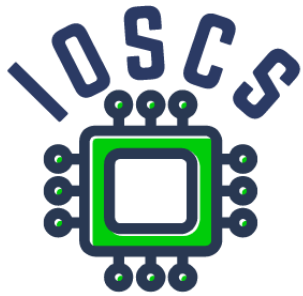


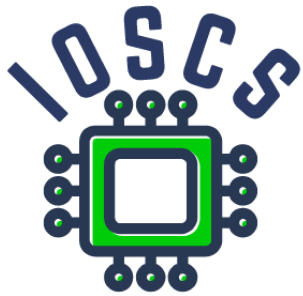
Project: Innovative Open Source Courses for Computer Science

Projektowanie aplikacji mobilnych Laboratoria

Radosław Maciaszczyk
West Pomeranian University of Technology in Szczecin

30.05.2021

Innovative Open Source Courses for Computer Science



This teaching material was written as one of the outputs of the project "Innovative Open Source Courses for Computer Science", funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

Project information

Project was implemented under the Erasmus+.
Project name: **"Innovative Open Source courses for Computer Science curriculum"**
Project nr: **2019-1-PL01-KA203-065564**
Key Action: **KA2 – Cooperation for innovation and the exchange of good practices**
Action Type: **KA203 – Strategic Partnerships for higher education**

Consortium
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
MENDELOVA UNIVERZITA V BRNE
ZILINSKA UNIVERZITA V ZILINE

Co-funded by the
Erasmus+ Programme
of the European Union

Pierwsza aplikacja – Hello World

To laboratorium zawiera:

- Instalację środowiska Android Studio
- Stworzenie pierwszego projektu
- Prezentacja środowiska
- Implementacja cyklu życia aktywności

Zadanie 1. Pobranie i instalacja Android Studio

Ze strony https://developer.android.com/studio pobierz i zainstaluj najnowszą wersję Android Studio

Wymagania systemowe [1]:

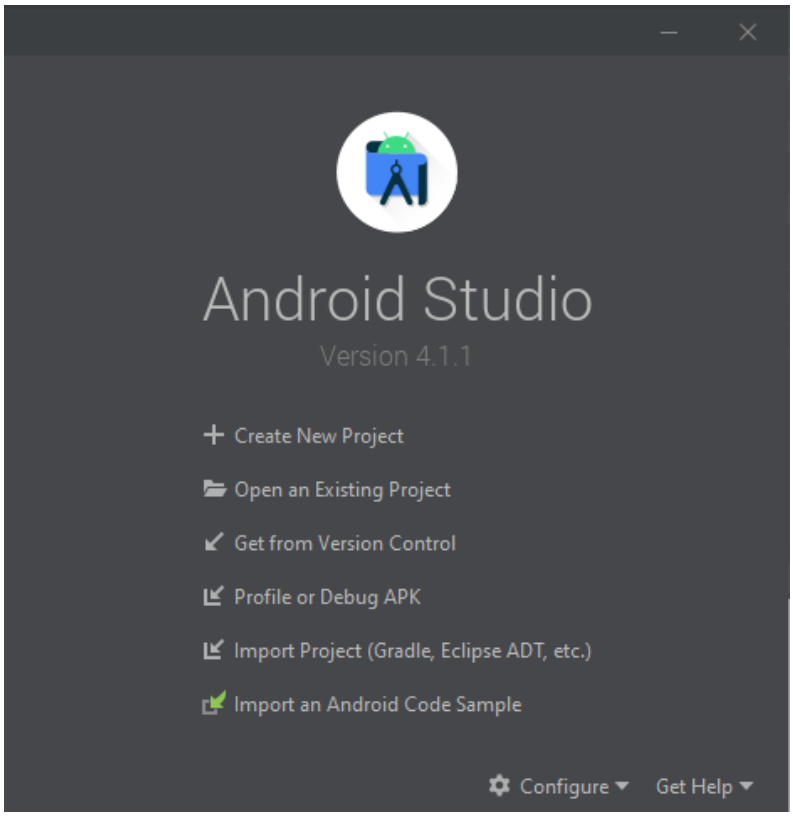
Windows	**Mac**	**Linux**
• 64-bit Microsoft® Windows® 8/10 • x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor • 8 GB RAM or more • 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) • 1280 x 800 minimum screen resolution	• MacOS® 10.14 (Mojave) or higher • ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework • 8 GB RAM or more • 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) • 1280 x 800 minimum screen resolution	• Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later. • x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3 • 8 GB RAM or more • 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) • 1280 x 800 minimum screen resolution

Zadanie 2. Utwórz projekt Hello World

Co-funded by the Erasmus+ Programme of the European Union

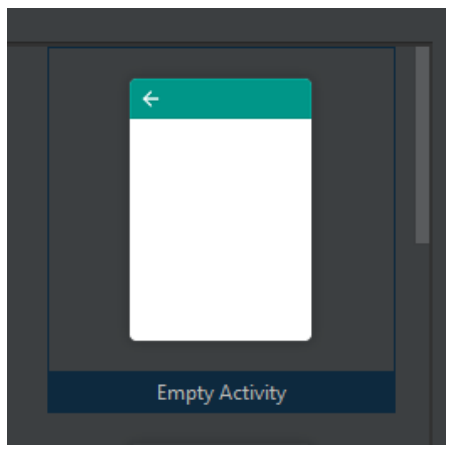
I. Stworzenie projektu

1. Utwórz nowy projekt

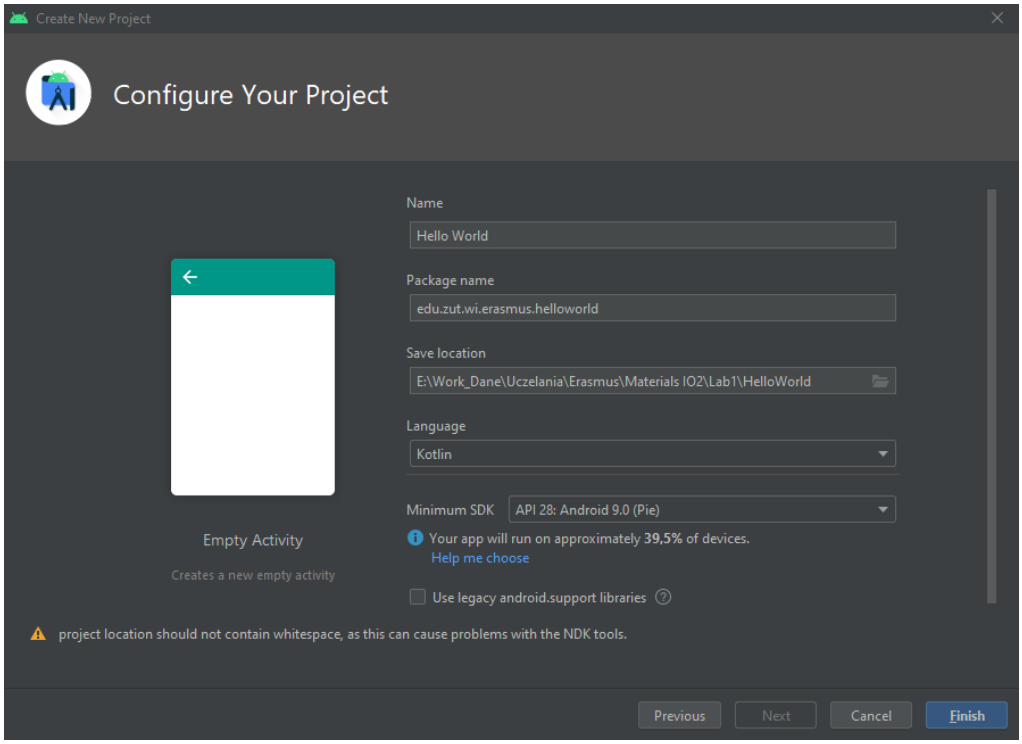


2. Wybierz „Empty Activity” -> next
3. Skonfiguruj projekt:
   Configure:
   - Name
   - Package name
   - Location project
   - Chose Language
   - Choose SDK

   Finish



Co-funded by the
Erasmus+ Programme
of the European Union



II. Stworzenie maszyny wirtualnej Android Virtual Device

Środowisko Android Studio pozwala na emulację urządzeń z systemem Android. Pozwala to na bezproblemowe testowanie stworzonych aplikacji.

1. Wybierz z menu: Tools->AVD Manager
2. Jeśli AVD nie istnieje, to należy je utworzyć
    a. Wybierz urządzenie np. Pixel 2
    b. Wybierz obraz systemu, np. Android 9
    c. Zdefiniuj nazwę
    d. Wejdź w ustawienia zaawansowane i sprawdź je.
    e. Finish

III. Uruchom aplikację

1. Wybierz z menu: Run -> Run ‘app’ or Shift + F10

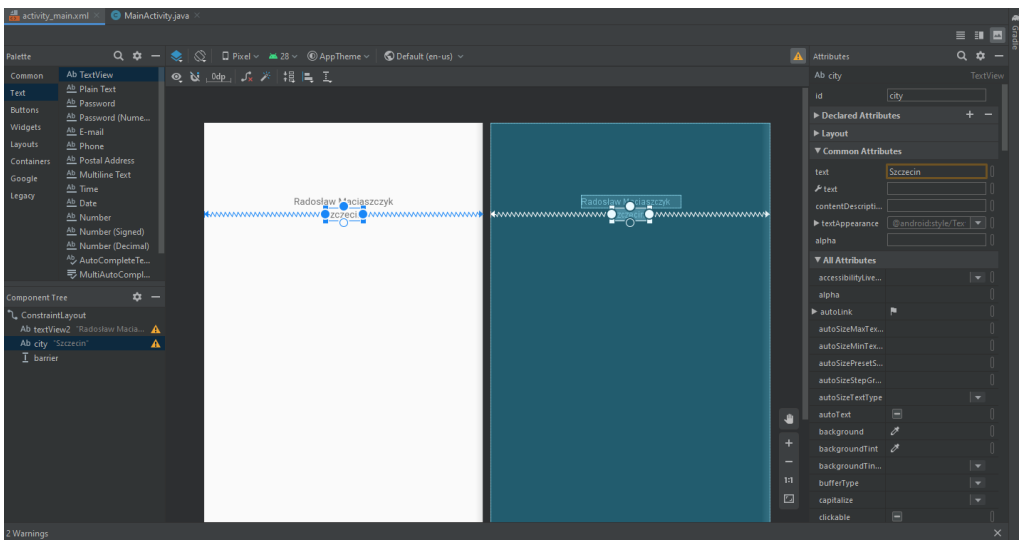
IV. Zmień tekst na ekranie i dodaj nowy obiekt TextView

1. Wyszukaj plik w projekcie ***R.layout.activity_main.xml*** i wyświetl (double click)
2. Wybierz tekst “Hello word” i go zmień
3. Dla TextView zmień **andoriod:id ->** android:id="@+id/name"
4. Z palety widgetów dodaj nowy widget: TextView
5. Dla TextView zdefinuj: **id** and **text**

Co-funded by the
Erasmus+ Programme
of the European Union

Wycinek kodu: ***R.layout.activity_main.xml***

```
<TextView
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Radosław Maciaszczyk"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.148" />

<TextView
    android:id="@+id/city"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Szczecin"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/name" />
```



Zadanie 3. Cykl życia aktywności (Acivity)

Zadanie to pokazuje cały cykl życia aktywności, jak zaimplementować intencję oraz jak dodać przycisk. Dodatkowo pokazuje jak korzystać z logowania przy użyciu **logcat**.

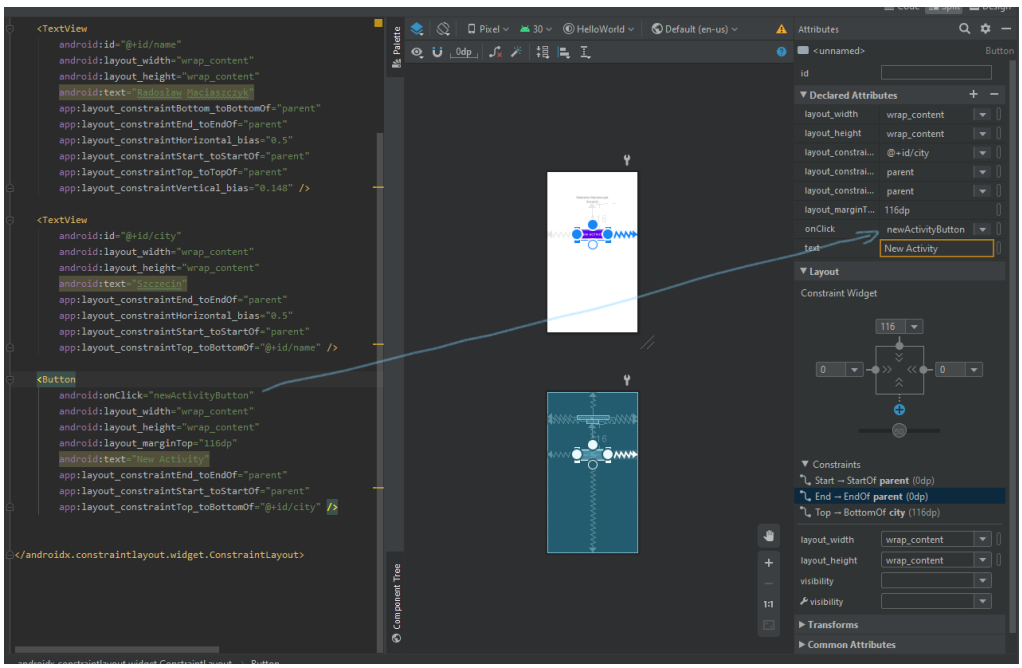
Znajomość cyklu życia aktywności jest kluczowe aby stworzyć poprawnie działającą aplikację, która potrafi przechować stany aplikacji pomiędzy uruchomieniami.

I. Dodaj przycisk do ekranu
1. W pliku ***R.layout.activity_main.xml*** dodaj przycisk z palety widgetów.
2. Zdefiniuj dla przycisku:
   id: newActivity
   text: New Activity

3. Idź do ***MainActivity.kt*** i zdefiniuj metodę która obsłuży zdarzenie onClick.

```
fun newActivityButton(view: View){

}
```

4. Idź do ***activity_main.xml*** i dodaj metodę onClick do przycisku.



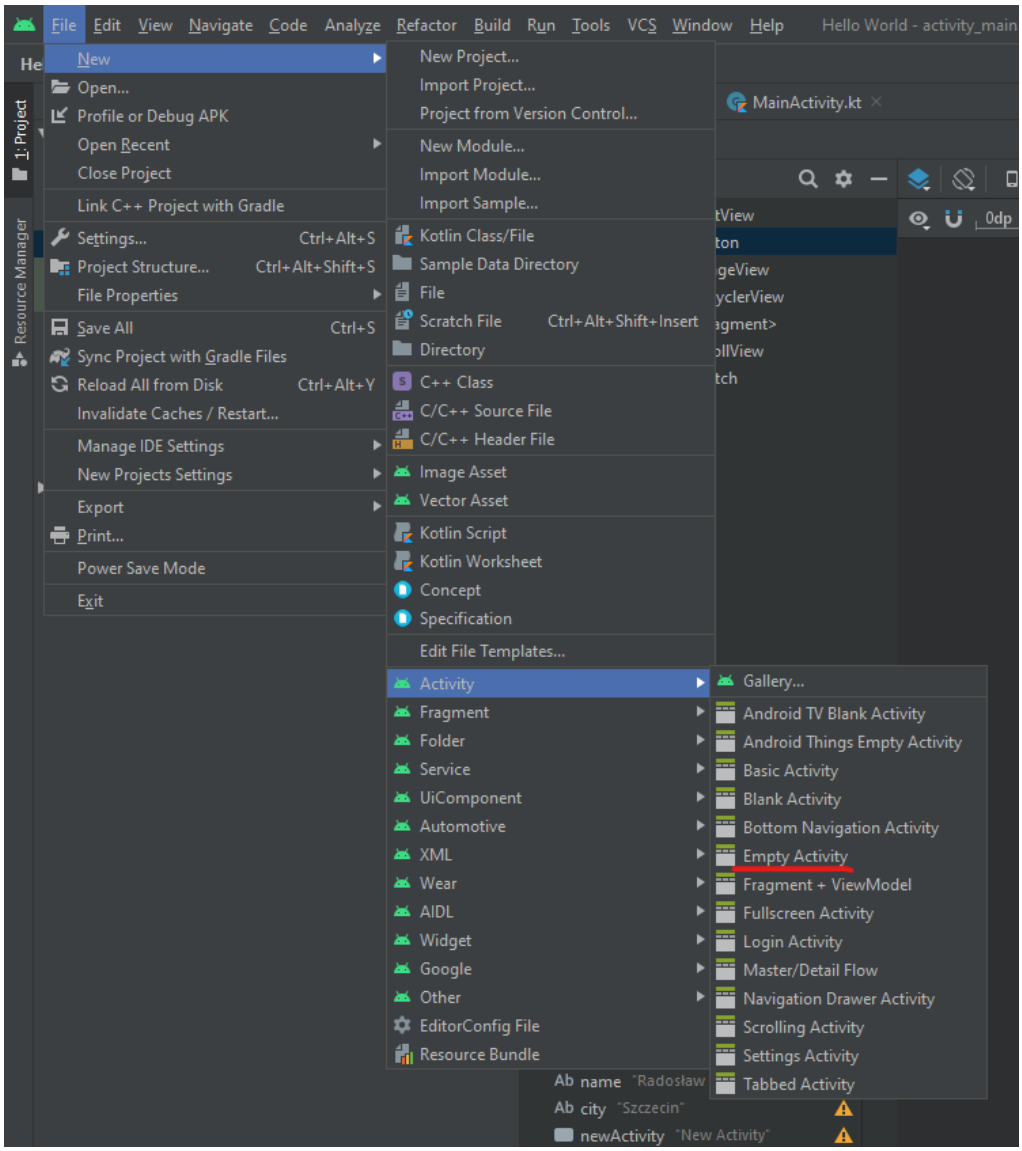
5. Ostateczna wersja xml dla przycisku

```
<Button
    android:onClick="newActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="116dp"
    android:text="New Activity"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/city" />
```

