Mobilne Interfejsy Multimedialne

Projekt 1 - Pogodynka

Jakub Karmański

rok 3 semestr 6 - letni
Wydział Matematyki Stosowanej
kierunek Informatyka

Spis treści:

Użyte narzędzia i źródła

W projekcie “Pogodynka” wykorzystane zostały darmowe narzędzia:

1. Android Studio,
2. GIMP,
3. GIT

oraz strony:

https://design.google/
https://developer.android.com/guide/topics/ui/look-and-feel
https://www.interaction-design.org/literature/topics/material-design
https://stackoverflow.com/
https://grafmag.pl/artykuly/czym-jest-material-design-teoria-zasady-materialy-i-przyklady
https://www.online-convert.com/
https://material.io/design/
https://openweathermap.org/api
https://angrytools.com/android/button/

Problemy w projekcie

Zaplanowanie widoku aplikacji

Był to jeden z najważniejszych problemów projektowych.Problem ten wymaga odpowiedniego rozplanowania dla przejrzystości i ułatwionego korzystania z aplikacji dla jej użytkowników.
Layout musiał być prosty , przejrzysty a zarazem bogaty w informacje potrzebne użytkownikowi

Rozpoznanie danych przychodzących z używanego API

Problem ten sprowadza się tak naprawdę do szczegółowego zapoznania się ze wszelkimi danymi przychodzącymi z API oraz możliwościami udostępnionymi przez udostępniony punkt dostępu do endpointa - wszelkie zmiany np. języka otrzymywanych danych , formatu , oraz innych kluczowych parametrów endpoitu.

Najważniejsze punkty projektu

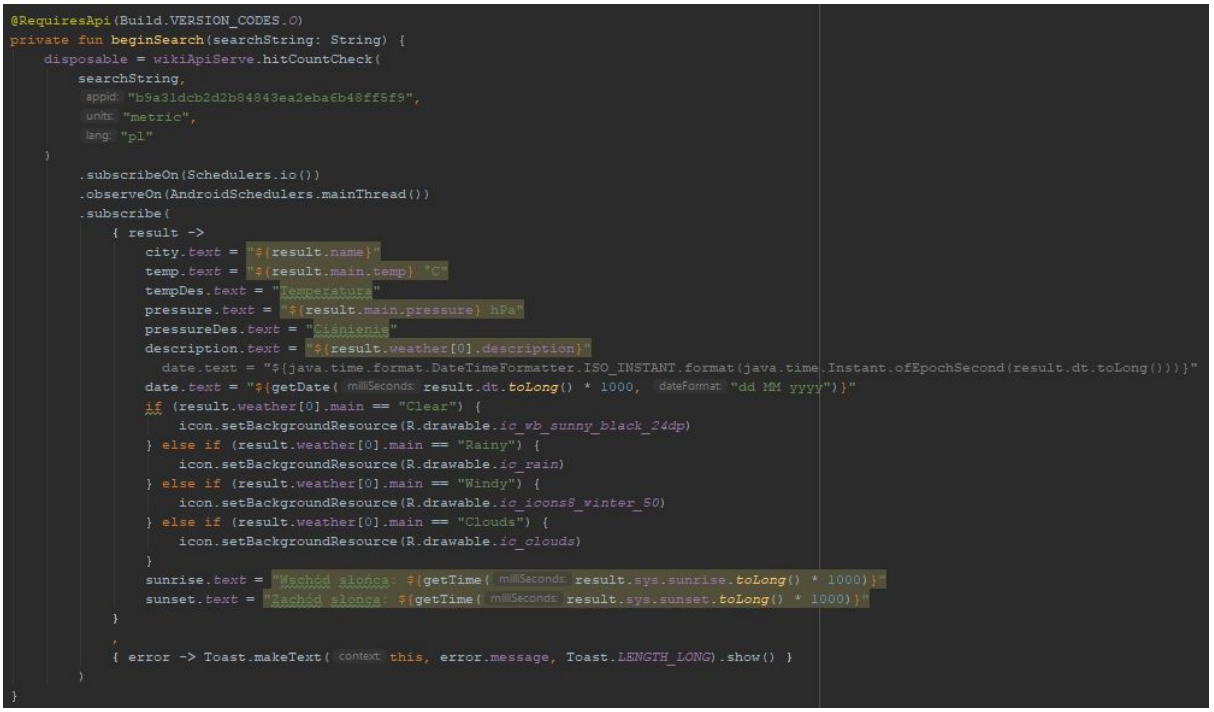
Klasa MainActivity, w której przeprowadzane jest odczyt danych z serwisu,wpisanie danych do przygotowanych komponentów oraz konwersja niektórych danych z racji ich przychodzącego z API formatu.

