# Dokumentacja projektu
# Weather App
# Mobilne interfejsy multimedialne

Dawid Bitner

27 marca 2020

Politechnika Śląska
Wydział Matematyki Stosowanej
Rok akademicki 2019/2020

# Spis treści

# 1 Narzędzia, skrypty, źródła zewnętrzne

Do realizacji projektu użyłem jako środowiska programistycznego *Android Studio* w wersji *3.5.1.*
Postanowiłem, że użyję frameworka do języka *Dart - Flutter'a* w wersji *v1.12.13+hotfix.8.*
Emulator symulował urządzenie *Nexus 5X*, na którym był zainstalowany system *Android* w wersji *8.1 API 27.*

Podczas realizacji projektu użyłem kilku zależności dodanych do projektu we Flutterze - opiszę je krótko na podstawie pliku *pubspec.yaml*

```
    name: weatherapp
description: Simply aplication using Open Weather Map API

version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^0.1.2
  retrofit: ^1.3.1
  flutter_simple_dependency_injection: ^1.0.2
  json_serializable: ^3.2.5
  bloc_pattern: ^2.5.1
  rxdart: ^0.23.1
  fluttertoast: ^3.1.3
  intl: ^0.16.1

dev_dependencies:
  flutter_test:
    sdk: flutter
  retrofit_generator: ^1.3.1
  build_runner: ^1.7.4

flutter:
  uses-material-design: true
  assets:
    - assets/
```

**cupertino-icons** - jest to repozytorium zasobów zawierające domyślny zestaw zasobów ikon używanych przez Flutter - wykorzystałem ikonę lupy do wyszukiwania miasta.

**retrofit** - wykorzystany w celu łatwego dostępu do API, bez pisania dużej ilości kodu, doskonale współgra z serializatorem i deserializatorem *JSON*.

**flutter-simple-dependency-injection** - dzięki temu mogłem wykorzystać wstrzykiwanie zależności w swoich programie.

**json-serializable** - wykorzystany w celu serializacji danych, ułatwia pisanie programu poprzez tworzenie plików, przy użyciu komendy na podstawie prostych zależności zostały stworzone pliki *g.dart* dzięki który można było pobierać dane z API. Przykład:

plik weather.dart:

```
    import 'package:json_annotation/json_annotation.dart';

part 'weather.g.dart';

@JsonSerializable()
class Weather {
  String main;
  String description;
  String icon;
  Weather({this.main, this.description, this.icon});

  //https://flutter.dev/docs/development/data-and-backend/json
  factory Weather.fromJson(Map<String, dynamic> json) => _$WeatherFromJson(json);
  Map<String, dynamic> toJson() => _$WeatherToJson(this);
}
```

Oraz stworzony na jego podstawie plik weather.g.dart:

```
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'weather.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

Weather _$WeatherFromJson(Map<String, dynamic> json) {
  return Weather(
    main: json['main'] as String,
    description: json['description'] as String,
```

```
    icon: json['icon'] as String,
  );
}

Map<String, dynamic> _$WeatherToJson(Weather instance) => <String, dynamic>{
      'main': instance.main,
      'description': instance.description,
      'icon': instance.icon,
    };
```

**bloc-pattern** - w celu prostszego zastosowania logiki biznesowej w projekcie

**rxdart** - w celu wykorzystania programowania reaktywnego **fluttertoast** - dołączone standardowo - w celu używania wyskakujących powiadomień - *Toastów*.

**intl** - ten pakiet zapewnia funkcje internacjonalizacji i lokalizacji, w tym tłumaczenie wiadomości, liczbę mnogą i płcie, formatowanie i analizę dat / liczb oraz tekst dwukierunkowy. Dołączany standardowo.