V. Stworzenie nowej aktywności

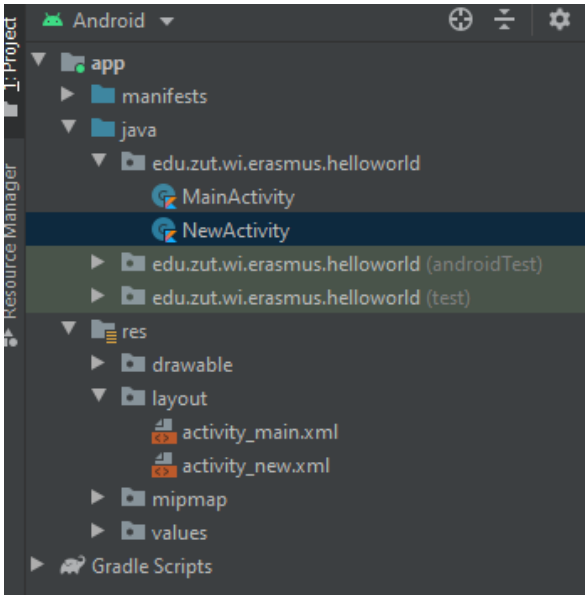
Co-funded by the Erasmus+ Programme of the European Union

1. Z menu wybierz File-> New -> Activity -> Empty Activity
2. Zdefiniuj Activity
   Activity name: NewActivity
   Click Finish



Po utworzeniu mamy następującą strukturę katalogów

Co-funded by the
Erasmus+ Programme
of the European Union



VI. Dodaj metody cyklu życia aktywności

1. Użyj skrótu klawiszowego CTRL+O (Show Override Members)
2. Wybierz metody z cyklu życia ***onPause, onStart, onRestart, onStop, onDestory***

VII. Wykorzystanie klasy Log

1. Zdefiniuj stałą TAG – (Dobra praktyka: TAG ma tę samą nazwę co klasa)

```
private val TAG = "NewActivity"
```

2. Dodaj wywołanie metody **Log** do wszystkich metod cyklu życia
   Np.

```
Log.i(TAG, "OnPause")
```

3. Ostateczna wersja NewActivity.kt

```
package edu.zut.wi.erasmus.helloworld

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class NewActivity : AppCompatActivity() {
    private val TAG = "NewActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_new)
        Log.i(TAG, "OnCreate")
    }

    override fun onPause() {
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
        super.onPause()
        Log.i(TAG, "OnPause")
    }

    override fun onRestart() {
        super.onRestart()
        Log.i(TAG, "OnRestart")
    }

    override fun onStart() {
        super.onStart()
        Log.i(TAG, "OnStart")
    }

    override fun onStop() {
        super.onStop()
        Log.i(TAG, "OnStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i(TAG, "OnDestroy")
    }
}
```

VIII. Dodanie obsługi powrotu za pomocą zdarzenia **UP**

1. Otwórz AndroidManifest.xml i zdefiniuj nadrzędną aktywność

```
<activity android:name=".NewActivity">
    android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

2. Wróć do MainActivity.kt
    1. Dodaj wszystkie metody cyklu życia do tej aktywności, podobnie jak w poprzedniej aktywności.
    2. Dodaj wywołanie metody ***Log***

3. Wewnątrz metody obsługującej zdarzenie onClick() ***newActivityButton*** dodaj

```
= Intent(this,NewActivity::class.java)
startActivity(intent)
```

Dodaj również informację o logowaniu

4. Ostateczna wersja MainActivity.kt

```
package edu.zut.wi.erasmus.helloworld

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
import android.view.View

class MainActivity : AppCompatActivity() {
    private val TAG = "MainActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.i(TAG, "OnCreate")
    }
    fun newActivityButton(view: View){
        Log.i(TAG, "newActivityButton")
        val intent = Intent(this,NewActivity::class.java).apply {  }
        startActivity(intent)
    }

    override fun onPause() {
        super.onPause()
        Log.i(TAG, "OnPause")
    }

    override fun onRestart() {
        super.onRestart()
        Log.i(TAG, "OnRestart")
    }

    override fun onStart() {
        super.onStart()
        Log.i(TAG, "OnStart")
    }

    override fun onStop() {
        super.onStop()
        Log.i(TAG, "OnStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i(TAG, "OnDestroy")
    }
}
```

3. Uruchom projekt

Spójrz na ***logcat*** I obserwuj zalogowane wartości.

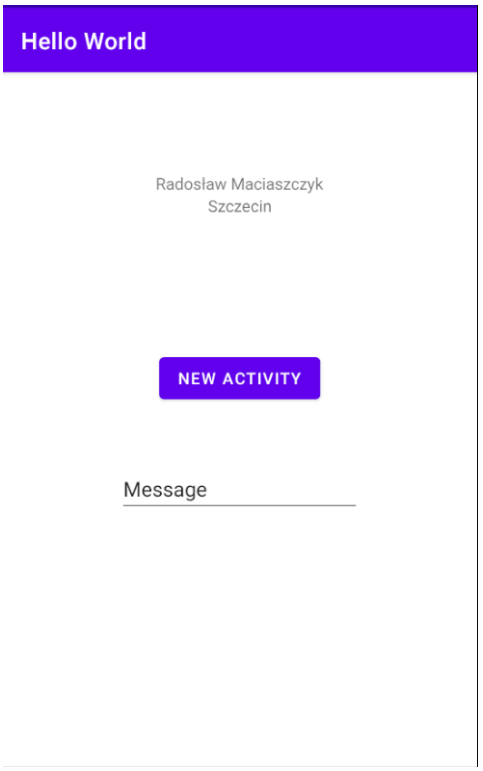
Zadanie dodatkowe:

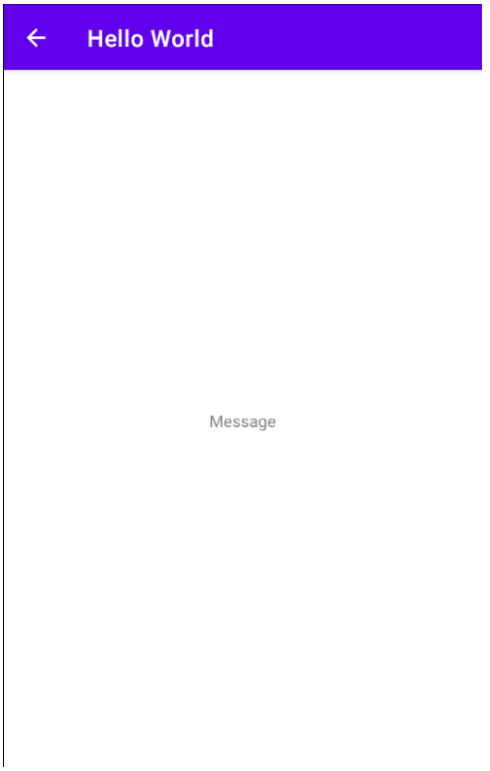
Do pierwszej aktywności dodaj ***Input Text*** i używając ***Intent*** wyślij wiadomość do drugiej aktywności, następnie ją wyświetl.

Stwórz widok podobnie jak poniżej:

Pierwsza aktywność MainActivity wysyła wiadomość po naciśnięciu przycisku do Second Activity

Co-funded by the Erasmus+ Programme of the European Union





Więcej informacji jak wysłać dane z wykorzystaniem intencji:
https://developer.android.com/training/basics/firstapp/starting-activity

Kod wynikowy aplikacji znajduje się na https://github.com/matam/Erasmus_Lab1

Literatura

[1] - https://developer.android.com/studio#downloads

[2] - https://developer.android.com/studio/intro/keyboard-shortcuts

Dodatkowe informacje:

https://developer.android.com/studio/intro

https://developer.android.com/training/basics/firstapp

https://developer.android.com/training/basics/firstapp/starting-activity

Co-funded by the
Erasmus+ Programme
of the European Union

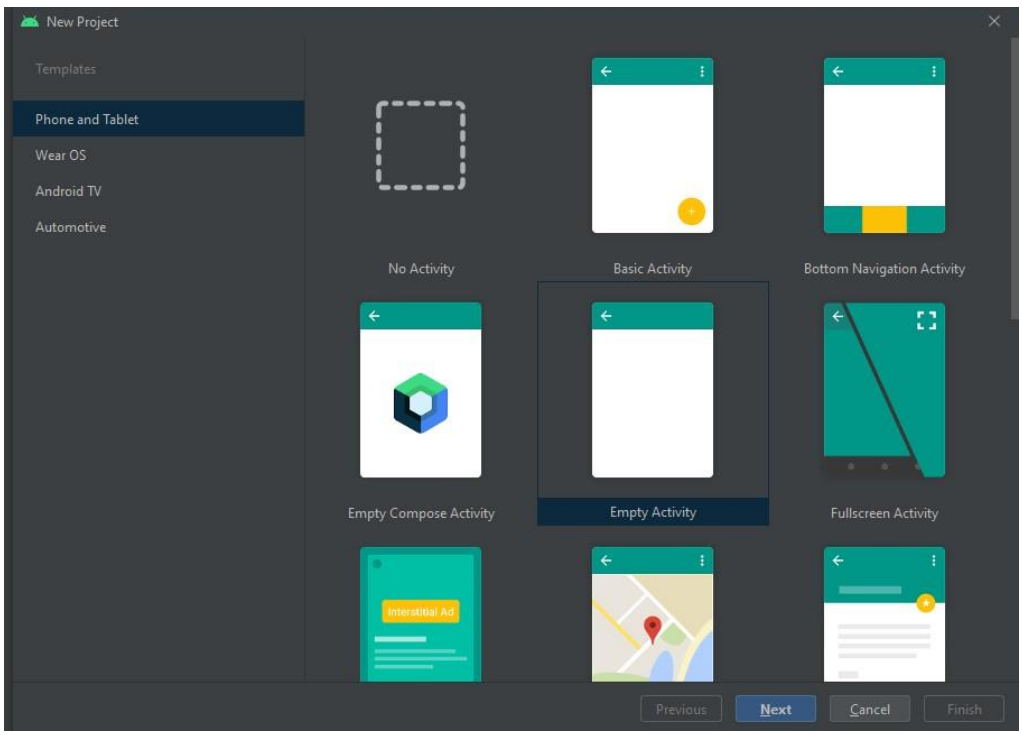
Interfejs użytkownika

Interfejs użytkownika - część programu, aplikacji lub systemu operacyjnego odpowiedzialna za komunikację z użytkownikiem. Jego najczęściej spotykaną formą jest graficzny interfejs użytkownika (GUI). Składa się on z elementów graficznych, które ujednolicają wygląd aplikacji i prezentują ją w rozpoznawalnej i przewidywalnej formie. Projektując interfejs, oprócz warstwy graficznej (ikony, menu, pola tekstowe, listy), należy pamiętać o tworzeniu ścieżek ruchu użytkownika, architekturze informacji oraz procesach interakcji. Dlatego połączenie UI i UX (user experience) jest obecnie tak ważne w procesie projektowania aplikacji. W systemie Android kompletny system UI i UX reprezentowany jest przez Material Design (https://material.io/). Obecnie w wersji 3.

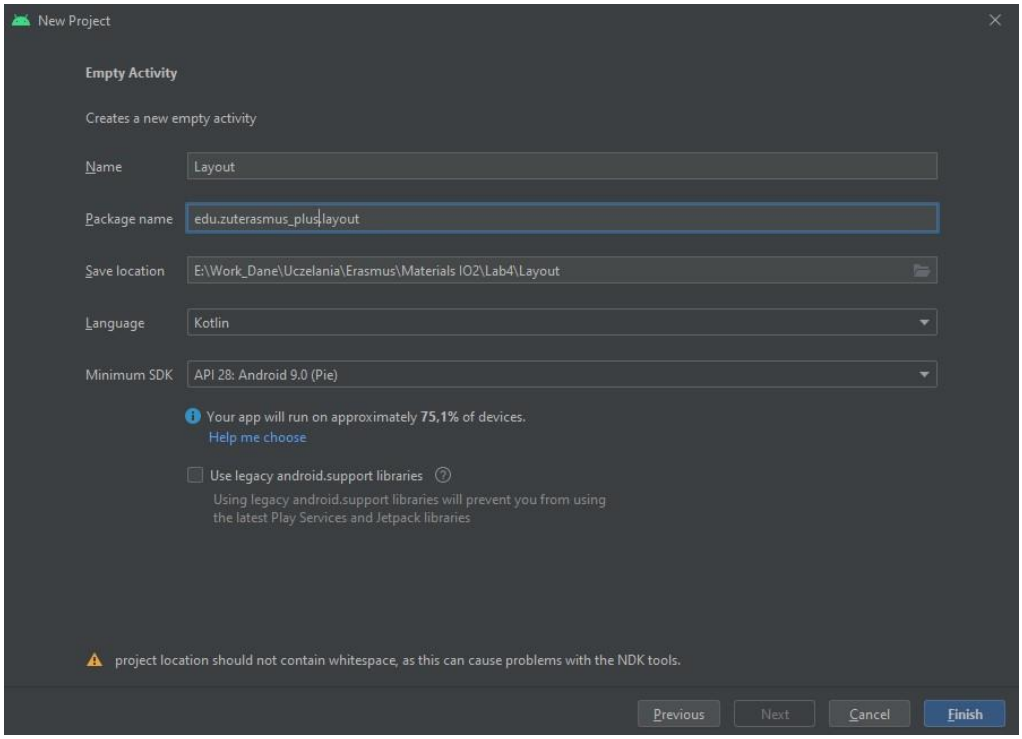
To laboratorium ma na celu pokazanie podstaw tworzenia interfejsów, wraz z wprowadzeniem do używania narzędzi pomagających w tworzeniu interfejsów. Najpierw stworzymy typową aplikację „Hello World", a następnie aplikację „Kalkulator".

Projekt Hello World

1. Uruchom Android Studio
2. Utwórz nowy projekt - wybierz „Empty Activity"



3. Wpisz nazwę projektu, nazwę pakietu, wybierz wersję API i ścieżkę.



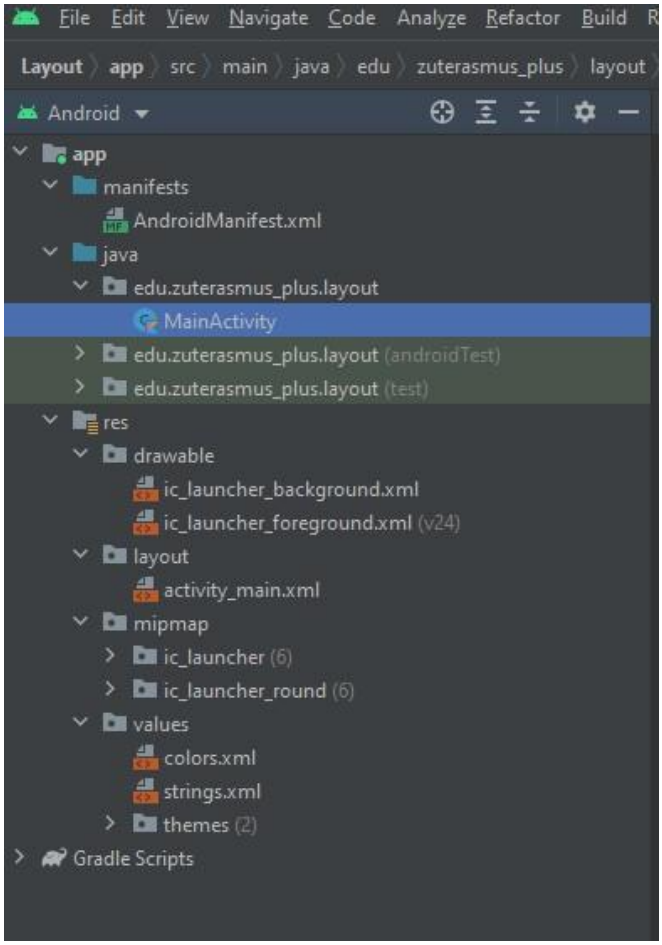
4. Naciśnij Finish i poczekaj na utworzenie szkieletu projektu.

Poniżej przedstawiono widok początkowy projektu, a także pokazano strukturę projektu.

Kod przechowujemy w katalogu ***JAVA*** (również w przypadku pisania z użyciem języka Kotlin). W katalogu ***manifest*** przechowywany jest plik **AndroidManifest.xml** zawierający ważne dla kompilatora informacje, w tym definicje komponentów aplikacji czy uprawnień.

Katalog ***res*** służy do przechowywania informacji o zasobach aplikacji, w tym katalogu ***layouts***, gdzie w pliku xml definiujemy wygląd aplikacji.

Co-funded by the
Erasmus+ Programme
of the European Union

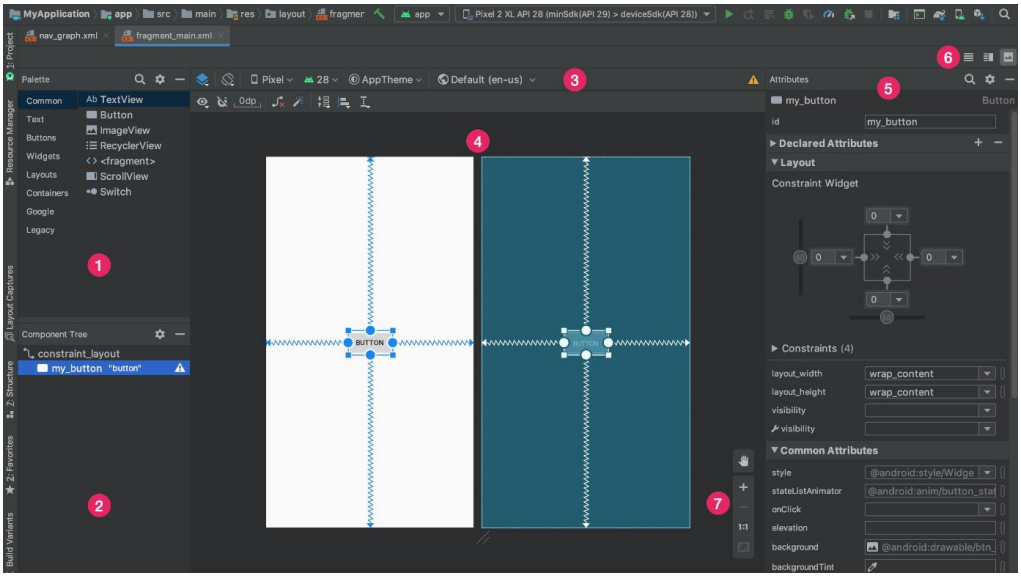


5. Tworzenie interfejsu użytkownika

Interfejs tworzony jest w plikach xml. Android Studio pozwala na tworzenie kodu bezpośrednio lub za pomocą edytora grafiki.

Wybierz plik **activity_main.xml** z katalogu układu. Zostanie otwarte narzędzie edytor układu (Layout Editor) (https://developer.android.com/studio/write/layout-editor):

Co-funded by the
Erasmus+ Programme
of the European Union

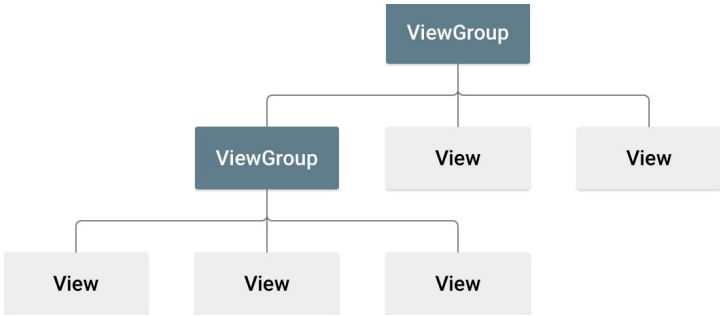


1) **Palette**: Zawiera różne widoki i grupy widoków, które możesz przeciągnąć do swojego układu.
2) **Component Tree**: Pokazuje hierarchię komponentów w twoim układzie.
3) **Toolbar**: Służy do skonfigurowania wyglądu układu w edytorze, pozwala zmienić atrybuty układu.
4) **Design editor**: Edytuj swój układ w widoku Projekt, widoku Blueprint lub obu.
5) **Attributes**: Kontrolki atrybutów wybranego widoku.
6) **View mode**: Zobacz swój układ w trybie ikony Code code mode, Design design mode icon lub Split split mode icon. Tryb podzielony pokazuje oba okna Code i Design w tym samym czasie.
7) **Zoom and pan controls**: Kontroluj rozmiar i pozycję podglądu w obrębie edytora.

Więcej informacji o interfejsie można znaleźć na stronie:
https://developer.android.com/studio/write/layout-editor

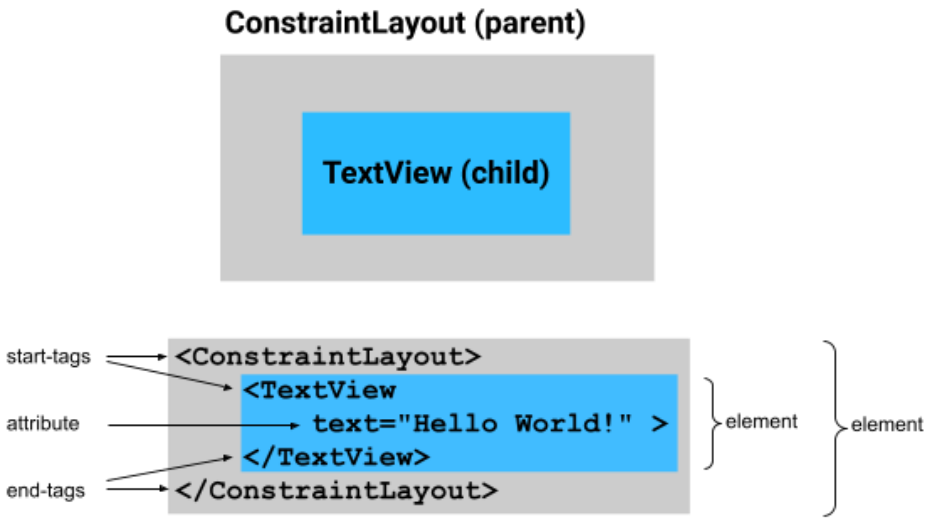
Interfejs użytkownika jest zbudowany hierarchicznie przy użyciu obiektów **ViewGroup** (układ) i **View** (widżet)

Co-funded by the
Erasmus+ Programme
of the European Union



Jak widać na powyższym rysunku obiekty **ViewGroup** pozwalają na tworzenie hierarchii, w których zawarte są poszczególne obiekty

6. Interfejs użytkownika w tworzonym projekcie jest zdefiniowany w następujący sposób



Plik xml z definicją układu graficznego

Co-funded by the
Erasmus+ Programme
of the European Union

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Do definiowania wyglądu obiektów używamy atrybutów. Poszczególne znaczniki xml (np. TextView) w pliku XML odpowiadają nazwom klas, w których te obiekty są zdefiniowane.

Spójrz na tag dla **ConstraintLayout**, i zauważ, że używa **androidx.constraintlayout.widget.ConstraintLayout** zamiast po prostu **ConstraintLayout** To dlatego, że jest częścią Android Jetpack. Android Jetpack posiada dodatkowe funkcjonalności ułatwiające budowanie aplikacji. Rozpoznasz, że ten komponent UI jest częścią Jetpack, ponieważ zaczyna się od "**androidx**".

Ćwiczenie 1

a) Zamień "Hello World" na swoje imię.
b) Dodaj nowy obiekt TextView z nazwą miasta, z którego pochodzisz.
c) Umieść wszystkie napisy w pliku strings.xml (przewodnik - https://developer.android.com/guide/topics/resources/string-resource)

Ćwiczenie 2 (dodatkowe)

Wykonaj ćwiczenie na tej stronie

https://developer.android.com/codelabs/constraint-layout

Projekt Kalkulator

1. Utwórz nowy układ lub pobierz nowy projekt z witryny

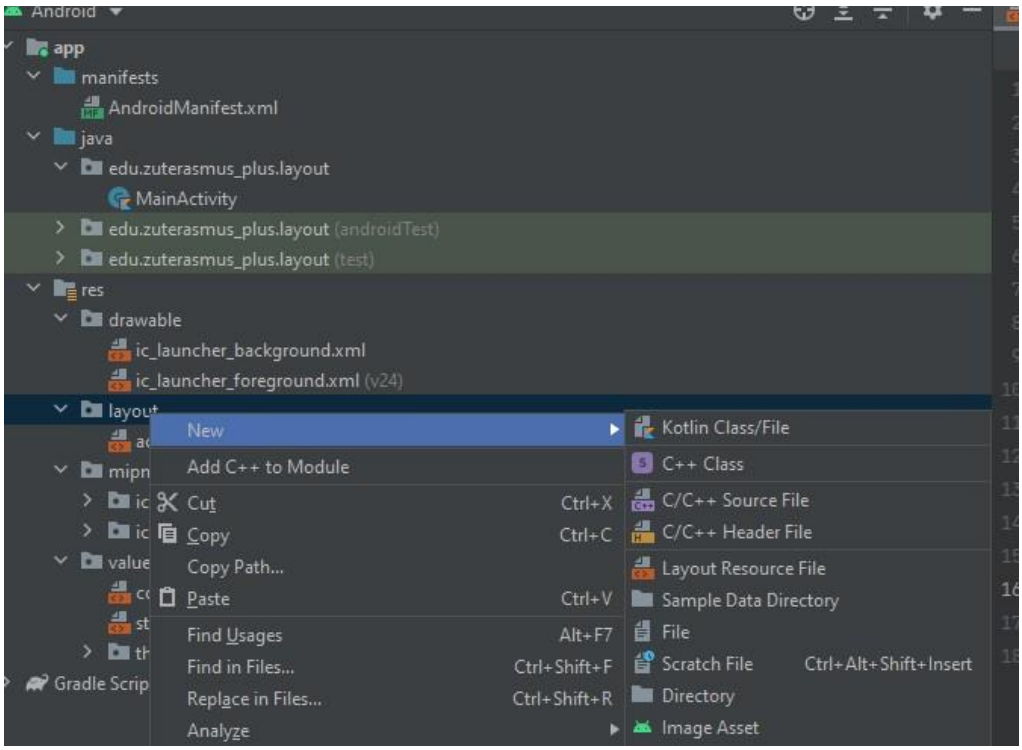
Co-funded by the Erasmus+ Programme of the European Union

https://github.com/matam/Erasmus_Lab1

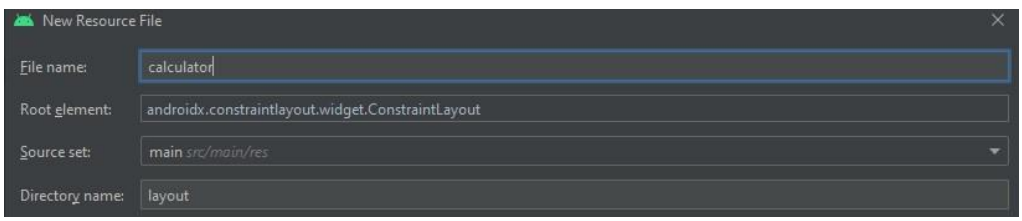
Jeśli pobierasz projekt z repozytorium, przejdź do punktu nr. 2

Kliknij prawym przyciskiem myszy na folder

layout->New->Layout Resourse File



Utwórz nowy plik układu "**calculator.xml**"



Skopiuj zawartość z pliku:
https://raw.githubusercontent.com/rmaciaszczyk/SummerSchool_Lab1/main/app/src/main/res/layout/calculator.xml

Co-funded by the
Erasmus+ Programme
of the European Union

Stwórz nowy plik styles.xml w katalogu values i skopiuje zawartość z poniższego źródła
https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/styles.xml

Podmień plik colors.xml w katalogu values i skopiuje zawartość z poniższego źródła
https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/colors.xml

Pobierz Ikonę Backspace 24 px i wgraj do katalogu drawable
https://github.com/rmaciaszczyk/SummerSchool_Lab1/blob/main/app/src/main/res/drawable/ic_baseline_backspace_24.xml
Ikonę możesz za importować z wykorzystaniem VectorAsset Studio
https://developer.android.com/studio/write/vector-asset-studio#svg

2. Zapoznaj się z plikiem ***calculator.xml***. Przeanalizuj jego strukturę

Ćwiczenie 3

a) Jakie i ile obiektów typu **ViewGroup** zostało użytych
b) Jakie i ile obiektów typu **View** zostało wykorzystanych

3. Kolejnym krokiem jest stworzenie kodu kalkulatora.

Przypisanie układu

a) Przejdź do pliku ***MainActivity.kt***
b) Wewnątrz ***onCreate()*** zmień

