

POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA

SPECJALNOŚĆ: SYSTEMY I SIECI KOMPUTEROWE

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja mobilna na platformę Android
umożliwiająca użytkowanie i uzupełnianie
tworzonej społecznościowo bazy danych
geoprzestrzennych

Mobile application for the Android platform
facilitating use of and contribution to a
crowdsourced geospatial database

AUTOR:

Bartosz Rodziewicz

PROWADZĄCY PRACĘ:

Dr inż. Paweł Trajdos, W4/K2

OCENA PRACY:

Spis treści

1 Wstęp	3
2 Opis koncepcji	4
2.1 Aplikacja mobilna	4
2.2 Serwer	4
2.3 Zakres realizacji	4
3 Wybrane technologie	6
3.1 Platforma	6
3.2 Język programowania	6
3.3 Zewnętrzne biblioteki	6
3.4 Komunikacja z serwerem	7
3.5 Środowisko deweloperskie	7
3.6 Kontrola wersji	8
3.7 System budowania	8
3.8 Testy jednostkowe	9
4 Opis implementacji	10
4.1 Struktura	10
4.1.1 Podział plików w projekcie	10
4.1.2 Przepływ danych	11
4.1.3 Widoki i nawigacja	13
4.2 Uruchomienie aplikacji i załadowanie mapy	14
4.3 Wyszukiwanie punktów	15
4.4 Przejście z użyciem data binding	15
4.5 Wyświetlenie listy miejsc	16
4.6 Wyświetlenie widoku szczegółów	17
4.7 Usuwanie obiektu	17
4.8 Dodawanie/edykcja obiektu	17
4.9 Implementacja repozytorium	18
4.10 Przechowywanie obiektów	19
5 Opis interfejsu użytkownika	20
5.1 Ekrany główne	20
5.2 Wyszukiwanie punktów	21
5.3 Tryb edycji i dodawania obiektów	22
5.4 Widok szczegółowy i usuwanie miejsc	23
6 Podsumowanie	24
Literatura	25
Indeks rycin	26
Indeks listingów	26

1 Wstęp

W dzisiejszych czasach praktycznie każdy posiada smartfon. Większość z nich jest wyposażona w najczęściej stałe połoczenie z Internetem. Pozwoliło to na wykształcenie się na rynku wielu aplikacji, których celem jest pomoc użytkownikowi w znalezieniu konkretnych informacji, w tym również geolokalizacyjnych. Istnieje wiele serwisów oferujących dostęp do map, ułatwiających znajdywanie firm, wyświetlających aktualne położenie autobusów komunikacji miejskiej, czy umożliwiających zlokalizowanie publicznych toalet w okolicy.

Celem tej pracy jest opracowanie aplikacji na urządzeniu z systemem Android umożliwiającej użytkownikom na łatwiejsze znajdywanie miejsc użyteczności publicznej, gdzie goście mają dostęp do energii elektrycznej, np. restauracji, w której można usiąść przy stoliku z laptopem i podłączyć się do prądu. Aplikacja będzie klientem do niezależnie rozwijanego serwera z bazą danych. Zaoferuje ona możliwość wyświetlania zebranych w niej danych, oraz umożliwi operowanie na nich w przystępny i szybki dla użytkownika sposób.

Program ten będzie pisany, wykorzystując najlepsze wzorce projektowe oraz odpowiednią warstwę abstrakcji, tak aby jego dalszy rozwój był łatwy, również dla innych osób. Dodatkowo umożliwi to łatwą zmianę przechowywanych danych w bazie, jeśli w przyszłości będzie taka konieczność bądź chciano by wykorzystać ten projekt do stworzenia innej aplikacji podobnego typu.

Praca jest podzielona w następujący sposób. Rozdział 2 bardziej szczegółowo opisuje koncepcję aplikacji, jej zastosowanie i to, co zostało zrealizowane. Rozdział 3 przedstawia technologie wybrane do stworzenia programu. Rozdział 4 opisuje aspekty implementacji kodu oraz stanowi jego dokumentację. Rozdział 5 został wykorzystany do prezentacji i opisania interfejsu użytkownika aplikacji. W rozdziale 6 zamieszone jest podsumowanie pracy. Na końcu dokumentu znajdują się odwołania do literatury, z której korzystano w trakcie przygotowywania tej pracy oraz indeks użytych rycin i listingów.

2 Opis koncepcji

2.1 Aplikacja mobilna

Pomysł na aplikację powstał obserwując inne aplikacje użytkowe, które są bardzo popularne na rynku. Głównie była ona bazowana na dwóch serwisach — aplikacji do wyszukiwania publicznych toalet (Flush), oraz publicznych sieci Wi-Fi (WiFi Map). W obu przypadkach, głównym punktem wokół którego kręci się cała idea, jest mapa ze społecznie zbieranymi danymi przez użytkowników, którzy w założeniu sami dbają o poprawność i aktualność danych.

Tutaj też głównym miejscem miał być widok mapy, umożliwiający na przeglądanie danych w łatwy dla użytkownika sposób. Dodatkowo dane miały być możliwe do wyświetlenia w formie listy, sortowanej rosnąco odległością od użytkownika.

Aby zmniejszyć obciążenie urządzenia, oraz umożliwić pracę na dużej bazie danych, klient miał pobierać punkty znajdujące się blisko lokalizacji użytkownika, oraz te znajdujące się w zakresie widoku mapy, wyświetlanej na ekranie. Aby zmniejszyć ilość zapytań do serwera aplikacja miała zapewniać metodę na buforowanie danych na urządzeniu. Wprowadzenie takiego rozwiązania jest możliwe, ponieważ program wyświetla rzadko zmieniające się informacje. Dane przechowywane miały być przez z góry zdefiniowany czas, bądź do sytuacji, gdy aplikacja przekroczy dozwolony rozmiar pamięci buforu. Klient miał również oferować ręczne sterowanie, który obszar będzie zapisany lokalnie, aby umożliwić dostęp do danych w trybie offline.

Baza miała przechowywać podstawowe dane o miejscu, jak jego lokalizacja czy nazwa, oraz bardziej specyficzne dla tego zastosowania informacje, takie jak ilość publicznie dostępnych gniazdek, czy szkic poglądowy z zaznaczonymi punktami na planie budynku (co np. w przypadku restauracji ułatwiałoby klientowi wybranie odpowiedniego stolika, bez konieczności pytania obsługi). Planowana była też szersza integracja z Google Maps, aby miejsca w aplikacji można było przypisać do punktów POI (*Points of interests*) znajdujących się w tym serwisie.

W aplikacji miał być dostępny widok szczegółowy danego miejsca, umożliwiający podejrzenie dodatkowych informacji o nim oraz historię jego ostatnich modyfikacji. Widok dodawania nowego miejsca miał umożliwić podanie wszystkich szczegółów dotyczących tego punktu, oraz miał ułatwiać dodanie planu budynku naszkicowanego ręcznie, bądź poprzez wgranie zdjęcia fizycznego wykonanego planu.

Klient miał oczywiście pozwalać również na edycje istniejących punktów, jak i usuwanie błędnie stworzonych, bądź już nieistniejących. Planowane było też wprowadzenie kont użytkownika, aby śledzić kto dokonuje modyfikacji, oraz zablokować tych, którzy pogarszają jakość danych w aplikacji.

2.2 Serwer

Do działania aplikacja potrzebowałaby również internetowego serwera z bazą danych, jednak nie stanowi to przedmiotu tej pracy. Serwer miał być napisany w sposób umożliwiający tej aplikacji, na połączenie się z nim, oraz wykorzystywać odpowiednią warstwę abstrakcji, by w przyszłości możliwe było stworzenie klientów na inne platformy (klient webowy, na system iOS itp.)

2.3 Zakres realizacji

Oczywiście powyższy opis konceptu przekracza to, co było planowane w zakresie tej pracy, oraz to, co udało się zrealizować. W trakcie wykonywania pracy stworzone zostały następujące funkcjonalności:

- przeglądanie punktów w formie mapy i listy,
- dodawanie, edycja i usuwanie punktów,
- przeszukiwanie bazy danych,
- połączenie z serwerem i synchronizacja danych,
- zapisywanie danych z serwera na urządzeniu, aby zminimalizować ilość zapytań.

Jest to całkiem solidny zakres podstawowych funkcji, umożliwiający na używanie aplikacji w założonych celach i jej dalszy rozwój.

3 Wybrane technologie

3.1 Platforma

Z punktu widzenia projektu wybór platformy mobilnej był jedynym słusznym wyborem, ponieważ aplikacja dostarcza informacje, które są potrzebne użytkownikom, gdy są w biegu, a nie siedzą w zaciszu swojego domu przed komputerem. Na rynku platform mobilnych aktualnie istnieje jedynie dwójka graczy, czyli Android od firmy Google oraz iOS od firmy Apple, mających udziały w rynku odpowiednio 77% i 22% [1]. Z tych danych wychodzi jasny obraz, że aby dotrzeć do jak największej liczby użytkowników, należy wybrać platformę Android. Dodatkowym aspektem skłaniającym do wyboru tej platformy był fakt posiadania przez autora pracy urządzeń działających pod kontrolą tego systemu.

