje użytkownika, zachowując spójność między logiką biznesową a interfejsem.

```

FutureOr<void> _onNextPageFetched(
    _NextPageFetched event,
    Emitter<AccommodationListState> emit,
) async {
    if (state.hasReachedMax || state.recommendations.isEmpty) return;

    emit(state.copyWith(nextPageStatus: StateStatus.loading));

    final result = await _recommendationsFacade.fetchAccommodations(
        state.destination,
        pageSize: _pageSize,
        lastRecommendation: state.recommendations.last,
    );

    result.fold(
        (_) => emit(state.copyWith(nextPageStatus: StateStatus.failure)),
        (result) => emit(
            state.copyWith(
                nextPageStatus: StateStatus.success,
                recommendations: [...state.recommendations, ...result],
                hasReachedMax: result.length != _pageSize,
            ),
        ),
    );
}

```

Rysunek 4.9 Zarządzanie stanem zakwaterowań

```
@freezed
class AccommodationListEvent with _$AccommodationListEvent {
  const factory AccommodationListEvent.fetched({
    @Default('') String destination,
  }) = _Fetched;

  const factory AccommodationListEvent.destinationChanged(String destination) =
    _DestinationChanged;

  const factory AccommodationListEvent.nextPageFetched() = _NextPageFetched;

  const factory AccommodationListEvent.deleted(
    AccommodationRecommendations accommodation) = _Deleted;
}
```

Rysunek 4.10 Eventy zakwaterowań

```
@freezed
class AccommodationListState with _$AccommodationListState {
  const factory AccommodationListState({
    required StateStatus getAccommodationsStatus,
    required StateStatus nextPageStatus,
    required List<AccommodationRecommendations> recommendations,
    required bool hasReachedMax,
    required String destination,
    required StateStatus deleteStatus,
  }) = _AccommodationListState;

  factory AccommodationListState.initial() => AccommodationListState(
    getAccommodationsStatus: StateStatus.initial,
    nextPageStatus: StateStatus.initial,
    hasReachedMax: false,
    recommendations: [],
    destination: '',
    deleteStatus: StateStatus.initial,
  );
}
```

Rysunek 4.11 State zakwaterowań

4.5. Integracja z Firebase Firestore

Aplikacja przechowuje dane w Firebase Firestore i wykorzystuje DTO (Data Transfer Object) do mapowania danych między modelem Firestore a warstwą domenową. Dzięki bibliotece *freezed* automatycznie generowane są metody *fromJson* i *toJson*[2], co upraszcza obsługę danych i przyspiesza pisanie kodu (Rysunek 4.12 i 4.13).

```
@freezed
class ActivityRecommendationsDto with _$ActivityRecommendationsDto {
  const ActivityRecommendationsDto._();

  @JsonSerializable(explicitToJson: true)
  const factory ActivityRecommendationsDto({
    @JsonKey(includeFromJson: false, includeToJson: false) String? id,
    required List<ActivityDto> activities,
    required String destination,
    required String destinationLowerCase,
    required String additionalNotes,
    required String requestId,
    String? userId,
    @FirebaseTimestampJsonConverter() required DateTime createdAt,
  }) = _ActivityRecommendationsDto;

  factory ActivityRecommendationsDto.fromJson(Map<String, dynamic> json) =>
    _$ActivityRecommendationsDtoFromJson(json);

  factory ActivityRecommendationsDto.fromFirebase(
    DocumentSnapshot documentSnapshot,
  ) {
    return ActivityRecommendationsDto.fromJson(
      documentSnapshot.data() as Map<String, dynamic>,
      .copyWith(id: documentSnapshot.id);
  }

  factory ActivityRecommendationsDto.fromApi(dynamic data) {
    return ActivityRecommendationsDto.fromJson(
      convertDynamicToMapStringDynamic(data),
      .copyWith(id: data['id']);
  }

  ActivityRecommendations toDomain() {
    return ActivityRecommendations(
      id: id!,
      destination: destination,
      additionalNotes: additionalNotes,
      activities: activities.map((r) => r.toDomain()).toList(),
    );
  }
}
```

Rysunek 4.12 DTO rekomendacji aktywności

```
@freezed
class ActivityDto with _$ActivityDto {
  const ActivityDto._();

  @JsonSerializable()
  const factory ActivityDto({
    required String placeName,
    required String description,
    required String bestTimeToVisit,
    required String safetyTips,
    required String combinationTips,
    required String budgetTips,
  }) = _ActivityDto;

  factory ActivityDto.fromJson(Map<String, dynamic> json) =>
    _$ActivityDtoFromJson(json);

  factory ActivityDto.fromApi(List<dynamic> json) {
    return ActivityDto.fromJson(json as Map<String, dynamic>);
  }

  factory ActivityDto.fromFirebase(DocumentSnapshot documentSnapshot) {
    return ActivityDto.fromJson(
      documentSnapshot.data() as Map<String, dynamic>;
  }

  Activity toDomain() {
    return Activity(
      bestTimeToVisit: bestTimeToVisit,
      combinationTips: combinationTips,
      description: description,
      placeName: placeName,
      safetyTips: safetyTips,
      budgetTips: budgetTips,
    );
  }
}
```

Rysunek 4.13 DTO aktywności

Na przedstawionym przykładzie (Rysunek 4.14) wyszukiwane są zapisane rekomendacje aktywności w kolekcji *activity_recommendations* dla aktualnie zalogowanego użytkownika. Dodatkowo, użytkownik może podać filtr *destination*, który zostanie uwzględniony przy wyszukiwaniu. Wyniki są sortowane według daty dodania w kolejności malejącej, następnie mapowane i zwracane w postaci listy rekomendacji.

```

@Override
Future<Either<RecommendationsFailure, List<ActivityRecommendations>>>
fetchActivities(
    String destination,
    int pageSize = 20,
    ActivityRecommendations? lastRecommendation,
) async {
    try {
        final userId = _auth.currentUser!.uid;
        var query = _firestore
            .collection('activity_recommendations')
            .where('userId', isEqualTo: userId);

        if (destination.isNotEmpty) {
            query = query.where(
                'destinationLowerCase',
                isEqualTo: destination.toLowerCase(),
            );
        }

        query = query.orderBy('createdAt', descending: true).limit(pageSize);

        if (lastRecommendation != null) {
            final lastDoc = await _firestore
                .collection('activity_recommendations')
                .doc(lastRecommendation.id)
                .get();

            query = query.startAfterDocument(lastDoc);
        }

        final querySnapshot = await query.get();
        final result = querySnapshot.docs.map(
            (doc) => ActivityRecommendationsDto.fromFirebase(doc).toDomain(),
        );
        return right(result.toList());
    } on FirebaseException catch (e) {
        _logger.e(e);
        return left(RecommendationsFailure.unexpected());
    }
}

```

Rysunek 4.14 Integracja z Firebase Firestore

Aby zapewnić jednolitą obsługę błędów i uniknąć nadmiernego stosowania wyjątków, zastosowano bibliotekę *dartz*, która umożliwia użycie typu *either*. Dzięki temu każda operacja zwraca albo sukces, albo błąd, co pozwala na bardziej przewidywalne i bezpieczne zarządzanie stanem w warstwie aplikacji. W przypadku powodzenia aplikacja emitemuje stan *success*, natomiast w przypadku niepowodzenia *failure*, co pozwala na odpowiednie reagowanie w interfejsie użytkownika (Rysunek 4.8 i 4.9).

Firestore wprowadza pewne ograniczenia, które wymagają dodatkowej obsługi. Przykładem jest brak możliwości filtrowania po *lowerCase*, co sprawia, że konieczne jest przechowywanie dodatkowego pola *destinationLowerCase* (Rysunek 4.12 i 4.14), aby

umożliwić wyszukiwanie niezależnie od wielkości liter. Dodatkowo Firestore wymaga indeksowania zapytań, które filtryują według wielu pól jednocześnie (Rysunek 4.15).

Collection ID	Fields indexed	Query scope	Status
accommodation_recommendations	destinationLowerCase Ascending userId Ascending createdAt Descending __name__ Descending	Collection	Enabled
accommodation_recommendations	userId Ascending createdAt Descending __name__ Descending	Collection	Enabled
activity_recommendations	destinationLowerCase Ascending userId Ascending createdAt Descending __name__ Descending	Collection	Enabled
activity_recommendations	userId Ascending createdAt Descending __name__ Descending	Collection	Enabled

Rysunek 4.15 Indeksy w Firestore

4.6. Integracja z Firebase Authentication

Dzięki bibliotece *firebase_auth* można w prosty sposób zaimplementować kluczowe funkcje związane z uwierzytelnianiem, takie jak rejestracja, logowanie, resetowanie hasła oraz obsługa sesji użytkownika. Firebase zapewnia gotowe mechanizmy, które znaczco upraszczają te procesy.

```
@override
Future<Either<AuthFailure, Unit>> signIn({
    required String email,
    required String password,
}) async {
    try {
        await _auth.signInWithEmailAndPassword(
            email: email,
            password: password,
        );
        return right(unit);
    } on FirebaseAuthException catch (e) {
        _logger.e(e.message);
        final failure = firebaseAuthErrors[e.code];
        return left(failure ?? const AuthFailure.unexpected());
    }
}
```

Rysunek 4.16 Integracja z Firebase Authentication

Na przedstawionym przykładzie (Rysunek 4.16) zaimplementowano logowanie użytkownika przy użyciu metody `signInWithEmailAndPassword`. Metoda ta zwraca rezultat w przypadku poprawnego logowania lub rzuca wyjątek `FirebaseAuthException`, jeśli operacja zakończy się niepowodzeniem.

```
const firebaseAuthErrors = {
  'account-exists-with-different-credential': AuthFailure.accountExists(),
  'invalid-credential': AuthFailure.invalidCredential(),
  'invalid-email': AuthFailure.invalidEmail(),
  'user-disabled': AuthFailure.userDisabled(),
  'email-already-in-use': AuthFailure.emailInUse(),
  'weak-password': AuthFailure.weakPassword(),
  'user-not-found': AuthFailure.userNotFound(),
  'wrong-password': AuthFailure.wrongPassword(),
  'already-exists': AuthFailure.accountExists(),
  'not-found': AuthFailure.userNotFound(),
  'app-not-authorized': AuthFailure.deviceNotSupported(),
};
```

