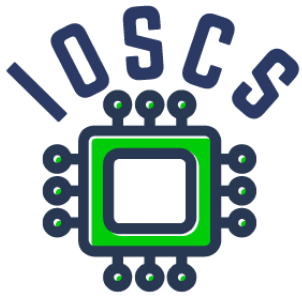


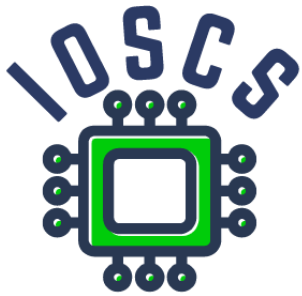
Project: Innovative Open Source Courses for Computer Science

Vývoj mobilných aplikácií
Materiál pre cvičenia

Radosław Maciaszczyk
West Pomeranian University of Technology in Szczecin

30.05.2021

Innovative Open Source Courses for Computer Science



This syllabus 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

Prvá aplikácia – Hello World

To laboratorium zawiera:

- Inštalácia prostredia Android Studio
- Vytvorenie prvého projektu
- Prezentácia prostredia
- Realizácia životného cyklu činnosti

Úloha 1. Stiahnutie a inštalácia aplikácie Android Studio

Zo stránky https://developer.android.com/studio si stiahnite a nainštalujte najnovšiu verziu aplikácie Android Studio

Systémové požiadavky [1]:

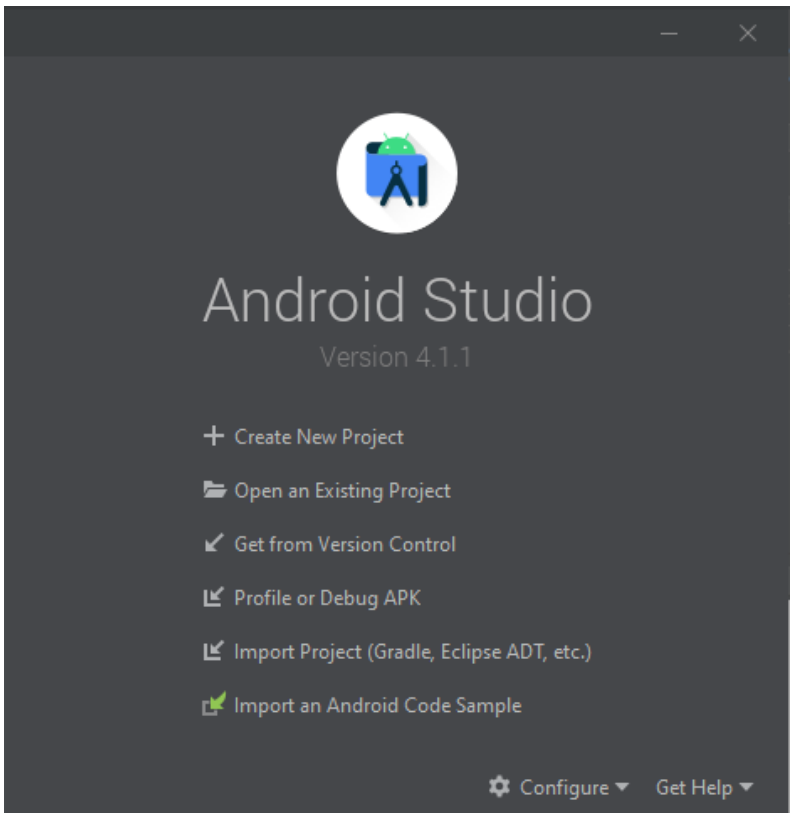
Windows	**Mac**	Lin **Toto laboratórium obsahuje ux**
• 64-bit Microsoft® Windows® 8/10 • x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor • 8 GB RAM or more • 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) • 1280 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

