Teleinformatyczne systemy mobilne

Projekt

Aplikacja Mobilna “Check Weather”

Autor:
Michał Stancelewski
Politechnika Poznańska

GITHUB: https://github.com/zerhaz/POGODA

1. Wprowadzenie

1.1. Temat, cel i zakres pracy

Celem niniejszego projektu było stworzenie aplikacji mobilnej na urządzenia typu “smartfon” obsługującej wyświetlanie reklam w jednym z dostępnych systemów RTB oraz oferującej użytkownikowi dowolną praktyczną funkcjonalność.

Niniejsza aplikacja nazwana została “Check Weather” i funkcjonalnością jaką oferuje jest wyświetlanie podstawowych danych pogodowych dla 16 największych polskich miast (miasta wojewódzkie).

Wyświetlane są dane pogodowe dla chwili obecnej oraz dane prognozowane dla dwóch najbliższych dni.

1.2. Uzasadnienie wyboru tematu

Temat ten został wybrany, aby wykazać umiejętność posługiwania się interfejsem API w aplikacjach mobilnych. Tego typu podejście do tworzenia aplikacji mobilnych jest bardzo popularne, gdyż ze względu na ograniczone możliwości sprzętowe telefonu konieczna jest wymiana danych ze zdalnymi serwerami, które wykonują większość obliczeń lub przechowują duże ilości danych.

2. Wykorzystane technologie i narzędzia

2.1. Java + Android SDK

Aplikacja została napisana z przeznaczeniem na urządzenia mobilne z systemem Android. W tym celu wykorzystany został język JAVA z ANDROID SDK w wersji 28.

2.2. Retrofit + GSON

Do komunikacji poprzez API wykorzystane zostały biblioteki Retrofit oraz GSON. Retrofit jest popularnym klientem HTTP dla Androida [https://square.github.io/retrofit/]. GSON to natomiast biblioteka konwertująca obiekty JSON w obiekty języka JAVA (i odwrotnie) [https://github.com/google/gson].

2.3. Google AdMob

Do wyświetlania reklam w aplikacji wykorzystana została platforma Google AdMob. Aplikacja korzysta z upublicznionej konfiguracji testowej dla reklam AdMob.

2.4. OpenWeatherMap.org

Dane pogodowe pobierane są z serwisu OpenWeatherMap.org [https://openweathermap.org/]. Wykorzystywany jest w tym celu darmowy klucz API, który uzyskuje się po założeniu konta w serwisie.

2.5. Android Studio

Aplikacja mobilna tworzona jest w IDE Android Studio.

3. Implementacja

3.1. Wykorzystanie API

W celu pobrania danych z API openweathermap.com aplikacja wykorzystuje interfejs *JsonPlaceHolderApi*, który za pomocą metod GET pobiera obiekty JSON z dwóch endpointów:

- “weather” - zawiera dane pogodowe obecne
- “forecast” - zawiera dane pogodowe prognozowane

W metodzie GET przekazywane są trzy parametry:

- “id” - id miasta, dla którego pobieramy dane (spis id jest dostępny na stronie internetowej)
- “lang” - język danych (dana opcjonalna)
- “appid” - klucz API do autoryzacji użycia API

```
package pl.zerhaz.checkweather;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface JsonPlaceHolderApi {

    @GET("weather")
    Call<Post> getPosts(
            @Query("id") String cityID,
            @Query("lang") String language,
            @Query("appid") String appAPI
    );

    @GET("forecast")
    Call<Forecast> getForecast(
            @Query("id") String cityID,
            @Query("lang") String language,
            @Query("appid") String appAPI
    );
}

```

Rys 1. Interfejs *JsonPlaceHolderApi*

3.2. Przetwarzanie danych pogodowych

Zapytanie API wysyłane jest w metodzie *callRequest(String id)*. Metoda ta przyjmuje id miasta, dla którego pobrane mają być dane.