```
setContentView(R.layout.activity_main)
```

activity_main.xml -> ***calculator.xml***

b) Uruchom aplikację

4. Definicja obiektów

Teraz należy zdefiniować wszystkie przyciski, do których będziemy przypisywać akcje.

W Kotlinie wszystkie zmienne muszą być zainicjalizowane lub musisz jawnie określić, że mogą przyjąć wartość ***null***. Możliwe jest również określenie, że zostaną one zainicjalizowane później przed pierwszym użyciem(lateinit).

Na przykład:

```
//Regular initialization means non-null by default
private var myName: String = "Erasmus"
//To allow nulls, you can declare a variable as a nullable string by writing String?:
private var myNameNullable: String? = null
//You should be very sure that your lateinit variable will be initialized before
accessing
private lateinit var lateMyName: String
```

Zmienne lokalne tylko do odczytu definiuje się za pomocą słowa kluczowego ***val***. Można im przypisać wartość tylko raz. Zmienne, które mogą być ponownie przypisane, używają słowa kluczowego ***var.***

a) Definicja obiektów w kodzie
Przejdźmy do zdefiniowania obiektów w klasie

```
class MainActivity : AppCompatActivity() {
private var one: TextView? =null
private lateinit var two:TextView
private var three:TextView? = null
private lateinit var four:TextView
private var five:TextView? = null
private var six:TextView? = null
private var seven:TextView? = null
private var eight:TextView? = null
private var nine:TextView? = null
private var zero:TextView? = null
private var div:TextView? = null
private var multi:TextView? = null
private var sub:TextView? = null
private var plus:TextView? = null
private var dot:TextView? = null
private var equals:TextView? = null
private lateinit var display:TextView
private var clear:TextView? = null
private var backDelete: ImageButton? = null
```

Jeśli nazwy klas są podkreślone to należy dodać import.
Można również użyć kombinacji klawiszy **ALT + ENTER** i automatycznie dodać import

5. Powiązanie obiektów w układzie z obiektami w kodzie

   Teraz musimy powiązać układ formularza obiektowego z kodem formularza obiektowego. Robimy to w ***onCreate()***

```
one = findViewById(R.id.one)
two = findViewById(R.id.two)
three = findViewById(R.id.three)
four = findViewById(R.id.four)
five = findViewById(R.id.five)
six = findViewById(R.id.six)
seven = findViewById(R.id.seven)
eight = findViewById(R.id.eight)
nine = findViewById(R.id.nine)
zero = findViewById(R.id.zero)
div = findViewById(R.id.div)
multi = findViewById(R.id.multi)
sub = findViewById(R.id.sub)
equals = findViewById(R.id.equals)
display = findViewById(R.id.display)
clear = findViewById(R.id.clear)
backDelete = findViewById(R.id.backDelete)
```

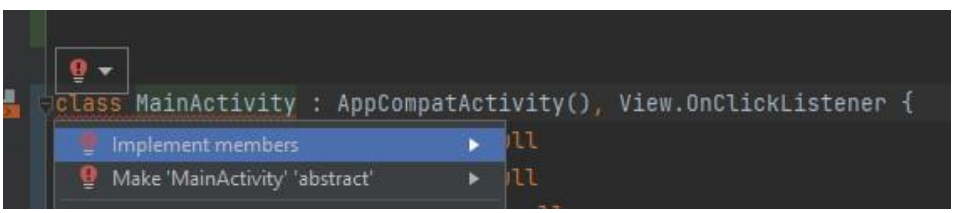
6. Dodanie zdarzenia ***OnClick()*** do przycisku
   Jest kilka sposobów na obsługę zdarzenia, w tym przypadku zdefiniujemy globalny dla danej klasy listener zdarzenia OnClick poprzez zdefiniowanie interfejsu.
   a. Dodaj interfejs ***View.OnClickListener*** do definicji klasy

```
class MainActivity : AppCompatActivity(), View.OnClickListener {.
```

   b. Zdefiniuj metodę calback-ową listenera wewnątrz klasy

Dodanie interfejsu wymusza implementację jego metod. Klikamy na podkreśloną nazwę klasy **MainActivity** (podkreślenie na czerwono oznacza błąd), pojawia się czerwona żarówka, po jej wybraniu mamy możliwość skorzystania z szybkich akcji. W tym przypadku wybieramy **Implement members**

Możesz również użyć kombinacji klawiszy **ALT + ENTER,** aby wywołać menu podręczne



Po akcji otrzymujemy:

```
override fun onClick(p0: View?) { TODO("Not yet implemented")
}
```

Co-funded by the
Erasmus+ Programme
of the European Union

c. Przypisanie listenerów do przycisków (wewnątrz **onCreate()** poniżej **setContentView()**)

```
one?.setOnClickListener(this)
two.setOnClickListener(this)
three?.setOnClickListener(this)
four.setOnClickListener(this)
five?.setOnClickListener(this)
six?.setOnClickListener(this)
seven?.setOnClickListener(this)
eight?.setOnClickListener(this)
nine?.setOnClickListener(this)
zero?.setOnClickListener(this)
div?.setOnClickListener(this)
multi?.setOnClickListener(this)
div?.setOnClickListener(this)
multi?.setOnClickListener(this)
sub?.setOnClickListener(this)
plus?.setOnClickListener(this)
dot?.setOnClickListener(this)
equals?.setOnClickListener(this)
display?.setOnClickListener(this)
clear?.setOnClickListener(this)
backDelete?.setOnClickListener(this)
```

d. Dokończ definiowanie listenera

Co-funded by the
Erasmus+ Programme
of the European Union

```
override fun onClick(p0: View?) {
    if (isError) {
        display.text=""
        isError=false
    }

    when  (p0?.id){
        R.id.one-> display.append("1")
        R.id.two-> display.append("2")
        R.id.three-> display.append("3")
        R.id.four-> display.append("4")
        R.id.five-> display.append("5")
        R.id.six-> display.append("6")
        R.id.seven-> display.append("7")
        R.id.eight-> display.append("8")
        R.id.nine-> display.append("9")
        R.id.zero-> display.append("0")
        R.id.div-> display.append("/")
        R.id.multi-> display.append("*")
        R.id.sub-> display.append("-")
        R.id.plus-> display.append("+")
        R.id.dot-> display.append(".")
        R.id.clear-> display.text=""
        R.id.equals-> evaluateExpression(display.text.toString())
        R.id.backDelete -> {
            display.text =
              if((display.text.length -1 )>=0)
                  display.text.subSequence(0,display.text.length -1)
              else display.text
              }
    }
}
```

Powyższy kod zawiera zmienną, która nie została jeszcze zdefiniowana (**isError**). Służy ona do określenia, czy dane wyrażenie jest błędne. Musi być zadeklarowana globalnie.