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

Úloha 2 Vytvorenie projektu Hello World

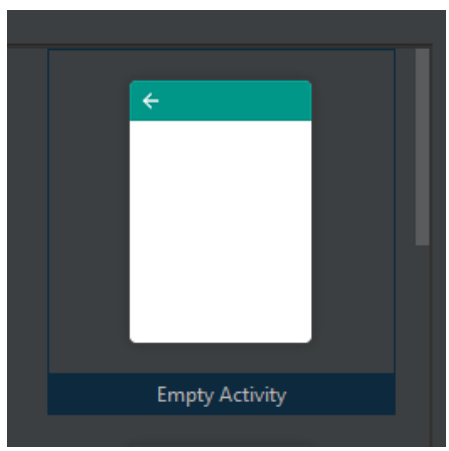
I. Vytvorenie projektu

1. Create New Project

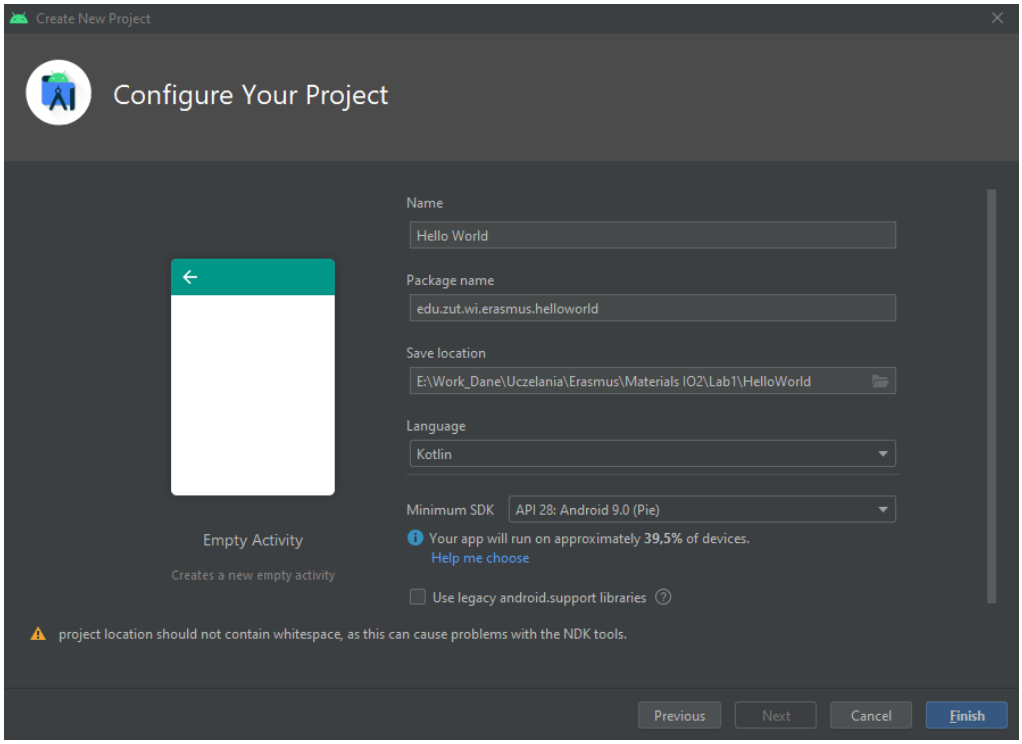


2. Vyberte „Empty Activity" -> next
3. Konfigurácia projektu:
   Konfigurácia:
   - Name
   - Package name
   - Location project
   - Chose Language
   - Choose SDK

   Finish



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



II. Vytvorenie virtuálneho počítača Android Virtual Device

Prostredie Android Studio umožňuje emuláciu zariadení so systémom Android. To umožňuje bezproblémové testovanie vytvorených aplikácií.

1. Vyberte z ponuky: Tools->AVD Manager
2. Ak AVD neexistuje, vytvorte virtuálne zariadenie
    a. Vyberte zariadenie, napr. Pixel 2
    b. Vyberte obraz systému, napr. Android 9
    c. Definujte názov
    d. Prejdite na rozšírené nastavenia a skontrolujte ich
    e. Finish

III. Spustenie projektu

1. Vyberte z ponuky: Run -> Run ‘app’ or Shift + F10

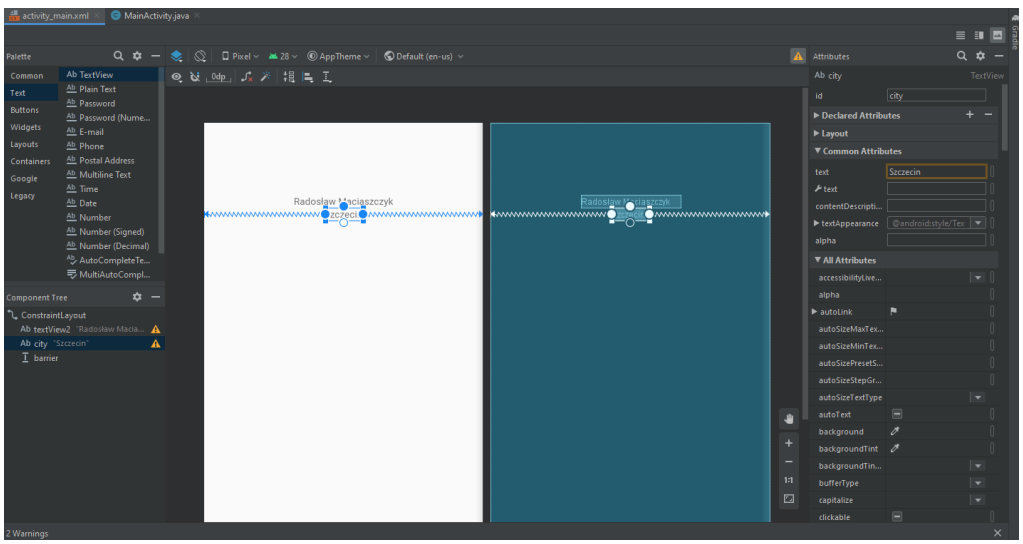
IV. Zmena textu na obrazovke a pridanie nového objektu TextView

1. Vyhľadajte súbor v projekte ***R.layout.activity_main.xml*** i wyświetl (double click)
2. Vyberte text “Hello word” a zmeniť ho
3. Pre TextView zmień **andoriod:id ->** android:id="@+id/name"
4. Pridanie nového widgetu z palety widgetov: TextView
5. Pre TextView definovať: **id** and **text**

