Pokedex

Jan Fidor

Tomasz Owienko

Anna Schäfer

Jakub Woźniak

Cel projektu

Celem projektu była implementacja aplikacji “Pokedex” na platformę Android w języku Kotlin korzystając z publicznego PokeApi.

Postanowiliśmy zaprojektować aplikację, która pozwoli użytkownikowi zapoznać się z informacjami na temat danego Pokemona. Stwierdziliśmy, że napiszemy aplikację na androida w języku Kotlin korzystającą z PokeApi, aby dać możliwość sprawdzenia co robi Pokemon, bez konieczności używania przeglądarki.

Założenia

Funkcjonalność:

1. Możliwość wyświetlania pobieżnych informacji o Pokemonach (nazwa, ikona, typy)
2. Widok szczegółowych informacji na temat danego Pokemona
3. Możliwość zapisywania wybranych Pokemonów w kolekcji
4. Implementacja wyszukiwarki oraz filtrów w widokach opartych na listach

Implementacja:

1. Aplikacja *offline first* - minimalizacja liczby koniecznych odwołań do zewnętrznego API

2. Utrzymanie rekomendowanej przez Google Developers czystej architektury
3. Praca na nowoczesnej platformie (*API level 31*)

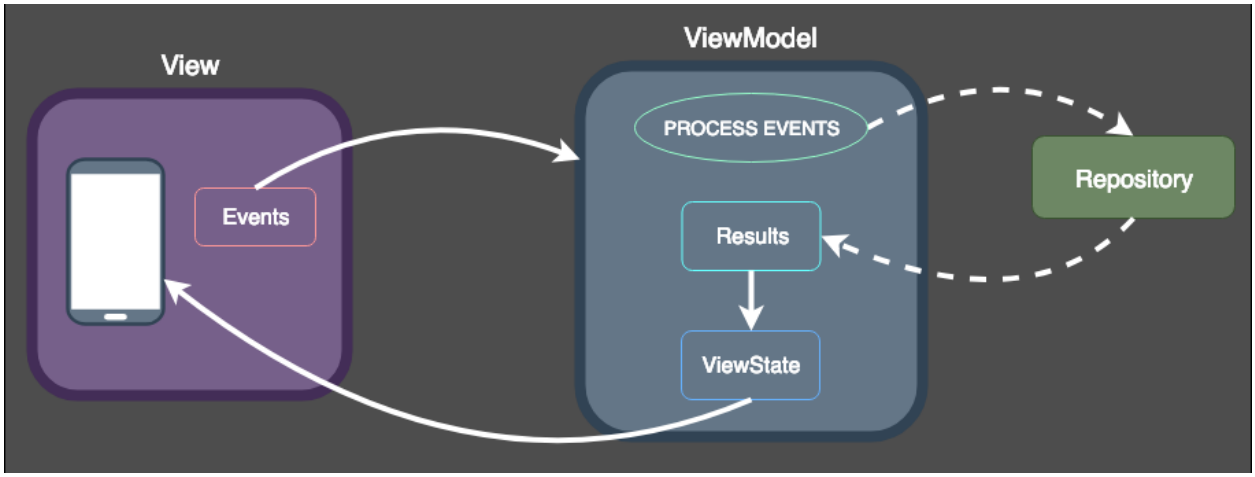
Wykorzystane technologie

- Kotlin, w szczególności:
    - Kotlin Coroutines - kod asynchroniczny
    - Kotlinx Serialization - serializacja i deserializacja danych z API
- Android SDK (API level 31)
- Jetpack Compose - interfejs użytkownika
- Jetpack Navigation Component - UI, obsługa nawigacji między ekranami
- Retrofit - klient HTTP zapewniający bezpieczeństwo typów
- Room - klient SQLite
- Hilt - biblioteka realizująca wstrzykiwanie zależności
- Gradle - zautomatyzowana budowa aplikacji
- Timber - logging
- JUnit - testy jednostkowe

Struktura aplikacji

W projekcie zastosowano rekomendowaną przez Google Developers [czystą architekturę](). Zrealizowano następujące założenia:

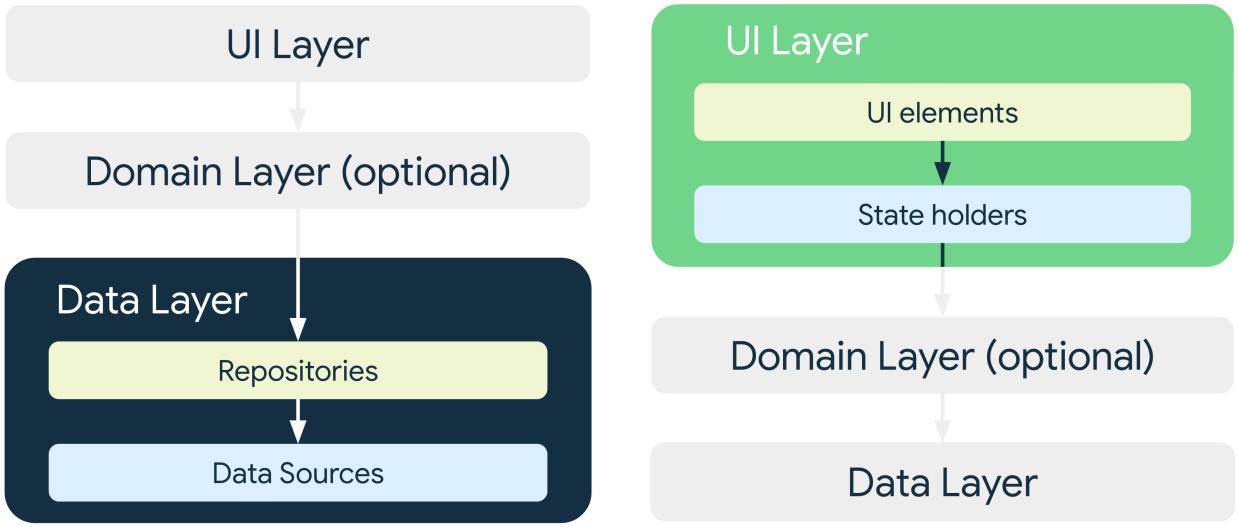
Jednokierunkowy przepływ danych



Zgodnie z założeniami czystej architektury, konieczne jest odgórne ustalenie strategii propagacji zmian w stanie aplikacji. Zgodnie z techniką jednokierunkowego przepływu danych, dozwolone jest, aby zmiany w stanie aplikacji powstałe w interfejsie użytkownika były propagowane “w głąb” aplikacji, a zmiany w prezentowanych użytkownikowi danych - w przeciwnym kierunku. W aplikacjach przeznaczonych na platformę Android ta technika realizowana jest zgodnie ze wzorcem *Model-View-ViewModel*. Zakłada on separację odpowiedzialności za wyświetlanie danych, ich przetwarzanie (logikę biznesową), oraz przechowywanie - co niejako jest już zapewniane przez warstwową strukturę aplikacji. Za wyświetlanie danych odpowiadają widoki tworzone przez framework *Compose*, za persystencję danych opowiada repozytorium, a komunikacja między nimi odbywa się za pomocą klas *ViewModel*. Definiują one strategie reagowania na zmiany stanu interfejsu użytkownika w kontekście funkcji repozytorium oraz propagują zmiany stanu danych (*MutableState*) do UI. Istotne jest, że zastosowanie obiektów *ViewModel* wymusza obsługę zdarzeń UI bez jednoczesnego modyfikowania interfejsu

użytkownika (klasa *ViewModel* nie wie o istnieniu interfejsu użytkownika). Ponadto, podczas propagacji zmian w danych aplikacji niemożliwe jest ich jednoczesne dalsze mutowanie - zamiast wywołań zwrotnych korzysta się tu z obiektów *MutableState,* które obserwowane są przez widoki z klasy UI (wiązaniem zajmuje się framework *Compose*).

Warstwowa struktura aplikacji



Zgodnie z założeniami czystej architektury zastosowano warstwową strukturę aplikacji z podziałem na następujące warstwy:

1. Data

   Dostęp do bazy danych i zdalnego API

2. Domain

   Enkapsulacja logiki biznesowej aplikacji, pośrednictwo w komunikacji między warstwami *data* oraz *UI*

3. UI

   Interfejs użytkownika.

Data

Zadaniem warstwy *data* było zapewnienie klas obsługujących komunikację z bazą danych aplikacji oraz zdalnym API.

Persystencja danych została zapewniona przez klasy *PokemonDatabase* oraz *PokemonDao* dostarczane przez bibliotekę *Room* korzystającą z mapowania obiektowo-relacyjnego. Obiekty *PokemonDAO* udostępniają metody dostępu do lokalnej bazy danych, z których dalej korzysta klasa repozytorium w warstwie *domain*. Schemat bazy danych oraz dozwolone operacje, wraz z więzami integralności, zdefiniowane są bezpośrednio w kodzie aplikacji, co pozwala na zapewnienie pełnego bezpieczeństwa typów. Warstwa *data* posiada własne obiekty transferu danych o strukturze odpowiadającej relacjom w bazie - mogą być one zmapowane na klasy z warstwy *domain* przy użyciu zapewnionych funkcji.