```
private var isError: Boolean = true
```

Obliczenia będą wykonywane z wykorzystaniem biblioteki **exp4j** - https://github.com/fasseg/exp4j
Służy ona do obliczania wyrażeń matematycznych opisanych jako ciąg znaków. W powyższym kodzie obliczenia zostaną wykonane jeśli użytkownik wybierze przycisk "=" (R.id.equals), wtedy wartość wprowadzonego ciągu znaków zostanie przekazana do metody ***evaluateExpression()***, korzysta ona ze wspomnianej biblioteki.

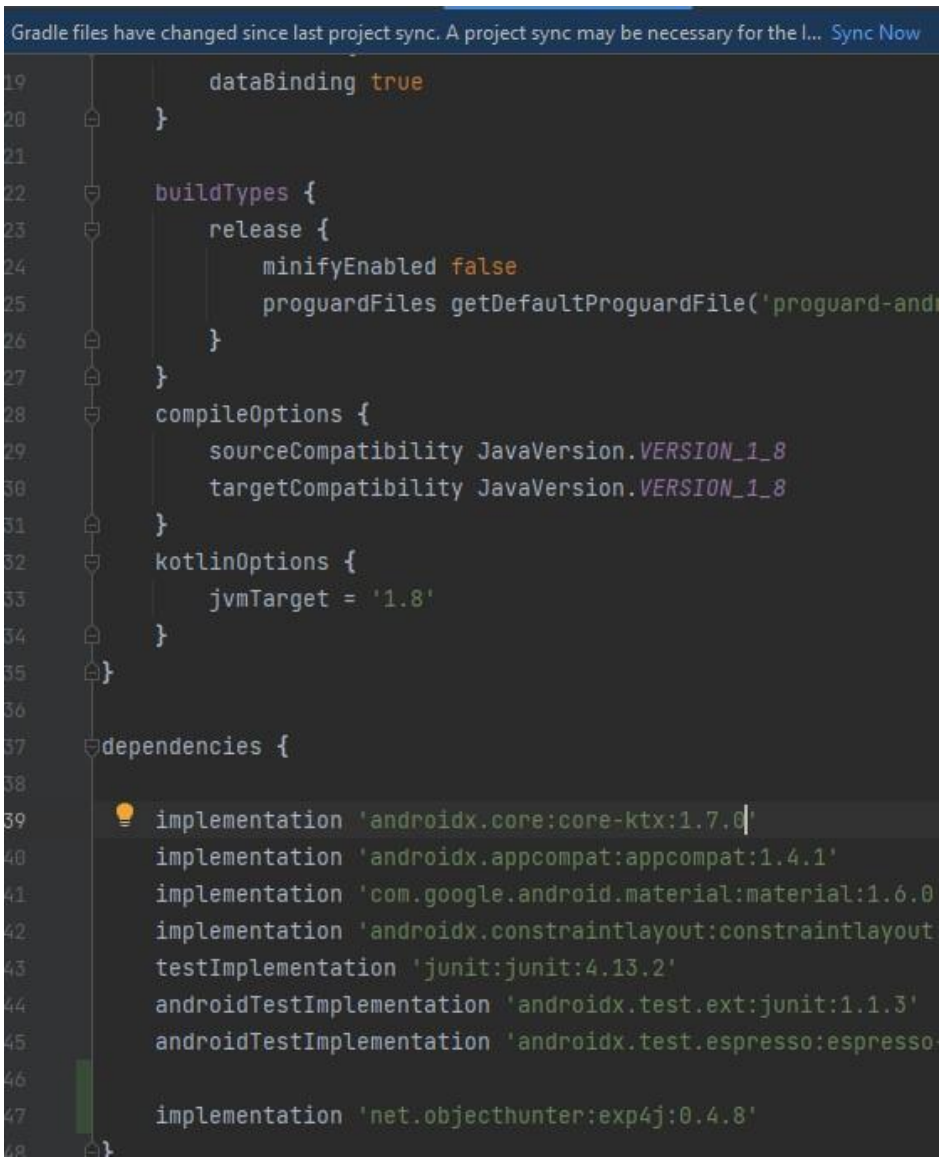
Co-funded by the
Erasmus+ Programme
of the European Union

7. Definiowanie metody **evaluateExpression(),**
   a) Dodawanie biblioteki do aplikacji

Przejdź do **build.gradle(app)** i w sekcji zależności dodaj

```
implementacion 'net.objecthunter:exp4j:0.4.8'
```

po czym należy zsynchronizować projekt. Naciśnij: ***Sync Now***



   b) Dodaj import

Co-funded by the
Erasmus+ Programme
of the European Union

```
import net.objecthunter.exp4j.ExpressionBuilder
```

c) Stwórz metodę ***evaluateExpression()*** w pliku MainActivity.kt

```
private fun evaluateExpression(inputString: String) {
    val expression = ExpressionBuilder(inputString).build()
    try {
        // Calculate the result and display
        val result = expression.evaluate()
        display.text = result.toString()
        //lastDot = true // Result contains a dot
    } catch (ex: Exception)
    {
        when(ex) {
            is IllegalArgumentException, is ArithmeticException -> {
                display.text = "Error"
                isError = true
            }
            else -> throw ex
        }

    }

}
```

8. Uruchom aplikację

View Binding

„View Binding" jest funkcją pozwalającą na łatwiejsze pisanie kodu, który współdziała z widokami. Kiedy wiązanie widoków jest włączone w module, generuje ono klasę wiązania dla każdego pliku XML layoutu obecnego w tym module. Instancja klasy wiążącej zawiera bezpośrednie odniesienia do wszystkich widoków, które mają ID w odpowiednim układzie.

W większości przypadków wiązanie widoku zastępuje ***findViewById().***

1. Włączanie powiązania widoku

Aby włączyć wiązanie widoku w module, ustaw opcję **viewBinding** build na true w pliku build**.gradle** na poziomie modułu, jak pokazano w poniższym przykładzie**:**

```
android {
 . . .
    buildFeatures {
 . . .
        viewBinding true
    }
 . . .
}
```

Co-funded by the
Erasmus+ Programme
of the European Union

2. Zastosowanie

   Jeśli wiązanie widoków jest włączone dla modułu, generowana jest klasa wiązania dla każdego pliku układu XML, który zawiera moduł. Każda klasa wiązania zawiera odniesienia do widoku głównego i wszystkich widoków, które mają ID. Nazwa klasy wiążącej jest generowana przez konwersję nazwy pliku XML na nazwę wg „Pascal Case" oraz dodanie na końcu słowa "Binding". Na przykład: calculator.xml -> wygenerowana klasa wiążąca CalculatorBinding

3. Używanie wiązania widoku w aktywnościach

   ViewBinding pozwala nam zastąpić definicje wszystkich obiektów z układu.

   a) Zastępujemy je jednym obiektem. W naszym kodzie utwórzmy obiekt (**MainActivity.kt**)

```
private lateinit var binding: CalculatorBinding
```

Wszystkie wcześniej zdefiniowane obiekty związane z ekranem powinny zostać usunięte z kodu.

Aby skonfigurować instancję klasy wiążącej do użycia z aktywnością, wykonaj następujące kroki w metodzie **onCreate():**

b) Wywołaj statyczną metodę **inflate()** zawartą w wygenerowanej klasie wiążącej. Tworzy to instancję klasy wiążącej dla aktywności do wykorzystania.

c) Uzyskaj referencję do widoku głównego, wywołując metodę **getRoot()** lub używając składni właściwości Kotlin.

d) Przekaż widok główny do **funkcji setContentView()**, aby uczynić go aktywnym widokiem na ekranie.

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    private var isError: Boolean = true
    private lateinit var binding: CalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = CalculatorBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
```

Co-funded by the
Erasmus+ Programme
of the European Union

e) Możemy teraz usunąć również kod wiążący elementy układu (jest to robione automatycznie przez kompilator). Usuwamy z kodu linię zawierającą wywołanie metody **findViewById()** oraz wszystkie zadeklarowane obiekty

f) Teraz będziemy się odwoływać do poszczególnych obiektów za pomocą obiektu **wiążącego**

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = CalculatorBinding.inflate(layoutInflater)
    val view = binding.root
    setContentView(view)