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

Úryvok kódu: ***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" />
```



Úloha 3 Životný cyklus činnosti (Acivity)

Táto úloha ukazuje celý životný cyklus aktivity, ako implementovať zámer a ako pridať tlačidlo. Okrem toho ukazuje, ako používať protokolovanie pomocou **logcat**.
Znalost životního cyklu aktivity je klíčem k vytvoření správně fungující aplikace, která dokáže ukládat stavy aplikace mezi jednotlivými spuštěními.

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

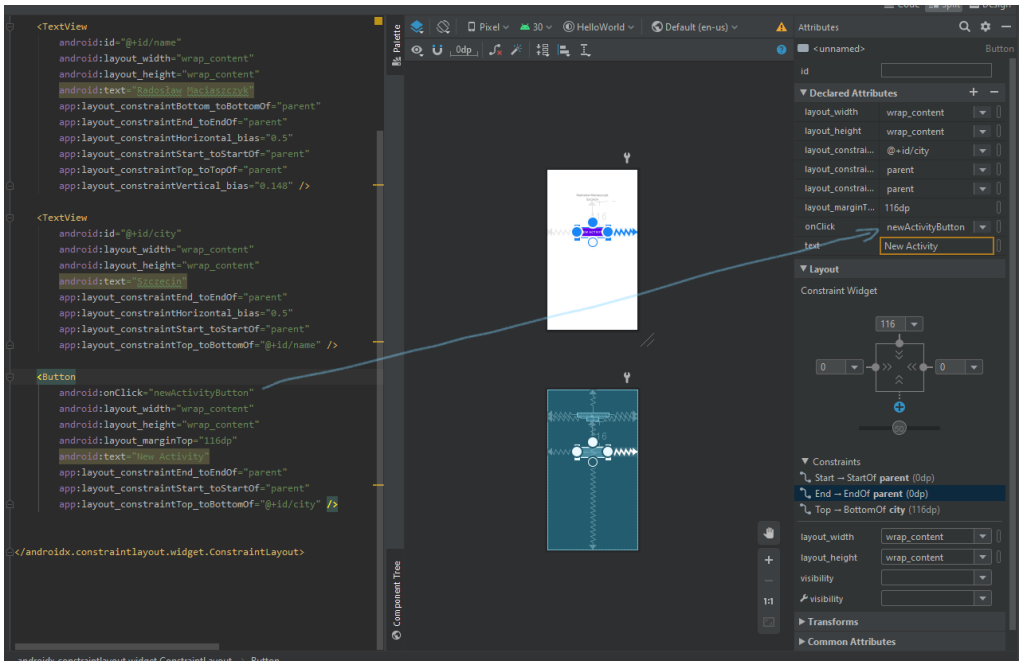
I. Pridanie tlačidla na obrazovku

1. V súbore ***R.layout.activity_main.xml*** pridať tlačidlo z palety widgetov.
2. Definovať pre tlačidlo:
   id: newActivity
   text: New Activity

3. Prejdite na ***MainActivity.kt*** i definovať metódu, ktorá bude spracovávať udalosť onClick.

```
fun newActivityButton(view: View){

}
```

4. Prejdite na ***activity_main.xml*** i pridať metódu **onClick** do tlačidla.



5. Konečná verzia súboru xml pre tlačidlo

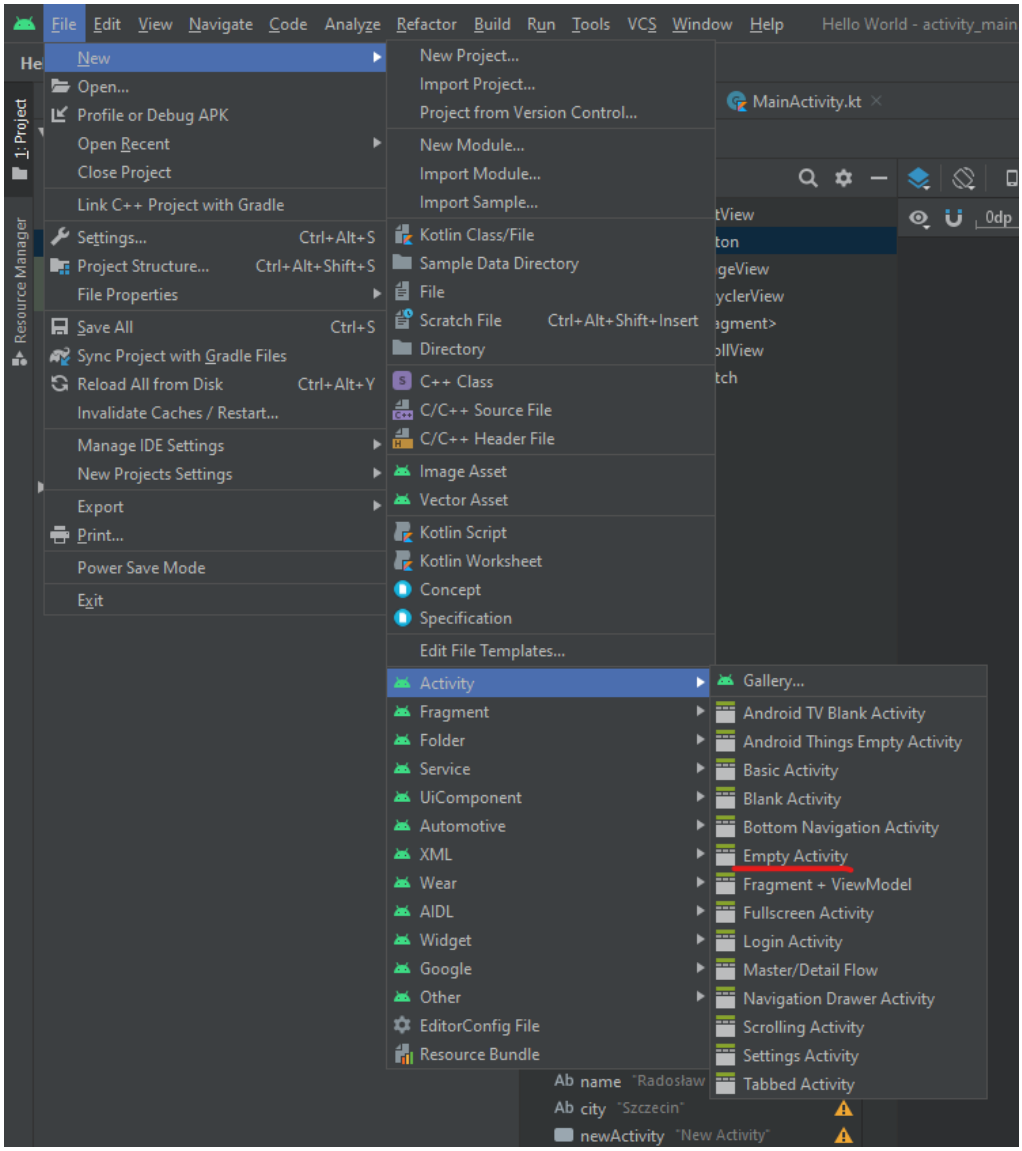
```
<Button
    android:onClick="newActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="116dp"
    android:text="New Activity"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
  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