Poniżej nadanie akcji przyciskowi do szukania oraz ukrycie klawiatury po jego wciśnięciu.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    btn_search.setOnClickListener { it: View!
        if (edit_search.text.toString().isNotEmpty()) {
            beginSearch(edit_search.text.toString())
            val imm = this.getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
            imm.hideSoftInputFromWindow(btn_search.windowToken,  flags: 0)
        }
    }
}
```

Metoda beginSearch wysyłająca oraz odbierająca dane z serwisu. Wpisuje ona również dane do komponentów.



```
@RequiresApi(Build.VERSION_CODES.O)
private fun beginSearch(searchString: String) {
    disposable = wikiApiServe.hitCountCheck(
        searchString,
        appid: "b9a31dcb2d2b84843ea2eba6b48ff5f9",
        units: "metric",
        lang: "pl"
    )
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(
            { result ->
                city.text = "${result.name}"
                temp.text = "${result.main.temp} °C"
                tempDes.text = "Temperatura"
                pressure.text = "${result.main.pressure} hPa"
                pressureDes.text = "Ciśnienie"
                description.text = "${result.weather[0].description}"
                  date.text = "${java.time.format.DateTimeFormatter.ISO_INSTANT.format(java.time.Instant.ofEpochSecond(result.dt.toLong()))}"
                date.text = "${getDate( milliSeconds: result.dt.toLong() * 1000,  dateFormat: "dd MM yyyy")}"
                if (result.weather[0].main == "Clear") {
                    icon.setBackgroundResource(R.drawable.ic_wb_sunny_black_24dp)
                } else if (result.weather[0].main == "Rainy") {
                    icon.setBackgroundResource(R.drawable.ic_rain)
                } else if (result.weather[0].main == "Windy") {
                    icon.setBackgroundResource(R.drawable.ic_icons8_winter_50)
                } else if (result.weather[0].main == "Clouds") {
                    icon.setBackgroundResource(R.drawable.ic_clouds)
                }
                sunrise.text = "Wschód słońca: ${getTime( milliSeconds: result.sys.sunrise.toLong() * 1000)}"
                sunset.text = "Zachód słońca: ${getTime( milliSeconds: result.sys.sunset.toLong() * 1000)}"
            }
            ,
            { error -> Toast.makeText( context: this, error.message, Toast.LENGTH_LONG).show() }
        )
}
```

Metody konwertujące czas przychodzący z API w milisekund do poprawnego formatu daty i czasu.

```
fun getDate(milliSeconds: Long, dateFormat: String?): String? {
    val formatter = SimpleDateFormat(dateFormat)
    val calendar: Calendar = Calendar.getInstance()
    calendar.setTimeInMillis(milliSeconds)
    return formatter.format(calendar.getTime())
}

fun getTime(milliSeconds: Long): String? {
    return String.format(
        " %d:%d",
        ((milliSeconds / (1000 * 60 * 60)) % 24),
        ((milliSeconds / (1000 * 60)) % 60)
    )
}
```

Model budowy danych przychodzących z endpointu oraz wykorzystywanych w aplikacji.

```
object Model {
    data class Result(val weather: List<Weather>,val main: Main,val sys:Sys,val dt:String,val name: String)
    data class Weather(val main: String,val description: String)
    data class Main(val temp: String,val pressure: String,val humidity: String)
    data class Sys(val sunrise: String,val sunset: String)
}
```

Serwis odbierający dane ze wskazanego punktu, działający ze wtyczką retrofit dla androida.

```
interface WeatherApiService {

    @GET( value: "weather?")
    fun hitCountCheck(@Query( value: "q") city: String,
                      @Query( value: "appid") appid: String,
                      @Query( value: "units") units: String,
                      @Query( value: "lang") lang: String): Observable<Model.Result>

    companion object {
        fun create(): WeatherApiService {

            val retrofit = Retrofit.Builder()
                    .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                    .addConverterFactory(GsonConverterFactory.create())
                    .baseUrl( baseUrl: "https://api.openweathermap.org/data/2.5/")
                    .build()

            return retrofit.create(WeatherApiService::class.java)
        }
    }

}
```

Podsumowanie

Projekt wymagał odpowiedniej wiedzy na temat API oraz Material Design. Odpowiednie rozplanowanie widoku danych oraz przenoszenie ich pomiędzy interfejsem serwisu oraz główną aktywnością jest kluczowe dla odpowiedniego efektu aplikacji.

Dzięki prawidłowemu rozplanowaniu danych na widoku oczekiwany efekt jest zadowalający a co najważniejsze funkcjonalny nie tylko w celach edukacyjnych ale także z możliwością wykorzystania dla własnego użytku.