3.2 Język programowania

Wybór platformy w dużej mierze uwarunkował wybór języka programowania. Istnieją oczywiście różne metody, by pisać na Androida w wielu różnych językach (np. COBOL), jednak oficjalnie wspierane są następujące języki [2]:

- Kotlin — nowy język, działający w JVM i będący w pełni interoperacyjny z Java, od niedawna zalecany przez Google jako główny język dla nowych aplikacji [3];
- Java — standardowy język, w którym od dawna powstają aplikacje na tę platformę;
- C++ — istnieje możliwość wykorzystania bibliotek napisanych w C++ za pomocą NDK udostępnionego przez Google, przydatne przy oprogramowaniu, dla którego kluczowa jest wydajność, czyli np. gier;
- HTML+CSS+JS — częściowo wspierane jest tworzenie nowoczesnych stron internetowych zachowujących się jako aplikacje wykorzystując PWA.

Taka sytuacja sprowadza się do wyboru pomiędzy dwoma językami: Kotlinem i Java. Mimo pewnego wcześniejszego doświadczenia autora pracy z Java wybrany został język Kotlin, który jest traktowany przez Google jako przyszłość dla tej platformy. Zdecydowano się na ten język, aby poznać nową technologię, która zaczyna zdobywać coraz większą popularność na rynku oraz ułatwić w przyszłości dalszy rozwój tego projektu.

3.3 Zewnętrzne biblioteki

Przy budowie projektu wykorzystano kilka zewnętrznych bibliotek, z których większość wchodzi w skład *Android Framework*, czyli oficjalnych bibliotek wymaganych do tworzenia aplikacji na tę platformę.

Podstawową użytą biblioteką jest biblioteka języka Kotlin, pozwalająca na pisanie kodu w tym języku oraz biblioteka *Core* z pakietu *AndroidX*, czyli główna biblioteka wymagana przez system, implementująca podstawowe definicje takich elementów jak aktywności czy widoki programu [3]. Większość widoków w aplikacji stworzono wykorzystując *ConstraintLayout*, którego podstawowa definicja znajduje się w bibliotece o tej samej nazwie.

Przy tworzeniu wykorzystano również bibliotekę *AppCompat*, która pozwoliła na modyfikacje paska akcji na górze widoku aplikacji oraz implementację mechanizmu wyszukiwania miejsc.

Do wsparcia przy budowie nawigacji pomiędzy ekranami programu wykorzystano bibliotekę o nazwie *Navigation*, ponieważ jest to zalecona metoda tworzenia przejść przez Google.

Z tego samego powodu wykorzystano bibliotekę *Lifecycle* do lepszego wsparcia zarządzania życiem poszczególnych elementów aplikacji.

W programie zaimplementowano bazę danych w technologii *SQLite* wykorzystując bibliotekę *Room*, która pozwala w prosty sposób tworzyć bazy danych do aplikacji na platformę Android.

Do obsługi map oraz lokalizacji wykorzystano biblioteki *Google Maps Services*, ponieważ jest to oficjalnie wspierana biblioteka do tych zastosowań.

Obsługa połączenia z serwerem została zaimplementowana z pomocą biblioteki *Retrofit*, która pomaga w obsłudze zapytań HTTP wraz z biblioteką *Moshi*, która służy do automatyzacji parsowania modelu programistycznego do JSON i w drugą stronę. Biblioteka *Moshi* jest pierwszą użytą w projekcie biblioteką niewchodząącą w skład *Android Framework*.

Spoza *Android Framework* wykorzystano jeszcze jedną bibliotekę — *Timber*, która znacząco ułatwia korzystanie z systemowych logów platformy Android.

3.4 Komunikacja z serwerem

Aplikacja do kontaktu z serwerem wykorzystuje jego publiczne API oparte o zapytania HTTP i architekturę typu REST [4]. Serwer nie posiada jeszcze domeny, więc aby się z nim połączyć konieczne jest uruchomienie go w sieci lokalnej i zmiana kodu aplikacji mobilnej podając jej poprawny adres działającego serwera. Serwer jednak nie był tematem tej pracy i został on stworzony przez inną osobę.

Aplikacja implementuje następujące zapytania do serwera:

- pobranie wszystkich punktów z bazy danych — zapytanie typu GET, które zwraca listę punktów;
- dodanie nowego punktu — zapytanie typu PUT, które w ciele przyjmuje obiekt z wszystkimi parametrami miejsca, jednak bez Id, zwraca dodany obiekt do bazy danych z poprawnym Id;
- edycja istniejącego punktu — zapytanie typu PATCH, które przyjmuje zmieniony obiekt i zwraca go;
- usunięcie punktu — zapytanie typu DELETE, które przyjmuje obiekt do usunięcia i zwraca pustą odpowiedź z kodem HTTP 200.

Wszystkie zapytania wykorzystują to samo URI, jednak różnią się typem i zawartością swojego ciała.

3.5 Środowisko deweloperskie

Projekt realizowany był z użyciem komputera wyposażonego w procesor Intel Pentium G3258@4.4GHz, 16GB pamięci RAM DDR3-1333 oraz układ graficzny AMD Pitcairn XT na karcie Radeon HD 7870 z 2 GB pamięci, pracujący pod kontrolą systemu operacyjnego Microsoft Windows 10 Pro w wersji 1903 (Build 18362.476).

Główny telefon wykorzystany do projektu to OnePlus 5T posiadający układ Snapdragon 835 i 6GB pamięci operacyjnej działający pod kontrolą systemu Android w wersji 9 Pie z nakładką OxygenOS w wersji 9.0.9.

Program komplikowany był z użyciem Kotlina w wersji 1.3.60 do kodu bajtowego zgodnego z JVM w wersji 1.6. Proces budowania był sterowany przez narzędzie Gradle w wersji 5.4.1 z wtyczką do Androida (*Android Gradle Plugin*) w wersji 3.5.1. Kod edytowany był w środowisku deweloperskim (IDE) Android Studio w wersji 3.5.2.

Najniższa wspierana przez aplikację wersja systemu Android to 5.0 Lollipop. Oprogramowanie było komplikowane używając SDK w wersji 29, odpowiadającej najnowszej wersji Androida — 10.

Aplikacja została przetestowana również na następujących urządzeniach:

- OnePlus One — Android 9 Pie
- Xiaomi Redmi Note 5A Prime — Android 7.1.2 Nougat
- Nokia 8 — Android 9 Pie
- Samsung Galaxy Tab 8.4 Pro — Android 7.1.2 Nougat (*tablet*)
- Nexus 5X — Android 10 (*maszyna wirtualna*)
- Nexus 4 — Android 5.0 Lollipop (*maszyna wirtualna*)

Na każdym z wymienionych urządzeń aplikacja działała w pełni poprawnie. Na maszynach wirtualnych nie została przetestowana funkcjonalność połączenia z serwerem z uwagi na problem z konfiguracją połączenia maszyny z siecią lokalną.

3.6 Kontrola wersji

W większości projektów informatycznych narzędzia systemu kontroli wersji pomagają zachować porządek i ułatwiają nad nim pracę [5]. Z tego też powodu ten projekt również wykorzystywał narzędzie tego rodzaju — Git.

Repozytorium Git wykorzystywane było głównie, aby lepiej zorganizować zmiany i łatwiej się w nich odnajdywać. Pozwalało na jednoczesną pracę nad kilkoma funkcjonalnościami jednocześnie, zachowując uporządkowaną historię zmian. Dodatkowo zabezpieczało ono sprawnie działający kod, pozwalając na cofnięcie niechcianych zmian w każdej chwili używając jednej komendy. System ten zapewniał też swojego rodzaju kopię zapasową, ponieważ umożliwiał on na synchronizację wszystkich zmian z serwerem serwisu GitHub, gdzie kod przechowywany był w formie prywatnego projektu.

3.7 System budowania

Do automatyzacji procesu budowania wykorzystano narzędzie Gradle. Pozwalało ono, po sprecyzowaniu podstawowych ustawień, na w pełni automatyczne tworzenie pliku apk, gotowego do instalacji na urządzeniu.

W przypadku projektów na platformę Android konfiguracja Gradle podzielona jest na dwa osobne pliki — konfiguracja do całego projektu oraz poszczególnego modułu. Mimo, że projekt posiadał tylko jeden moduł, zachowany został ten podział, ponieważ uważany jest on za dobrą praktykę [6].