    binding.one.setOnClickListener(this)
    binding.two.setOnClickListener(this)
. . .
```

g) Odniesienia powinny być zmieniane w całym kodzie

4. Uruchom aplikację
5. Usuń zbędne importy

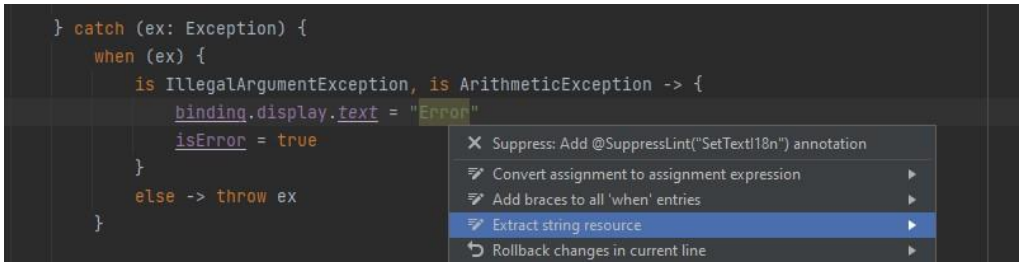
Po wykonaniu tych operacji można usunąć niepotrzebny import, ręcznie lub za pomocą skrótu klawiaturowego. Aby zoptymalizować import w pliku, można również nacisnąć klawisze Ctrl+Alt+Shift+L , wybrać opcję „Optimize imports" i kliknąć przycisk Uruchom. Uwaga: skrót odnosi się do przeformatowania kodu i pozwala również na „Code Cleaup"

6. Usuwanie ostrzeżeń

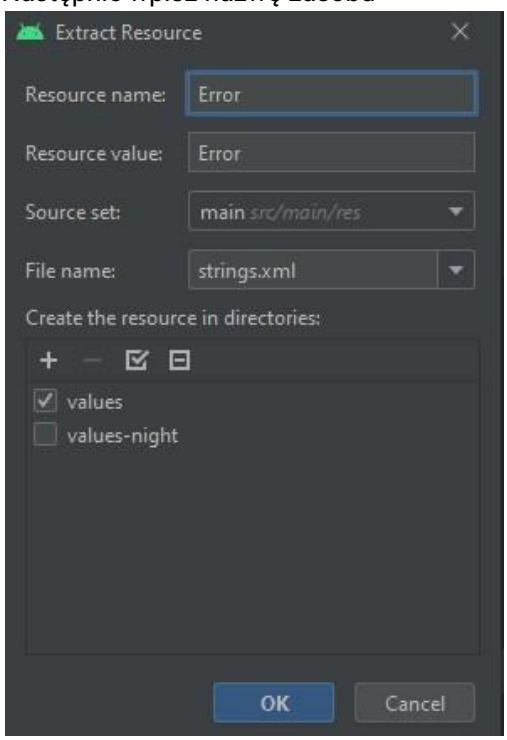
Kompilator analizujący kod każe nam stosować dobre praktyki. Jedną z nich jest umieszczenie wszystkich stringów w dedykowanym pliku zasobów. (**res/values/strings.xml**). Dzięki temu rozwiązaniu możemy bez problemu przetłumaczyć naszą aplikację na inny język. Więcej informacji (https://developer.android.com/training/basics/supportingdevices/languages)

a) Tworzenie stałych za pomocą, podpowiedzi AndroidStudio:
W metodzie evaluateExpression mamy hardcodowany obiekt String - "Error". Najedź na ten obiekt i użyj **ALT+Enter**, aby wybrać **„Extract string resource".**

Co-funded by the
Erasmus+ Programme
of the European Union



Następnie wpisz nazwę zasobu



Kod po zmianach

b) Popraw kod w swojej aplikacji, aby nie było ostrzeżeń.

Ćwiczenie 4

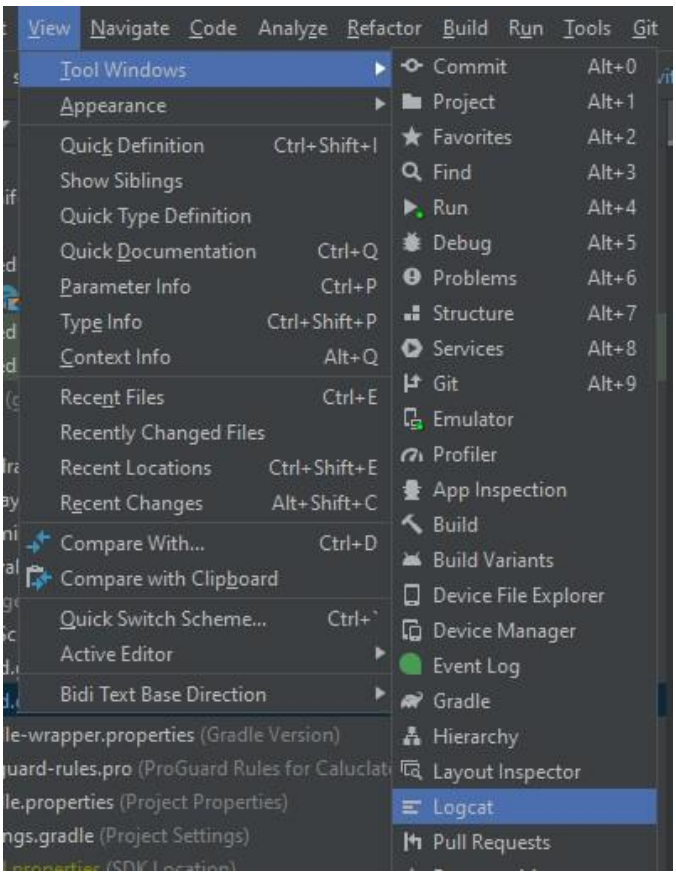
Co-funded by the
Erasmus+ Programme
of the European Union

Proszę o przetestowanie aplikacji. Napraw błąd, gdy użytkownik w aplikacji wykonuje następujące czynności:

a) Naciśnij "2"
b) Naciśnij "5"
c) Naciśnij "/"
d) Naciśnij "="
e) Naciśnij "="

☹ ☹ ☹

Aby zobaczyć, co jest przyczyną błędu, użyj narzędzia Logcat



Kod wynikowy aplikacji znajduje się na https://github.com/matam/Erasmus_Lab2.

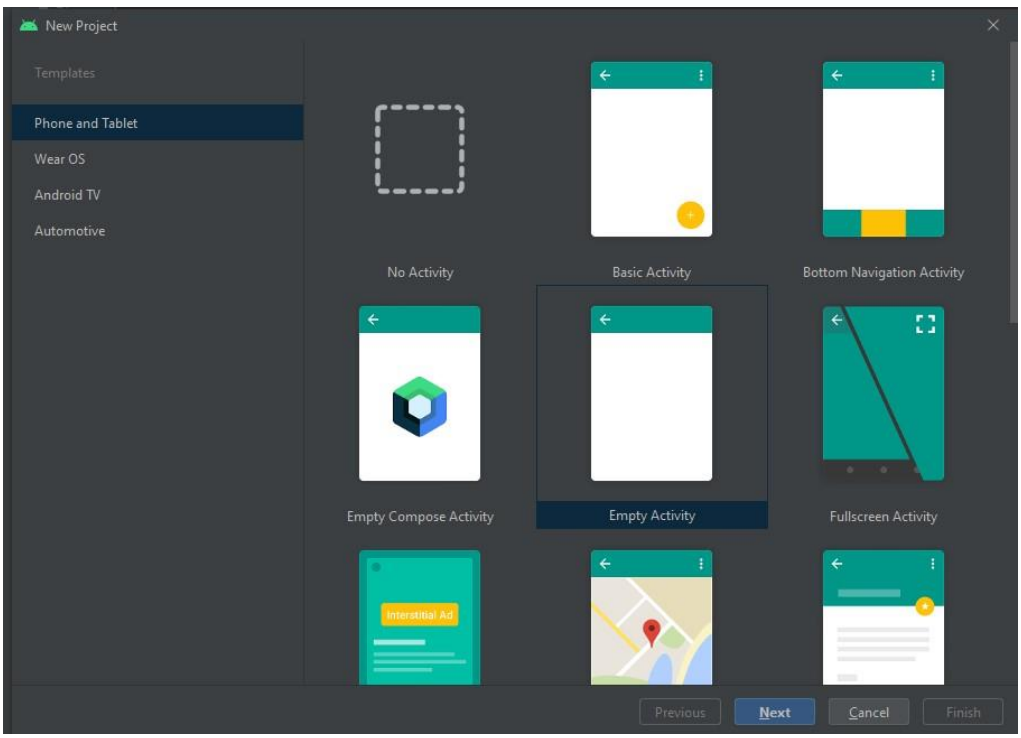
Co-funded by the
Erasmus+ Programme
of the European Union

Sensory

Większość urządzeń mobilnych ma wbudowane czujniki, które mierzą ruch, orientację i różne warunki środowiskowe. Czujniki te są w stanie dostarczyć surowe dane z dużą precyzją i dokładnością. Są przydatne, jeśli chcesz monitorować trójwymiarowy ruch czy pozycjonowanie urządzenia. Pozwalają monitorować zmiany środowiskowe w pobliżu urządzenia. Na przykład gra może śledzić odczyty z czujnika grawitacyjnego urządzenia, aby wnioskować o złożonych gestach i ruchach użytkownika(przechylanie, potrząsanie, obracanie czy kołysanie). Podobnie, aplikacja pogodowa może wykorzystywać czujnik temperatury i czujnik wilgotności urządzenia do obliczania i zgłaszania punktu rosy lub aplikacja podróżnicza może wykorzystywać czujnik pola geomagnetycznego czy akcelerometr do określenia kierunku przemieszczania się.

Podczas tych laboratoriów zostanie stworzona aplikacja do odczytu czujników, a następnie przebudowana tak, aby wykorzystać wzorzec projektowy MVVM.

1. Create a New project -> Empty Activity



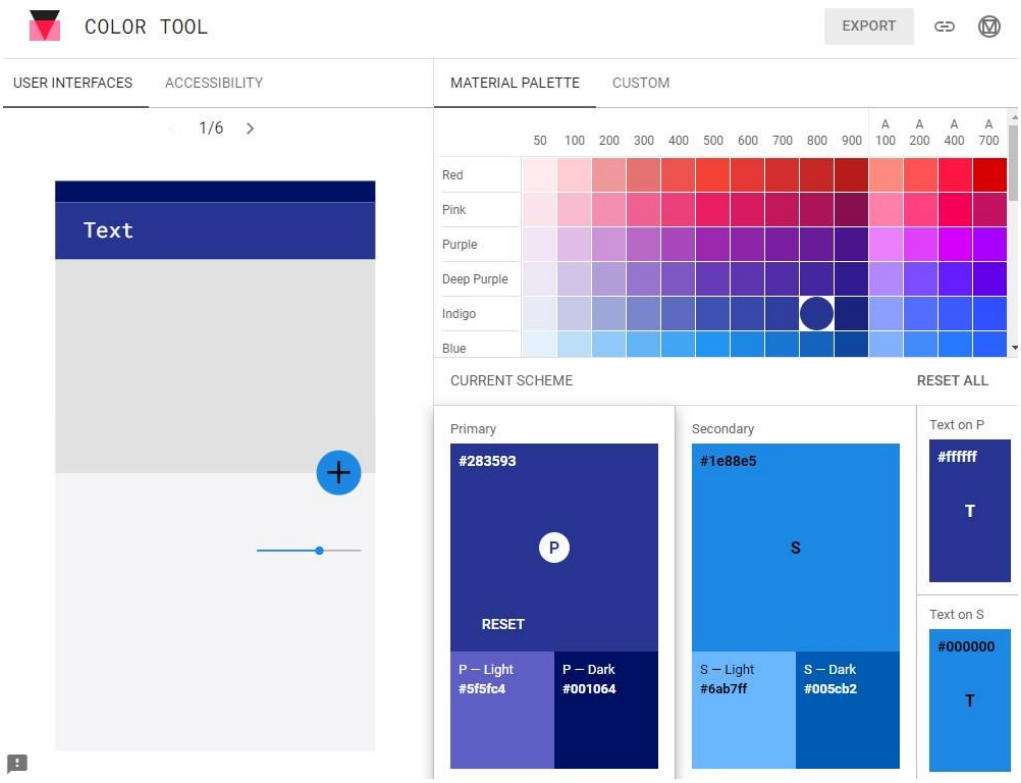
Co-funded by the
Erasmus+ Programme
of the European Union

2. ***Zdefiniuj*** nazwę aplikacji (***Lab5***) i pakiet (***edu.zut.erasmus_plus.sensors***), wybierz minimalne API (***API28***)
3. Zapoznaj się z kodem
    1. Znajdź i przeanalizuj pliki: **MainActivity.kt, activity_main.xml, AndroidManifest.xml**
4. Przejdź do build.gradle -> Uaktualnij wszystkie zależności i biblioteki dla projektu i modułu (możemy pominąć)

5. Uruchom aplikację

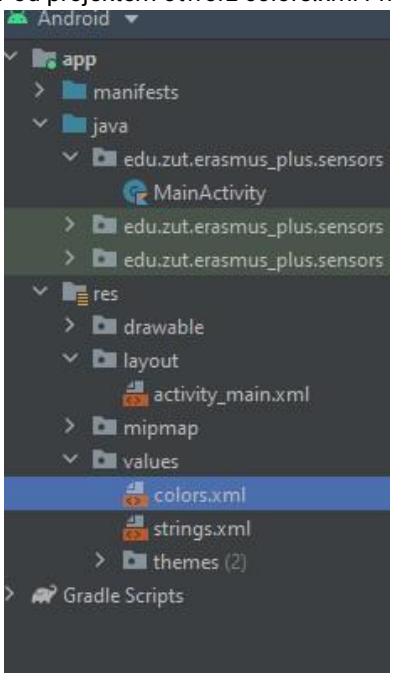
6. Tworzenie schematu kolorów (nieobowiązkowe)

    Przed zdefiniowaniem układu użyj [narzędzia Color Tool - Material Design] i zdefiniuj kolory swojej aplikacji, a następnie użyj tego koloru przy definiowaniu elementów.
    Przykładowy kolor:
    https://material.io/resources/color/#!/?view.left=0&view.right=0&primary.color=283593&secondary.color=1E88E5
    Więcej informacji o kolorach:
    https://material.io/design/color/the-color-system.html#color-theme-creation



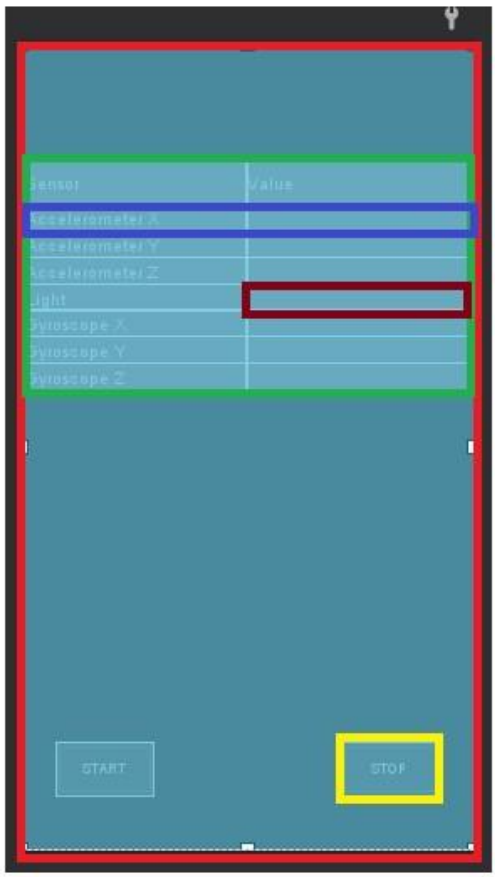
Po wybraniu odpowiedniej kolorystyki przejdź do zakładki DOSTĘPNOŚĆ i sprawdź ostrzeżenia. Kolejnym krokiem jest eksport konfiguracji. W górnej części ekranu wybieramy przycisk EXPORT i zapisujemy colors.xml.

Pod projektem otwórz colors.xml i wstaw wartość z pobranego pliku.



7. Projekt interfejsu graficznego

Proszę zaprojektować układ w taki sposób:



Constraint Layout

Table Layout
Table Row
TextView

Button

Ten interfejs został zbudowany przy użyciu dwóch typów „layout”, ConstraintLayout i TableLayout

Zarys interfejsu – należy dokończyć

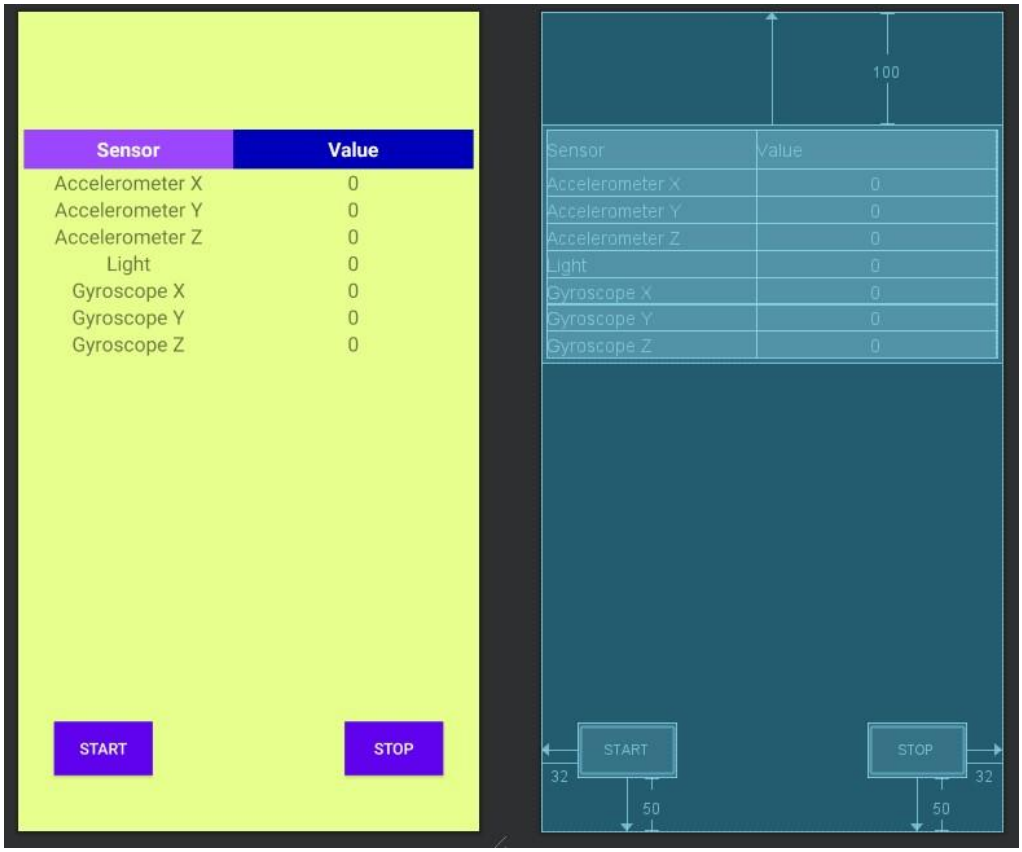
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:background="@color/design_default_color_background"
    tools:context=".MainActivity">

    <TableLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:padding="5dp"
        android:stretchColumns="2"
        app:layout_constraintTop_toTopOf="parent"
        tools:context=".MainActivity"
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
    tools:layout_editor_absoluteX="16dp">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="187dp"
            android:layout_height="match_parent"
            android:layout_column="1"
            android:background="@color/primaryLightColor"
            android:gravity="center"
            android:text="@string/sensor_name"
            android:textColor="@color/primaryTextColor"
            android:textSize="18sp"
            android:textStyle="bold" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="35dp"
            android:layout_column="2"
            android:background="@color/primaryDarkColor"
            android:gravity="center"
            android:text="@string/value_sensor"
            android:textColor="@color/primaryTextColor"
            android:textSize="18sp"
            android:textStyle="bold" />
    </TableRow>
```

Please finish, this is only the beginning

Co-funded by the
Erasmus+ Programme
of the European Union



8. Przejdź do **MainActivity.kt** i wewnątrz **onCreate()** dodaj obiekty **SensorManager** i **Sensor**

```
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT) mGyroscope =
mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
```

Wcześniej należy zdefiniować obiekty jako globalne dla klasy np.

```
private lateinit var mSensorManager : SensorManager private
var mAccelerometer : Sensor ?= null ...
```

9. Dodaj interfejs ***SensorEventListener*** do klasy MainActivity, który pozwoli na odbieranie zdarzeń od czujników.

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```

10. Dodaj metody wywołania zwrotnego ***onAccuracyChanged, onSensorChanged***

```
override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    print("accuracy changed")
}

override fun onSensorChanged(event: SensorEvent?) {
    if (event != null && resume) {
        if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
            findViewById<TextView>(R.id.acc_X).text = event.values[0].toString()
            findViewById<TextView>(R.id.acc_Y).text = event.values[1].toString()
            findViewById<TextView>(R.id.acc_Z).text = event.values[2].toString()
        }

        if (event.sensor.type == Sensor.TYPE_LIGHT) {
            findViewById<TextView>(R.id.light).text = event.values[0].toString()
        }

        if (event.sensor.type == Sensor.TYPE_GYROSCOPE) {
            findViewById<TextView>(R.id.gyro_x).text = event.values[0].toString()
            findViewById<TextView>(R.id.gyro_y).text = event.values[1].toString()
            findViewById<TextView>(R.id.gyro_z).text = event.values[2].toString()
        }
    }
}
```

11. Upewnij się, że nazwy obiektów w kodzie (R.id.XXXX) były zgodne z nazwami ID w układzie.
12. Definiowanie metod pomocniczych

```
private fun registerListener()
{
    this.resume = true

    mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL)
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL)
    mSensorManager.registerListener(this, mGyroscope, SensorManager.SENSOR_DELAY_NORMAL)

    changeButtonStatus()
}
private fun unRegisterListener()
{
    this.resume = false

    mSensorManager.unregisterListener(this)
    changeButtonStatus()
}
private fun changeButtonStatus()
{
    findViewById<Button>(R.id.start_button).isEnabled = !resume
    findViewById<Button>(R.id.stop_button).isEnabled = resume
}
```

Co-funded by the
Erasmus+ Programme
of the European Union

a także dodać zmienną **resume**

```
private var resume = false
```

13. Użyj poprzednich metod, aby zarejestrować i wyrejestrować SensorsListener.

```
override fun onResume() {
    super.onResume()

    registerListener()
}

override fun onPause() {
    super.onPause()

    unRegisterListener()
}
```

Dlaczego rejestrujemy i wyrejestrowujemy się?

14. Dodaj metody ***onClick()*** (zdefiniuj również w layoucie)

```
fun resumeReading(view: View) {
    registerListener()
}
fun pauseReading(view: View) {
    unRegisterListener()
}
```

15. Uruchom aplikację

Dodatkowe zadanie

16. Definiowanie różnych kolorów w zależności od stanu przycisku.

Utwórz background_button.xml i dodaj kod

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@color/primaryColor" android:state_enabled="true" />
    <item android:drawable="@color/secondaryColor" android:state_enabled="false" />
    <!-- default state-->
    <item android:drawable="@color/primaryColor" />
</selector>
```

17. Zmiana tła definicji dla każdego przycisku

```
android:background="@drawable/button_background"
```

18. Uruchom aplikację

Co-funded by the
Erasmus+ Programme
of the European Union

MVVM, LiveData

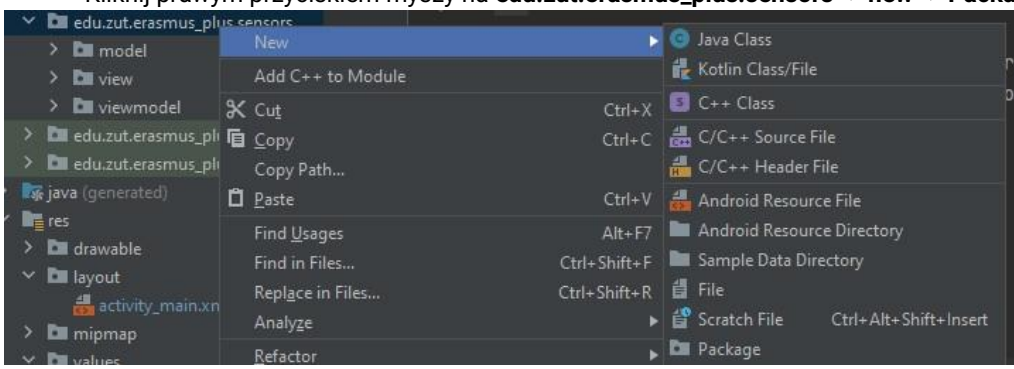
Powyższa aplikacja pokazuje podstawowe podejście do programowania w systemie Android. Obecnie zaleca się, aby aplikacje na Androida były tworzone z wykorzystaniem wzorca projektowego MVVM (Model - View - ViewModel).

Poniżej przedstawiono rozwiązanie wykorzystujące komponenty architektury wprowadzone w bibliotece AndroidX.

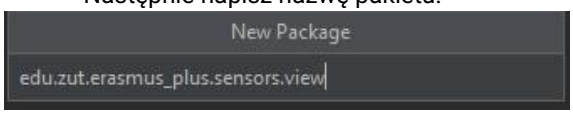
Zamierzamy zmienić aplikację, którą właśnie zbudowaliśmy w aplikację zgodną z wzorcem projektowym MVVM.

1. Przygotowanie pakietów
   - **edu.zut.erasmus_plus.sensors.viewmodel**
   - **edu.zut.erasmus_plus.czujniki.widok**
   - **edu.zut.erasmus_plus.sensors.model**

Kliknij prawym przyciskiem myszy na **edu.zut.erasmus_plus.sensors -> new -> Package**



Następnie napisz nazwę pakietu.



2. Przenieś plik **MainActivity.kt** do **edu.zut.erasmus_plus.sensors.view**

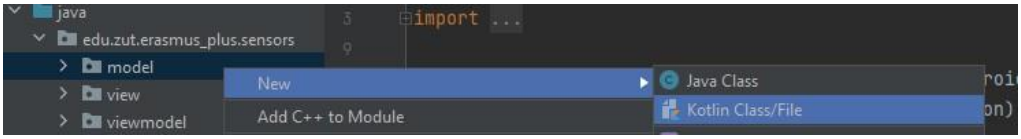
Wszystkie aktywności i fragmenty są klasyfikowane jako widok w modelu MVVM.

3. Tworzenie klasy danych https://kotlinlang.org/docs/data-classes.html

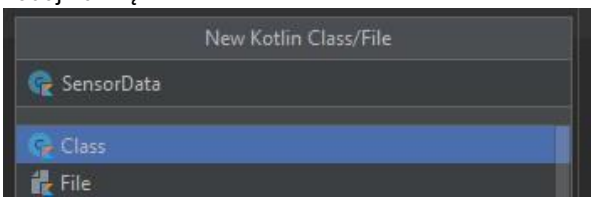
Data class - to klasa, która zawiera tylko pola i surowe metody dostępu do nich (gettery i setery). Są to po prostu kontenery dla danych używanych przez inne klasy. Klasy te nie zawierają żadnej dodatkowej funkcjonalności i nie mogą niezależnie operować na danych, które posiadają.

a) Utwórz plik SensorData.kt wewnątrz **edu.zut.erasmus_plus.sensors.model**

Kliknij prawym przyciskiem myszy na ***model->New-> Kotlin Class/File***



Podaj nazwę



b) Określenie klasy

Wewnątrz SensorData.kt zdefiniuj wszystkie zmienne

```
package edu.zut.erasmus_plus.sensors.model


data class SensorData(
    var accX: Float,
    var accY: Float,
    var accZ: Float,
    var gyroX: Float,
    var gyroY: Float,
    var gyroZ: Float,
    var light: Float
)
```

4. Data Binding.

Mechanizm rozszerzający View Binding i jednocześnie go zastępujący. Pozwala na dwustronne wiązanie.
Szczegółowe informacje: https://developer.android.com/topic/libraries/data-binding

Po utworzeniu interfejsu konieczne jest powiązanie istniejących obiektów z kodem. Obecnie zalecanym podejściem jest wykorzystanie Data Binding. Oprócz automatycznego tworzenia kodu pozwala ono również na aktualizację powiązanego widoku gdy dane ulegną zmianie przy użyciu **LiveData** (wzorzec Obserwator). Biblioteka Data Binding została zbudowana z myślą o obserwowalności. Wzorcu, który stał się dość popularny w rozwoju aplikacji mobilnych. Obserwowalność jest uzupełnieniem wiązania danych, którego podstawowa koncepcja uwzględnia jedynie widok i obiekty danych. Jednak to właśnie dzięki temu wzorcowi dane mogą automatycznie propagować swoje zmiany do widoku. Eliminuje to konieczność ręcznego aktualizowania widoków za każdym razem, gdy dostępne są nowe dane, co upraszcza i zmniejsza liczbę linii kodu.

Nasza aplikacja wykorzysta utworzoną wcześniej klasę data do rozgłaszania informacji o zmianach w czujnikach

5. Włączanie wiązania danych

Otwórz plik build.gradle aplikacji i dodaj tę linię wewnątrz tagu android.