Rysunek 4.17 Kody błędów Firebase

Firebase definiuje własne kody błędów[8] (Rysunek 4.17), które można obsłużyć w aplikacji i odpowiednio zmapować na komunikaty przyjazne dla użytkownika, np. wyświetlając wiadomość “Nieprawidłowe dane logowania” (Rysunek 4.18). Dzięki temu użytkownik otrzymuje jasną informację zwrotną, co zwiększa użyteczność aplikacji i poprawia *user experience*.

```

@freezed
class AuthFailure with _$AuthFailure {
  const factory AuthFailure.unexpected() = _Unexpected;

  const factory AuthFailure.accountExists() = _AccountExists;

  const factory AuthFailure.invalidCredential() = _InvalidCredential;

  const factory AuthFailure.invalidEmail() = _InvalidEmail;

  const factory AuthFailure.userDisabled() = _UserDisabled;

  const factory AuthFailure.emailInUse() = _EmailInUse;

  const factory AuthFailure.weakPassword() = _WeakPassword;

  const factory AuthFailure.userNotFound() = _UserNotFound;

  const factory AuthFailure.wrongPassword() = _WrongPassword;

  const factory AuthFailure.deviceNotSupported() = _DeviceNotSupported;
}

extension AuthFailureX on AuthFailure {
  String get message => when(
    unexpected: () => 'Wystąpił błąd, proszę spróbować ponownie',
    accountExists: () => 'Konto istnieje',
    invalidCredential: () => 'Nieprawidłowe dane logowania',
    invalidEmail: () => 'Nieprawidłowe dane logowania',
    wrongPassword: () => 'Nieprawidłowe dane logowania',
    userDisabled: () => 'Konto zablokowane',
    emailInUse: () => 'Email jest już w użyciu',
    weakPassword: () => 'Za słabe hasło',
    userNotFound: () => 'Użytkownik nie istnieje',
    deviceNotSupported: () => 'Urządzenie nie jest obsługiwane',
  );
}

```

Rysunek 4.18 Mapowanie kodów błędów Firebase

Na kolejnym przykładzie (Rysunek 4.19) widać, jak rezultat logowania jest obsługiwany w praktyce. *BlocConsumer* nasłuchuje zmian stanu i dynamicznie reaguje na jego aktualizacje. Jeśli stan zmieni się na *failure*, aplikacja wyświetla odpowiedni komunikat błędu, informując użytkownika o problemie, np. niepoprawnych danych logowania. Natomiast w przypadku *success*, użytkownik zostaje automatycznie przekierowany na ekran główny. Takie podejście pozwala na czytelną i intuicyjną obsługę procesu logowania, zapewniając płynne doświadczenie użytkownika.

```

child: BlocConsumer<SignInCubit, SignInState>(
    listenWhen: (previous, current) =>
        previous.status != current.status ||
        previous.authFailureOrSuccessOption != current.authFailureOrSuccessOption,
    listener: (context, state) async {
        switch (state.status) {
            case StateStatus.initial:
            case StateStatus.loading:
                context.unfocus();
                context.loaderOverlay.show();
            case StateStatus.failure:
                context.loaderOverlay.hide();
                state.authFailureOrSuccessOption.fold(
                    () {},
                    (failure) => context.showSnackBarMessage(failure.message),
                );
            case StateStatus.success:
                context.loaderOverlay.hide();
                await AutoRouter.of(context).replaceAll([const HomeRoute()]);
        }
    },
),

```

Rysunek 4.19 Implementacja ekranu logowania

4.7. Integracja z Cloud Functions i OpenAI API

```

@Override
Future<Either<AccommodationsSearchFailure, AccommodationRecommendations>>
    getAccommodationRecommendations(AccommodationRequest request) async {
try {
    final func =
        _firebaseFunctions.httpsCallable('getAccommodationRecommendations');

    final response =
        await func.call(AccommodationRequestDto.fromDomain(request).toJson());

    final result =
        AccommodationRecommendationsDto.fromApi(response.data).toDomain();

    return right(result);
} on Exception catch (e) {
    _logger.e(e);
    return left(AccommodationsSearchFailure.unexpected());
}
}

```

Rysunek 4.20 Integracja z Cloud Functions

Biblioteka *cloud-functions* umożliwia integrację aplikacji z API poprzez wywoływanie funkcji chmurowych. Na przedstawionym przykładzie (Rysunek 4.20), podczas generowania rekomendacji zakwaterowania, aplikacja wysyła żądanu HTTP do funkcji *getAccommodationRecommendations*.

```
no usages
export const getAccommodationRecommendations : Function<any, Promise<Accommod... = onCall(
{
    region: "europe-central2",
    secrets: ["OPEN_AI_KEY"],
},
async (request : CallableRequest<any> ) : Promise<AccommodationRecommendationsRe... => {
    const openai : OpenAI = new OpenAI({apiKey: process.env.OPEN_AI_KEY});
    const accommodationRequest : AccommodationRequest = request.data as AccommodationRequest;
    if (!accommodationRequest.destination) {
        throw new HttpsError("invalid-argument", "Destination is required.");
    }
    const systemPrompt : string = `
Jestes inteligentnym asystentem podrózniczym „LocAdvisor”, skoncentrowanym na autentycznym, lokalnym klimacie danego miejsca.
Twom priorytetem jest zapewnienie użytkownikowi autentycznych, lokalnych doświadczeń, unikając typowych turystycznych pułapek, biorąc pod uwagę
następujące wagi preferencji lokalizacyjnych użytkownika:

Podczas generowania rekomendacji przestrzegaj następujących zasad:
1. **Spójność preferencji:** Wszystkie rekomendacje muszą spełniać WSZYSTKIE wybrane preferencje użytkownika (np. lokalizacja, atmosfera, budżet).
2. **Unikal konfliktów:** Jeśli użytkownik wybiera "tętniące życiem", unikaj rekomendowania miejsc, które są spokojne lub odizolowane (np. Key Biscayne).
3. **Waga preferencji lokalizacji:** Jeśli użytkownik wybiera wiele lokalizacji, daj pierwszeństwo priorytetowi "Blisko plaży lub natury".
4. **Atmosfera:** Dopusz atmosferę miejsca do preferencji użytkownika.
    - "Tętniące życiem" = miejsca z bogatą ofertą wydarzeń, klubów, koncertów.
    - "Spokojne" = miejsca ciche, relaksujące.
5. **Autentyczny vibe:** Skup się na lokalnych miejscach, omijając typowe turystyczne atrakcje.
6. **Bezpieczeństwo:** Podaj szczegółowe wskazówki dotyczące bezpieczeństwa dla każdej lokalizacji.
7. **Filtr budżetowy:** Dopusz rekomendacje do zakresu finansowego użytkownika, unikając miejsc zbyt drogich lub zbyt tanich.

Jeśli użytkownik wybiera wiele preferencji, połącz je w taki sposób, aby rekomendacje były możliwie najbardziej zrównoważone i dopasowane
do oczekiwania. Unikaj typowych pułapek turystycznych, bazuj na wiedzy o miejscach polecanych przez lokalnych mieszkańców.
Każdż duży naciśk na bezpieczeństwo - wskaz, jakie zagrożenia mogą wystąpić w danej okolicy oraz zachowania, których lepiej unikać.

Twoje odpowiedzi MUSZA:
- Uwzględniać preferencje „blisko plaży”, jeśli została zaznaczona.
- Łączyć preferencje „tętniące życiem” z lokalnymi miejscowościami znymi z życia nocnego.
- Być zgodne z budżetem i priorytetami użytkownika.
```

Rysunek 4.21 Integracja z OpenAI API 1

Funkcja napisana jest w TypeScript i uruchamiana w środowisku Node.js, generuje zapytanie do OpenAI na podstawie przekazanych parametrów (Rysunek 4.21). Podczas definiowania funkcji wykorzystywany jest klucz “OPEN_AI_KEY”, który jest bezpiecznie przechowywany w sekretach Google Cloud.

Zapytanie zawiera szczegółowe wytyczne, które model OpenAI musi uwzględnić, aby generować trafne rekomendacje (Rysunek 4.21, 4.22 i 4.23).

```

być zgodne z budżetem i przystępstwem do klimatu.

### Przykład zapytania i odpowiedzi:

**Zapytanie:**  

Użytkownik planuje wyjazd do Miami.  

Preferencje użytkownika:  

- Lokalizacja: Blisko plaży lub natury, Blisko centrum, Blisko kajpek i kawiarni.  

- Atmosfera: Spokojne.  

- Budżet: Średnie.

**Idealna odpowiedź:**  

{
  "destination": "Miami",
  "locations": [
    {
      "placeName": "North Beach",
      "description": "North Beach to spokojna okolica w Miami, idealna dla osób szukających relaksu blisko plaży. Znajdują się tu lokalne restauracje i kawiarnie oferujące autentyczne smaki.",
      "localVibe": "Autentyczny klimat Miami, z dala od tłumów South Beach. Popularne wśród lokalnych mieszkańców.",
      "safetyTips": "Bezpieczna okolica, ale unikaj spacerów w mniej oświetlonych miejscach w nocy.",
      "budgetTips": "Ceny zakwaterowania są średnie, z dużą liczbą opcji Airbnb. Lokalne restauracje oferują przystępne ceny w porównaniu do turystycznych miejsc w South Beach.",
      "transportTips": "Najlepiej poruszać się rowerem lub korzystać z usług Uber/Lyft. W okolicy kursują również autobusy miejskie, które łączą North Beach z centrum Miami."
    },
    {
      "placeName": "Coconut Grove",
      "description": "Coconut Grove to zielona i spokojna dzielnica Miami, znana z licznych parków, butików i lokalnych restauracji. Idealna dla osób ceniących kontakt z naturą i relaksującą atmosferą.",
      "localVibe": "Artystyczna atmosfera z domieszką historii. Często organizowane są lokalne festiwale i wydarzenia.",
      "safetyTips": "Bezpieczna okolica, jednak warto unikać mniej oświetlonych ulic w nocy.",
      "budgetTips": "Średni przedział cenowy, ale można znaleźć tańsze opcje zakwaterowania, jeśli zarezerwuje się z wyprzedzeniem. Lokalne restauracje oferują happy hours i promocje.",
      "transportTips": "Dzielnica jest dobrze skomunikowana z centrum Miami przez Metrorail. Warto też korzystać z wypożyczalni rowerów do zwiedzania okolicy."
    }
  ]
}