W pliku z globalną konfiguracją sprecyzowane zostały zdalne repozytoria, z których Gradle pobierał wymagane dependencje oraz zdefiniowane zostało użycie narzędzi wymaganych do zbudowania aplikacji na system Android.

```
compileSdkVersion 29
dataBinding {
    enabled = true
}
androidExtensions {
    experimental = true
}
buildToolsVersion "29.0.2"
defaultConfig {
    applicationId "test.mug.espresso"
    minSdkVersion 21
    targetSdkVersion 29
    versionCode 2
    versionName "0.1.1"
    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}
```

Listing 1: Konfiguracja narzędzia do budowy Gradle

W pliku konfiguracyjnym modułu programu znajdowały się podstawowe ustawienia, wymagane do budowy (listing 1).

Zdefiniowana została wersja systemu Android, użyta do zbudowania binarki, minimalna wersja wymagana do działania programu oraz wersja narzędzia używanego do budowania. Aktywowano technologię data binding, która w prosty sposób umożliwia tworzenie powiązań pomiędzy kodem a definicją układu widoku oraz aktywowało eksperymentalne funkcjonalności Android Framework.

Ustalona została nazwa kodowa programu oraz jego wersja. Zgodnie z konwencją systemu Android zalecane jest użycie jako nazwy kodowej odwrotności domeny przeznaczonej dla danej aplikacji. Z uwagi, że do projektu nie została zarejestrowana jeszcze żadna domena, na czas realizacji pracy zdecydowano się użyć domeny `test`, która przeznaczona jest do użycia w testowaniu oprogramowania i gwarantuje pewność, że nigdy nie nastąpi konflikt z żadną istniejącą domeną, ponieważ zablokowana jest możliwość jej rejestracji [7].

3.8 Testy jednostkowe

Do projektu podłączono biblioteki *JUnit* umożliwiające pisanie lokalnych testów jednostkowych oraz testów wykonywanych na fizycznym urządzeniu. Niestety z uwagi na ograniczenia czasowe do aplikacji nie powstały żadne testy.

4 Opis implementacji

4.1 Struktura

4.1.1 Podział plików w projekcie

Nawet prosty projekt na platformę Android ma dość skomplikowaną strukturę plików i wymaga chwili do zapoznania się z nią.

Aplikacje na ten system pisane są w formie modułowej. Ten projekt posiada tylko jeden moduł, który nazywa się `app`.

W głównym folderze projektu znajduje się kilka plików konfiguracyjnych dla narzędzia *Gradle*, które służy do automatyzacji procesu budowania. Folder `.idea/` zawiera pliki konfiguracyjne środowiska IDE — *Android Studio*.

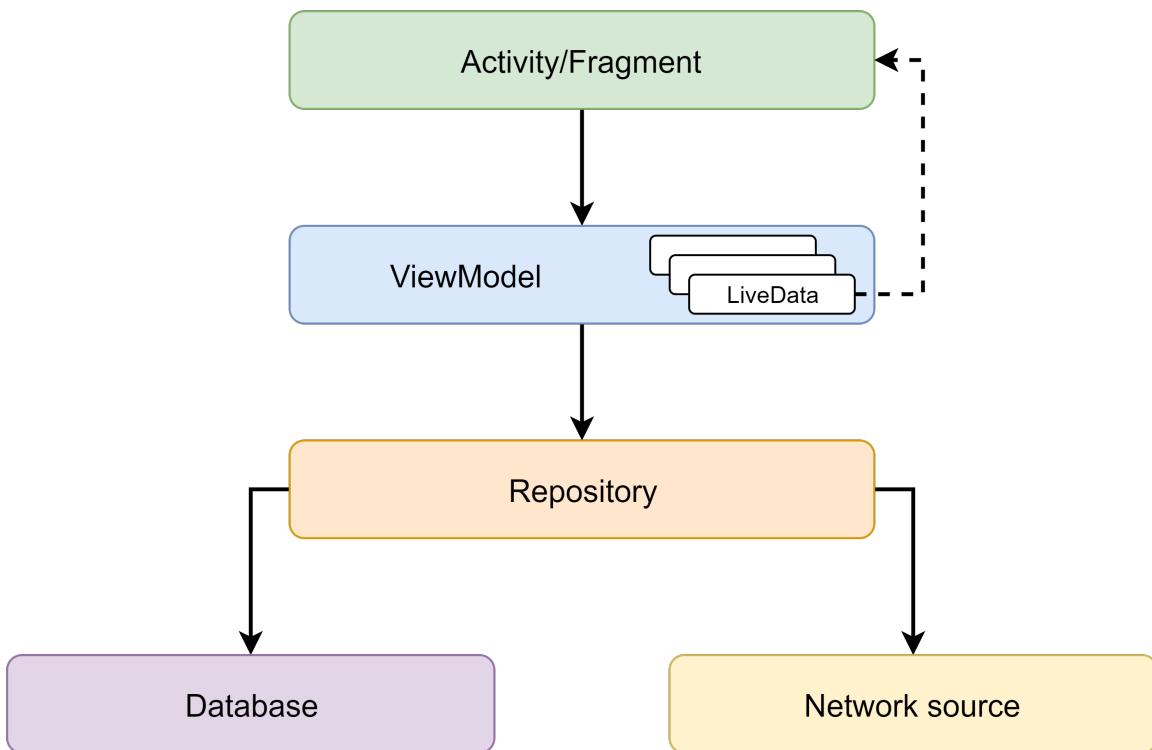
Wszystkie pliki w folderze `app/` to pliki należące do tego modułu, folder `app/src/` zawiera 5 podfolderów, które wprowadzają podstawowy podział kodu aplikacji:

- `main/` zawiera kod wspólny dla wszystkich metod budowania,
- `debug/` to kod przeznaczony do budowania w tym trybie,
- `release/` to kod przeznaczony do użycia przy budowaniu w trybie `release`, w tym projekcie te dwa powyższe foldery są używane do przechowywania osobnych plików z kluczem API do Google Maps,
- `test/` i `androidTest/` to foldery wykorzystywane przez testy jednostkowe.

Główny podział kodu znajduje się w katalogu `app/src/main/`:

- folder `java/` zawiera kod pisany w języku Java lub Kotlin (w tym projekcie tylko Kotlin), struktura pod folderów odpowiada nazwie kodowej aplikacji,
- folder `res/` zawierający zasoby zapisane w formacie XML, gdzie najważniejsze to pliki definicji układu widoków (`res/layout/`) oraz definicji menu (`res/menu/`),
- `AndroidManifest.xml` — plik informujący system Android o strukturze tej aplikacji.

4.1.2 Przepływ danych



Rysunek 1: Schemat architektury

Z perspektywy obsługi danych architektura aplikacji była pisana wzorując się na sugerowanej przez Google strukturze [8][6]. Została ona przedstawiona na Ryc. 1.

Stworzona została klasa pełniąca funkcję repozytorium, które jest pośrednikiem w dostępie do danych z bazy oraz obsługuje zapytania sieciowe, by aktualizować dane znajdujące się w niej. Dostęp do repozytorium wykonywany jest z poziomu klas typu **ViewModel**. Starano się w nich zawrzeć jak największą część kodu odpowiedzialnego za manipulowanie danymi. W klasach aktywności i fragmentów starano się zostawić tylko kod odpowiedzialny za obsługę interfejsu użytkownika. Odświeżanie danych wyświetlanych na ekranie wykonywane jest z poziomu klas **ViewModel** korzystając ze zmiennych typu **LiveData** i technologii data binding.

```
└── EspressoApplication.kt
└── mainView
    ├── DataViewModel.kt
    ├── ListViewFragment.kt
    ├── MainActivity.kt
    └── MapViewFragment.kt
└── detailView
    ├── DetailViewFragment.kt
    └── DetailViewModel.kt
└── addEditView
    ├── AddViewFragment.kt
    └── AddViewModel.kt
└── domain
    ├── PowerMug.kt
    └── PowerMugWithDistance.kt
└── repository
    └── PowerMugRepository.kt
└── database
    ├── DbPowerMug.kt
    ├── PowerMugDatabase.kt
    └── PowerMugDatabaseDao.kt
└── network
    ├── NetworkPowerMug.kt
    └── ServerApiService.kt
└── BindingAdapters.kt
└── Utils.kt
```

Listing 2: Lista plików z kodem źródłowym (*Pliki są wyświetlane w recznie ustalonej kolejności*)

Schemat ten ma swoje odbicie w układzie plików z folderu zawierającego kod źródłowy (listing 2).