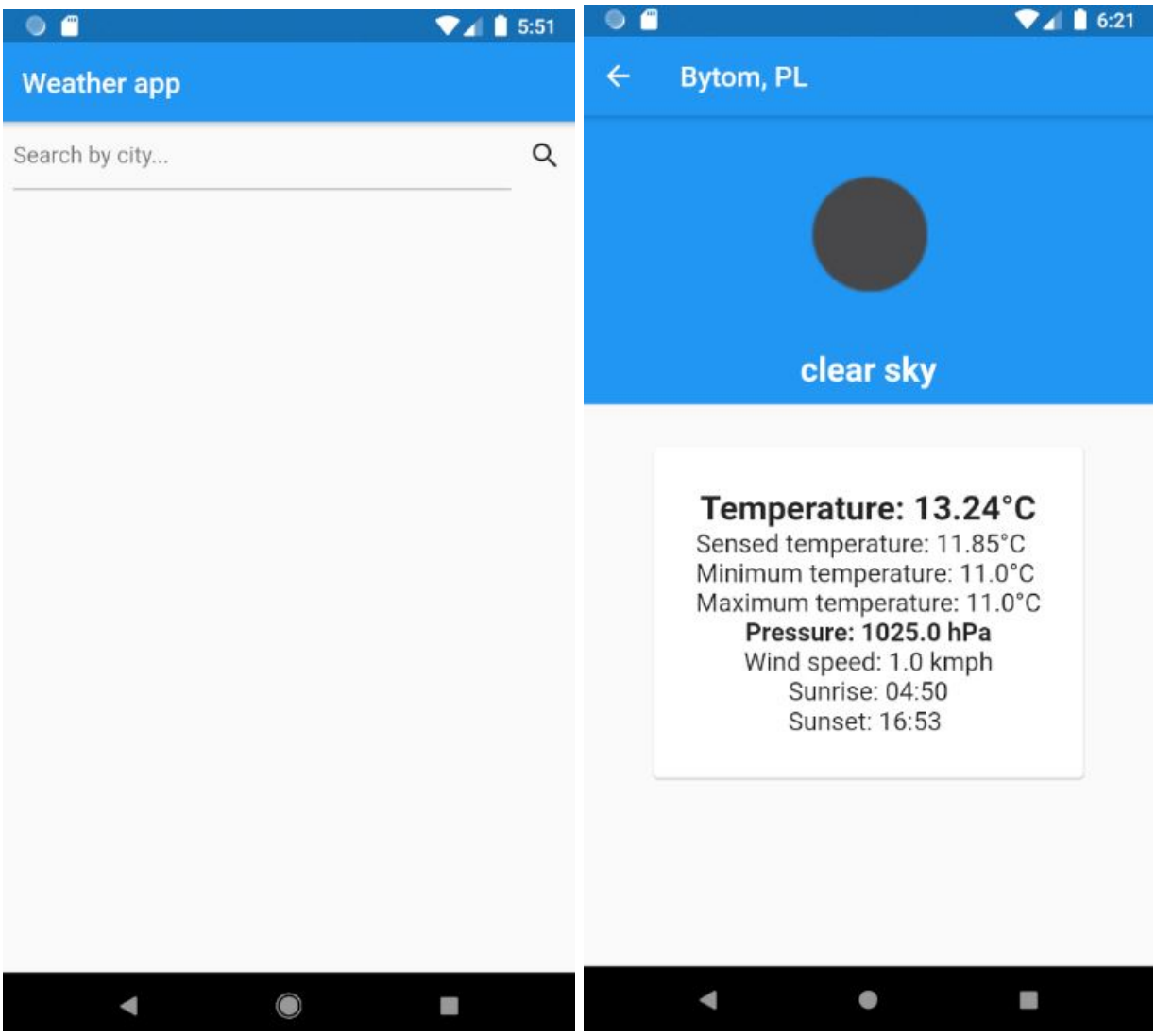
**uses-material-design** - w celu używania *Google Design*.

# 2 Problemy podczas realizacji

Nie miałem większych problemów podczas realizacji tego projektu we Flutterze którego dopiero poznaję, więc stosowałem się do zaleceń przedstawianych np. na *Stack Overflow*, czy posiłkując się opinią znajomych z doświadczeniem w tej technologii. Posiłkowałem się również poradnikami zamieszczonymi pod tym adresem:
`https://bloclibrary.dev/#/?id=examples`
Jedyne problemy jakie wystąpiły to te związane z designem aplikacji, pierwszy raz spotkałem się z taką strukturą budowania UI.

# 3 Główna część projektu

Jako stronę powitalną mamy do użytku wyszukiwarkę pogody po nazwie miasta:




Dwie karty dostępne w aplikacji

Po wyszukaniu miasta zostajemy przenoszeni do części aplikacji która wyświetla informacje o pogodzie w wybranym mieście. Dostępne informacje to: nazwa miasta, kraj, ikona aktualnej pogody (zgodna z ikonami udostępnianymi przez API), opis słowny aktualnej pogody, temperatury: nominalna, odczuwalna, minimalna i maksymalna na dany dzień, ciśnienie, prędkość wiatru, oraz wschód i zachód słońca.

# 4 Opinia

W mojej opinii pomimo prostoty projektu mogłem nauczyć się dzięki niemu wielu aspektów tworzenia aplikacji we Flutterze. Użyłem wielu popularnych i używanych komercyjnie zależności. Skorzystałem z poradników i pomocy innych osób, największe problemy sprawiało mi

graficzne ułożenie interfejsu, ponieważ struktura i sposób tworzenia UI jest dla mnie mało przejrzysty, możliwe że istnieją narzędzia wspomagające programistę w tym zakresie, których jeszcze nie poznałem.