```

Rysunek 4.22 Integracja z OpenAI API 2

```

const userPrompt : string = `

Użytkownik planuje wyjazd do: ${accommodationRequest.destination}
Preferencje użytkownika:
- Lokalizacja: ${accommodationRequest.locationPreferences.join(", ")}.  

- Atmosfera: ${accommodationRequest.atmosphereOption}.
- Budżet: ${accommodationRequest.budgetOption}.
- Dodatkowe uwagi: ${accommodationRequest.additionalNotes}.

Wygeneruj proszę rekommendacje, gdzie najlepiej zatrzymać się w ${accommodationRequest.destination} zgodnie z powyższymi preferencjami.
Podaj szczegółowe wskazówki dotyczące bezpieczeństwa, klimatu okolicy oraz ewentualnych porad,
czego unikać, aby nie wpaść w pułapki turystyczne.
Skup się na autentycznym klimacie lokalnym oraz na unikaniu pułapek turystycznych.

Uwagi do generowania rekommendacji:
1. Jeśli użytkownik wybrał „Blisko plaży lub natury”, przynajmniej jedna rekommendacja MUSI spełniać tę preferencję.
2. Unikaj konfliktów preferencji - np. „spokojne” i „tętniące życiem” są wzajemnie wykluczające się.
3. Wszystkie rekommendacje muszą być zgodne z wybranym budżetem.
4. Podaj szczegółowe wskazówki bezpieczeństwa oraz informacje o lokalnym klimacie każdej lokalizacji.
5. Uzgadnij wyłącznie miejsca, które reprezentują autentyczny lokalny vibe.

Na koniec zwróć odpowiedź WYŁĄCZNIE w formacie JSON o strukturze:
{
  "destination": "string",
  "locations": [
    {
      "placeName": "string",
      "description": "string",
      "localVibe": "string",
      "safetyTips": "string",
      "transportTips": "string",
      "budgetTips": "string"
    }
  ],
  "additionalNotes": "string"
}
`;

```

Rysunek 4.23 Integracja z OpenAI API 3

Po jego utworzeniu funkcja wysyła żądanie do OpenAI za pomocą SDK i metody *openai.chat.completions.create* (Rysunek 4.24). Do generowania rekomendacji został użyty model gpt-4o oraz ustawiono temperaturę na 0.7, co pozwala na uzyskanie nieco bardziej kreatywnych rekomendacji. Jeśli OpenAI nie zwróci odpowiedzi, funkcja rzuca wyjątek i kończy działanie. W przypadku pomyślnego wygenerowania rekomendacji, dane wraz z parametrami wejściowymi są zapisywane do Firestore przy użyciu *db.collections.add*, a następnie wynik jest zwracany do aplikacji.

```

const completion : ChatCompletion = await openai.chat.completions.create({
    model: "gpt-4o",
    messages: [
        {role: "system", content: systemPrompt},
        {role: "user", content: userPrompt},
    ],
    temperature: 0.7,
});
const response : ChatCompletionMessage = completion.choices[0].message;

if (!response.content) {
    throw new HttpsError("internal", "Failed to generate response.");
}

const recommendations: AccommodationRecommendationsResponse = JSON.parse(response.content);
const userId : string | null = request.auth ? request.auth.uid : null;
const now : Timestamp = admin.firestore.Timestamp.now();

accommodationRequest.userId = userId;
accommodationRequest.createdAt = now;

recommendations.userId = userId;
recommendations.createdAt = now;
recommendations.destinationLowerCase = accommodationRequest.destination.toLowerCase();

const requestDoc : DocumentReference<DocumentData, DocumentData> = await db
    .collection("accommodation_requests")
    .add({...accommodationRequest});

recommendations.requestId = requestDoc.id;

const recommendationDoc : DocumentReference<DocumentData, DocumentData> = await db
    .collection("accommodation_recommendations")
    .add({...recommendations});

recommendations.id = recommendationDoc.id;

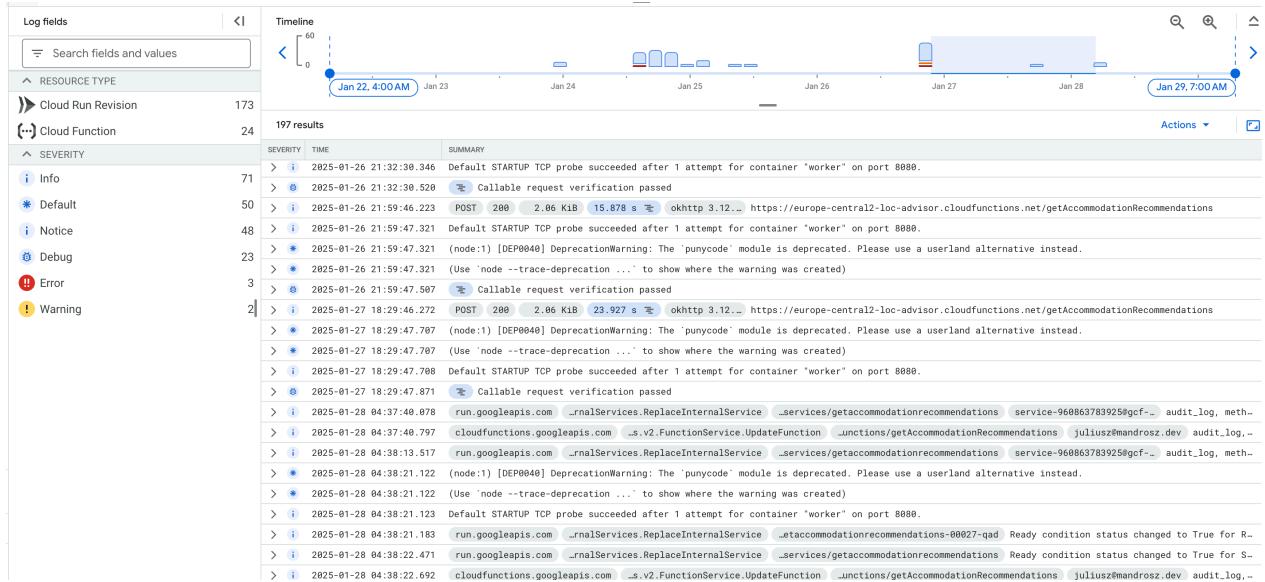
return recommendations;

```

Rysunek 4.24 Integracja z OpenAI API 4

Dzięki panelowi administracyjnemu Google Cloud, można przeglądać logi funkcji i analizować ewentualne błędy (Rysunek 4.25). Cloud Functions są hostowane w regionie *europe-central2* (Warszawa), działają w modelu *serverless* i skalują się automatycznie, co

spełnia wymagania niefunkcjonalne aplikacji LocAdvisor, zapewniając dostępność i nieograniczone zasoby serwerowe[3].



Rysunek 4.25 Logi Cloud Functions

4.8. Testowanie aplikacji

W ramach testów, przeprowadzono testy jednostkowe dla pól formularza w aplikacji.

Aplikacja korzysta z biblioteki *formz*, która ułatwia testowanie poszczególnych funkcjonalności, na przedstawionym przykładzie zaprezentowano zastosowanie tej biblioteki w praktyce dla pola adresu e-mail. (Rysunek 4.26). Do walidacji użyto skomplikowanego wyrażenia regularnego, więc testy jednostkowe były kluczowe, żeby upewnić się, że wszystko działa poprawnie.

```
enum EmailError {
    empty,
    invalid;

    String get message {
        switch (this) {
            case EmailError.empty:
                return 'Proszę podać adres e-mail';
            case EmailError.invalid:
                return 'Proszę podać poprawny adres e-mail';
        }
    }
}

class EmailInput extends FormzInput<String, EmailError> {
    const EmailInput.pure() : super.pure('');

    const EmailInput.dirty(String value) : super.dirty(value);

    static final _emailRegex = RegExp(
        r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$',
    );

    @override
    EmailError? validator(String value) {
        if (isPure) {
            return null;
        }

        if (value.isEmpty) {
            return EmailError.empty;
        }

        if (!_emailRegex.hasMatch(value)) {
            return EmailError.invalid;
        }

        return null;
    }
}
```

Rysunek 4.26 Model adresu e-mail

W kolejnym przykładzie przedstawiono testy jednostkowe dla pola adresu e-mail (Rysunek 4.27 i 4.28).

```
void main() {
  group('EmailInput', () {
    test('should return null for a valid email address', () {
      const email = 'test@example.com';
      const input = EmailInput.dirty(email);
      expect(input.validator(email), isNull);
    });

    test('should return EmailError.empty for an empty email address', () {
      const email = '';
      const input = EmailInput.dirty(email);
      expect(input.validator(email), EmailError.empty);
    });

    test('should return EmailError.invalid for an invalid email address', () {
      const email = 'invalid-email';
      const input = EmailInput.dirty(email);
      expect(input.validator(email), EmailError.invalid);
    });

    test('should return EmailError.invalid for an email address missing domain',
        () {
      const email = 'test@';
      const input = EmailInput.dirty(email);
      expect(input.validator(email), EmailError.invalid);
    });
  });
}
```