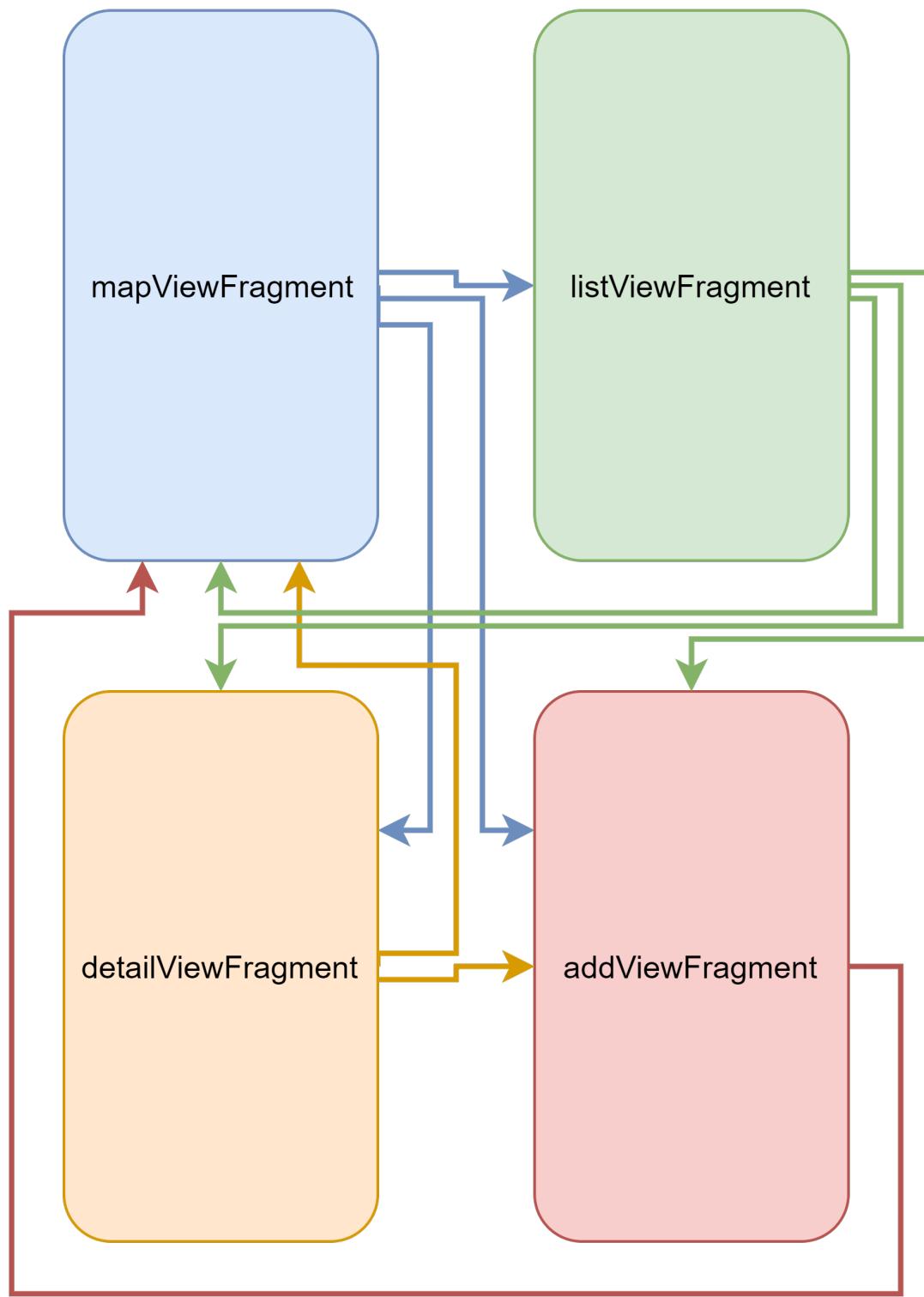
Aplikacja składa się z jednej aktywności, której jedynym zadaniem jest załadowanie odpowiedniego pełnoekranowego fragmentu.

Do każdego fragmentu (widoku) stworzony został odpowiedni ViewModel, z wyjątkiem dwóch głównych fragmentów — MapView i ListView, które posiadają jeden wspólny obiekt tego typu.

Klasy reprezentujące struktury używane przez ViewModel zostały wydzielone do folderu domain. W kolejnych folderach pogrupowane są klasy wykorzystywane przez repozytorium, bazę danych i obsługę sieci.

Na samym końcu znajdują się dwa zbiorcze pliki zawierające pomocnicze wolne funkcje (niebędące częścią żadnej klasy).

4.1.3 Widoki i nawigacja



repozytorium

Rysunek 2: Diagram nawigacji pomiędzy fragmentami

Aplikacja posiada jedną aktywność i cztery widoki, w tym dwa główne. Aktywność przy tworzeniu aktywuje szablon nawigacji (znajdujący się w res/navigation/), który pomaga

w bezproblemowym zarządzaniu przechodzeniem pomiędzy fragmentami. Schematyczny diagram tego szablonu został przedstawiony na Ryc. 2.

Głównym i domyślnym widokiem jest widok mapy. Uruchamia się on zaraz po starcie aplikacji. Umożliwia przeglądanie danych w postaci punktów na mapie zajmującej cały ekran. Kliknięcie w punkt powoduje wyświetlenie szczegółów danego miejsca.

Drugim głównym widokiem jest widok pozwalający na wyświetlenie miejsc w formie sortowanej listy. Przejście do niego jest możliwe z ekranu mapy za pomocą przycisku na dole. Kliknięcie pozycji reprezentującej konkretne miejsce powoduje wyświetlenie się jego szczegółów.

Oba widoki główne posiadają przycisk do dodania nowego miejsca, który przenosi do widoku dodawania. Po dodaniu następuje przejście do ekranu mapy. Ekran ze szczegółami poza wyświetlaniem dodatkowych informacji o miejscu pozwala na usunięcie miejsca z bazy. Po usunięciu następuje przejście do widoku mapy.

Android oferuje dwa rodzaje przejść pomiędzy ekranami [2] — te opisane wyżej to przejścia w przód, które wykonywane są poprzez interakcje użytkownika z aplikacją. Drugim typem przejść są przejścia w tył wykonywane przez użycie przycisku wstecz. Tego typu przejścia powracały do poprzedniego ekranu. Aby nawigacja w aplikacji działała poprawnie i intuicyjnie dla użytkownika wszystkie przejścia "w przód" powracające do ekranu mapy powodują wy czyszczenie kolejki wcześniejszych widoków, aby kliknięcie przycisku wstecz na widoku mapy powodowało opuszczenie aplikacji.

4.2 Uruchomienie aplikacji i załadowanie mapy

Pierwszym krokiem po uruchomieniu aplikacji jest wykonanie się kodu z klasy Aplikacji (EspressoApplication.kt). Kod tam znajdujący się należy ograniczyć do minimum [6] i w tym projekcie znajduje się tam jedynie aktywacja zewnętrznej biblioteki *Timber* pomagającej w tworzeniu logów.

Następnie tworzona jest aktywność, która aktywuje szablon nawigacji ładujący główny widok, którym jest ekran mapy.

Podczas tworzenia się widoku mapy (metoda `onCreateView()`) następuje załadowanie definicji układu elementów widoku (znajdującej się w `res/layout/`). Dzięki wykorzystaniu data binding już w pliku layout przypisane są odpowiednie metody z ViewModelem, które mają się wykonać po kliknięciu przycisków. Następnie jest tworzona (bądź podłączona, jeśli już istnieje) instancja klasy ViewModel dla tego widoku.

Przy tworzeniu ViewModel, tworzona jest instancja repozytorium. Po stworzeniu repozytorium zostaje momentalnie zwrócona lista miejsc znajdująca się w bazie i następuje żądanie aktualizacji bazy poprzez połączenie z serwerem. Takie działanie powoduje, że użytkownik od razu po uruchomieniu aplikacji może z niej korzystać, zamiast czekać na pobranie się danych z serwera, które wcale nie musiały ulec zmianie. Po zakończeniu procesu pobierania repozytorium aktualizuje w tle dane w bazie oraz dzięki data binding zmiany te propagują się do wszystkich widoków aplikacji.

Po stworzeniu ViewModel następuje stworzenie obserwatorów zmiennych `LiveData` w ViewModelu używanych do nawigacji oraz żądanie inicjalizacji mapy do biblioteki Google Maps Services i zapytanie użytkownika o pozwolenie na dostęp do lokalizacji (jeśli wcześniej nie zostało ono udzielone).

Po poprawnej inicjalizacji mapy następuje stworzenie obserwatora listy punktów i wypełnienie mapy punktami. Następuje też przypisanie działania, które ma się wykonać po kliknięciu na dany punkt. Widok mapy przechodzi na aktualną pozycję użytkownika.

Obecność elementu mapy zaburza poprawną implementację wzorca projektowego MVVM (Model-View-ViewModel) [9], ponieważ zgodnie z nim część kodu odpowiedzialnego za ob-

slugę map powinna zostać przeniesiona do ViewModela. Nie udało się tego jednak dokonać podczas realizacji tego projektu.

4.3 Wyszukiwanie punktów

Aplikacja oferuje wyszukiwanie punktów po nazwie lub adresie. Przeszukiwanie dostępne jest z poziomu przycisku znajdującego się na panelu na górze ekranu mapy.