V. Vytvorenie novej činnosti

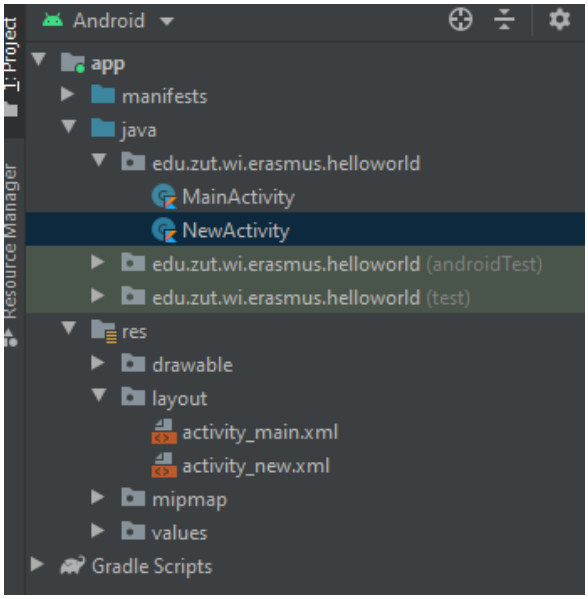
1. V ponuke vyberte položku File-> New -> Activity -> Empty Activity

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

2. Definovať Activity
   Activity name: NewActivity
   Click Finish



Po vytvorení máme nasledujúcu adresárovú štruktúru



VI. Pridanie metód životného cyklu aktivity

1. Použite klávesovú skratku
2. CTRL+O (Show Override Members)
3. Vyberte metódy z životného cyklu ***onPause, onStart, onRestart, onStop, onDestory***

VII. Používanie triedy **Log**

1. Definujte konštantu TAG – (Dobrá prax: TAG má rovnaký názov ako trieda)

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

2. Pridať volanie metódy ***Log*** na všetky metódy životného cyklu

Napr.

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

3. Konečná verzia 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")
    }
```

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

```
    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")
    }
}
```

VIII. Pridanie spracovania návratov pomocou udalosti UP

1. Otvorte súbor AndroidManifest.xml a definujte nadradenú aktivitu

```
<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. Vrátiť sa na MainActivity.kt
    1. Pridajte všetky metódy životného cyklu do tejto aktivity, rovnako ako v predchádzajúcej aktivite.
    2. pridať volanie metódy ***Log***

3. Vnútri metódy, ktorá spracováva udalosť onClick() ***newActivityButton*** pridať

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

Pridajte aj prihlasovacie údaje

4. Konečná verzia 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
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. Spustenie projektu

Pozrite sa na ***logcat*** a sledujte zaznamenané hodnoty.

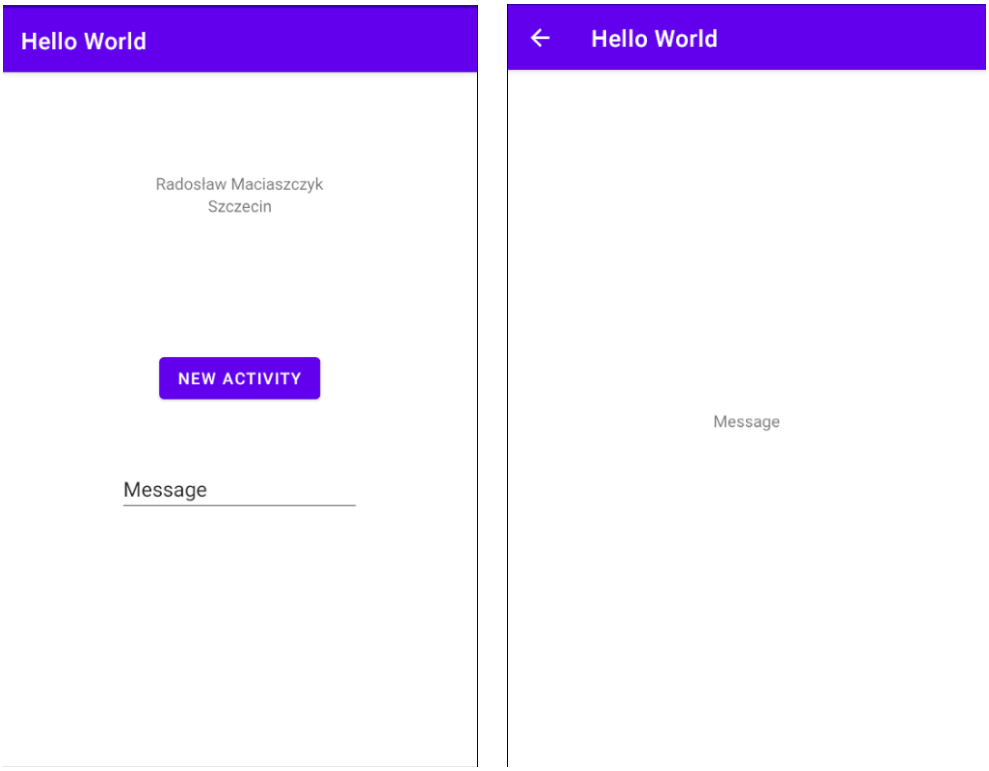
Dodatočná úloha:

K prvej aktivite pridajte *Input Text* a používanie Intent poslať správu inej aktivite a potom ju zobraziť.

Vytvorte pohľad podobný tomu, ktorý je uvedený nižšie:

Prvá činnosť MainActivity odošle správu po stlačení tlačidla na Second Activity

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



Ďalšie informácie o odosielaní údajov pomocou zámeru:
https://developer.android.com/training/basics/firstapp/starting-activity

Výsledný kód aplikácie nájdete na adrese https://github.com/matam/Erasmus_Lab1.