Rysunek 4.27 Testy adresu e-mail 1

```
test(
  'should return EmailError.invalid for an email address missing "@" symbol',
  () {
  const email = 'testexample.com';
  const input = EmailInput.dirty(email);
  expect(input.validator(email), EmailError.invalid);
});

test(
  'should return EmailError.invalid for an email address missing country code',
  () {
  const email = 'test@example';
  const input = EmailInput.dirty(email);
  expect(input.validator(email), EmailError.invalid);
});

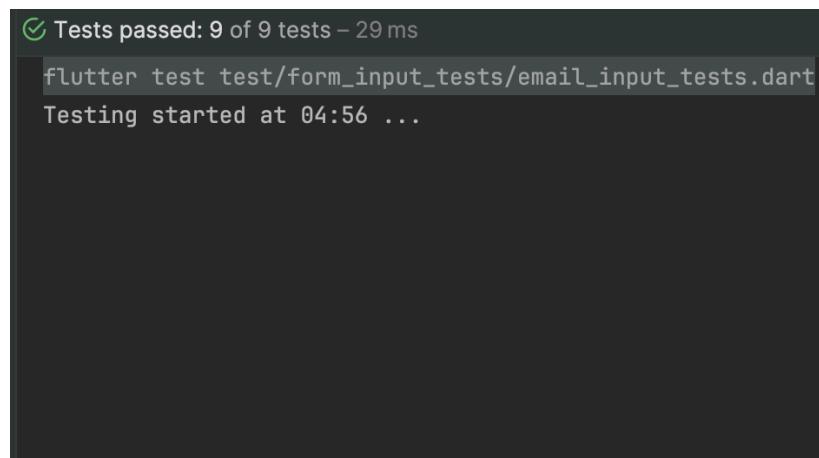
test('should return null for a valid email address with a subdomain', () {
  const email = 'test@mail.example.com';
  const input = EmailInput.dirty(email);
  expect(input.validator(email), isNull);
});

test(
  'should return null for a valid email address with numbers and special characters',
  () {
  const email = 'user123+test@example.co.uk';
  const input = EmailInput.dirty(email);
  expect(input.validator(email), isNull);
});

test('should return null for a pure input', () {
  const input = EmailInput.pure();
  expect(input.validator(''), isNull);
});
```

Rysunek 4.28 Testy adresu e-mail 2

Po uruchomieniu testów, wszystkie zakończyły się sukcesem (Rysunek 4.29).



A terminal window displaying the output of a Flutter test run. The output shows that 9 tests passed in 29 ms, starting at 04:56. The command used was flutter test test/form_input_tests/email_input_tests.dart.

```
Tests passed: 9 of 9 tests - 29 ms
flutter test test/form_input_tests/email_input_tests.dart
Testing started at 04:56 ...
```

Rysunek 4.29 Wyniki testów jednoskowych

5. Prezentacja aplikacji

W tym rozdziale omówione zostanie działanie aplikacji, wraz z opisem kluczowych funkcjonalności i interakcji użytkownika. Poszczególne ekrany zostaną zaprezentowane w kolejności, od pierwszego uruchomienia po generowanie i przeglądanie rekomendacji.



Rysunek 5.1 Ekran powitalny

5.1. Onboarding użytkownika

Po uruchomieniu aplikacji użytkownik trafia na ekran powitalny z trzema opcjami (Rysunek 5.1):

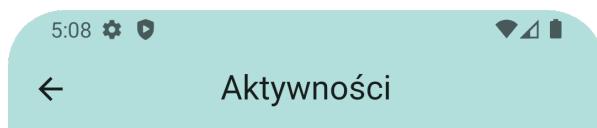
- Znajdź aktywności.
- Znajdź zakwaterowanie.
- Zaloguj się.

Nowi użytkownicy, którzy jeszcze nie mają konta, mogą od razu skorzystać z wersji demo. Pozwala im ona zapoznać się z funkcjonalnościami aplikacji i sprawdzić, co oferuje, bez konieczności rejestracji. Dzięki temu od początku mogą skupić się na wyszukiwaniu interesujących ich opcji, bez konieczności wypełniania dodatkowych formalności.

Zakładając, że użytkownik wybrał opcję „Znajdź aktywności” na ekranie powitalnym (Rysunek 5.1), zostaje przekierowany do ekranu wyszukiwania, gdzie może dostosować wyniki do swoich preferencji (Rysunek 5.2 i 5.3).

Proces rozpoczyna się od wyboru lokalizacji za pomocą pola tekstowego. Następnie użytkownik wskazuje termin, decydując, czy szuka aktywności na dziś, czy planuje coś na później. W sekcji „Co chcesz robić?” dostępne są kategorie takie jak jedzenie, życie nocne, relaks, ukryte perełki, natura czy sport. Ta sekcja umożliwia wybór wielu kategorii jednocześnie.

Kolejnym krokiem jest określenie preferencji dotyczących budżetu (tanie, średnie, luksusowe) oraz atmosfery (spokojne lub tętniące życiem), gdzie można zaznaczyć tylko jedną opcję w każdej kategorii. Dodatkowo użytkownik ma do dyspozycji opcjonalne pole tekstowe na dodatkowe uwagi, które pozwala na wpisanie specyficznych wymagań lub sugestii.



Gdzie szukasz?

Wpisz lokalizację (np. Madryt)

Na kiedy szukasz?

Na dzisiaj

Na później

Co chcesz robić?

Jedzenie

Życie nocne

Relaks

Ukryte perelki

Natura

Sport

Preferencje

Budżet:

Tanie

Średnie

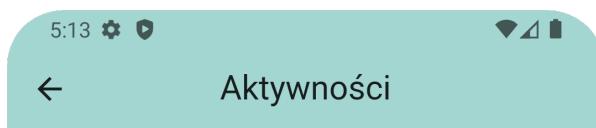
Luksusowe



Atmosfera:



Rysunek 5.2 Wyszukiwanie aktywności 1



Aktywności

Jedzenie

Życie nocne

Relaks

Ukryte perelki

Natura

Sport

Preferencje

Budżet:

Tanie

Średnie

Luksusowe

Atmosfera:

Spokojne

Tętniące życiem

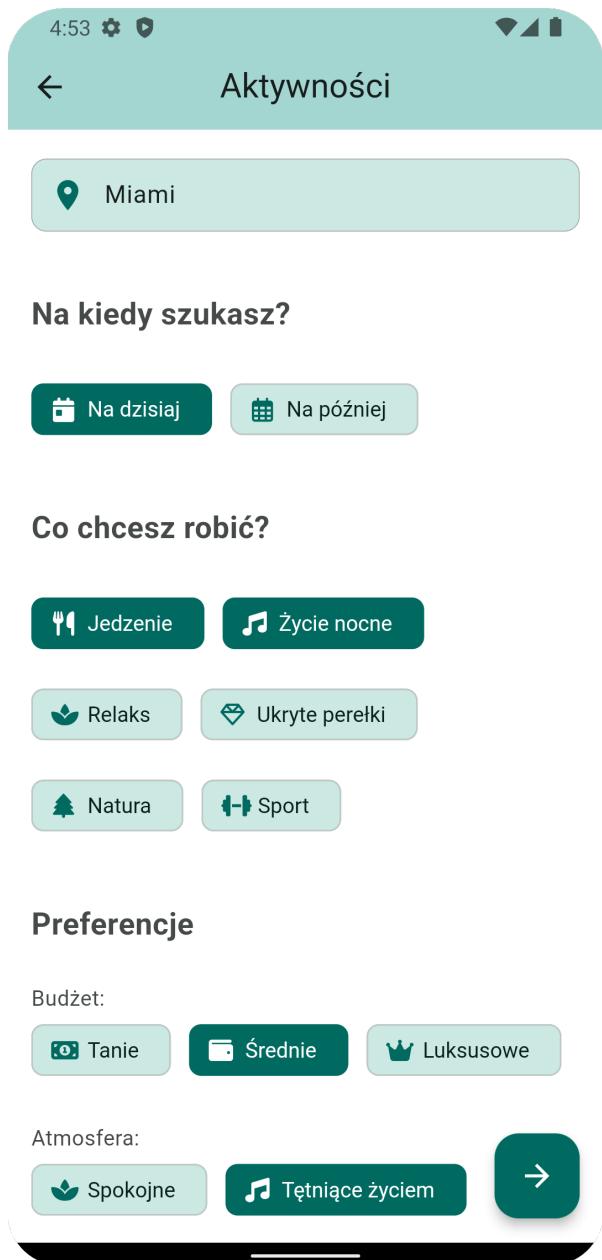
Masz dodatkowe uwagi?

Dodaj coś od siebie (opcjonalne)



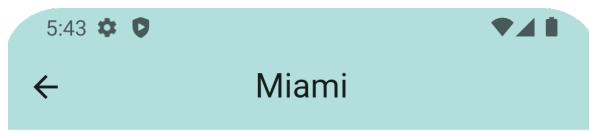
Rysunek 5.3 Wyszukiwanie aktywności 2

Ekran na kolejnym przykładzie (Rysunek 5.4) pokazuje stan po dokonaniu wyborów przez użytkownika. W sekcji „Gdzie szukasz?” użytkownik wpisał lokalizację „Miami”. W części „Na kiedy szukasz?” wybrał opcję „Na dzisiaj”. W sekcji „Co chcesz robić?” zaznaczył dwie kategorie: „Jedzenie” i „Życie nocne”. W sekcji „Preferencje” określił budżet jako „Średnie” oraz atmosferę jako „Tętniące życiem”. Po kliknięciu przycisku aplikacja skorzysta z wybranych preferencji, aby znaleźć propozycje najlepiej dopasowane do oczekiwania użytkownika.



Rysunek 5.4 Wybór preferencji aktywności

Po wybraniu preferencji użytkownik przechodzi do ekranu z wygenerowanymi rekomendacjami aktywności (Rysunek 5.5 i 5.6). W górnej części ekranu znajduje się sugestia, która przedstawia spójny plan wieczoru, łącząc kilka miejsc. Poniżej wyświetlane są karty z bardziej szczegółowymi propozycjami, zawierającymi krótki opis, optymalny czas wizyty, sugestie dotyczące bezpieczeństwa, oraz orientacyjne koszty. Na dole ekranu znajduje się przycisk, który pozwala przejść do kolejnego kroku.



i Zaczni j wieczór od kolacji w Batch Gastropub, następnie wybierz się na Rosa Sky na before, a potem baw się w Space Club.