Przy tworzeniu menu (w trakcie tworzenia widoku) do obiektu wyszukiwania przypisywany jest queryTextListener, który zawiera definicje metody, którą należy wywołać po zatwierdzeniu wyszukiwania.

Wyszukiwanie polega na wykonaniu queryTextListener.onQueryTextSubmit(String) [3], czyli wysłaniu do repozytorium tekstu z pola wyszukiwania i czekaniu na wyniki. Repozytorium odpytuje bazę danych i zwraca do ViewModelu listę znalezionych miejsc.

Po pojawienniu się wyników mapa jest czyszczona i pokazywane są na niej tylko znalezione miejsca, oraz widok automatycznie przechodzi na pierwsze znaleziony punkt. W przypadku braku wyników pojawia się stosowny komunikat.

Powrót do wyświetlania wszystkich miejsc jest możliwy wychodząc z wyszukiwania przy ciskiem na górnym pasku aplikacji. Następuje wtedy wywołanie metody closeListener, która czyści mapę i ponownie ustawia obserwatora na domyślną zmienną LiveData z listą wszystkich miejsc.

4.4 Przejście z użyciem data binding

Większość przejść w tym projekcie realizowana jest zgodnie z poniższym schematem. Jest to zalecana metoda wykonywania przejść, ponieważ już w definicji układu widoku widać, do czego służy dany przycisk [6].

```
private var _navigateToSecondView = MutableLiveData<Boolean>()
val navigateToSecondView: LiveData<Boolean>
    get() = _navigateToSecondView

fun goToSecondView() {
    _navigateToSecondView.value = true
}

fun wentToSecondView() {
    _navigateToSecondView.value = false
}
```

Listing 3: Kod znajdujący się w klasie typu ViewModel potrzebny do przejścia

```

viewModel.navigateToSecondView.observe(viewLifecycleOwner, Observer {
    if (it == true) {
        this.findNavController()
            .navigate(R.id.action_mapViewFragment_to_listViewFragment)
        viewModel.wentToSecondView()
    }
})

```

Listing 4: Kod obserwatora potrzebny do przejścia

Metoda polega na przypisaniu na kliknięcie układu widoku metody z ViewMode'a (w tym wypadku `goToSecondView()`). Metoda ta, zmienia wartość zmiennej `LiveData` (listing 3) na którą ustawiony jest obserwator w widoku (listing 4), który aktywuje się jeśli zmienna ma wartość `true`. Wywołuje on kontroler nawigacji i wykonuje odpowiednie przejście, po czym ustawia zmienną `LiveData` (w tym wypadku `_navigateToSecondView`) na `false` wykorzystując drugą metodę (w tym wypadku `wentToSecondView()`). Ustawienie jej bezpośrednio jest niemożliwe, ponieważ jest to zmienna prywatna, a `navigateToSecondView` jest zmienną publiczną, której wartość nie może być modyfikowana — jest to zdefiniowane w ten sposób, aby zachować enkapsulację klasy.

Konieczność wykonania takiego przepływu jest spowodowana faktem, że użycie kontrolera nawigacji jest możliwe tylko z poziomu kodu fragmentu, a nie kodu ViewMode'a [3].

4.5 Wyświetlenie listy miejsc

Wyświetlenie listy miejsc wymaga przejścia do widoku listy. Przejście z widoku mapy do widoku listy jest użyte jako przykład do opisu przejścia w podrozdziale powyżej.

Zaimplementowany w tym projekcie widok listy wykorzystuje specjalny widok Android Framework o nazwie `RecyclerView`. Jest to widok przystosowany do wyświetlania bardzo dużych zbiorów danych bez wpływu na pamięć urządzenia. Widok ten, zamiast tworzyć jedną długą listę i wyświetlać tylko jej fragment, tworzy listę zawierającą niewiele więcej elementów niż mieści się w danej chwili na ekranie i wraz z jej przewijaniem ponownie wykorzystuje obiekty w pamięci, które znikają z ekranu, aby wyświetlić nowe obiekty. Do działania wymaga definicji dodatkowego obiektu typu `ViewAdapter` [3].

Do stworzenia obiektu `ViewAdapter` wymagany jest obiekt `ViewHolder`, który operuje na definicji układu widoku pojedynczego elementu z listy. W tak podstawowej liście jak w tym projekcie obiekt ten nie definiuje nic poza nazwą używanej definicji układu.

Nawigacja do tego fragmentu powołuje wywołanie metody `onCreateView()`, która na samym początku ładuje definicję układu widoku oraz przypisuje się do ViewMode'a (tego samego co `mapView`, więc przechodząc z niego mamy pewność, że ten ViewMode'już istnieje).

Z uwagi na brak konieczności ładowania mapy już na tym etapie ustawiani są obserwatorzy na zmienne wykorzystywane do nawigacji, jak i zmienne z listą punktów. Ten widok z uwagi na konieczność sortowania listy po odległości wykorzystuje jednak inną listę niż widok mapy. Lista potrzebna do tego widoku jest zdefiniowana w ViewMode'u i jest tworzona w formie transformacji podstawowej listy wykorzystując dodatkowo aktualną lokalizację użytkownika.

Do obliczenia odległości pomiędzy lokalizacją użytkownika a punktem wykorzystywany jest wzór 1, który zabezpiecza przed błędami zaokrągleń wynikających z precyzji arytmetyki zmiennoprzecinkowej [10].

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Phi_1 - \Phi_2}{2}\right) + \cos(\Phi_1)\cos(\Phi_2)\sin^2\left(\frac{\lambda_1 - \lambda_2}{2}\right)}\right), \quad (1)$$

gdzie:

- d : odległość, pomiędzy dwoma punktami [w kilometrach];
- φ_1, φ_2 : szerokość geograficzna punktu 1 i punktu 2 [w radianach];
- λ_1, λ_2 : wysokość geograficzna punktu 1 i punktu 2 [w radianach];
- r : uśredniony promień kuli ziemskiej [w kilometrach].

Na samym końcu tworzony jest obiekt typu `ViewAdapter` i zostaje on przypisany do parametrów `RecyclerView`.

4.6 Wyświetlenie widoku szczegółów

Przejście do tego fragmentu jest możliwe po kliknięciu w znacznik na mapie w widoku mapy bądź pozycje na liście w widoku listy. Przy przejściu następuje przekazanie jednego parametru (`Id` obiektu), którego szczegóły mają zostać wyświetcone.

Tworzenie widoku rozpoczyna od załadowania definicji układu widoku i uzyskanie dostępu do repozytorium. Następnie wykonywane jest zapytanie do repozytorium o obiekt podając jego `Id`. Zapytanie nie korzysta z bazy danych, więc wykonywane jest na głównym wątku. Po otrzymaniu obiektu tworzony jest `ViewModel` wykorzystując do tego wzorzec projektowy fabryki (stworzenie `ViewModela` z parametrami w konstruktorze wymaga zdefiniowania dodatkowej klasy, pełniącej rolę fabryki).

Następnie tworzeni są obserwatorzy zmiennych używanych do nawigacji, następuje żądanie inicjalizacji mapy oraz uzupełnione zostaje menu w pasku na górze ekranu.

Po inicjalizacji mapy kamera zostaje przeniesiona na marker miejsca.

4.7 Usuwanie obiektu

Usunięcie wybranego obiektu jest możliwe poprzez opcję w menu na pasku akcji. Schemat wykorzystany do usuwania jest podobny do schematu przejścia opisanego w rozdziale 4.4.

W momencie wybrania opcji następuje wyświetlenie nieskończonego paska postępu oraz zapytanie do repozytorium o usunięcie obiektu. Z uwagi na wykonywanie zapytania do serwera oraz do lokalnej bazy działanie to musi być wykonane na osobnym wątku.

Po obsłużeniu zapytania repozytorium zwraca `Boolean`, czy udało się usuwanie, czy nie. Bazując na nim ustawiana jest trzystanowa `LiveData`, której obserwator podejmuje stosowne działanie przy zmianie wartości.

Gdy usuwanie się uda następuje przejście do widoku mapy, w razie niepowodzenia pokazany zostaje komunikat i znika pasek postępu. Po obsłużeniu tej zmiany UI zmienna jest przywracana do stanu `UNSET`.

4.8 Dodawanie/edycja obiektu

Aby wykonać dodanie bądź edycję obiektu konieczne jest przejście do fragmentu `addViewFragment`. Dodanie jest możliwe z obu głównych widoków, natomiast edycja z widoku szczegółów. Wywołanie nawigacji wymaga podania `Id` zmienianego miejsca. Jeśli przejście następuje w celu stworzenia nowego obiektu, podawany jest parametr `-1`. Jest to specjalnie zarezerwowany `Id`, który nigdy nie może pojawić się jako poprawny `Id` w bazie danych.