```
android {
    compileSdk 31

    dataBinding {
        enabled true
    }
```

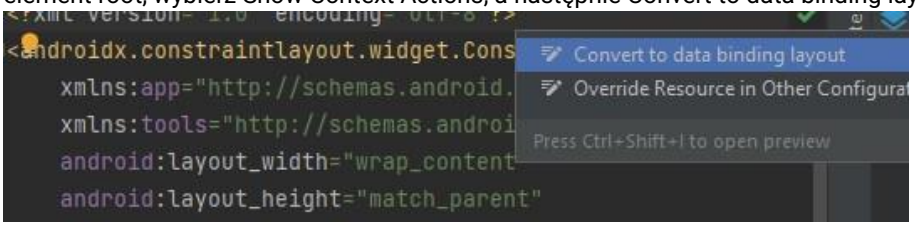
Powiązanie danych w widoku

Aby przekonwertować zwykły układ na układ z wykorzystaniem Data Binding:

1. Opakuj swój układ graficzny w tag <layout>
2. Dodaj zmienne układu (opcjonalnie)
3. Dodaj wyrażenia układu (opcjonalnie)

Albo

Android Studio oferuje automatyczną konwersję: Kliknij prawym przyciskiem myszy na element root, wybierz Show Context Actions, a następnie Convert to data binding layout:




```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:background="@color/cardview_shadow_start_color"
        tools:context=".view.MainActivity">
```

Znacznik **<data>** będzie zawierał zmienne układu. Będziemy tam później dodawać wartości.

Zmienne układu są używane do pisania **wyrażeń układu**. Wyrażenia układu są umieszczane w wartościach atrybutów elementów i używają formatu *@{wyrażenie}*. Poniżej zawarto przykładowe wyrażenia (proszę nie wpisywać do laboratorium):

Co-funded by the
Erasmus+ Programme
of the European Union

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
android:transitionName='@{"image_" + id}'

// Bind the name property of the viewmodel to the text attribute
android:text="@{viewmodel.name}"
// Bind the nameVisible property of the viewmodel to the visibility attrib-
ute
android:visibility="@{viewmodel.nameVisible}"
// Call the onLike() method on the viewmodel when the View is clicked.
android:onClick="@{() -> viewmodel.onLike()}">
```

Więcej informacji na temat wyrażenia layout
https://developer.android.com/topic/libraries/databinding/expressions#expression_language

6. Dodaj zależności do LiveData w pliku build.gradle(app)

```
dependencies {
    //ViewModel
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
    implementation 'androidx.activity:activity-ktx:1.4.0'
    //Lifecycle
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
```

7. Utwórz klasę do odczytu wartości z czujników

Teraz przeniesiemy cały kod odpowiedzialny za zbieranie wartości dla czujników do osobnej klasy

a) Utwórz plik **SensorDataLiveData.kt** wewnątrz ***edu.zut.erasmus_plus.sensors.model***
b) Przenieś kod z Activity_Main.kt (wszystkie funkcje oprócz **onCreate()**)
c) Usuń zbędne obiekty z klasy oraz kod odpowiedzialny za tworzenie instancji Sensor Managera i sensrów.

Po wykonaniu tych czynności MainActivity.kt wygląda tak:

```
package edu.zut.erasmus_plus.sensors

import ...
class MainActivity : AppCompatActivity(){

    override fun onCreate(savedInstanceState: Bundle?) {
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Praktycznie cały istniejący kod został usunięty.

Klasa **SensorDataLiveData** wygląda następująco:

```
class SensorDataLiveData(
    context: Context,
    private val sensorDelay: Int = SensorManager.SENSOR_DELAY_UI
) : LiveData<SensorData>(), SensorEventListener {

    private val mSensorManager: SensorManager =
        context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
    private val accelerometer: Sensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    private val gyroscope: Sensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
    private val light: Sensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)

    private val mAccelerometerReading = FloatArray(3)
    private val mGyroscopeReading = FloatArray(3)
    private val mLightReading = FloatArray(1)

    override fun onActive() {
        super.onActive()
        registerListeners()
    }
    override fun onInactive() {
        super.onInactive()
        unregisterListeners()
    }
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {}

    override fun onSensorChanged(event: SensorEvent) {
        if (event.sensor == accelerometer) {
            System.arraycopy(event.values, 0, mAccelerometerReading, 0,
mAccelerometerReading.size)
        } else if (event.sensor == gyroscope) {
            System.arraycopy(event.values, 0, mGyroscopeReading, 0,
mGyroscopeReading.size)
        } else if (event.sensor == light) {
            System.arraycopy(event.values, 0, mLightReading, 0,
mLightReading.size)
        }

        value = SensorData(
            mAccelerometerReading[0], mAccelerometerReading[1],
mAccelerometerReading[2],
            mGyroscopeReading[0], mGyroscopeReading[1],
mGyroscopeReading[2],
            mLightReading[0]
        )
    }

    fun unregisterListeners() {
        mSensorManager.unregisterListener(this)
    }

    fun registerListeners() {
        mSensorManager.registerListener(
            this,
            accelerometer,
            SensorManager.SENSOR_DELAY_NORMAL,
            sensorDelay
        )
        mSensorManager.registerListener(
```

Teraz ta klasa jest odpowiedzialna za obsługę czujników. Naszym celem jest, aby każda zmiana była przekazywana za pomocą obserwatorów. Do tego celu wykorzystamy klasę **LiveData**. Nasza klasa ją rozszerza i dzięki temu będzie obserwować zmiany z wykorzystaniem modelu **SensorData**. Definicja klasy przyjmuje następującą postać.

```
class SensorDataLiveData(context: Context ) : LiveData<SensorData>(), SensorEventListener
```

Dzięki metodom z klasy LiveData (**onActive()** i **onInactive())** może ona rozpocząć lub zakończyć zbieranie zdarzeń z czujników. Metoda **onActive()** jest uruchamiana, gdy pierwszy obserwator dołączy do naszej klasy, a druga, gdy ostatni się rozłączy.
Reszta kodu jest zgodna z tym, co było wcześniej zaimplementowane w **MainActivity(),** z tym że w metodzie **onSensorChanged()** obiekt ***value*** zwraca wynik. W naszym przypadku jest to obiekt klasy **SensorData** zawierający ostatnie wartości z odczytów czujników.

8. Utwórz ViewModel

Klasa ViewModel jest przeznaczona do przechowywania i zarządzania danymi związanymi z UI w sposób świadomy cyklu życia. Klasa ViewModel pozwala danym przetrwać zmiany konfiguracji, takie jak obracanie ekranu.

Utwórz plik SensorViewModel.kt wewnątrz **edu.zut.erasmus_plus.sensors.viewmodel**

Wewnątrz pliku dodajemy dwie zmienne prywatne i ich metody get.

```
class SensorViewModel(application: Application): AndroidViewModel(applica-
tion) {
  private val _sensor = SensorDataLiveData(application)
  private var _pauseReading = MutableLiveData<Boolean>()

  val sensor: LiveData<SensorData>
    get() = _sensor

  fun getPauseReading(): MutableLiveData<Boolean> {
    return _pauseReading
  }
```

Pierwsza zmienna służy do odczytywania wartości sensorów, z wcześniej stworzonej klasy SensorDataLiveData, gdzie konieczne jest przekazanie kontekstu aplikacji. Stąd modyfikujemy definicję klasy jak powyżej. Proszę zauważyć iż nasza klasa dziedziczy po AndroidViewModel.

Zmienna **_pauseReading** określa, czy zatrzymujemy/rozpoczynamy odczytywanie czujników. Trzeba ją zainicjalizować, więc do SensorViewModel dodajemy następujący kod:

```
init {
    _pauseReading = MutableLiveData(false)
}
```

Ostatnią metodą, którą należy dodać, jest prawidłowa reakcja na zatrzymanie/uruchomienie odczytu czujników. Dodaj następującą funkcję do modelu SensorViewModel:

Co-funded by the
Erasmus+ Programme
of the European Union

```
fun changeButtonStatus()
{
  if(_pauseReading.value==true) _sensor.registerListeners()
  else _sensor.unregisterListeners()
  _pauseReading.value?.let {
    _pauseReading.value = !it
  }
}
```

9. Zmiana układu

Po utworzeniu VM musimy zmienić layout i dodać link do ViewModelu

a) Dodaj informację o zmiennej

Otwórz activity_mail.xml

Wewnątrz właściwości **<data> </data>** wstaw nową zmienną

```
<data>
    <variable
        name="sensorViewModel"
        type="edu.zut.erasmus_plus.sensors.viewmodel.SensorViewModel" />
</data>
```

Dodanie zmiennej (**sensorViewModel**) pozwoli na wykorzystanie obiektów z klasy **SensorViewModel** w pliku interfejsu.

b) Zmień właściwości w activity_main.xml

Znajdź obiekt **TextView** odpowiedzialny za wyświetlanie wartości **acc_X** (okolice linii 64) i zmień właściwość **android:text**.

```
<TextView
    android:id="@+id/acc_X"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_column="2"
    android:text="@{String.valueOf(sensorViewModel.sensor.accX)}"
    android:textAlignment="center"
    android:textSize="18sp" />
```

Zmień właściwość **android:text** dla wszystkich obiektów reprezentujących wartość odczytu z czujników (7 razy)

c) Zmień właściwości ***onClick*** i ***enable*** dla Button w następujący sposób.

Co-funded by the
Erasmus+ Programme
of the European Union

```
<Button
    android:id="@+id/start_button"
. . .
    android:enabled="@{sensorViewModel.pauseReading}"
    android:onClick="@{()->sensorViewModel.changeButtonStatus()}"


. . .
 />

<Button
    android:id="@+id/stop_button"
. . .
    android:enabled="@{!sensorViewModel.pauseReading}"
    android:onClick="@{()->sensorViewModel.changeButtonStatus()}"
. . .
 />
```

Powyżej pokazane są tylko zmienione fragmenty.

10. Zmiany w aktywności- MainActivity.kt
    a) Otwórz MainActivity.kt
    b) Wewnątrz onCreate zmień kod w ten sposób:

```
class MainActivity : AppCompatActivity() {
    private var resume = false

    private lateinit var binding: ActivityMainBinding
    private val sensorViewModel: SensorViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.sensorViewModel = sensorViewModel
        binding.lifecycleOwner = this
    }
}
```

Zauważ, że dodaliśmy dwa obiekty, **binding** i **sensorViewModel**. Obiekt **binding** wykorzystuje **DataBinding** i zapewnia nam automatyczny dostęp do zmiennych w układzie bez konieczności używania **findViewById()**.

Obiekt **sensorViewModel** wiąże utworzoną klasę ViewModel z widokiem.

11. Uruchom aplikację

Powyższe kroki pozwoliły nam na zastosowanie zalecanego obecnie modelu tworzenia aplikacji z wykorzystaniem komponentów Androida zdefiniowanych w bibliotece androidX.

Kod wynikowy aplikacji znajduje się na https://github.com/matam/Erasmus_Lab3-4

Co-funded by the
Erasmus+ Programme
of the European Union

Korzystanie z bazy danych i widoku RecyclerView

Przechowywanie danych jest jedną z najczęściej wykorzystywaną funkcjonalnością w aplikacji. Jeżeli dane posiadają strukturę, wtedy wykorzystywane są relacyjne bazy do przechowywania danych. W środowisku Android baza zaimplementowana jest z wykorzystaniem bazy danych SQLite, jednak jej bezpośrednie zastosowanie jest dość skomplikowane. Biblioteka AndroidX wprowadziła framework Room, który znacznie ułatwił wykorzystanie bazy danych.

Do wyświetlenia danych których liczba elementów jest zmienna wykorzystuje się dynamiczne obiekty typu View, jednym z najczęściej stosowanych jest RecyclerView.

W niniejszym dokumencie zaprezentowano aplikację bazodanową do przechowywania informacji o książkach.

Laboratoria te zawierają następujące elementy:

1. Tworzenie nowej aplikacji i konfigurowanie build.grade
2. Tworzenie modelu danych
3. Stworzenie instancji bazy danych
4. Tworzenie RecyclerView
5. Powiązanie danych z bazy danych z RecyclerView

1. Create New project -> Empty Activity

2. Zdefiniuj App Name (***Lab4***) and nazwę pakietu (***edu.zut.wi.erasmus.lab4***), wybierz minimum API (***API28***)

3. Dodaj zależności do ***build.grade***

```
def room_version = "2.3.0"
def lifecycle_version = "2.3.1"
implementation("androidx.room:room-runtime:$room_version")
annotationProcessor "androidx.room:room-compiler:$room_version"
// To use Kotlin annotation processing tool (kapt)
kapt("androidx.room:room-compiler:$room_version")
// To use Kotlin Symbolic Processing (KSP)
//ksp("androidx.room:room-compiler:$room_version")
// optional - Kotlin Extensions and Coroutines support for Room
implementation "androidx.lifecycle:lifecycle-livedata-
ktx:$lifecycle_version"
implementation("androidx.room:room-ktx:$room_version")
// optional - RxJava2 support for Room
implementation "androidx.room:room-rxjava2:$room_version"
// optional - RxJava3 support for Room
implementation "androidx.room:room-rxjava3:$room_version"
// optional - Guava support for Room, including Optional and
ListenableFuture
implementation "androidx.room:room-guava:$room_version"
// optional - Test helpers
testImplementation("androidx.room:room-testing:$room_version")
// optional - Paging 3 Integration
implementation("androidx.room:room-paging:2.4.0-alpha04")
```

oraz plugin

Co-funded by the
Erasmus+ Programme
of the European Union

```
id "org.jetbrains.kotlin.kapt"
```

4. Zdefiniuj nowy pakiet: database(***edu.zut.wi.erasmus.lab4.database***)
5. Dodaj nową klasę do tego pakietu ***Book.kt***

```
package edu.zut.wi.erasmus.lab4.database

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class Book(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "title") val title: String?,
    @ColumnInfo(name = "subtitle") val subtitle: String?,
    @ColumnInfo val publisher: String?
)
```

6. Stwórz Dao –interface – **BookDao.kt**

```
@Dao
interface BookDao {
    @Query("SELECT * FROM book ORDER BY title ASC")
    fun getAlphabetizedBooks(): Flow<List<Book>>

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(book: Book)

    @Query("DELETE FROM book")
    suspend fun deleteAll()
}
```

7. Zaimplementuj Room database – ***BookRoomDatabase.kt***

```
@Database(entities = arrayOf(Book::class), version = 1, exportSchema =
false)
public abstract class BookRoomDatabase: RoomDatabase() {

    abstract fun bookDao(): BookDao

    companion object {
        // Singleton prevents multiple instances of database opening at the
        // same time.
        @Volatile
        private var INSTANCE: BookRoomDatabase? = null

        fun getDatabase(
            context: Context,
            scope: CoroutineScope
        ): BookRoomDatabase {
            // if the INSTANCE is not null, then return it,
            // if it is, then create the database
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    BookRoomDatabase::class.java,
                    "book_database"
                ).addCallback(BookDatabaseCallback(scope))
                 .build()
                INSTANCE = instance
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
                // return instance
                instance
            }
        }
    }
    suspend fun populateDatabase(bookDao: BookDao) {
        // Delete all content here.
        bookDao.deleteAll()