Wykonuje ona dwa zapytania. Jedno dla pobrania obiektu JSON z endpointu “weather”. Obiekt JSON jest konwertowany do obiektu klasy

Post, która korzysta także z obiektów klas *Weather* i *Wind*. Pobrane dane wstawiane są w odpowiadające im obiekty Text View.

```
call.enqueue(new Callback<Post>() {
    @Override
    public void onResponse(Call<Post> call, Response<Post> response) {
        if (!response.isSuccessful()) {
            textViewError.setText("Code: " + response.code());
            textViewCity.setText("");
            textViewTemperature.setText("-");
            textViewWind.setText("-");
            textViewPressure.setText("-");
            return;
        }
        textViewError.setText("");
        textViewCity.setText(response.body().getCityName());
        textViewTemperature.setText(response.body().getWeather().getTemperature() + " °C");
        textViewWind.setText(response.body().getWind().getWindSpeed() + " m/s");
        textViewPressure.setText(response.body().getWeather().getPressure() + " hPa");

    }

    @Override
    public void onFailure(Call<Post> call, Throwable t) {
        textViewError.setText(t.getMessage());
        textViewCity.setText("");
        textViewTemperature.setText("-");
        textViewWind.setText("-");
        textViewPressure.setText("-");
    }
});
```

Rys 2. Zapytanie do endpointu “weather”

```
package pl.zerhaz.checkweather;

import com.google.gson.annotations.SerializedName;

public class Post {
    @SerializedName("id")
    private int cityId;
    @SerializedName("name")
    private String cityName;
    @SerializedName("main")
    private Weather weather;
    @SerializedName("wind")
    private Wind wind;

    public Weather getWeather() { return weather; }

    public int getCityId() { return cityId; }

    public String getCityName() { return cityName; }

    public Wind getWind() { return wind; }
}
```

Rys 3. Klasa *Post*

```java
package pl.zerhaz.checkweather;

import com.google.gson.annotations.SerializedName;

import java.text.DecimalFormat;

class Weather {

    @SerializedName("temp")
    private Double temperature;
    @SerializedName("pressure")
    private Double pressure;

    private static DecimalFormat df = new DecimalFormat( pattern: "0.0");

    public String getTemperature() {
        temperature = temperature - 273.15;
        return df.format(temperature);
    }

    public String getPressure() {
        int p = (int) Math.round(pressure);
        return Integer.toString(p);
    }

}
```

Rys 4. Klasa *Weather*

```java
package pl.zerhaz.checkweather;

import com.google.gson.annotations.SerializedName;

import java.text.DecimalFormat;

public class Wind {

    @SerializedName("speed")
    private Double windspeed;

    private static DecimalFormat df = new DecimalFormat( pattern: "0.0");

    public String getWindSpeed() {
        return df.format(windspeed);
    }

}
```