Bibliografia

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

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

Ďalšie informácie:

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

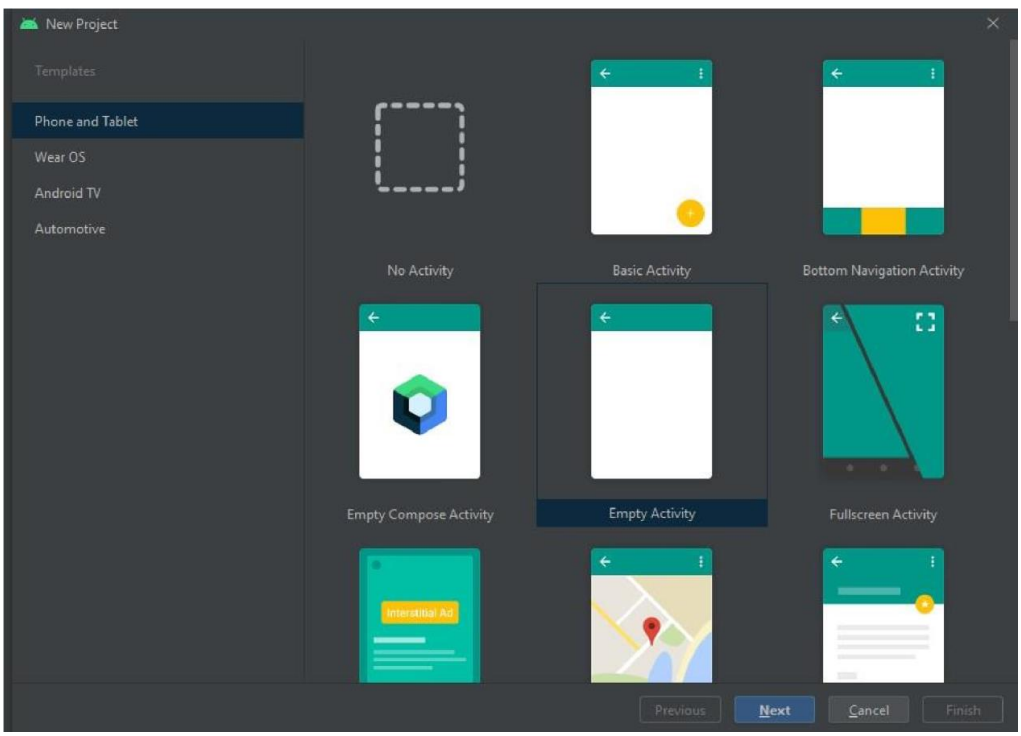
Používateľské rozhranie

Používateľské rozhranie - časť programu, aplikácie alebo operačného systému zodpovedná za komunikáciu s používateľom. Jeho najbežnejšou formou je grafické používateľské rozhranie (GUI). Pozostáva z grafických prvkov, ktoré zjednocujú vzhľad aplikácie a prezentujú ju v rozpoznateľnej a predvídateľnej podobe. Pri návrhu rozhrania je okrem grafickej vrstvy (ikony, ponuky, textové polia, zoznamy) dôležité pamätať na vytvorenie ciest pohybu používateľa, informačnú architektúru a interakčné procesy. Preto je v súčasnosti v procese návrhu aplikácie taká dôležitá kombinácia UI a UX (user experience). V systéme Android predstavuje kompletný systém UI a UX Material Design (https://material.io/), ktorý je v súčasnosti vo verzii 3.

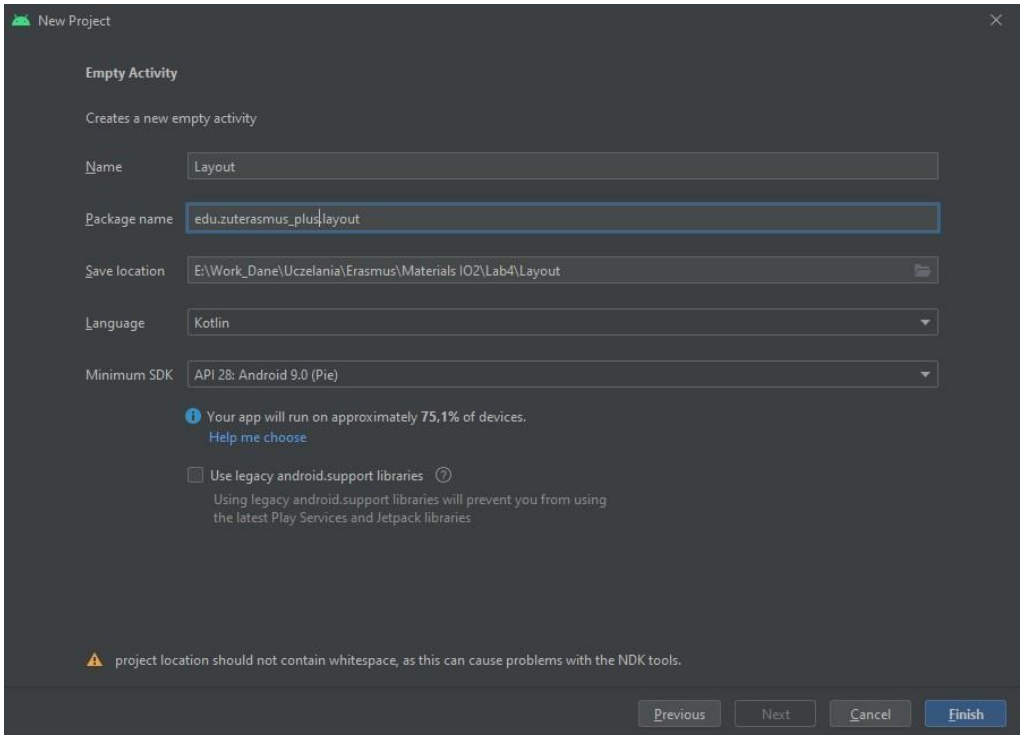
Toto laboratórium je navrhnuté tak, aby ukázalo základy vytvárania rozhraní spolu s úvodom do používania nástrojov, ktoré pomáhajú vytvárať rozhrania. Najprv vytvoríme typickú aplikáciu "Hello World" a potom aplikáciu "Kalkulačka".