        // Add sample words.
        var book = Book(title = "new title 1",publisher = "publisher
1",subtitle = "subtitle 1")
        bookDao.insert(book)
        book = Book(title = "new title 2",publisher = "publisher
2",subtitle = "subtitle 2")
        bookDao.insert(book)
        book = Book(title = "new title 3",publisher = "publisher
3",subtitle = "subtitle 3")
        bookDao.insert(book)
        book = Book(title = "new title 4",publisher = "publisher
4",subtitle = "subtitle 4")
        bookDao.insert(book)
        book = Book(title = "new title 5",publisher = "publisher
5",subtitle = "subtitle 5")
        bookDao.insert(book)

    }

    private class BookDatabaseCallback(
        private val scope: CoroutineScope
    ) : RoomDatabase.Callback() {

        override fun onCreate(db: SupportSQLiteDatabase) {
            super.onCreate(db)
            INSTANCE?.let { database ->
                scope.launch {
                    database.populateDatabase(database.bookDao())
                }
            }
        }

    }
}
```

8. Zaimplementuj repozytorium – BookRepository.kt

```
class BookRepository(private val bookDao: BookDao) {
    val allBooks: Flow<List<Book>> = bookDao.getAlphabetizedBooks()

    @WorkerThread
    suspend fun insert(book: Book) {
        bookDao.insert(book)
    }
}
```

9. Stórz nowy pakiet: viewmodel(***edu.zut.wi.erasmus.lab4.viewmodel***)
10. Zaimplementuj ViewModel and ViewModelFactory – BookViewModel.kt

```
class BookViewModel(private val repository: BookRepository) : ViewModel()
{
    val allWords: LiveData<List<Book>> = repository.allBooks.asLiveData()
```

```
    fun insert(book: Book) = viewModelScope.launch {
        repository.insert(book)
    }
}

class BookViewModelFactory(private val repository: BookRepository) :
ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(BookViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return BookViewModel(repository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Part II . Implementacja RecyclerView

11. Stwórz nowy layout dla recycler item (recyclerview_item.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/design_default_color_background">

    <TextView
        android:id="@+id/idBook"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:minWidth="100dp"
        style="@style/book_id"
        android:text="TextView" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="11"
        android:orientation="vertical">

        <TextView
            android:id="@+id/title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/book_title"
            android:text="TextView" />

        <TextView
            android:id="@+id/subtitle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/book_subtitle"
            android:text="TextView" />

        <TextView
            android:id="@+id/publisher"
            android:layout_width="match_parent"
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
            android:layout_height="wrap_content"
            style="@style/book_publisher"
            android:text="TextView" />
    </LinearLayout>

</LinearLayout>
```

12. Dodaj do *activity_main_xml* kod dla RecyclerView i FAB

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="0dp"
        android:layout_height="0dp"
        tools:listitem="@layout/recyclerview_item"

        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:src="@drawable/ic_baseline_add_24"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:contentDescription="@string/add_book"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

13. Stwórz nowy pakiet : adapter(***edu.zut.wi.erasmus.lab4.adapter***), stwórz nową klasę BookListAdapter.kt

```
class BookListAdapter : ListAdapter<Book,
BookListAdapter.BookViewHolder>(BooksComparator()) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
BookViewHolder {
        return BookViewHolder.create(parent)
    }

    override fun onBindViewHolder(holder: BookViewHolder, position: Int) {
        val current = getItem(position)

holder.bind(current.uid.toString(),current.title,current.subtitle,current.p
ublisher)


    }
```

```
    class BookViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        private val titleBook: TextView = itemView.findViewById(R.id.title)
        private val subtitleBook: TextView =
itemView.findViewById(R.id.subtitle)
        private val idBook: TextView = itemView.findViewById(R.id.idBook)
        private val publisherBook: TextView =
itemView.findViewById(R.id.publisher)

        fun bind(id: String?,title: String?,subtitle: String?,publisher:
String?) {
            idBook.text = id
            titleBook.text = title
            subtitleBook.text = subtitle
            publisherBook.text = publisher
        }

        companion object {
            fun create(parent: ViewGroup): BookViewHolder {
                val view: View = LayoutInflater.from(parent.context)
                    .inflate(R.layout.recyclerview_item, parent, false)
                return BookViewHolder(view)
            }
        }
    }

    class BooksComparator : DiffUtil.ItemCallback<Book>() {
        override fun areItemsTheSame(oldItem: Book, newItem: Book): Boolean
{
            return oldItem === newItem
        }

        override fun areContentsTheSame(oldItem: Book, newItem: Book):
Boolean {
            return oldItem.title == newItem.title
        }
    }
}
```

14. Dodaj wywołanie adaptera w MainActivity.kt
    Wstaw kod poniżej ***setContentView***

```
val recyclerView = findViewById<RecyclerView>(R.id.recyclerview)
val adapter = BookListAdapter()
recyclerView.adapter = adapter
recyclerView.layoutManager = LinearLayoutManager(this)
```

15. Stwórz instancję ***Application*** oraz w niej instancję bazy danych oraz repozytorium, klasa ***BookApplication.kt***

```
class BooksApplication: Application() {
    val applicationScope = CoroutineScope(SupervisorJob())
    // Using by lazy so the database and the repository are only created
when they're needed
    // rather than when the application starts
    val database by lazy {
BookRoomDatabase.getDatabase(this,applicationScope) }
    val repository by lazy { BookRepository(database.bookDao())}
}
```

16. Zmień ***AndroidManifest.xml*** i dodaj wewnątrz tagu <application … kod poniżej

```
android:name=".BooksApplication"
```

17. Uruchom aplikację
18. Dodaj nową aktywność do dodawania nowych książek– NewBookActivity.kt – stwórz używając kreatora

```
class NewBookActivity : AppCompatActivity() {
    private val bookViewModel: BookViewModel by viewModels {
        BookViewModelFactory((application as BooksApplication).repository)
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_new_book)
        val editTitle = findViewById<EditText>(R.id.edit_title)
        val editSubTitle = findViewById<EditText>(R.id.edit_subtitle)
        val editPublisher = findViewById<EditText>(R.id.edit_publisher)

        val button = findViewById<Button>(R.id.button_save)
        button.setOnClickListener {
            val replayIntent = Intent()
            if(TextUtils.isEmpty(editTitle.text)){
                setResult(Activity.RESULT_CANCELED,replayIntent)
            } else {
                bookViewModel.insert(Book(title =
editTitle.text.toString(),publisher =
editPublisher.text.toString(),subtitle = editSubTitle.text.toString()))
            }
            finish()
        }
    }
}
```

19. Dostosuj layout – activity_new_book.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".NewBookActivity">

    <EditText
        android:id="@+id/edit_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="@dimen/min_height"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_title"
        android:inputType="textAutoComplete"
        android:layout_margin="@dimen/big_padding"
        android:textSize="18sp" />

    <EditText
        android:id="@+id/edit_subtitle"
        android:layout_width="match_parent"
```

Co-funded by the
Erasmus+ Programme
of the European Union

```
        android:layout_height="wrap_content"
        android:minHeight="@dimen/min_height"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_subtitle"
        android:inputType="textAutoComplete"
        android:layout_margin="@dimen/big_padding"
        android:textSize="18sp" />

    <EditText
        android:id="@+id/edit_publisher"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/big_padding"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_publisher"
        android:inputType="textAutoComplete"
        android:minHeight="@dimen/min_height"
        android:textSize="18sp"
        android:autofillHints="@string/hint_publisher" />

    <Button
        android:id="@+id/button_save"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_save"
        android:layout_margin="@dimen/big_padding" />
</LinearLayout>
```

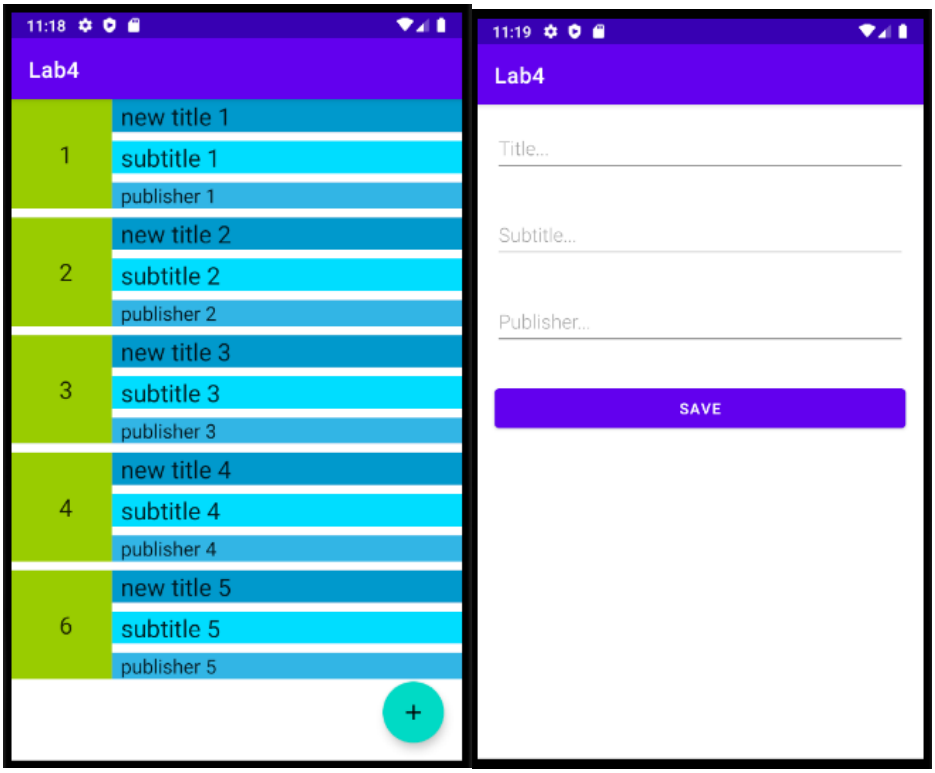
20. Dodaj kod dla FloatingActionButton – MainActivity.kt

```
val fab = findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val intent = Intent(this@MainActivity, NewBookActivity::class.java)
    startActivity(intent)
}
```

21. Uruchom aplikację

Końcowy widok aplikacji

Co-funded by the
Erasmus+ Programme
of the European Union



Main Activity New Book Activity

Dodatkowe informacje: https://developer.android.com/codelabs/android-room-with-a-view-kotlin

Kod wynikowy aplikacji znajduje się na https://github.com/matam/Erasmus_Lab5.

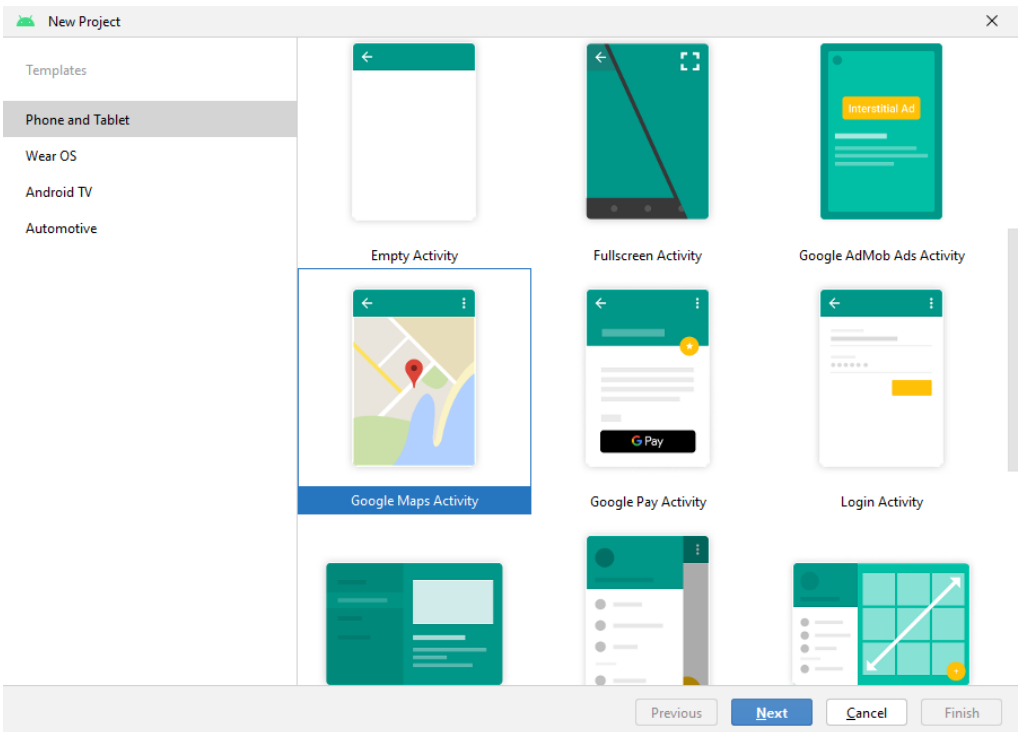
Co-funded by the
Erasmus+ Programme
of the European Union

Lokalizacja

Lokalizacja daje możliwość stworzenia aplikacji uwzględniającej kontekst położenia użytkownika. Pozwala na stworzenie zawartości, która będzie dynamicznie zmienna w zależności od pozycji urządzenia. Lokalizacja pozwala też developerem na zbieranie danych statystycznych, ułatwia profilowanie, jednak jest to pewne zagrożenie prywatności. Dostęp do lokalizacji wymaga nadania uprawnień aplikacji przez użytkownika.

Ten dokument opisuje aplikację, która wyświetla pozycję użytkownika w sposób ciągły lub jednorazowy. Dodatkowo opisany zostanie proces nadawania i weryfikowania uprawnień. Aplikacja korzysta z gotowego projektu wyświetlającego mapę.

1. Utwórz projekt korzystając z kreatora

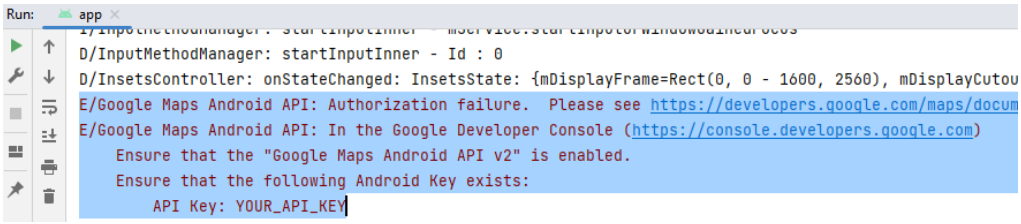


Wybierz aktywność w Mapach Google.

2. Zapoznaj się z kodem. Zauważ jakie interfejsy są zaimplementowane, przeanalizuj stworzone metody
3. Run the application.
   Po uruchomieniu powinna pokazać się mapa z markerem w miejscowości Sydney. Jednak nie pokazuje się z powodu braku prawidłowego klucza API.

W Run mamy informację o braku prawidłowego klucza

Co-funded by the
Erasmus+ Programme
of the European Union



4. Dodanie właściwego klucza do projektu: Dokładna informacją jest pod poniższym adresem

   https://developers.google.com/maps/documentation/android-sdk/get-api-key

   a. Dodaj swój API_KEY into AndroidManifest.xml
   b. Dodaj w konsli dla twojego klucza API_KEY – add Android Maps
5. Przejdziemy teraz do dodania do aplikacji wyświetlania pozycji urządzenia.

6. Do lokalizacji wykorzystamy zalecane przez Google wykorzystanie z Google Play Services

   Jak dodać Google Play services - https://developers.google.com/android/guides/setup

Dodaj zależności do build.gradle (wybierz najnowszą wersję)

```
apply plugin: 'com.android.application'

...

dependencies {
  implementation 'com.google.android.gms:play-services-location:21.0.1'
}
```

7. Zmienimy projekt graficzny **activity_main.xml**, tak aby dodać do niego dwa przyciski.

Co-funded by the
Erasmus+ Programme
of the European Union

```
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <androidx.fragment.app.FragmentContainerView
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        tools:context=".MapsActivity"
        />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btGetLastPosition"
            android:onClick="getLastPos"
            android:text="@string/get_position"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:layout_margin="10dp"/>
        <Button
            android:id="@+id/btContinousPosition"
            android:onClick="startStopRequestLocation"
            android:text="@string/start_loop"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:layout_margin="10dp"/>

    </LinearLayout>
</android.widget.LinearLayout>
```

Popraw błędy, poprzez dodanie odpowiednich definicji do pliku **strings.xml** (get_position-”Get Position”, start_loop -> „Start Loop”), dodaj także nową definicję stop_loop ->”Stop_Loop”

8. Pobierz ostatnio znaną pozycję - https://developer.android.com/training/location/retrieve-current
   Pierwszym krokiem będzie dodanie dodanie kodu, który pozwoli ustalić ostatnio znaną pozycję. Tę funkcję podepniemy pod przycisk „Get Position”, metoda obsługująca będzie nazywała się ***getLastPos()***
   Dodamy tą metodę do głównego pliku aktywności.

Co-funded by the Erasmus+ Programme of the European Union

```
fun getLastPos(view: View)
{
    //checkPermission()
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location : Location? ->
            val myPosition = location?.let {
                LatLng(it.latitude,it.longitude)
            }
            myPosition?.let {

mMap.addMarker(MarkerOptions().position(myPosition).title("My position"))
                mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))
            }
        }

}
```

oraz dodaj definicję obiektu w klasie ***MapsActivity***

```
private lateinit var fusedLocationClient: FusedLocationProviderClient
```

Obiekt należy zaincjalizować. Dodać poniższy kod do metody ***onCreate()***.

```
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
```

Powyższy kod nie zawiera sprawdzania uprawnień. Zgodnie z zasadami przed każdorazowym użyciem funkcjonalności, która musi posiadać pozwolenie od użytkownika konieczne jest sprawdzenie uprawnień.