Rys 5. Klasa *Wind*

Drugie zapytanie wykonywane jest do endpointu “forecast”. Obiekt JSON konwertowany jest do obiektu klasy *Forecast*. Następnie dokonywane jest parsowanie danych, gdyż darmowe API wykorzystywane w niniejszym projekcie udostępnia tylko dane zbiorcze dla kilku kolejnych dni z trzygodzinnymi odstępami czasowymi. Aplikacja natomiast ma wyświetlać prognozowaną temperaturę dla środka dnia i środka nocy. Obiekt *Forecast* zwraca listę w której wyszukiwane są dane dotyczące temperatury dla godziny 15.00 i 3.00 dla dwóch następnych dni. Dane te są wstawiane w odpowiadające im obiekty Text View.

```
@Override
public void onResponse(Call<Forecast> call, Response<Forecast> response) {
    if (!response.isSuccessful()) {
        textViewError.setText("Code: " + response.code());
        textViewTomorrowMax.setText("-");
        textViewTomorrowMin.setText("-");
        textViewAfterTomorrowMax.setText("-");
        textViewAfterTomorrowMin.setText("-");
        return;
    }

    Date currentDate = new Date();
    Calendar localDateTime1 = Calendar.getInstance();
    Calendar localDateTime2 = Calendar.getInstance();
    localDateTime1.setTime(currentDate);
    localDateTime2.setTime(currentDate);
    localDateTime1.add(Calendar.DATE,  amount: 1);
    localDateTime2.add(Calendar.DATE,  amount: 2);
    Date currentDatePlusOneDay = localDateTime1.getTime();
    Date currentDatePlusTwoDays = localDateTime2.getTime();
    String tomorrowDateMAX = dateFormat.format(currentDatePlusOneDay) + " 15:00:00";
    String tomorrowDateMIN = dateFormat.format(currentDatePlusOneDay) + " 03:00:00";
    String afterTomorrowDateMAX = dateFormat.format(currentDatePlusTwoDays) + " 15:00:00";
    String afterTomorrowDateMIN = dateFormat.format(currentDatePlusTwoDays) + " 03:00:00";

    for (int i = 0; i < 30; i++) {
        if (response.body().getForecastList(i).contains(tomorrowDateMAX)) {
            String[] words = response.body().getForecastList(i).split( regex: " ");
            for (String word : words) {
                if (word.startsWith("main={temp=")) {
                    int t = Integer.parseInt(word.substring(11, 14)) - 273;
                    String s = "";
                    if (t >= 0) {
                        s = "+" + t + " °C";
                        textViewTomorrowMax.setText(s);
                    } else {
                        s = t + " °C";
                        textViewTomorrowMax.setText(s);
                    }
                }
            }
        }
    }
```

Rys 6. Fragment zapytania do endpointu “forecast” wraz z paserem wartości temperatury dla godziny 15.00 następnego dnia.

3.3. Wybór miasta

Jak zostało wspomniane wcześniej - aplikacja pobiera dane dla wskazanego po id miasta. Miasta te zdefiniowane są w tablicy *citiesList[]* i wyświetlone użytkownikowi w formie obiektu typu *Spinner* (więcej w pkt 4.3). Jednocześnie mogą być wyświetlane dane dla tylko jednego miasta.

Aplikacja przechowuje informacje o wybranym mieście (id i nazwa) w obiektach typu SharedPreferences. Jest to klasa umożliwiająca zapis danych jako klucz-wartość (oraz ich odczyt), a dane te są przechowywane również po zamknięciu aplikacji. Dzięki temu aplikacja zapamiętuje jakie miasto było ostatnio wybrane przez użytkownika i po ponownym otwarciu wyświetla mu się właśnie ono bez potrzeby ręcznego wyboru z listy. Gdy aplikacja uruchamiana jest po raz pierwszy domyślnym wyborem jest Warszawa.

```
SharedPreferences getPref = getPreferences(Context.MODE_PRIVATE);
cityID = getPref.getString( key: "cityID", defValue: null);
cityName = getPref.getString( key: "cityName", defValue: null);
spinnerPosition = getPref.getString( key: "spinnerPosition", defValue: "13");
```

Rys 7. Odczyt wartości z SharedPreferences i zapisanie ich do zmiennych typu String. spinnerPosition określa pozycję na liście citiesList[], wartością domyślną jest 13, czyli “Warszawa”

3.4. Wyświetlanie reklam

Aplikacja wyświetla jedną reklamę typu SMART_BANER, która ładowana jest po zakończeniu uruchamiania aplikacji.

```
MobileAds.initialize( context: this, new OnInitializationCompleteListener() {
    @Override
    public void onInitializationComplete(InitializationStatus initializationStatus) {
    }
});
mAdView = findViewById(R.id.adViewBaner_1);
AdRequest adRequest = new AdRequest.Builder().build();
mAdView.loadAd(adRequest);
```