Projekt Hello World

1. Spustenie aplikácie Android Studio
2. Vytvorenie nového projektu - vyberte možnosť "Empty Activity"



3. Zadajte názov projektu, názov balíka, vyberte verziu API a cestu.



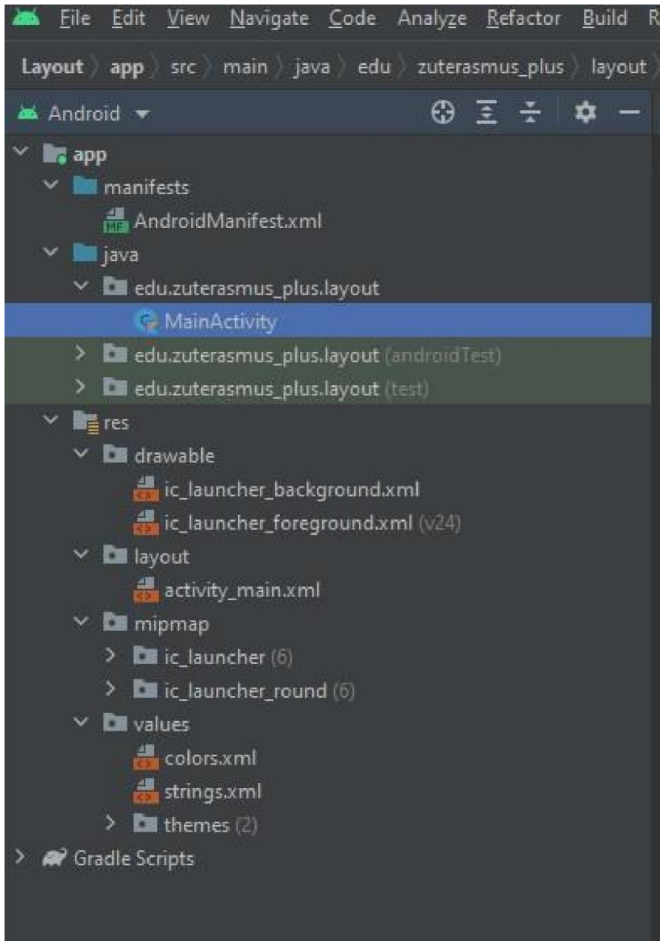
4. Stlačte tlačidlo Dokončiť a počkajte, kým sa vytvorí kostra projektu.

Počiatočný pohľad na projekt je zobrazený nižšie a zobrazená je aj štruktúra projektu.

Kód ukladáme do adresára ***JAVA (***aj pri písaní pomocou jazyka Kotlin). V adresári ***manifest*** je uložený súbor **AndroidManifest.xml, ktorý** obsahuje dôležité informácie pre kompilátor vrátane definícií komponentov aplikácie alebo oprávnení.

Adresár res sa používa na ukladanie informácií o zdrojoch aplikácie vrátane adresára ***layouts***, v ktorom definujeme vzhľad aplikácie v súbore xml.

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

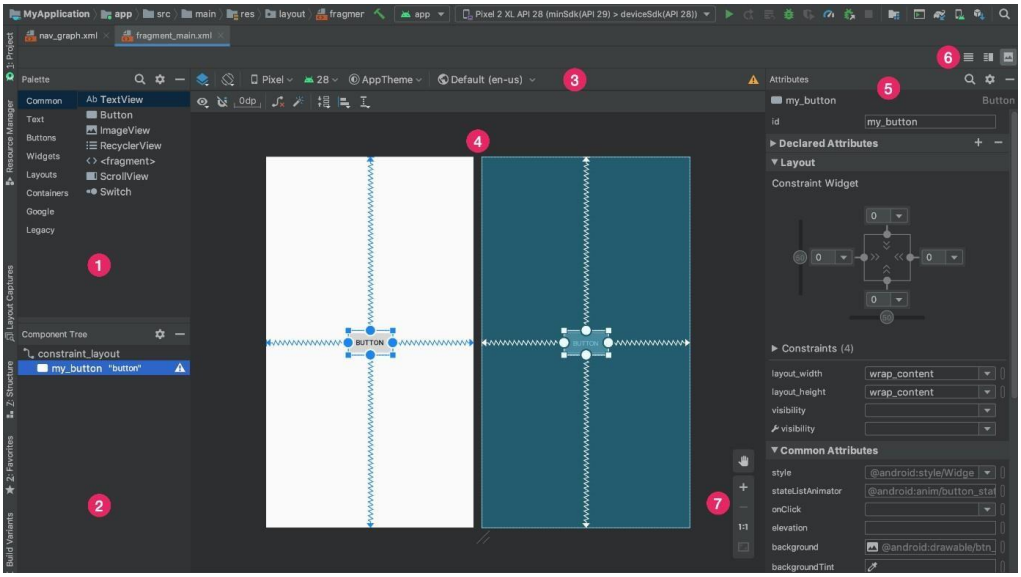


5. Vytvorenie používateľského rozhrania

Rozhranie je vytvorené v súboroch xml. Android Studio umožňuje vytvárať kód priamo alebo pomocou grafického editora.