Polecane aktywności

Batch Gastropub

Lokalny pub/restauracja w Brickell z luźną atmosferą i pysznym jedzeniem, idealnym na start wieczoru. Serwują klasyczne amerykańskie dania z twistem oraz szeroki wybór piw kraftowych.

- i** Najlepiej między 19:00 a 21:00, aby złapać dobrą atmosferę i uniknąć tłumów.
- !** Unikaj zostawiania rzeczy bez opieki, szczególnie w weekendowe wieczory.
- !** Po kolacji warto udać się do Rosa Sky na before party i podziwiać widok na Miami.
- !** Średni – dania główne od \$18, drinki od \$12.

Rosa Sky Rooftop



Rysunek 5.5 Propozycje aktywności 1



Rosa Sky Rooftop

Stylowy rooftop bar na Brickell z panoramicznym widokiem na centrum Miami, serwujący kreatywne koktajle i lekkie przekąski.

- !** Najlepiej między 21:00 a 23:00, aby cieszyć się pięknymi widokami przed nocną zabawą.
- !** Zarezeruj miejsce z wyprzedzeniem, aby mieć gwarancję wejścia, szczególnie w weekendy.
- !** Po Rosa Sky najlepiej złapać Ubera do Space Club, gdzie impreza trwa do rana.
- !** Średni – koktajle od \$16, przekąski od \$14.

Space Club

Legendarny klub w Miami z całonocnymi imprezami, występami topowych DJ-ów i niepowtarzalnym klimatem.

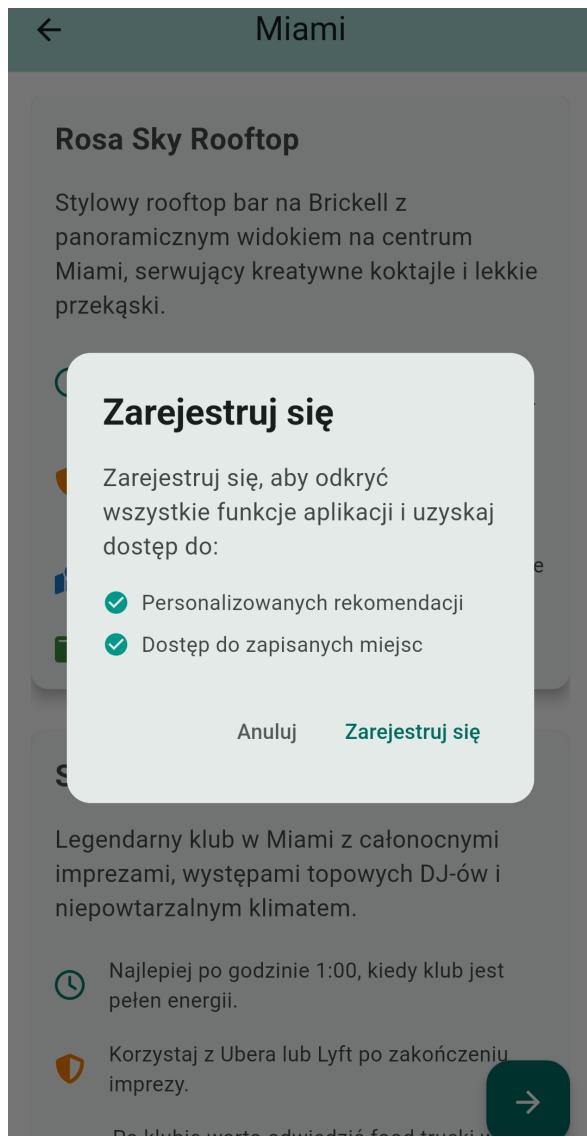
- !** Najlepiej po godzinie 1:00, kiedy klub jest pełen energii.
- !** Korzystaj z Ubera lub Lyft po zakończeniu imprezy.
- !** Po klubie warto odwiedzić food trucki w pobliżu na późnonocna przekąskę.



Rysunek 5.6 Propozycje aktywności 2

Po kliknięciu przycisku nawigacyjnego aplikacja wyświetla okno z zachętą do rejestracji (Rysunek 5.7). W oknie widoczny jest nagłówek „Zarejestruj się”, a poniżej informacja o korzyściach z założenia konta. Użytkownik dowiaduje się, że rejestracja umożliwia dostęp do wszystkich funkcji aplikacji. Na dole okna znajdują się dwa przyciski:

- „Anuluj” – zamyka okno i pozwala dalej korzystać z aplikacji w trybie demo.
- „Zarejestruj się” – przekierowuje użytkownika do formularza rejestracyjnego.

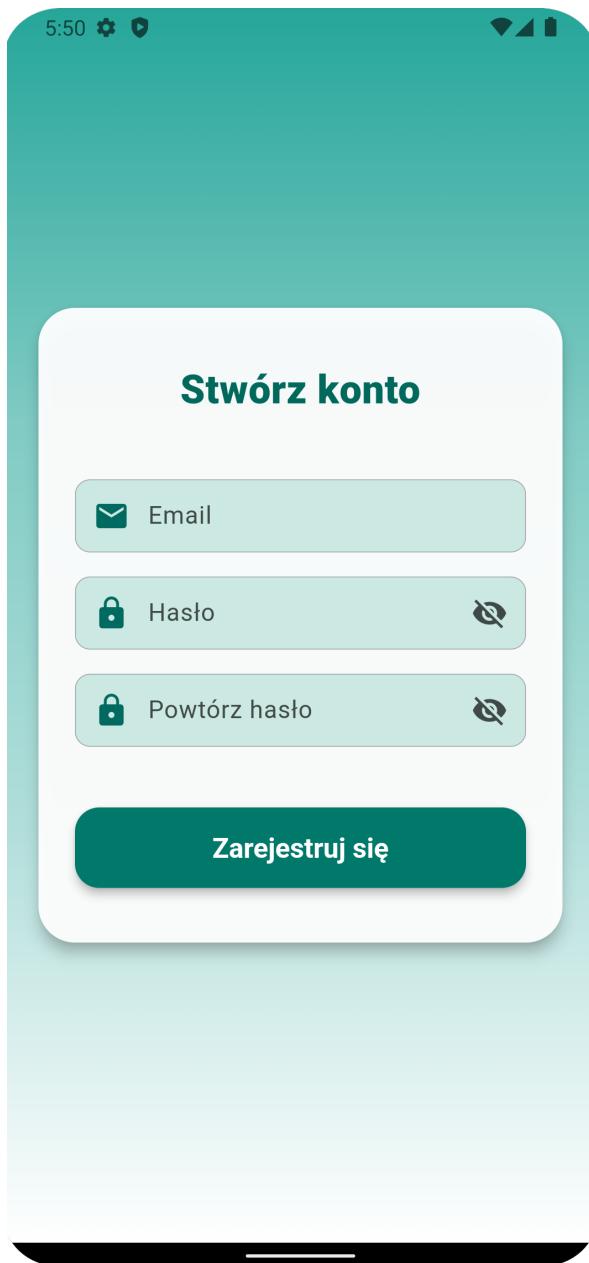


Rysunek 5.7 Modal przejścia do rejestracji

Po wybraniu opcji „Zarejestruj się” użytkownik przechodzi na ekran rejestracji (Rysunek 5.8), gdzie może utworzyć swoje konto w aplikacji. Ekran zawiera formularz z trzema polami:

- Email - użytkownik wprowadza swój adres e-mail, który będzie używany jako login.
- Hasło - pole do wpisania hasła.
- Powtórz hasło - użytkownik ponownie wpisuje hasło, aby uniknąć pomyłek.

Obok pól hasła znajdują się ikony umożliwiające podgląd wpisanych znaków, co pozwala łatwo sprawdzić ich poprawność. Na dole ekranu znajduje się przycisk „Zarejestruj się”, który przesyła dane i kończy proces rejestracji.

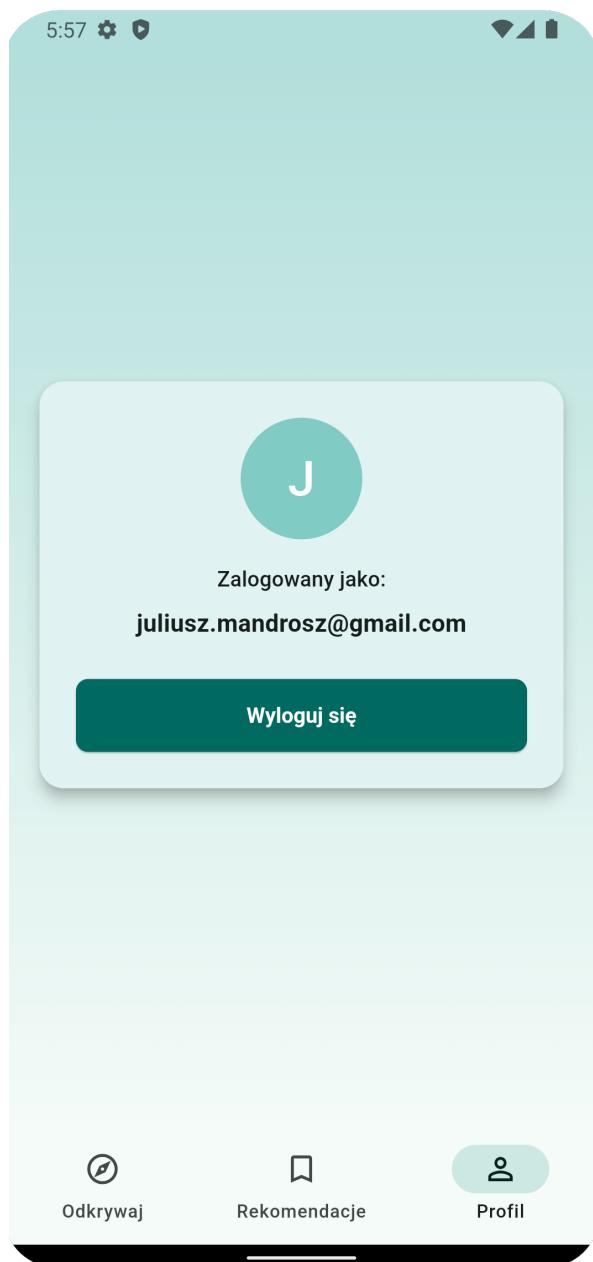


Rysunek 5.8 Ekran rejestracji

Po zakończeniu rejestracji użytkownik zostaje przekierowany na ekran „Odkrywaj” (Rysunek 5.9), który pełni rolę głównego punktu startowego w aplikacji. Ekran wituje użytkownika komunikatem „Cześć, dobrze Cię widzieć!” i umożliwia wybór jednej z dwóch opcji: „Znajdź aktywności” lub „Znajdź zakwaterowanie”, pozwalając na dalsze eksplorowanie aplikacji.