Stworzenie widoku wygląda bardzo podobnie do widoku szczegółów. Jeśli podanym argumentem jest `-1`, zamiast poprawnego obiektu, do ViewModela przekazywany jest `null`. Następuje żądanieinicjalizacji map i dostępu do lokalizacji użytkownika. Następnie tworzony jest pusty obiekt w ViewModelu i uzupełniany aktualną lokalizacją użytkownika. Na mapie w tym miejscu pojawia się też marker.

Jeśli został podany `Id` poprawnego miejsca ViewModel, jest tworzony z tym obiektem i po inicjalizacji map kamera przechodzi do lokalizacji edytowanego miejsca, a pola do edycji parametrów miejsca uzupełniają się aktualnymi wartościami.

Kliknięcie na jakiekolwiek miejsce na mapie powoduje przeniesienie wskaźnika w nowe miejsca.

Zapisanie dodania/edykcji miejsca możliwe jest poprzez użycie przycisku na dole ekranu. Wykorzystywany jest podobny schemat jak przy usuwaniu i przejściu. Wysyłane jest wtedy odpowiednie zapytanie do repozytorium i następuje oczekiwanie na wynik. Przy poprawnej akcji program przechodzi do widoku mapy, przy negatywnym następuje informacja o błędzie.

4.9 Implementacja repozytorium

Model repozytorium w tym projekcie pełni funkcję warstwy abstrakcji pomiędzy zapytaniami do serwera i bazy oraz powoduje skupienie całego kodu odpowiedzialnego za te operacje w jednym miejscu. Repozytorium jest zaimplementowane w formie wzorca projektowego typu Singleton, zapewniającego, że w trakcie życia aplikacji będzie istniał tylko jeden obiekt repozytorium i bazy danych [11].

```
val powerMugs: LiveData<List<PowerMug>>

suspend fun refreshCache()

suspend fun updatePlace(powerMug: PowerMug): Boolean
suspend fun insertPlace(powerMug: PowerMug): Boolean
suspend fun deletePlace(powerMug: PowerMug): Boolean

fun search(query: String): LiveData<List<PowerMug>>

fun returnPlace(key: Long): PowerMug?
```

Listing 5: Publiczny interfejs repozytorium

Użycie repozytorium sprowadza się do wywołania jednej z publicznych metod jego interfejsu z dowolnego miejsca aplikacji. Metody wypisane są w listingu 5.

Użycie metod typu `suspend` konieczne jest z poziomu współprogramów (*ang. coroutines*) [12], aby zapobiec blokowaniu wątku głównego programu odpowiedzialnego za wyświetlanie elementów interfejsu użytkownika.

Wszystkie metody działające we współprogramach mają podobny schemat działania — najpierw konkretne działanie jest wykonywane na serwerze, po czym w przypadku powodzenia zmiana zapisywana jest w lokalnej bazie danych, z której korzysta program. Zwracany obiekt typu `Boolean` informuje o sukcesie bądź niepowodzeniu danej akcji.

4.10 Przechowywanie obiektów

Mimo, że program w każdym miejscu wykorzystuje bardzo podobną definicję klasy poszczególnego punktu na mapie, zdecydowano się rozdzielić ich definicje na 3 osobne — definicję dla logiki programu (domenową — PowerMug), do bazy danych (DbPowerMug) i do zapytań sieciowych (NetworkPowerMug oraz EphemeralNetworkPowerMug). Tego typu rozdzielenie nie było konieczne, jednak powoduje, że kod aplikacji staje się bardziej elastyczny i modyfikacja np. modelu bazy nie powoduje konieczności wprowadzenia modyfikacji w kodzie całej aplikacji.

```
fun PowerMug.asDbModel(): DbPowerMug
fun PowerMug.asNetworkModel(): NetworkPowerMug
fun PowerMug.asEphemeralNetworkModel(): EphemeralNetworkPowerMug

fun DbPowerMug.asDomainModel(): PowerMug
fun List<DbPowerMug>.asDomainModel(): List<PowerMug>

fun NetworkPowerMug.asDatabaseModel(): DbPowerMug
fun List<NetworkPowerMug>.asDatabaseModel(): Array<DbPowerMug>
```

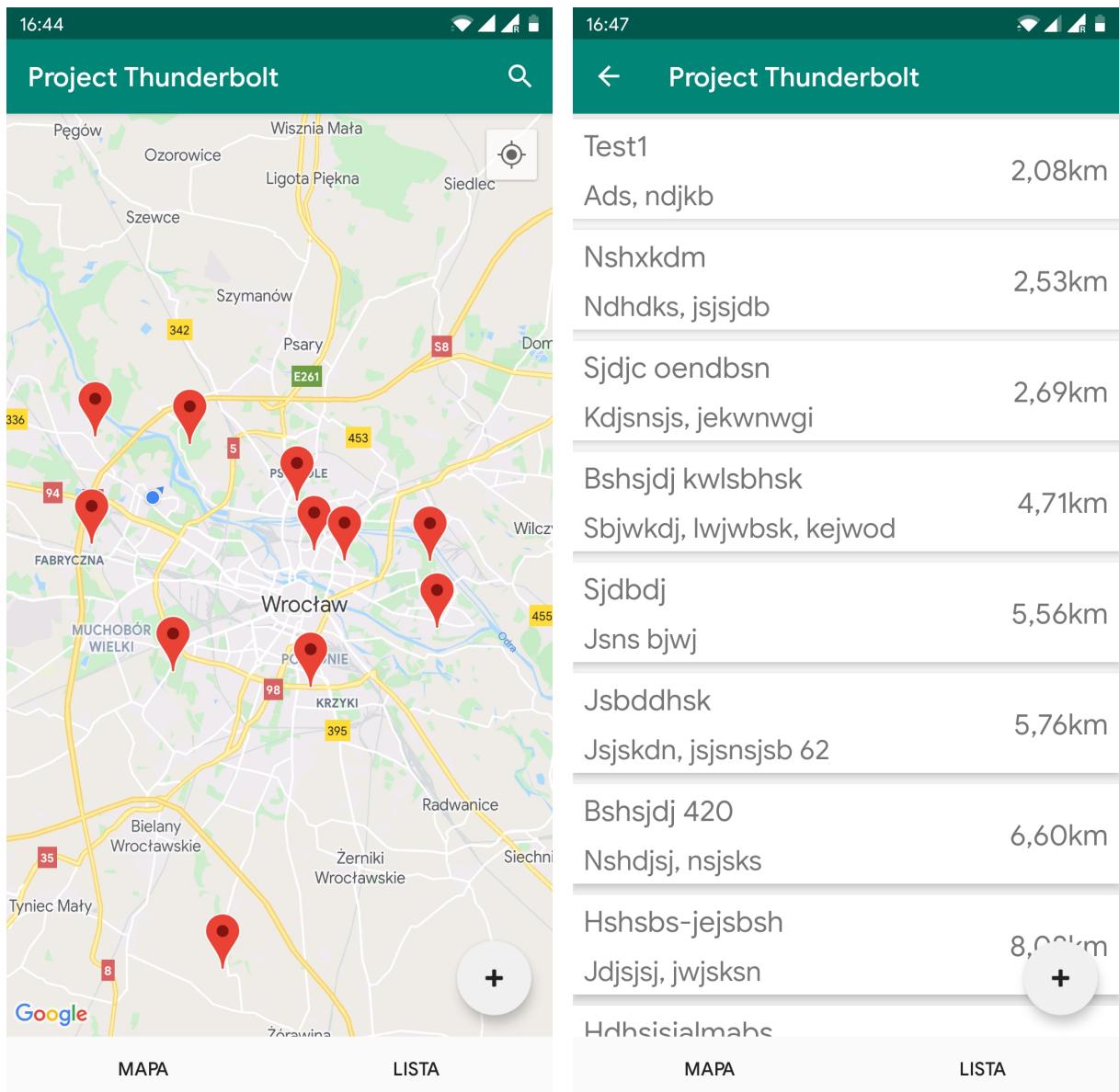
Listing 6: Metody używane do konwersji pomiędzy modelami obiektów

Aby uprościć zamianę obiektów, jednej reprezentacji w drugą przygotowano specjalne metody poszczególnych klas (listing 6).

5 Opis interfejsu użytkownika

Program posiada zaimplementowany interfejs użytkownika w dwóch językach — angielskim i polskim. W tej pracy wszystkie zdjęcia ukazują interfejs polski, aby zachować zgodność językową z językiem dokumentu.