Vyberte súbor **activity_main.xml** z adresára rozloženia. Otvorí sa nástroj Layout Editor (https://developer.android.com/studio/write/layout-editor):

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

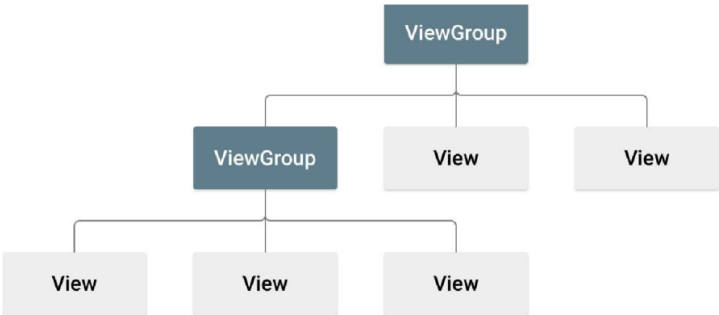


1) **Palette**: Obsahuje rôzne zobrazenia a skupiny zobrazení, ktoré môžete pretiahnuť do rozloženia.
2) **Component Tree:** Zobrazuje hierarchiu komponentov vo vašom rozložení.
3) **Toolbar**: Slúži na konfiguráciu vzhľadu rozloženia v editore, umožňuje meniť atribúty rozloženia.
4) **Design editor**: Upravujte rozloženie v zobrazení Design, Blueprint alebo v oboch.
5) **Attributes**: Ovládacie prvky **atribútov** pre vybrané zobrazenie.
6) **View mode**: Zobrazte svoje rozvrhnutie v ikone režimu kódu, ikone režimu návrhu alebo ikone režimu rozdelenia. Režim Split (Rozdelenie) zobrazuje súčasne okná Code (Kód) aj Design (Návrh).
7) **Zoom and pan controls**: Ovládajte veľkosť a polohu náhľadu v rámci editora.

Viac informácií o rozhraní nájdete na adrese:
https://developer.android.com/studio/write/layout-editor.

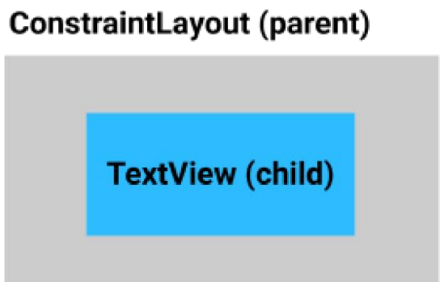
Používateľské rozhranie je hierarchicky vytvorené pomocou objektov **ViewGroup** (rozloženie) a **View** (widget)

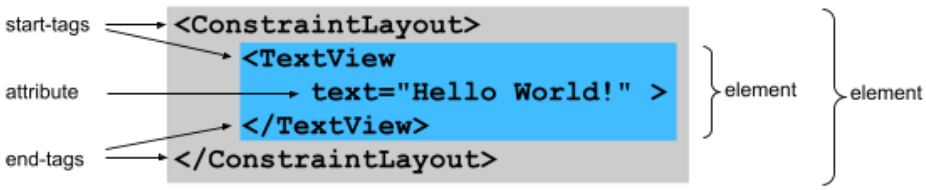
Co-funded by the
Erasmus+ Programme
of the European Union



Ako je vidieť na obrázku vyššie, objekty **ViewGroup** umožňujú vytvárať hierarchie, v ktorých sú obsiahnuté jednotlivé objekty

6. Používateľské rozhranie vyvíjaného projektu je definované takto





Súbor xml s definíciou rozloženia

```
<?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>
```

Na definovanie vzhľadu objektov používame atribúty. Jednotlivé značky xml (napr. TextView) v súbore XML zodpovedajú názvom tried, v ktorých sú tieto objekty definované.

Pozrite sa na značku **ConstraintLayout** a všimnite si, že používa **androidx.constraintlayout.widget.ConstraintLayout** namiesto len **ConstraintLayout** Je to preto, že je súčasťou balíka Android Jetpack. Android Jetpack má ďalšie funkcie, ktoré vám uľahčia vytváranie aplikácie. Spoznáte, že táto zložka používateľského rozhrania je súčasťou systému Jetpack, pretože začína na "**androidx**".

Cvičenie 1

a) Nahraďte "Hello World" svojím menom.
b) Pridajte nový objekt TextView s názvom mesta, z ktorého pochádzate.
c) Vložte všetky titulky do súboru strings.xml (návod - https://developer.android.com/guide/topics/resources/string-resource)

Cvičenie 2 (doplnkové)