Za komunikację z API odpowiada biblioteka *Retrofit*, w której odpowiedzialności leży przetwarzanie (w szczególności parsowanie) odpowiedzi ze zdalnego API oraz udostępnienie klas do komunikacji z nim. Podobnie, jak w przypadku biblioteki *Room*, *Retrofit* definiuje własne obiekty transferu danych, które mogą być zmapowane na klasy z warstwy *domain*.

Dowiązywanie zależności zapewniane jest przez bibliotekę *Hilt*, a kod odpowiedzialny za wiązanie klas wydzielony został do osobnego pliku w ramach separacji odpowiedzialności.

Domain

Warstwa *domain* pośredniczy w komunikacji między pozostałymi warstwami poprzez:

1. Dostarczenie modeli wykorzystywanych w pozostałym kodzie aplikacji (otrzymywanych z obiektów transferu danych w *data* oraz przekazywanych między *UI* a repozytorium)
2. Utworzenie repozytorium, które udostępnia interfejs dla *UI* do pośredniej komunikacji z warstwą persystencji oraz kapsułkuje większość logiki biznesowej aplikacji.

UI

Warstwa *UI* przechowuje klasy składające się na interfejs graficzny aplikacji.

Interfejsy graficzne zrealizowano z wykorzystaniem frameworku *Jetpack Compose*. Framework udostępnia adnotację `@Composable`, na szybkie budowania reużywalnych komponentów interfejsów, które mogą być ze sobą składane w dowolny sposób. Ponadto, odpowiedzialnością frameworku jest także zapewnienie spójności wyświetlanych danych z przechowywanymi w aplikacji - w skład *Compose* wchodzi m.in. wtyczka kompilatora języka Kotlin, która pozwala na generowanie w czasie kompilacji dodatkowych klas monitorujących stan wyświetlanych danych i wymuszającymi przerysowanie odpowiednich części interfejsu, jeśli dane ulegną zmianie - eliminuje to konieczność korzystania z klasycznych metod data binding opartych na wywołaniach zwrotnych.

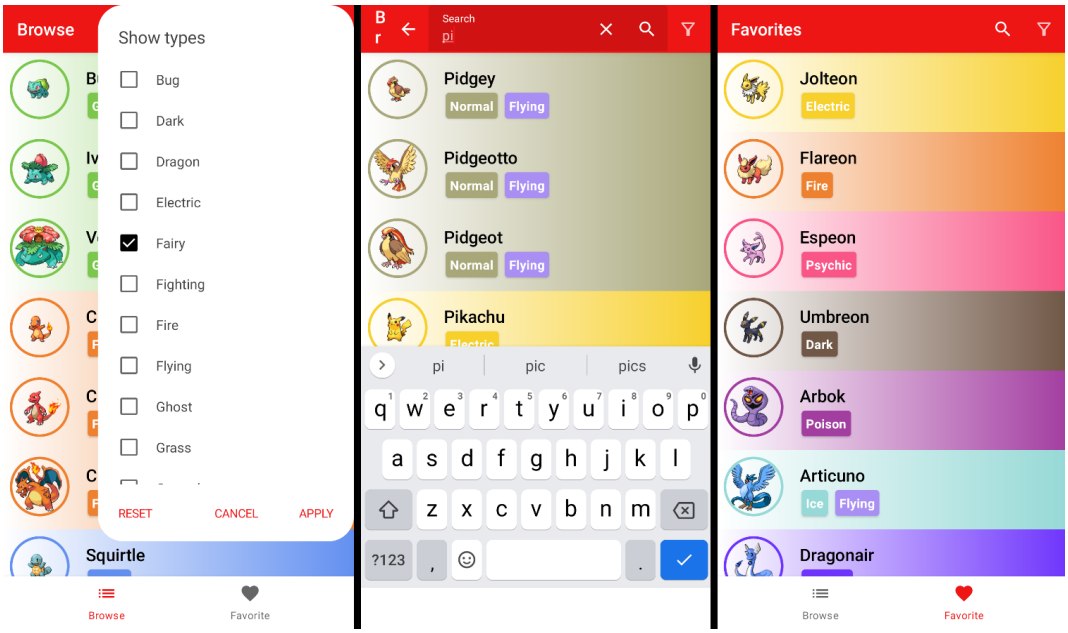
Ekrany aplikacji oraz nawigację między nimi zrealizowano z wykorzystaniem *Jetpack Navigation Component*. Framework pozwala na tworzenie ekranów (*fragments*) świadomych cyklu życia zarówno swojego, jak i aktywności, do której należą, oraz nawigacji między nimi z możliwością przekazywania argumentów. Bezpieczeństwo typów argumentów zapewniane jest przez przez wtyczkę do systemu Gradle wchodzącą w skład *Navigation Component* - podczas budowy projektu na podstawie zdefiniowanych przejść w *grafie nawigacji* generowane są klasy przechowujące argumenty na czas przejścia między *fragmentami*.