5.1 Ekrany główne



Rysunek 3: Dwa główne widoki programu

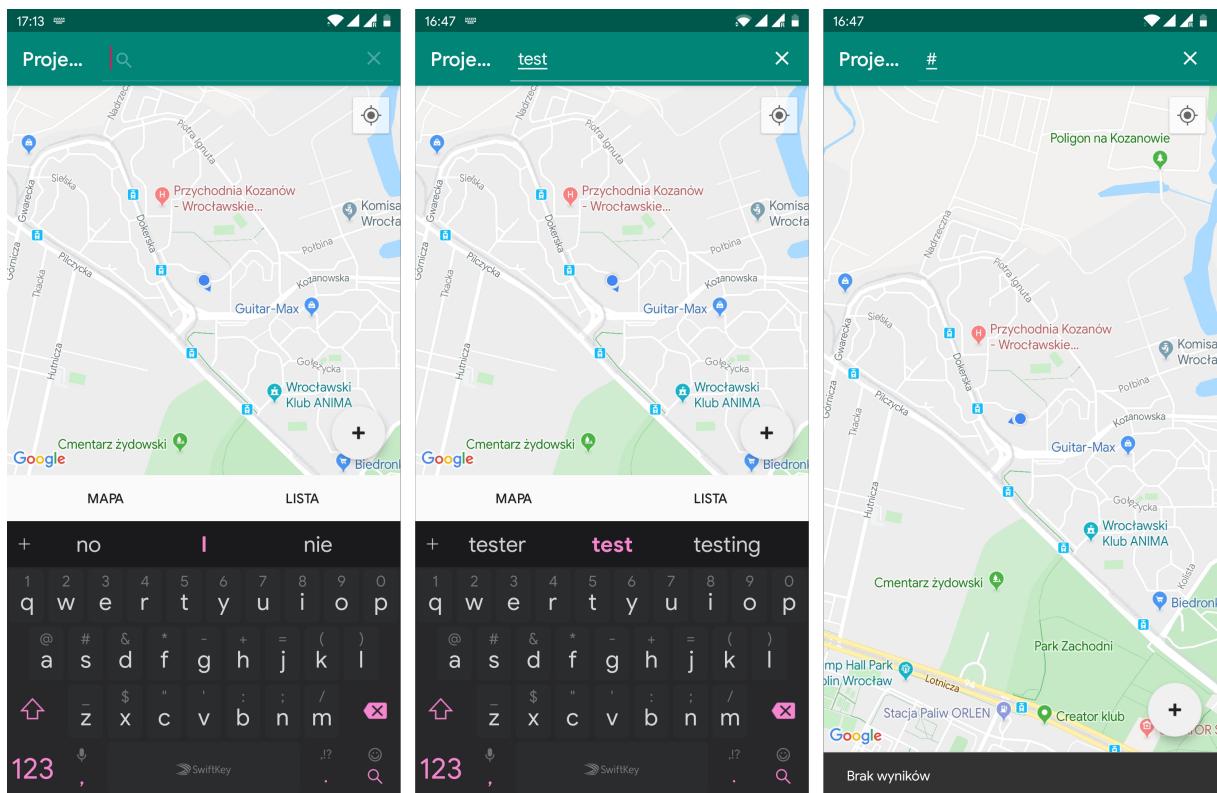
Aplikacja posiada dwa główne ekranы — widok mapy i widok listy. Ich wygląd został przedstawiony na Ryc. 3. Oba posiadają pasek na dole służący do nawigacji i przełączania pomiędzy nimi. W prawym dolnym rogu każdego z nich znajduje się "pływający" przycisk służący do dodawania nowego obiektu.

Na widoku mapy cały ekran zajęty jest przez fragment Google Maps, wyświetlający wszystkie punkty w formie markerów w odpowiednich miejscach geograficznych. Kliknięcie danego

markera przenosi do ekranu ze szczegółami o konkretnym miejscu. Na górnym pasku poza nazwą aplikacji wyświetlany jest przycisk uruchamiający tryb wyszukiwania.

W formie listy cały ekran jest zajęty przez przewijaną listę miejsc, posortowaną rosnącą odległością od aktualnej lokalizacji użytkownika. Kliknięcie w dowolną pozycję na liście powoduje przeniesienie do ekranu ze szczegółami o tym miejscu.

5.2 Wyszukiwanie punktów



Rysunek 4: Wygląd trybu wyszukiwania miejsc

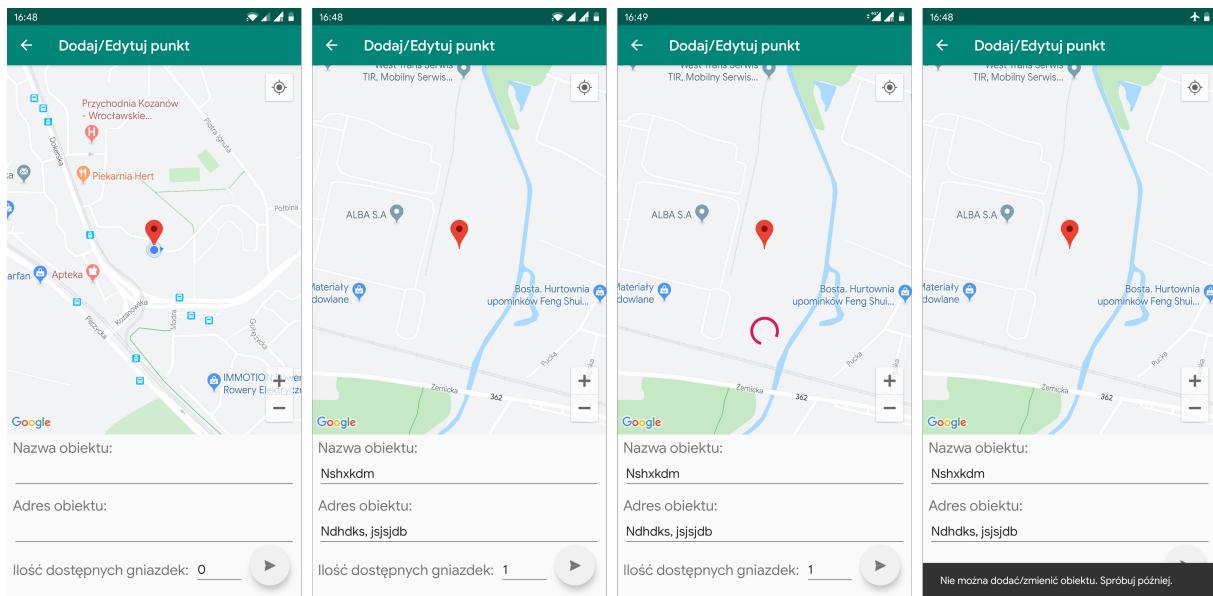
Wyszukiwanie markerów na mapie odbywa się w widoku mapy. Po kliknięciu ikony wyszukiwania pojawia się pasek, gdzie należy wpisać szukany ciąg znaków. Miejsca przeszukiwane są po nazwie i adresie. Wyszukiwanie zatwierdzane jest za pomocą przycisku na klawiaturze ekranowej.

W przypadku nieznalezienia żadnych wyników wyświetla się komunikat o błędzie, natomiast w przypadku powodzenia mapa ograniczy wyświetlane markery do tych spełniających kryteria wyszukiwania, a kamera przenosi się na pierwsze spełniające zapytanie miejsce.

Opuszczenie wyszukiwania i powrót do wszystkich miejsc następuje poprzez kliknięcie przycisku X na pasku szukania.

Ryc. 4 przedstawia po kolejno: pasek wyszukiwania, pasek uzupełniony szukaną frazą oraz błąd, gdy nie znaleziono żadnych wyników.

5.3 Tryb edycji i dodawania obiektów



Rysunek 5: Możliwy wygląd ekranu dodawania/edykcji punktu na mapie

Widok dodawania i edycji to jeden widok. W przypadku dodawania widok nie zawiera żadnych danych o miejscu, a marker na mapie wyświetla się na aktualnej lokalizacji użytkownika. W przypadku edycji widok wypełnia się aktualnymi informacjami o miejscu.