9. Specify app permission
   Pierwszym krokiem jest dodanie do pliku manifestu informacji jakie uprawnienia są niezbędne do działania aplikacji. W naszym przypadku dodamy uprawnienia zarówno o zgrubnej jak i dokładnej lokalizacji. Zwróć uwagę w której sekcji manifestu dodawane są uprawnienia.

```
<manifest ... >
 <!-- Always include this permission -->
 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

 <!-- Include only if your app benefits from precise location access. -->
 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

10. Teraz można przejść do stworzenia funkcji, która będzie weryfikowała uprawnienia.

Co-funded by the
Erasmus+ Programme
of the European Union

```
private fun checkPermission()
{
    if(ContextCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED)

requestPermissionLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION)
}
private val requestPermissionLauncher =
    registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) {
            isGranted: Boolean ->
        if (isGranted) {
            Log.i("Permission: ", "Granted")
        } else {
            Log.i("Permission: ", "Denied")
        }
    }
```

Podana powyżej metoda jest najprostsza i tylko jednokrotnie sprawdza uprawnienia, w przypadku gdy użytkownik odrzuci na stałe nie obsłuży tej sytuacji. Jako zadanie dodatkowe należy zaimplementować pełen cykl pytania o uprawnienia zgodnie z:
https://developer.android.com/guide/topics/permissions/overview#workflow

11. Teraz można dokonać edycji metody ***getLastPos()*** i od komentować sprawdzanie uprawnień.
12. Uruchom aplikację
13. Dodamy teraz obsługę ciągłej aktualizacji pozycji. Także będziemy korzystać Google Play services. Zadaniem tej funkcji będzie ciągła aktualizacja znacznika pozycji na mapie. W tym celu zostanie stworzona metoda zwrotna, która zadaniem będzie aktualizacja pozycji w przypadku otrzymania nowej pozycji. Należy w ***MapsActivity()*** zdefiniować następujące obiekty, mają być one dostępne dla wszystkich metod w klasie.

```
private var isRequestLoacation: Boolean = false
private lateinit var locationRequest: LocationRequest
private lateinit var locationCallback: LocationCallback
```

14. W metodzie ***onCreate()*** zainicjujemy teraz metody do lokalizacji.
    Pierwszy obiekt, służy do określania parametrów lokalizacji, drugi zaś definiuje metody zwrotne. Zwróć uwagę na częstość aktualizacji.

Co-funded by the
Erasmus+ Programme
of the European Union

```
locationRequest = LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY,
    500)
    .build()

locationCallback = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult) {
        if (locationResult != null) {
            super.onLocationResult(locationResult)
            locationResult.lastLocation?.let {
                val myPosition = it?.let {
                    LatLng(it.latitude,it.longitude)
                }
                myPosition?.let {

mMap.addMarker(MarkerOptions().position(myPosition).title("My position"))

mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))

                }
            }

        }
    }
}
```

15. Teraz możemy wywołać żądanie lokalizacji, zrobimy to w metodzie ***startStopRequestLocation()*** która została stworzona do obsługi drugiego przycisku
    Na początku sprawdzamy uprawnienia, później na podstawie statusu będziemy sprawdzać czy uruchomić zadanie aktualizacji pozycji czy też je zatrzymać. W naszym przypadku jedynie dodajemy punkt do mapy. Poniższy kod można zoptymalizować tworząc wspólne metody dodawania

```
fun startStopRequestLocation(view: View)
{
    checkPermission()
    if (!isRequestLoacation)
    {
        binding.btContinousPosition.text=getString(R.string.stop_loop)
        val addTask=
fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, Looper.myLooper())
        addTask.addOnCompleteListener {task->
            if (task.isSuccessful) {
                Log.d("startStopRequestLocation", "Start loop Location
Callback.")
            } else {
                Log.d("startStopRequestLocation", "Failed start  Location
Callback.")
            }
        }
    }
    else
    {
        binding.btContinousPosition.text=getString(R.string.start_loop)
        val removeTask =
fusedLocationClient.removeLocationUpdates(locationCallback)
        removeTask.addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Log.d("startStopRequestLocation", "Location Callback
removed.")
            } else {
                Log.d("startStopRequestLocation", "Failed to remove
Location Callback.")
            }
        }
    }
    isRequestLoacation=!isRequestLoacation
}
```

16. Uruchom aplikację. Jeżeli wykonujesz aplikację na emulatorze, pozycję możesz zmienić w ustawieniach maszyny wirtualnej.
17. Projekt początkowy jak i końcowy jest umieszczony w repozytorium https://github.com/matam/Erasmus_Lab6

Zadanie:

1. Zrealizować proces nadawania uprawnień zgodnie z zalecanym workflow
2. Na mapie pokazać jedynie 5 ostatnich znaczników.

Co-funded by the Erasmus+ Programme of the European Union

Networking

Podczas tworzenia aplikacji często pojawia się potrzeba korzystania z połączeń sieciowych. Żądania sieciowe są wykorzystywane do pobierania lub aktualizowania danych. Korzystanie z sieci może generować znaczne koszty, a także być szczególnym zagrożeniem dla prywatności użytkownika, dlatego też niezbędne jest zezwolenie na korzystanie z sieci, a wykonywanie operacji sieciowych i odczytywanie stanu sieci w aplikacji, w jej manifeście muszą znaleźć się uprawnienia. Wykonywanie operacji sieciowych wymaga, aby były one wykonywane w osobnym wątku, aby nie obciążać głównego wątku. Przed połączeniem dobrą praktyką jest sprawdzenie, czy urządzenie jest połączone z Internetem. Sprawdzenie sieci musi być wykonane przed każdą operacją pobierania.

Obecnie większość aplikacji wykorzystuje REST API danej usługi do pobierania danych. Czasami jednak konieczne jest pobranie pliku z serwera w tradycyjny sposób. Wówczas preferowany jest protokół HTTP/HTTPS.

W tym laboratorium tworzymy program, który pobiera obrazy za pomocą API NASA (Astronomy Picture of the day)

- https://api.nasa.gov/#browseAPI .

1. Aby zrealizować to laboratorium, musimy wygenerować klucz API - https://api.nasa.gov/#signUp .
   Proszę się zarejestrować i zapisać otrzymane **API_KEY**.

2. Create a New project -> Empty Activity

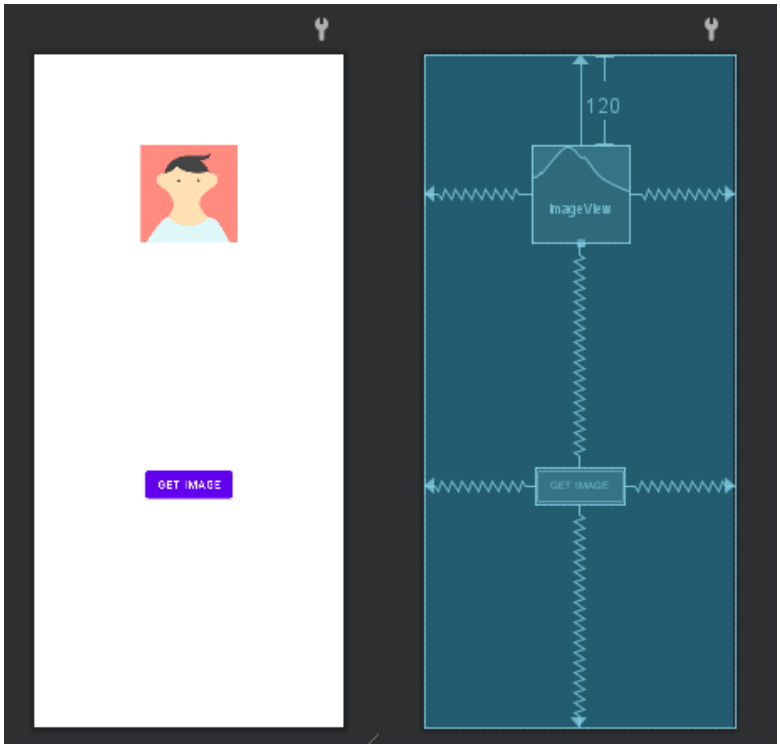
3. Zdefiniuj nazwę projektu (***Lab7***) oraz pakiet (***edu.zut.erasmus_plus.networking***), wybierz API (***API28***)

4. Zapoznaj się z kodem: MainActivity.kt, activity_main.xml, AndroidManifest.xml

5. Zaktualizuj zależności oraz biblioteki -> build.gradle (można ominąć)

6. Zaprojektuj układ w ten sposób:

Co-funded by the
Erasmus+ Programme
of the European Union



Ekran aplikacji będzie zawierał dwa elementy, przycisk do pobrania danych oraz zdjęcie pobrane z internetu.

7. Uruchom aplikację

8. Zdefiniuj uprawnienia AndroidManifest.xml

Korzystanie z internetu wymaga zdefiniowanie odpowiedniego uprawnienia, dodatkowo aby sprawdzać stan sieci również konieczne jest zdefiniowanie uprawnienia.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.zut.erasmus_plus. networking ">
...
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>

    <application>
...
    </application>

</manifest>
```

9. Stwórz metodę do sprawdzania połączenia z internetem.

Co-funded by the
Erasmus+ Programme
of the European Union

Poniższy kod będzie przydatny do sprawdzenia, czy istnieje połączenie internetowe. Proszę użyć tej metody przed pobraniem pliku z internetu.

```
private fun isNetworkConnected(): Boolean {
    val connectivityManager =
getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val activeNetwork = connectivityManager.activeNetwork
    val networkCapabilities =
connectivityManager.getNetworkCapabilities(activeNetwork)
    return networkCapabilities != null &&
networkCapabilities.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERN
ET)
}
```

10. Retrofit

Retrofit to biblioteka Android i Java, która wyróżnia się w pobieraniu i przesyłaniu danych strukturalnych, takich jak JSON i XML. Biblioteka ta wykonuje żądania HTTP za pomocą ***OkHttp***, innej biblioteki od Square. Za pomocą tej biblioteki definiujemy usługę, klasę danych i repozytorium.

Korzystanie z biblioteki polega na zdefiniowaniu danych (klasy danych), serwisu definiującego zapytania (interfejs) oraz metody wywołującej. W pierwszej kolejności zdefiniujemy zależności.

11. Zdefiniuj zależności

```
//Retrofit
implementation 'com.squareup.retrofit2:retrofit:2.9.0'

// JSON Converter
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

12. Zdefiniuj klasę danych

Dane z API będą ustrukturyzowane i zwrócone jako JSON. Utworzenie klasy danych ułatwi korzystanie z tych danych. Opis poszczególnych pól znajduje się pod adresem:

https://github.com/nasa/apod-api

Co-funded by the
Erasmus+ Programme
of the European Union

```
data class  AstronomyPictureDayEntity(
    @SerializedName("copyright")
    val copyright: String?,
    @SerializedName("date")
    val date: String?,
    @SerializedName("explanation")
    val explanation: String?,
    @SerializedName("hdurl")
    val hdurl: String?,
    @SerializedName("media_type")
    val mediaType: String?,
    @SerializedName("service_version")
    val serviceVersion: String?,
    @SerializedName("title")
    val title: String?,
    @SerializedName("url")
    val url: String?
)
```

13. Zdefiniuj serwis

Pobranie zdjęcia będzie przebiegało dwuetapowo, najpierw zostanie pobrany obiekt opisujący zdjęcie dnia wraz z adresem skąd można pobrać obraz. Drugi etap polega na pobraniu zdjęcia. Dlatego serwis zawiera zdefiniowane dwie metody pobierające dane.

```
//Nasa Astrology picture of the day
interface ApodService {
    companion object {
        const val BASE_URL_APOD_ITEM = "https://api.nasa.gov/"
        const val BASE_URL_APOD_IMAGE = "https://apod.nasa.gov/"
        const val API_KEY = "YIm2xWGByBtM12tptMjEGJZqxEIyONsd2hC6h2lB"
        fun getApodItem(): ApodService {
            val retrofit = Retrofit.Builder()
                .addConverterFactory(GsonConverterFactory.create())
                .baseUrl(BASE_URL_APOD_ITEM)
                .build()
            return retrofit.create();
        }
        fun getApodImage(): ApodService {
            val retrofit = Retrofit.Builder()
                .baseUrl(BASE_URL_APOD_IMAGE)
                .build()
            return retrofit.create();
        }

    }
    @GET
    fun downloadImageUrl(@Url fileUrl: String): Call<ResponseBody>

    @GET("planetary/apod")
    fun getApod(@Query("api_key") api_key: String, @Query("date") date:
String ): Call<AstronomyPictureDayEntity>

}
```

14. Użycie biblioteki ***Retrofit***

```
private fun getApodItem(){
    val service = ApodService.getApodItem()
    val serviceRequest = service.getApod(ApodService.API_KEY,formattedDate)
    serviceRequest.enqueue(object : retrofit2.Callback<AstronomyPicture-
DayEntity> {
        override fun onResponse(
            call: retrofit2.Call<AstronomyPictureDayEntity>,
            response: retrofit2.Response<AstronomyPictureDayEntity>
        ) {
            val apod = response.body()
            apod?.url?.let {
                Log.i(MainActivity::class.simpleName,"URL: " + apod.url)
                getApodImage(it)
            }
        }
        override fun onFailure(call: retrofit2.Call<AstronomyPicture-
DayEntity>, t: Throwable) {
            Log.i(MainActivity::class.simpleName, "on FAILURE!!!!")
        }
    })
}

private fun getApodImage(url: String){
    val service = ApodService.getApodImage()
    val serviceRequest = service.downloadImageUrl(url)
    serviceRequest.enqueue(object : retrofit2.Callback<ResponseBody> {
        override fun onResponse(
            call: retrofit2.Call<ResponseBody>,
            response: retrofit2.Response<ResponseBody>
        ) {
            response.body()?.let{readStream(it.byteStream())
                current = LocalDateTime.now().minusDays(clickCounter++)
                formattedDate = current.format(formatter)
                button.setText("Get Image " + formattedDate)
            }
        }

        override fun onFailure(call: retrofit2.Call<ResponseBody>, t: Thro-
wable) {
            Log.i(MainActivity::class.simpleName, "on FAILURE!!!!")
        }

    })
}
```

Wykorzystanie stworzonego service sprowadza się do zdefiniowania własnej instancji serwisy oraz odwołanie do zdefiniowanej metody w tym przypadku ***getApodItem()***

Następnie występuje wywołanie metody ***enqueue()***, a tym samym wysłanie zapytania sieciowego. Wynik zapytanie otrzymujemy w metodach callbacks ***onResponse()*** lub ***onFailure()*** w zależności od wyniku. W powyższym kodzie wsytępują dwie moetody, jedna pobiera metadane zdjęcia a druga pobiera zdjęcie

15. Dodajmy kod od obsługi przycisku oraz do wyświetlenia obrazka (onCreate())

Co-funded by the
Erasmus+ Programme
of the European Union

```
button = findViewById(R.id.button)
button.setText("Get Image " + formattedDate)

imageView = findViewById(R.id.imageView)

button.setOnClickListener {
    if (isNetworkConnected()) {
        getApodItem()
    }
}
```

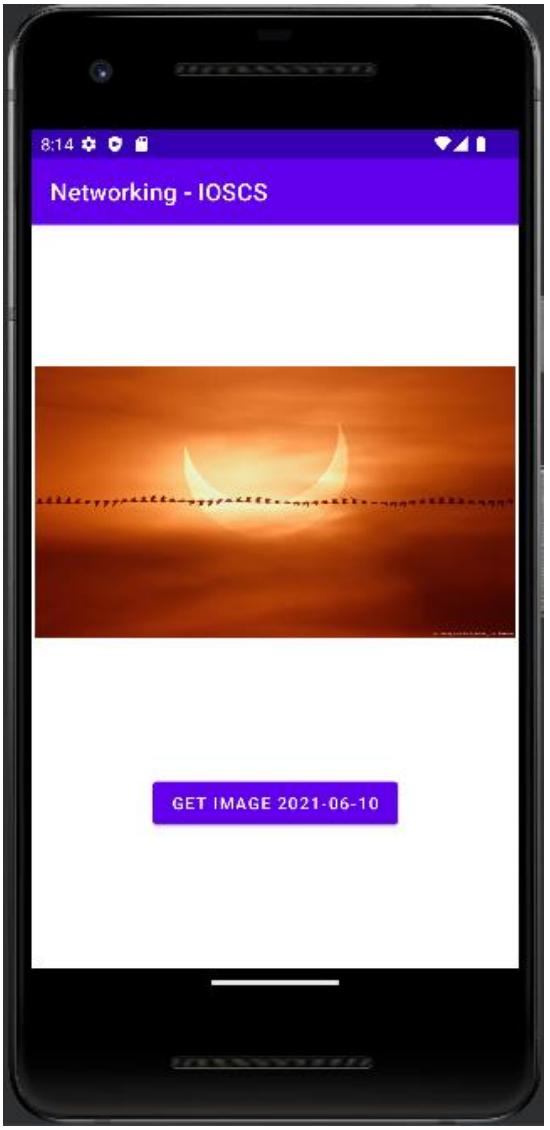
16. Oraz metodę która zamienia otrzymany strumień bitów na obrazek

```
private fun readStream(inputStream: InputStream) {
    val bitmapImage = BitmapFactory.decodeStream(inputStream)

    CoroutineScope(Dispatchers.Main).launch() {
        imageView.setImageBitmap(bitmapImage)}
}
```

17. Uruchom Aplikację

    W wyniku działania aplikacja może wyglądać jak poniżej:

Co-funded by the
Erasmus+ Programme
of the European Union



Pytania:

a) Czy należy pytać użytkownika o zgodę na korzystanie z Internetu?
b) Czy pozwolenia do korzystania z internetu uznane są jako niebezpieczne ?

Kod wynikowy aplikacji znajduje się na https://github.com/matam/Erasmus_Lab7

Co-funded by the
Erasmus+ Programme
of the European Union