Rysunek 5.9 Ekran “Odkrywaj”



Rysunek 5.10 Ekran profilu

W dolnej części ekranu znajduje się pasek nawigacyjny, który umożliwia przełączanie się między trzema głównymi ekranami aplikacji: „Odkrywaj”, „Rekomendacje” oraz „Profil”.

Po przejściu do zakładki „Profil” (Rysunek 5.10) użytkownik widzi swój adres e-mail podany podczas rejestracji, inicjał wyświetlany jako avatar oraz przycisk „Wyloguj się”, który pozwala wylogować się z aplikacji i przejść do ekranu logowania (Rysunek 5.11).

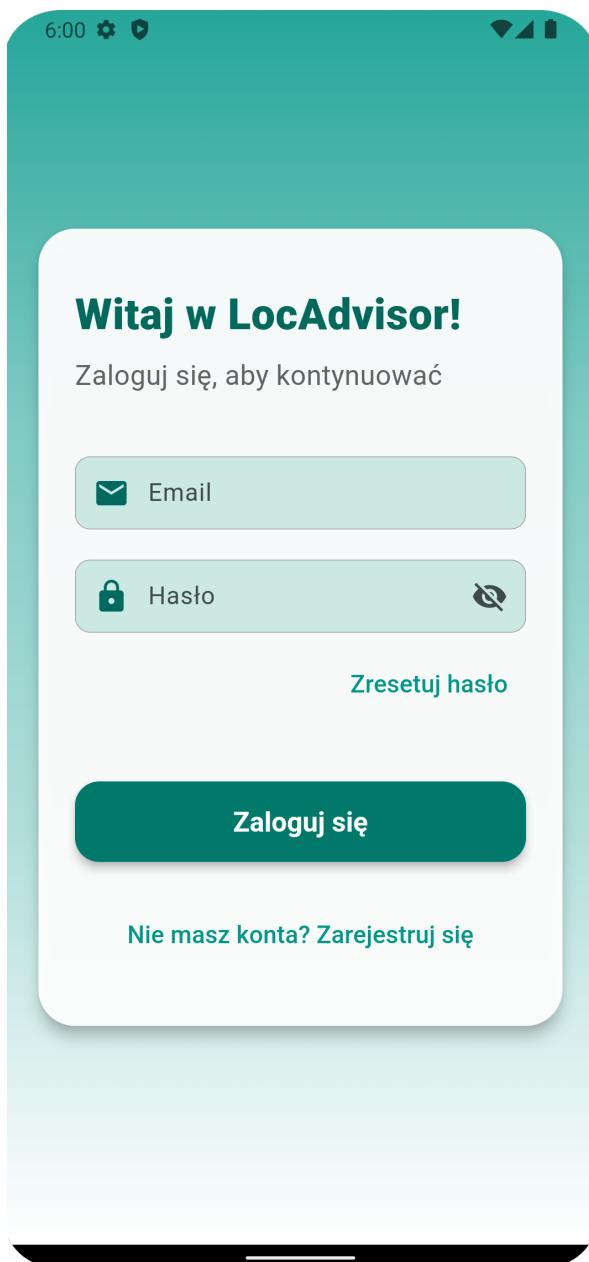
5.2. Logowanie do aplikacji

Proces logowania do aplikacji rozpoczyna się od ekranu logowania (Rysunek 5.11), gdzie użytkownik jest witany komunikatem „Witaj w LocAdvisor!”. Formularz składa się z dwóch pól:

- Email - pole do wpisania adresu e-mail użytego podczas rejestracji.
- Hasło - pole do wprowadzenia hasła. Obok pola znajduje się ikona umożliwiająca podgląd wpisanych znaków, co pozwala uniknąć błędów.

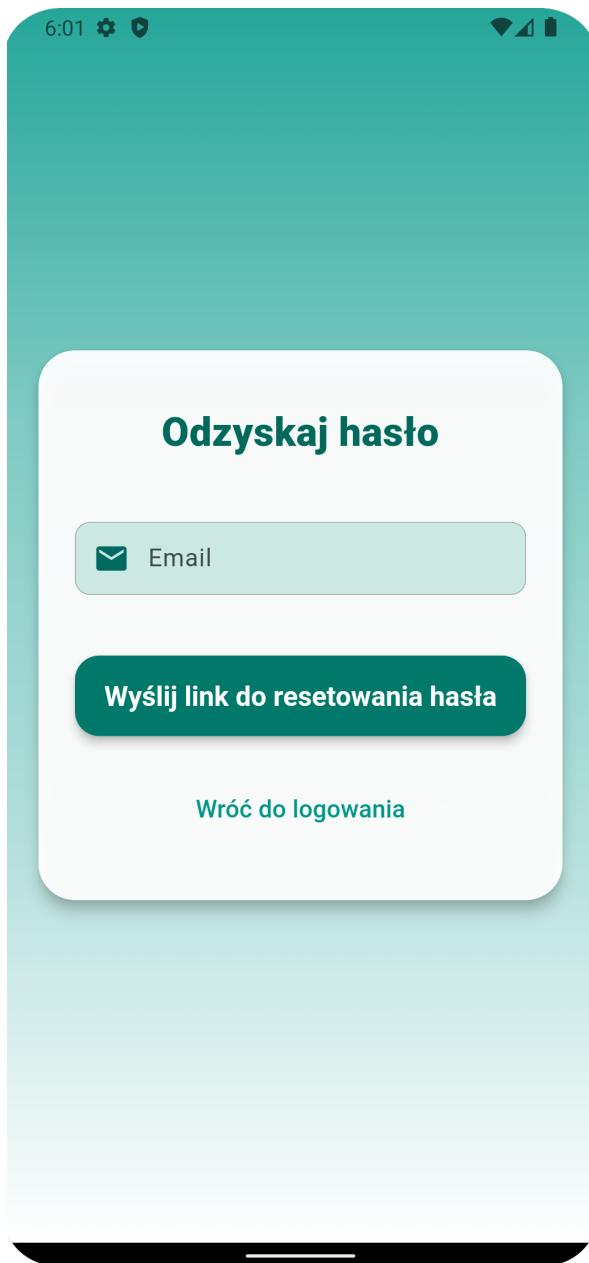
Pod formularzem znajdują się trzy opcje:

- „Zaloguj się” - przycisk wysyłający dane w celu zalogowania użytkownika i przekierowujący na ekran „Odkrywaj” (rysunek 5.9), jeśli dane logowania są poprawne.
- „Zresetuj hasło” - link prowadzący do ekranu odzyskiwania hasła.
- „Nie masz konta? Zarejestruj się” - link przekierowujący na ekran rejestracji (Rysunek 5.8), umożliwiający utworzenie nowego konta.



Rysunek 5.11 Ekran logowania

W przypadku wybrania opcji resetowania hasła użytkownik trafia na ekran odzyskiwania hasła (Rysunek 5.12). Ekran ten zawiera pole tekstowe, w którym należy wpisać adres e-mail, oraz przycisk „Wyślij link do resetowania hasła”. Po jego użyciu na podany adres e-mail wysyłana jest wiadomość z linkiem umożliwiającym zmianę hasła. Dodatkowo na ekranie znajduje się przycisk „Wróć do logowania”, który pozwala użytkownikowi powrócić do poprzedniego widoku.