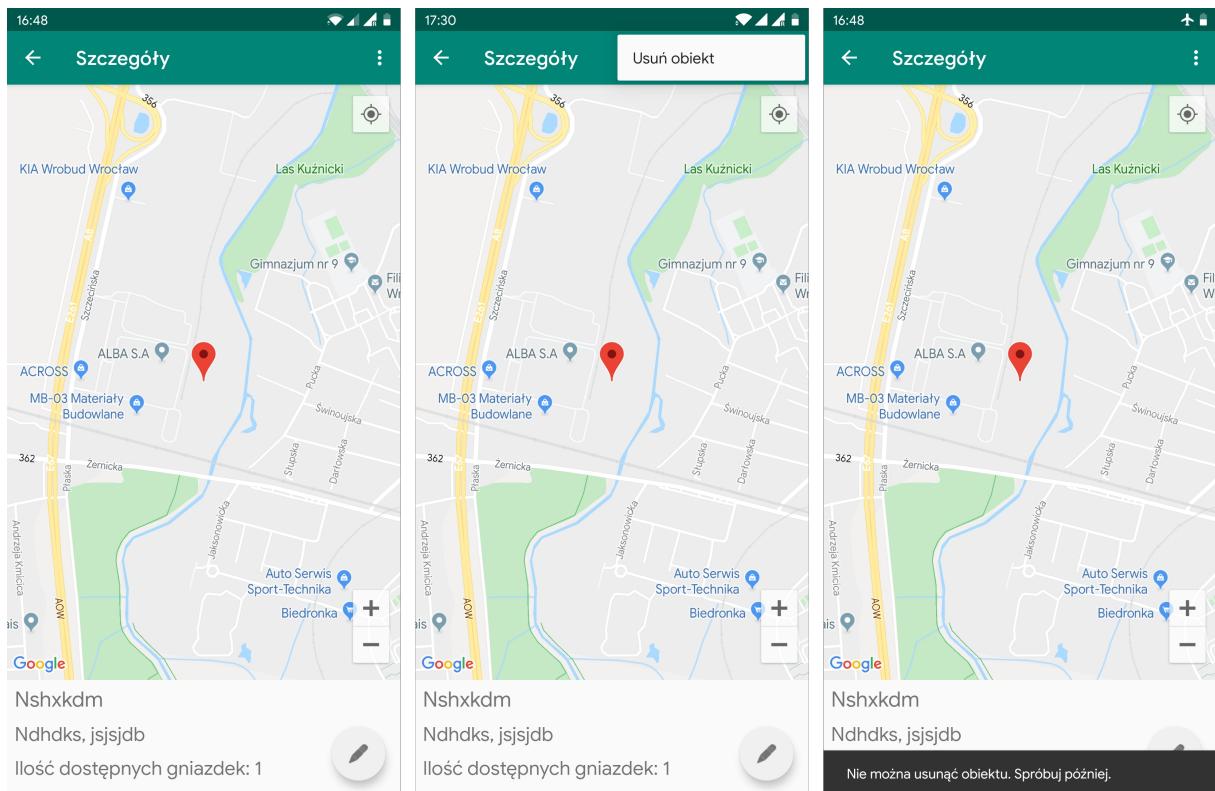
Zmiana pozycji markera odbywa się poprzez kliknięcie w nową lokalizację na mapie.

Zapisanie danych jest możliwe po kliknięciu przycisku w prawym dolnym rogu. Przy zapisywaniu danych pojawia się nieskończony pasek postępu, który znika po zakończeniu procesu dodawania. Przy niepowodzeniu następuje wyświetlenie stosownego komunikatu, natomiast po poprawnym dodaniu aplikacja wraca do głównego ekranu mapy.

Wyjście z ekranu bez zapisywania zmian jest możliwe, używając sprzętowego klawisza cofnij lub przycisku na górnjej belce widoku.

Ryc. 5 przedstawia po kolejci: widok po otwarciu trybu dodawania, widok trybu edycji, pasek postępu przy zapisywaniu zmian oraz komunikat o błędzie przy zapisywaniu.

5.4 Widok szczegółowy i usuwanie miejsc



Rysunek 6: Ekran szczegółów wraz z błędem usuwania

Fragment ze szczegółami niewiele różni się wyglądem od trybu edycji. Nie oferuje on jednak możliwości zmiany żadnych informacji.

Usunięcie miejsca jest możliwe z poziomu pozycji w menu na górnym pasku aplikacji. Po wywołaniu usuwania miejsca, tak jak w przypadku dodawania wyświetla się okrągły pasek postępu. Niepowodzenie jest sygnalizowane odpowiednim komunikatem, a sukces przejściem do głównego ekranu mapy.

Z tego widoku możliwe jest również przejście do widoku edycji, korzystając z przycisku w prawym dolnym rogu. Wyjście, czyli powrót do widoku mapy lub listy jest możliwe używając sprzętowego przycisku, bądź przycisku na górnym pasku.

Ryc. 6 przedstawia po kolej: widok szczegółów, menu widoku szczegółowego oraz błąd przy usuwaniu miejsca.

6 Podsumowanie

W ramach pracy dyplomowej zostały zaimplementowane podstawowe funkcjonalności opisane w koncepcie aplikacji, umożliwiające korzystanie z niej w założonych celach. Korzystając z programu da się oczywiście odczuć, że to nie jest jeszcze finalna wersja, jednak podstawowy zasób funkcji jest już dostępny. W dokumencie opisano metody realizacji każdej z tej funkcjonalności, na poziomie koncepcyjnym, jak i bardziej szczegółowym poziomie kodu.

Duża część czasu tworzenia tej pracy została poświęcona na zapoznanie się z językiem Kotlin oraz Android Framework. Podchodziąc do realizacji pracy autor nie miał żadnego doświadczenia w obu tych dziedzinach, więc potrzebna była nauka od zera. Wykonanie tej pracy dało jej autorowi solidny zasób podstawowej znajomości tych technologii oraz poszerzyło jego umiejętności w programowaniu ogółem.

Aplikacja została zaprojektowana tak by wykorzystać jak najwięcej gotowych bibliotek wchodzących w skład framework'u Androida, ponieważ naturalnie domyślna implementacja dla danej platformy jest z reguły najbardziej optymalna do ogólnych zastosowań. W projekcie wykorzystano również dwie zewnętrzne biblioteki — Timber i Moshi. Wykorzystanie biblioteki Timber było spowodowane chęcią zachowania bardziej przejrzystego kodu. Zakres, w którym została ona wykorzystana pozwolił na bardziej czytelny kod odpowiedzialny za korzystanie z logów systemowych. Biblioteka Moshi natomiast zaoszczędziła autorowi pracy sporo czasu wymaganego na ręczną implementację metod konwersji JSON na reprezentacje modelu danych i odwrotnie. Wykorzystanie zewnętrznej biblioteki gwarantuje, że dana funkcjonalność będzie poprawnie zaimplementowana i pozwala na skupienie się na implementacji funkcjonalności na wyższym poziomie. Dodatkowo zewnętrzna biblioteka została przez wiele osób już przetestowana i sprawdzona.

Dalszy rozwój projektu powinien skupić się na implementacji kolejnych funkcjonalności opisanych w rozdziale 2, głównie na implementacji obsługi kont użytkownika oraz poprawieniu dość ascetycznego interfejsu użytkownika. Warto by również przeanalizować, jakie dokładnie dane powinny być przechowywane na temat poszczególnych punktów na mapie oraz poszerzona powinna zostać integracja z usługą *Google Maps Services*. Rozwój tej aplikacji wymaga oczywiście pewnej dozy współpracy na poziomie planowania pomiędzy osobami odpowiedzialnymi za rozwój klienta oraz serwera internetowego.

Literatura

- [1] *Mobile Operating System Market Share Worldwide - October 2019*. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide>. (dostęp w dniu 24. listopada 2019).
- [2] Dawn Griffiths David Griffiths. *Head First Android Development, 2nd Edition*. O'Reilly Media, Inc., 2017.
- [3] *Android API Reference*. URL: <https://developer.android.com/reference>. (dostęp w dniu 24. listopada 2019).
- [4] *Representational state transfer*. URL: https://en.wikipedia.org/wiki/Representational_state_transfer. (dostęp w dniu 24. listopada 2019).
- [5] Tobias Günther. *Learn Version Control With Git: A Step-by-step Course for the Complete Beginner*. CreateSpace Independent Publishing Platform, 2017.
- [6] Antonio Leiva. *Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App*. CreateSpace Independent Publishing Platform, 2016.
- [7] *.test*. URL: <https://en.wikipedia.org/wiki/.test>. (dostęp w dniu 24. listopada 2019).
- [8] *Guide to app architecture*. URL: <https://developer.android.com/jetpack/docs/guide>. (dostęp w dniu 24. listopada 2019).
- [9] *Model–view–viewmodel*. URL: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>. (dostęp w dniu 24. listopada 2019).
- [10] *Haversine formula*. URL: https://en.wikipedia.org/wiki/Haversine_formula. (dostęp w dniu 24. listopada 2019).
- [11] *Singleton pattern*. URL: https://en.wikipedia.org/wiki/Singleton_pattern. (dostęp w dniu 24. listopada 2019).
- [12] *Kotlin Language Documentation*. URL: <https://kotlinlang.org/docs/reference/>. (dostęp w dniu 24. listopada 2019).

Spis rysunków

1	Schemat architektury	11
2	Diagram nawigacji pomiędzy fragmentami	13
3	Dwa główne widoki programu	20
4	Wygląd trybu wyszukiwania miejsc	21
5	Możliwy wygląd ekranu dodawania/edykcji punktu na mapie	22
6	Ekran szczegółów wraz z błędem usuwania	23

Indeks listingów

1	Konfiguracja narzędzia do budowy Gradle	9
2	Lista plików z kodem źródłowym (<i>Pliki są wyświetlane w ręcznie ustalonej kolejności</i>)	12
3	Kod znajdujący się w klasie typu ViewModel potrzebny do przejścia	15
4	Kod obserwatora potrzebny do przejścia	16
5	Publiczny interfejs repozytorium	18
6	Metody używane do konwersji pomiędzy modelami obiektów	19