Zadania

Sebastian Kulig, Karol Wnęk

April 26, 2020

1 Wstęp

Celem tych ćwiczeń jest stworzenie aplikacji opartej na fragmentach, która pozwoli nam pozyskać informację o warunkach pogodowych we wskazanym przez użytkownika mieście. Spora cześć aplikacji jest już gotowa:
https://github.com/SebastianKulig/Zadania.git
Naszym zadaniem będzie ją dokończyć zgodnie z poniższymi poleceniami. W kodzie znajdują się komentarze (TODO) w miejscach wymagających uzupełnienia. Efekt końcowy zaprezentowany jest na poniższych zdjęciach.

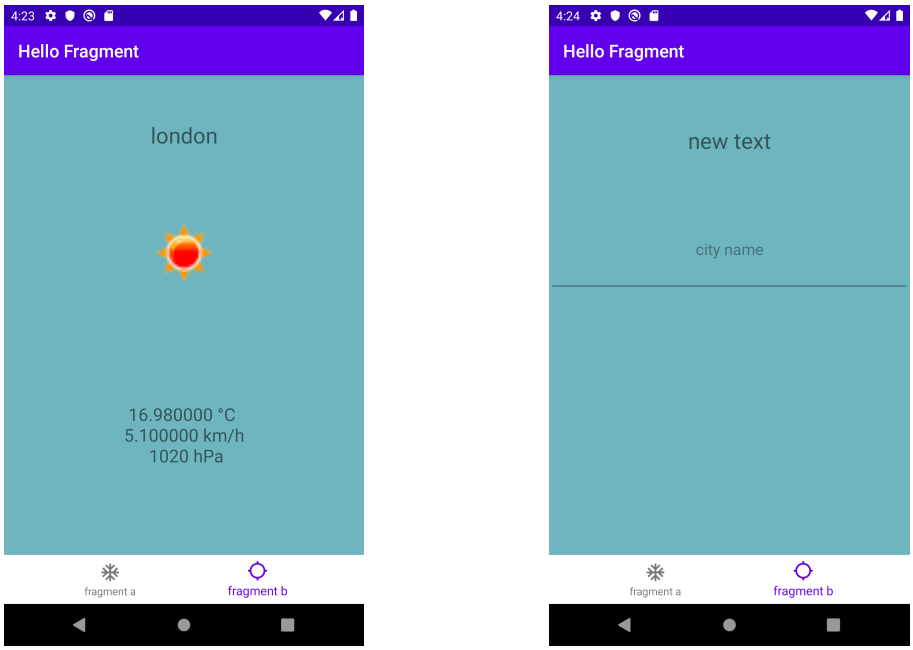

Figure 1: Spodziewany efekt

2 Dwa fragmenty

2.1 Uzupełnij klasę HelloFragmentA oraz HelloFragmentB.

Wyglądy poszczególnych fragmentów są zdefiniowane w plikach: *hello_fragment_a.xml* oraz *hello_fragment_b.xml*. Pamiętaj o "znalezieniu" elementów zdefiniowanych w tych plikach. Do tego celu wykorzystaj metodę *findViewById(...)*.

2.2 Gdzie się podział fragment A?

Po pierwszym uruchomieniu aplikacji widać, że fragment A zostaje wyświetlony dopiero po odwiedzeniu fragmentu B. Dzieje się tak dlatego, że w *MainActivity* w metodzie *onCreate(...)* nie dowiązaliśmy "startowego" fragmentu. Twoim zadaniem jest zmienić ten stan rzeczy. Wykorzystaj do tego celu następujące metody:

- *beginTransaction()*

- *commit()*
- *getSupportFragmentManager()*
- *replace(...)*

Po wykonaniu zadania aplikacja powinna się uruchamiać i pozwalać na swobodne przełącznie pomiędzy fragmentami.