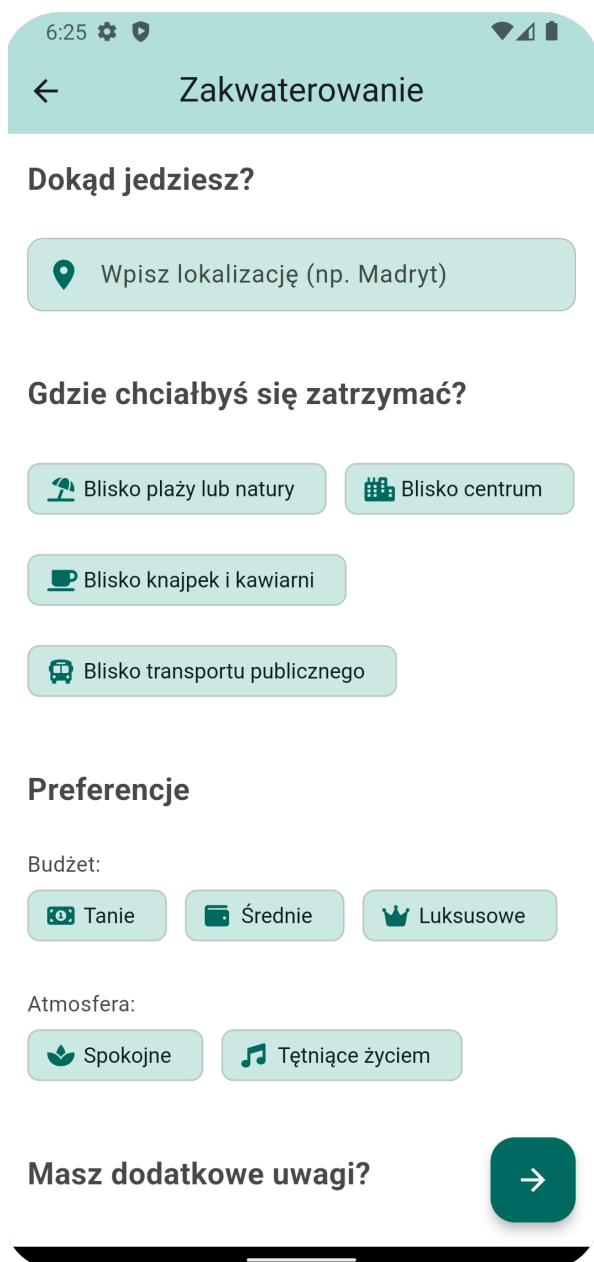
Rysunek 5.12 Ekran resetowania hasła

5.3. Wyszukiwanie rekomendacji zakwaterowania

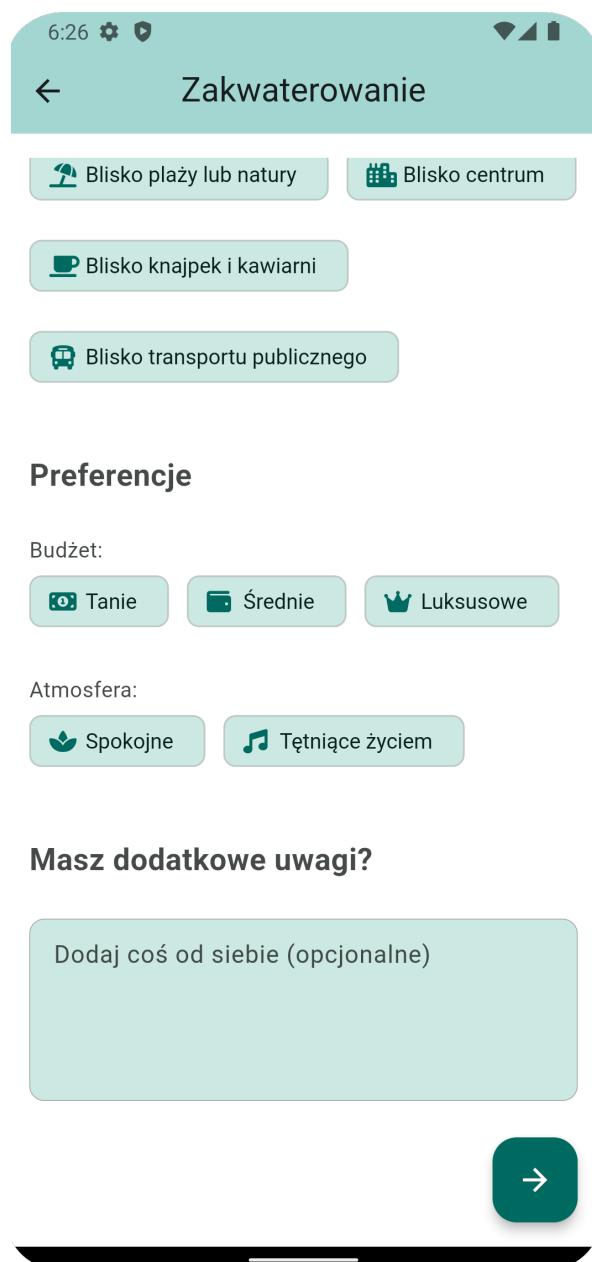
Proces wyszukiwania rekomendacji zakwaterowania rozpoczyna się na ekranie, gdzie użytkownik może dostosować swoje preferencje, aby uzyskać najbardziej dopasowane propozycje (Rysunek 5.13 i 5.14). Pierwszym krokiem jest wpisanie lokalizacji w polu „Dokąd jedziesz?”, gdzie użytkownik podaje nazwę miasta. Następnie, w sekcji „Gdzie chciałbyś się zatrzymać?”, można wybrać jedną lub kilka opcji, takich jak

„Blisko plaży lub natury”, „Blisko centrum”, „Blisko knajpek i kawiarni” czy „Blisko transportu publicznego”.

W kolejnych polach, w sekcji „Preferencje”, użytkownik określa swój budżet (tanie, średnie, luksusowe) oraz atmosferę (spokojne lub tętniące życiem). W każdej z tych kategorii można wybrać tylko jedną opcję. Na końcu ekranu znajduje się opcjonalne pole tekstowe „Masz dodatkowe uwagi?”, w którym użytkownik może wpisać dodatkowe informacje lub specyficzne wymagania, aby jeszcze lepiej dostosować wyniki wyszukiwania.



Rysunek 5.13 Wyszukiwanie zakwaterowania 1



Rysunek 5.14 Wyszukiwanie zakwaterowania 2

Na ekranie (Rysunek 5.15) przedstawiono stan po dokonaniu wyborów przez użytkownika. W sekcji „Co chcesz robić?” zaznaczono trzy kategorie: „Blisko plaży lub natury”, „Blisko centrum” oraz „Blisko knajpe i kawiarni”. W sekcji „Preferencje” użytkownik wybrał budżet „Średnie” oraz atmosferę „Spokojne”. Te ustawienia definiują, jakie oczekiwania aplikacja weźmie pod uwagę przy wyszukiwaniu zakwaterowania. Przycisk umieszczony na dole ekranu umożliwia przejście do kolejnego kroku. Po jego kliknięciu aplikacja wykorzystuje wybrane preferencje do wygenerowania najlepiej dopasowanych rekomendacji.



Rysunek 5.15 Wybór preferencji zakwaterowania

Po wybraniu preferencji użytkownik zostaje przeniesiony na ekran z wygenerowanymi propozycjami zakwaterowania (Rysunek 5.16 i 5.17). W górnej części ekranu znajduje się ogólna sugestia, podpowiadająca, jak unikać miejsc typowo turystycznych i odnaleźć bardziej autentyczne lokalizacje. Poniżej widoczne są karty z konkretnymi propozycjami zakwaterowania, które zawierają krótkie opisy miejsc, sugestie dotyczące bezpieczeństwa oraz orientacyjne koszty. Przycisk na dole ekranu pozwala na powrót do ekranu głównego.



Rysunek 5.16 Propozycje zakwaterowań 1

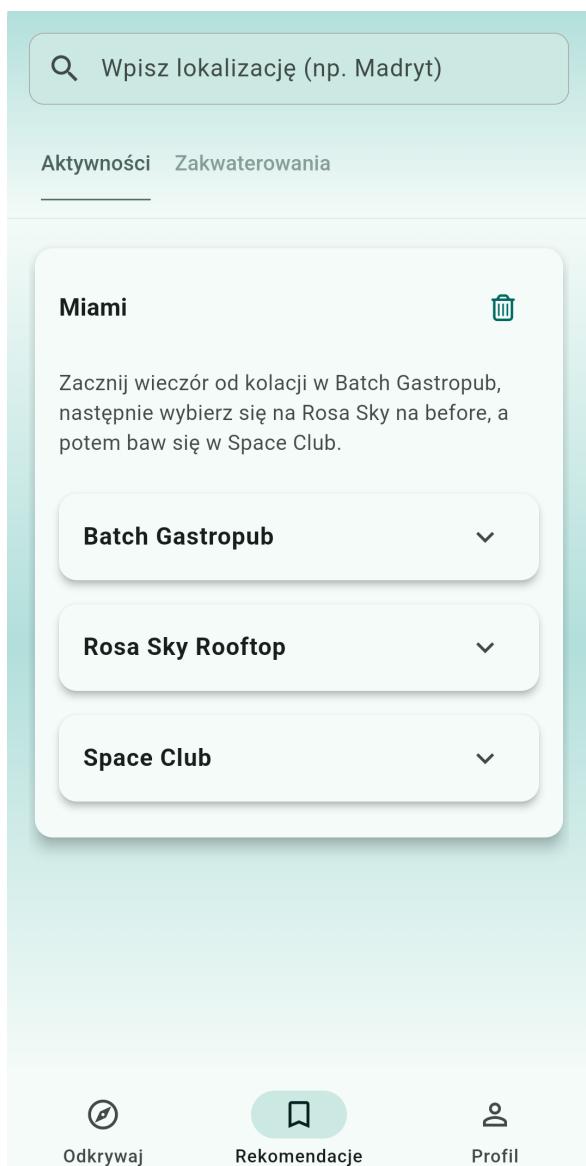


Rysunek 5.17 Propozycje zakwaterowań 2

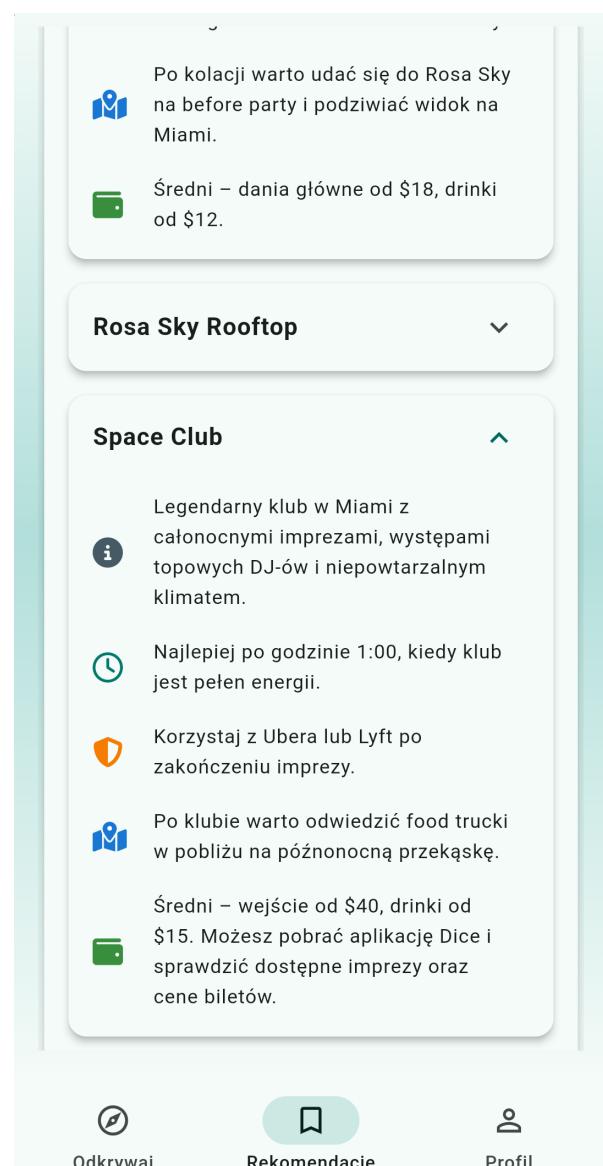
5.4. Przeglądanie zapisanych rekomendacji

Zakładka „Rekomendacje” (Rysunek 5.18 i 5.19) umożliwia przeglądanie zapisanych propozycji miejsc i aktywności, które zostały wygenerowane na podstawie wcześniejszych wyborów użytkownika. W górnej części ekranu znajduje się wyszukiwarka, która pozwala filtrować rekomendacje według nazwy miasta, np. „Miami” lub „Madryt”.