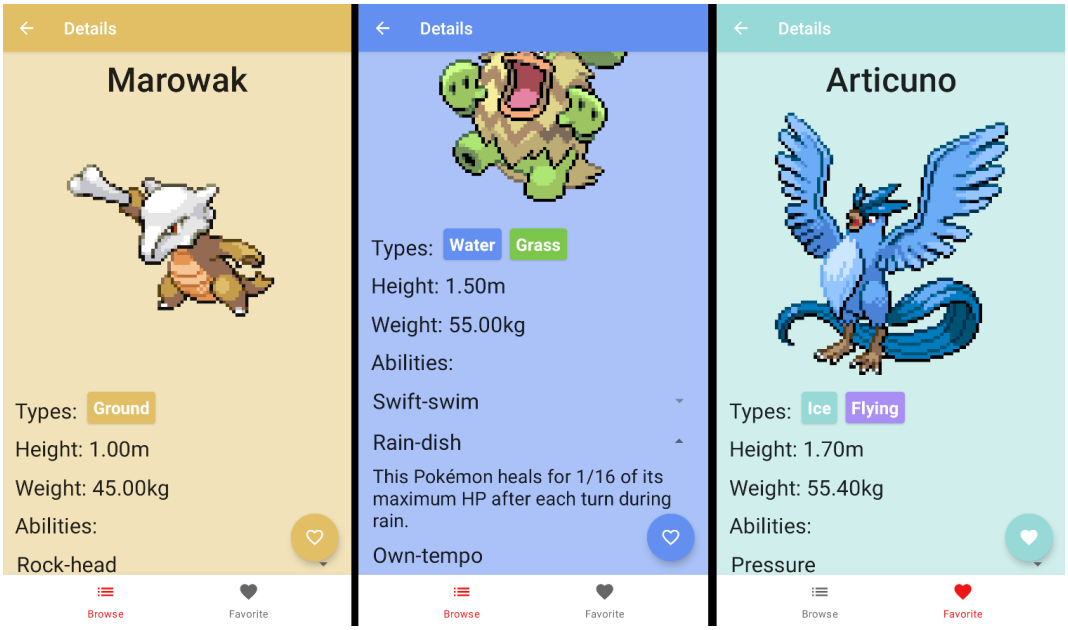
Użytkowanie

W zakładce “Browse” użytkownik może przeglądać Pokemony przewijając listę. Ma również możliwość wyszukania Pokemona po jego nazwie – po kliknięciu w ikonkę lupy na górnym pasku może zacząć wpisywać jego nazwę. Obok ikonki lupy znajduje się ikonka filtra, która pozwoli ograniczyć listę wyświetlanych Pokemonów tylko do typów, które interesują użytkownika. Drugą zakładką jest “Favourites”, gdzie znajdą się wybrane przez nas Pokemony.
Jeśli zaś chodzi o wybieranie Pokemonów do tej zakładki, musimy wejść w ekran szczegółów o Pokemonie z poziomu zakładki “Browse”. Wtedy w ekranie, gdzie `wyświetlą się szczegółowe informacje znajdziemy w prawym dolnym rogu ikonkę serduszka. Po kliknięciu wnętrze

serduszka pokoloruje się na biało, a Pokemon zostanie dodany do listy ulubionych.

W ekranie szczegółów można znaleźć takie informację jak nazwa, typ, wzrost, waga oraz listę umiejętności danego Pokemona. By poznać działanie umiejętności, wystarczy kliknąć w jej nazwę, po czym rozwinie się opis.





Budowa i instalacja

Do budowy aplikacji zaleca się skorzystanie z narzędzia Android Studio. Możliwa jest także budowa z poziomu linii polecenia:

```
chmod +x gradlew
./gradlew assembleDebug
```

Aplikacja może zostać zainstalowana na emulatorze z użyciem poleceń:

```
emulator -avd avd_name
adb install path/to/your_app.apk
```

Lub na fizycznym urządzeniu:

```
adb -d install path/to/your_app.apk
```

Możliwe kierunki rozwoju aplikacji

- Zwiększenie ilości informacji o Pokemonach wyświetlanych w widoku detali
- Wprowadzenie możliwości porównywania Pokemonów
- Adaptacja aplikacji do systemu Material 3.