Rys 8. Metoda inicjalizująca wczytanie reklamy

4. Dokumentacja użytkowa

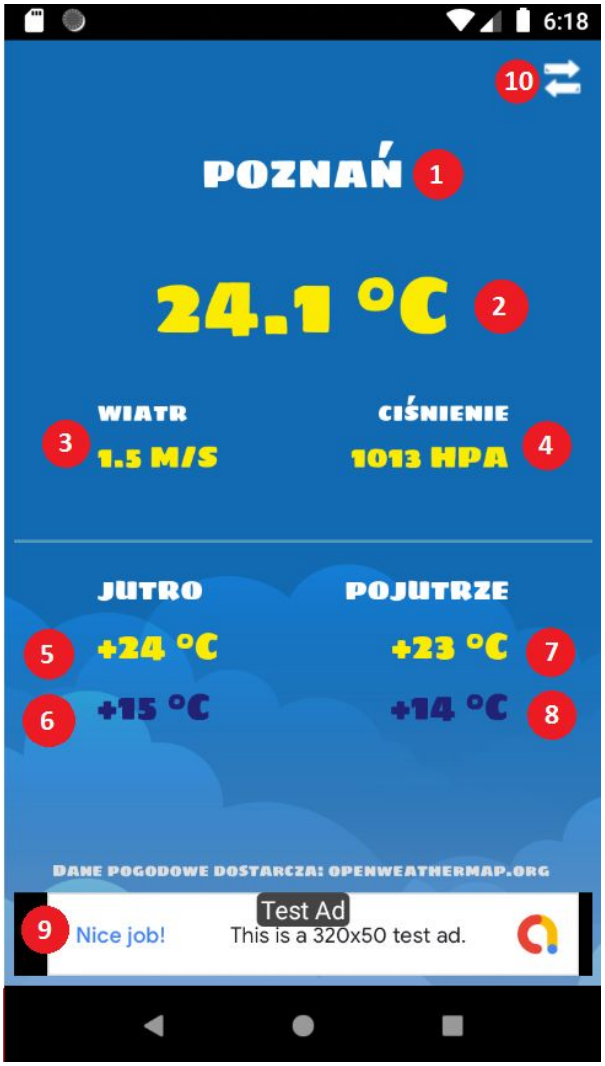
4.1. Informacje ogólne

Aplikację można zainstalować uruchamiając na telefonie z systemem Android plik "install.apk" [https://github.com/zerhaz/POGODA/blob/master/install.apk].

UWAGA: Kod źródłowy zamieszczony na platformie GITHUB **nie zawiera** klucza API opeanweathermap.com.

4.2. Informacje wyświetlane na ekranie

Aplikacja składa się z jednego widoku (activity), na którym widoczne są dane opisane na rys. 9.



Rys 9. Interfejs aplikacji

Objaśnienia poszczególnych elementów oznaczonych numerami:

1. Nazwa miasta, dla którego wyświetlana jest pogoda
2. Aktualna temperatura powietrza (w stopniach Celsjusza)
3. Aktualna prędkość wiatru (w metrach na sekundę)
4. Aktualne ciśnienie powietrza (w hektopaskalach)
5. Prognozowana temperatura powietrza w dniu jutrzejszym o godzinie 15.00 (w stopniach Celsjusza)
6. Prognozowana temperatura powietrza w dniu jutrzejszym o godzinie 3.00 (w stopniach Celsjusza)

7. Prognozowana temperatura pojutrze o godzinie 15.00 (w stopniach Celsjusza)
8. Prognozowana temperatura pojutrze o godzinie 3.00 (w stopniach Celsjusza)
9. Baner reklamowy
10. Przycisk zmiany miasta.

Po wciśnięciu przycisku oznaczonego jako “10” pojawia się lista miast, dla których można wybrać wyświetlanie pogody. Użytkownik może kliknąć na dowolne miasto, co spowoduje odświeżenie danych dla elementów od “1” do “8”.

Rys 10. Lista wyboru miasta