Lista jest podzielona na miejsca docelowe, a każda z nich zawiera zapisane miejsca. Kliknięcie w wybraną pozycję rozwija szczegóły, takie jak opis, sugerowany czas wizyty i dodatkowe wskazówki.



Rysunek 5.18 Przeglądanie listy rekomendacji 1



Rysunek 5.19 Przeglądanie listy rekomendacji 2

Każdą rekomendację można usunąć poprzez kliknięcie ikony kosza obok jej nazwy. Po wybraniu tej opcji pojawia się okno potwierdzenia z dwoma przyciskami (Rysunek 5.20):

- „Anuluj” - zamiera okno i pozostawia element na liście.
- „Usuń” - trwale usuwa rekomendację.



Rysunek 5.20 Usuwanie rekomendacji

6. Podsumowanie

Celem tej pracy było opracowanie aplikacji wspierającej odkrywanie lokalnych doświadczeń podróżniczych oraz pokazanie, jak sztuczna inteligencja ułatwia tworzenie personalizowanych rozwiązań i podnosi jakość doświadczeń użytkownika. Przeanalizowane zostały istniejące rozwiązania, które w większości skupią się na promowaniu najpopularniejszych miejsc i atrakcji turystycznych.

Cel pracy został osiągnięty, aplikacja została zaimplementowana i spełnia wszystkie założone wymagania funkcjonalne oraz niefunkcjonalne, co było możliwe dzięki trafnemu wyborowi technologii. Wykorzystanie sztucznej inteligencji pozwoliło na stworzenie aplikacji, która generuje precyzyjne rekomendacje dopasowane do preferencji użytkownika. Dzięki temu użytkownik otrzymuje dokładnie to, czego szuka, wraz z przydatnymi wskazówkami, jest to zupełnie inne podejście niż to, które oferują tradycyjne aplikacje promujące popularne atrakcje turystyczne.

Aplikacja LocAdvisor ma również duży potencjał do dalszego rozwoju. Możliwe jest dodanie nowych funkcjonalności, takich jak:

- Integracja z Google Custom Search API - automatyczne pobieranie odpowiednich zdjęć dla każdej lokalizacji, przechowywanie ich w Firebase Storage i wyświetlanie w interfejsie użytkownika.
- Integracja z Google Maps i Google Places API - umożliwienie wyświetlania mapy z zaznaczonymi strefami o różnym poziomie bezpieczeństwa. Potencjalnie niebezpieczne obszary mogłyby być oznaczone kolorem czerwonym, a miejsca wymagające większej ostrożności – żółtym, wraz z dodatkowymi wskazówkami dotyczącymi bezpieczeństwa.
- Integracja z Airbnb - wyświetlanie dostępnych ofert zakwaterowania w rekomendowanych dzielnicach, co mogłoby stanowić potencjalny model biznesowy poprzez nawiązanie partnerstwa.
- Zaawansowana analiza danych - wykorzystanie *machine learning* i *data science* do jeszcze lepszego dopasowywania rekomendacji i zwiększenia poziomu personalizacji.

- Moduł społecznościowy - przestrzeń, w której użytkownicy mogliby dyskutować o różnych miejscach, dzielić się własnymi doświadczeniami i rekomendować swoje ulubione lokalizacje.

Świat oferuje wiele do odkrycia, potrzeba jedynie narzędzia, które pomoże podróżnym w odnalezieniu autentycznych i wyjątkowych miejsc.

Literatura

1. Martin R. C., *Czysta architektura*, Helion S.A., 2018.
2. Miola A., *Flutter Complete Reference: Create beautiful, fast and native apps for any device*, Alberto Miola, 2020.
3. Kleppmann M., *Przetwarzanie danych w dużej skali. Niezawodność, skalowalność i łatwość konserwacji systemów*, Helion S.A., 2018.
4. Efthymiou P., *Clean Mobile Architecture: Become an Android, iOS, Flutter Architect*, Petros Efthymiou, 2022.
5. Khononov V., *Koncepcja Domain-Driven Design. Dostosowywanie architektury aplikacji do strategii biznesowej*, Helion S.A., 2022.
6. Cherny B., *Programming TypeScript*, O'Reilly Media, 2019.
7. Firebase Docs. Witryna internetowa. <https://firebase.google.com/docs>, stan z 20.01.2025.
8. Bloc Docs. Witryna internetowa. <https://bloclibrary.dev/>, stan z 20.01.2025.
9. OpenAI Docs. Witryna internetowa. <https://platform.openai.com/docs/overview>, stan z 20.01.2025.
10. Flutter Docs. Witryna internetowa. <https://docs.flutter.dev/>, stan z 20.01.2025.
11. Dart Docs. Witryna internetowa. <https://dart.dev/docs>, stan z 20.01.2025.

Spis rysunków

Rysunek 2.1 Aplikacja TripAdvisor	4
Rysunek 2.2 Aplikacja GetYourGuide	6
Rysunek 2.3 Aplikacja Spotted by Locals	7
Rysunek 3.1 Logo frameworka Flutter	9
Rysunek 3.2 Logo języka Dart	10
Rysunek 3.3 Logo Firebase	11
Rysunek 3.4 Logo Node.js	12
Rysunek 3.5 Logo języka TypeScript	13
Rysunek 3.6 Logo OpenAI	14
Rysunek 4.1 Model activity_requests	15
Rysunek 4.2 Model activity_recommenations	16
Rysunek 4.3 Model accommodation_requests	17

<u>Rysunek 4.4 Model accommodation_recommendations</u>	18
<u>Rysunek 4.5 Moduły aplikacji</u>	19
<u>Rysunek 4.6 Struktura modułów</u>	20
<u>Rysunek 4.7 Architektura BLoC</u>	21
<u>Rysunek 4.8 Implementacja ekranu zakwaterowań</u>	22
<u>Rysunek 4.9 Zarządzanie stanem zakwaterowań</u>	23
<u>Rysunek 4.10 Eventy zakwaterowań</u>	24
<u>Rysunek 4.11 State zakwaterowań</u>	24
<u>Rysunek 4.12 DTO rekommendacji aktywności</u>	25
<u>Rysunek 4.13 DTO aktywności</u>	25
<u>Rysunek 4.14 Integracja z Firebase Firestore</u>	26
<u>Rysunek 4.15 Indeksy w Firestore</u>	27
<u>Rysunek 4.16 Integracja z Firebase Authentication</u>	27
<u>Rysunek 4.17 Kody błędów Firebase</u>	28
<u>Rysunek 4.18 Mapowanie kodów błędów Firebase</u>	29
<u>Rysunek 4.19 Implementacja ekranu logowania</u>	30
<u>Rysunek 4.20 Integracja z Cloud Functions</u>	30
<u>Rysunek 4.21 Integracja z OpenAI API 1</u>	31
<u>Rysunek 4.22 Integracja z OpenAI API 2</u>	32
<u>Rysunek 4.23 Integracja z OpenAI API 3</u>	32
<u>Rysunek 4.24 Integracja z OpenAI API 4</u>	33
<u>Rysunek 4.25 Logi Cloud Functions</u>	34
<u>Rysunek 4.26 Model adresu e-mail</u>	35
<u>Rysunek 4.27 Testy adresu e-mail 1</u>	36
<u>Rysunek 4.28 Testy adresu e-mail 2</u>	36
<u>Rysunek 4.29 Wyniki testów jednoskowych</u>	37
<u>Rysunek 5.1 Ekran powitalny</u>	38
<u>Rysunek 5.2 Wyszukiwanie aktywności 1</u>	40
<u>Rysunek 5.3 Wyszukiwanie aktywności 2</u>	40
<u>Rysunek 5.4 Wybór preferencji aktywności</u>	41
<u>Rysunek 5.5 Propozycje aktywności 1</u>	42
<u>Rysunek 5.6 Propozycje aktywności 2</u>	42
<u>Rysunek 5.7 Modal przejścia do rejestracji</u>	43
<u>Rysunek 5.8 Ekran rejestracji</u>	44
<u>Rysunek 5.9 Ekran “Odkrywaj”</u>	45
<u>Rysunek 5.10 Ekran Profilu</u>	45
<u>Rysunek 5.11 Ekran logowania</u>	47
<u>Rysunek 5.12 Ekran resetowania hasła</u>	48
<u>Rysunek 5.13 Wyszukiwanie zakwaterowania 1</u>	49
<u>Rysunek 5.14 Wyszukiwanie zakwaterowania 2</u>	49
<u>Rysunek 5.15 Wybór preferencji zakwaterowania</u>	50
<u>Rysunek 5.16 Propozycje zakwaterowań 1</u>	51
<u>Rysunek 5.17 Propozycje zakwaterowań 2</u>	51

Rysunek 5.18 Przeglądanie listy rekomendacji 1	52
Rysunek 5.19 Przeglądanie listy rekomendacji 2	52
Rysunek 5.20 Usuwanie rekomendacji	53