3 Wzajemna komunikacja

Celem tego zadania jest zapewnienie komunikacji pomiędzy fragmentami. Wynik powinien być następujący: przy pierwszej wizycie w fragmencie B wyświetlany jest tekst *hello fragment B!*, przy kolejnej dowolny inny. W tym celu należy:

- zadaklarować metodę w interfejsie w HelloFragmentA, który posłuży do wysłania nowego tekstu
- upewnić się, że implementuje HelloFragmentAListener oraz dołączyć się do jego instancji - *onAttach(...)*, pamiętaj o zwolnienu w *onDetach()*
- uzupełnić *updateData(String text)* zmieniającą tekst w *HelloFragmentB*, a następnie wywołać ją w implementacji interfejsu w *MainActivity*.

Dane prześlij z metody *onPause()* (wywołaj odpowiednią metodę na instancji interfejsu).

4 Przygotowanie do współpracy z API

- fragment B:
  - zmień wygląd fragmentu na ten zdefiniowany w pliku: *hello_fragment_b_final.xml*
  - "znajdź" nowe elementy w układzie fragmentu
  - uzupełnij *onAttach(...)* oraz *onDetach()*
  - odkomentuj zaznaczone fragmenty kody
- fragment A:
  - zmień wygląd fragmentu na ten zdefiniowany w pliku: *hello_fragment_a_final.xml*
  - "znajdź" nowe elementy w układzie fragmentu
  - w *onCreateView(...)* ustaw tekst wyświetlany przez TextView na ten przechowywany w zmiennej *trxtViewString* - skorzystaj z metody *setText()*
  - uzupełnij metodę updateCity(String city), tak aby przypisywałae otrzymaną wartość do zmiennej przechowującej nazwę miasta - *cityNameString*.
  - odkomentuj zaznaczone fragmenty kody
- MainActivity:
  - odkomentuj zaznaczone fragmenty kody

5 Współpraca z API

Teraz zajmiemy się dostarczeniem informacji pogodowych do naszej aplikacji. Wykorzystamy w tym celu openweathermap.org.

5.1 Dodanie odpowiednich bibliotek i pozwoleń

- Pierwszym zadaniem będzie dodanie odpowiednich pozwoleń. W tym celu należy przejść do AndroidManifest.xml i dodać następujące pozwolenia:
  - <uses-permission android:name="android.permission.INTERNET"/>
  - `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>`
  - `<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />`

- Następnie przygotowywujemy potrzebne biblioteki. Przechodzimy do pliku build.gradle i w zakładce dependencies dodajemy:
  - implementation 'com.squareup.retrofit2:retrofit:2.7.2' //retrofit2
  - implementation 'com.squareup.retrofit2:converter-gson:2.7.2' //Gson
  - implementation 'com.squareup.picasso:picasso:2.71828' //Picasso

  W zakładce android dodajemy:

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8 //Language level of the java source code
    targetCompatibility JavaVersion.VERSION_1_8 //Version of the generated Java bytecode
}
```

5.2 Model odpowiedzi

- Teraz zbudujemy klasy modelu, które będą zwracane z serwera jako JSON. W celu przeanalizowania czego będziemy potrzebowali najlepiej sprawdzić dokumentację danego API, gdzie znajdziemy wszystkie niezbędne informacje. My jednak po prostu wpiszemy nasze zapytanie w przeglądarkę i sprawdzimy jak dokładnie wygląda opowiedź którą będziemy dostawać. Chcemy uzykać aktualną informację pogodową w wybranym mieście, dlatego zapytanie będzie wyglądało tak:

  api.openweathermap.org/data/2.5/weather?q={city name}&appid={your api key}

  Klucz możemy wygenerować sobie sami zakładając konto na OpenWeatherMap (oczywiście darmowe) lub skorzystać z przygotowanego klucza : "eccf917310f1c6acbf2acb9e85bf1a0d"

  Przykładowe gotowe zapytanie:

  api.openweathermap.org/data/2.5/weather?q=krakow&appid=eccf917310f1c6acbf2acb9e85bf1a0d

  Wpisz powyższe zapytanie do przeglądarki i sprawdź jak wygląda odpowiedź zwracana przez serwer. Istnieje wiele narzędzi pozwalających nam na przeanalizowanie takiego pliku json i przygotowanie odpowiednich klas (istnieją gotowe rozszerzenia do Android Studio). My nie będziemy musieli tworzyć całego modelu, więc wystarczy zapoznać się chociażby z kodem generowanym przez stronę www.jsonschema2pojo.org, żeby zobaczyć jak taki model jest tworzony (wklejamy naszą odpowiedź, wybieramy odpowiedzie opcje i naciskamy Preview).

  W pakiecie model znajdują się już przygotowane klasy, ale należy dopisać do nich settery, gettery. Czy możemy usunąć poszczególne zmienne lub nawet całe klasy z modelu, jeśli z nich nie korzystamy?

  Więcej szczegółów odnośnie wykorzystywanego API znajdziesz pod adresem:
  https://openweathermap.org/current

5.3 Interfejs

- Następnym krokiem jest przygotowanie interfejsu w którym zdefiniowane będą operacje HTTP, jego implementacją zajmie się już retrofit.

  Przejdź do interfejsu WeatherApi, a następnie zdefiniuj metodę HTTP która posłużyć do pobierania informacji z serwera. Wykorzystaj przykład z prezentacji.

  Podpowiedzi:
  - użyj adnotacji @GET
  - zdefiniuj endpoint URL
  - w delkaracji musimy uwzględnić że dane zwracane przez tę metodę muszą być zgodne z przygotowanym modelem (główna klasa naszego modelu otoczona interfejsem Retrofit tj. Call <WeatherResult>)

– w parametrach metody przekazujemy nazwę miasta, jednostkę i klucz (wykorzystując adnotację @Query z nazwą parametru, "q"-nazwa miasta, "units" - jednostka, "appid" - klucz)

Więcej szczegółów na temat biblioteki Retrofit znajdziesz pod adresem:
square.github.io/retrofit/

5.4 Builder

- Następnie tworzymy klasę NetworkClient która będzie nam zwracać klienta Retrofit. Sama biblioteka Retrofit dostarcza nam builder do tworzenia takich obiektów. Potrzebuję on bazowy adres url oraz "converter factory" który zajmie się parsowaniem odbieranych i wysyłanych danych (zamianą obiektów java na json i jsonów na obiekty java) my użyjemy GsonConverterFactory. Cała klasa jest już gotowa, więc twoim zadaniem jest tylko się jej przyjrzeć i przeanalizować, bo niedługo będziesz musiał z niej skorzystać.

5.5 Wykonanie zapytania

- Gdy wszystko mamy już przygotowane, to możemy przejść do HelloFragmentA. Mamy tam przygotowaną metodę fetchWeatherDetails() do której należy jednak dodać kilka elementów we wskazanych miejscach:
  – za pomocą klasy NetworkClient tworzymy klienta retrofit, nazwij go retrofit (wskazówka: metoda statyczna)
  – wywołujemy metodę zdefiniowaną w interfejsie przekazując do niej nazwę miasta uzyskaną z fragmentu B, jednostkę oraz klucz. Metodę tę wywołujemy na utworzonym linijkę wyżej obiekcie weatherApi. Zwracaną wartość przypisać do obiektu Call call.
  – zapytanie wykonuje się asynchronicznie (call.enqueue) dlatego Retrofit wymaga implementacji klasy Callback(z metodami onResponse i onFailure) w celu obsługi odpowiedzi. W onResponse sprawdzamy czy nasza odpowiedź nie jest null i dokonujemy mapowania klasy modelu na odpowiedź. Twoim zadaniem tutaj będzie wyświetlenie uzyskanych informacji wykorzystując przygotowany w string.xml string z miejscem na 3 argumenty, którymi będą temperatura, prędkość wiatru i ciśnienie.
  – do ładowania ikonek użyjemy biblioteki Pisacco. W naszej odpowiedzi z serwera mamy String icon, dodajemy go do url https://openweathermap.org/img/w/ i całość kończymy rozszerzeniem ".png". Następnie przekazujemy ten adres oraz miejsce w którym należy umieścić ikonkę (imageView) do Picasso:
    Picasso.get.load(url).into(imageView);

Więcej szczegółów na temat biblioteki Picasso znajdziesz pod adresem:
square.github.io/picasso/