Vyplňte cvičenie na tejto strane

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

Projektová kalkulačka

1. Vytvorte nové rozloženie alebo si stiahnite nový dizajn zo stránky

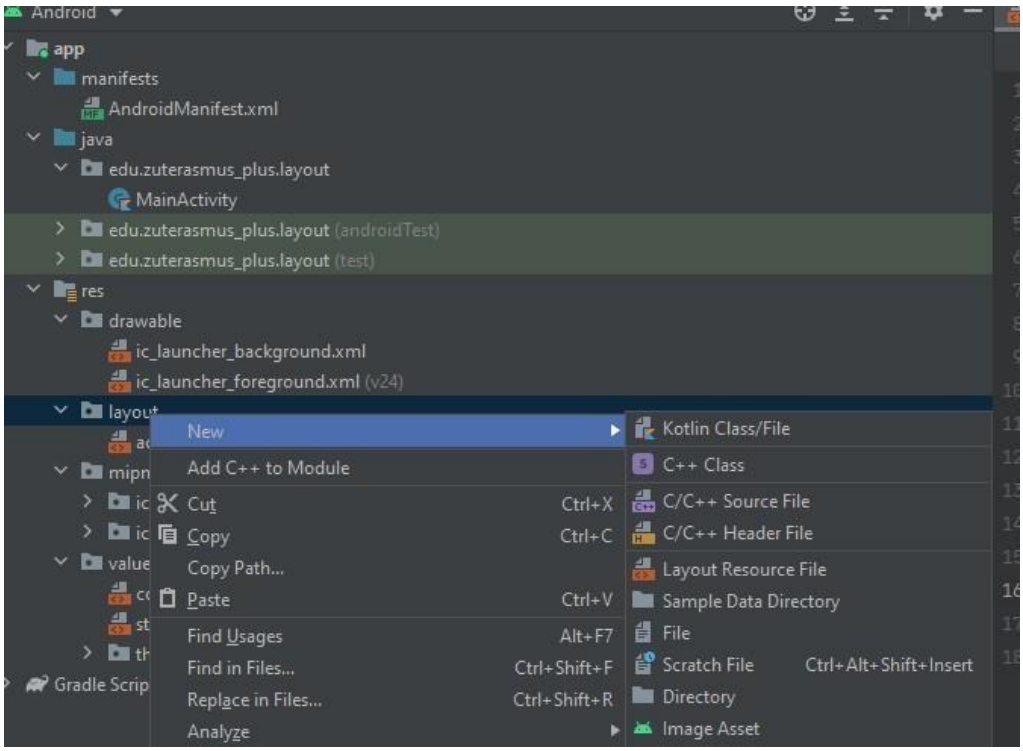
https://github.com/matam/Erasmus_Lab2

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

Ak sťahujete projekt z úložiska, prejdite na bod č. 2

Kliknite pravým tlačidlom myši na priečinok

layout->New->Layout Resourse File



Vytvorenie nového súboru rozloženia "**calculator.xml**"

Kopírovanie obsahu zo súboru:
https://raw.githubusercontent.com/rmaciaszczyk/SummerSchool_Lab1/main/app/src/main/res/layout/calculator.xml

Vytvorte nový súbor styles.xml v adresári values a skopírujte obsah z nasledujúceho zdroja

https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/styles.xml

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

Nahraďte súbor colors.xml v adresári values a skopírujte jeho obsah z nasledujúceho zdroja

https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/colors.xml

Stiahnite si ikonu Backspace 24 px a nahrajte ju do adresára 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

Importovať súbor SVG

https://developer.android.com/studio/write/vector-asset-studio#svg

2. Zoznámte sa so súborom ***calculator.xml***. Analyzujte jeho štruktúru

Cvičenie 3

a) Ktoré a koľko objektov **skupiny ViewGroup** bolo použitých
b) Ktoré a koľko objektov **zobrazenia** sa použilo

3. Ďalším krokom je vytvorenie kódu kalkulačky.

Priradenie rozloženia

a) Prejdite do súboru ***MainActivity.***kt
b) Vnútri ***funkcie onCreate()*** zmeniť

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

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

b) Spustenie aplikácie

4. Definícia objektov

Teraz je potrebné definovať všetky tlačidlá, ktorým budeme priraďovať akcie.

V jazyku Kotlin musia byť všetky premenné inicializované alebo musíte explicitne určiť, že môžu nadobúdať ***nulovú hodnotu***. Je tiež možné určiť, že budú inicializované neskôr pred prvým použitím(lateinit).

Napríklad:

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

```
//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
```

Lokálne premenné určené len na čítanie sa definujú pomocou kľúčového slova ***val***. Hodnotu im možno priradiť iba raz. Premenné, ktoré možno priradiť znova, používajú kľúčové slovo ***var***.

a) Definícia objektov v kóde
Prejdime k definovaniu objektov v triede

```
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
```

Ak sú názvy tried podčiarknuté, mal by sa pridať import.
Môžete tiež použiť kombináciu klávesov **ALT + ENTER** a automaticky pridať import

5. Prepojenie objektov v rozložení s objektmi v kóde

   Teraz musíme prepojiť rozloženie objektového formulára s kódom objektového formulára.

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

```
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)
```

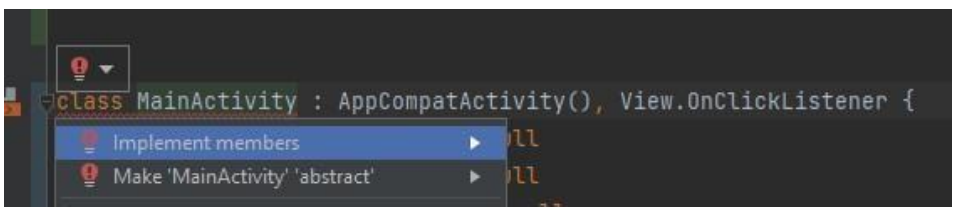
6. Pridanie udalosti ***OnClick() do*** tlačidla
   Existuje niekoľko spôsobov, ako spracovať udalosť, v tomto prípade definujeme globálneho poslucháča udalosti OnClick pre triedu definovaním rozhrania.
   a. Pridanie rozhrania ***View.OnClickListener*** do definície triedy

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

   b. Definovanie metódy spätného volania poslucháča vo vnútri triedy

Pridanie rozhrania si vynúti implementáciu jeho metód. Klikneme na podčiarknutý názov triedy **MainActivity** (podčiarknutie červenou farbou znamená chybu), objaví sa červená žiarovka, po jej výbere máme možnosť použiť rýchle akcie. V tomto prípade vyberieme možnosť **Implementovať** členy

Kombináciu klávesov **ALT + ENTER** môžete použiť aj na vyvolanie kontextového menu



Po akcii sa nám dostáva:

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

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

c. Priradenie poslucháčov tlačidlám (vnútri funkcie **onCreate()** pod **funkciou 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. Dokončenie definovania poslucháča

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
              }
    }
}
```

Vyššie uvedený kód obsahuje premennú, ktorá ešte nebola definovaná (**isError)**. Používa sa na určenie, či je výraz chybný. Musí byť deklarovaná globálne.

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

Výpočty sa budú vykonávať pomocou knižnice **exp4j** - https://github.com/fasseg/exp4j
Používa sa na výpočet matematických výrazov opísaných ako reťazec. Vo vyššie uvedenom kóde sa výpočet vykoná, ak používateľ vyberie tlačidlo '=' (R.id.equals), potom sa hodnota zadaného reťazca odovzdá metóde ***evaluateExpression(),*** tá využíva uvedenú knižnicu.

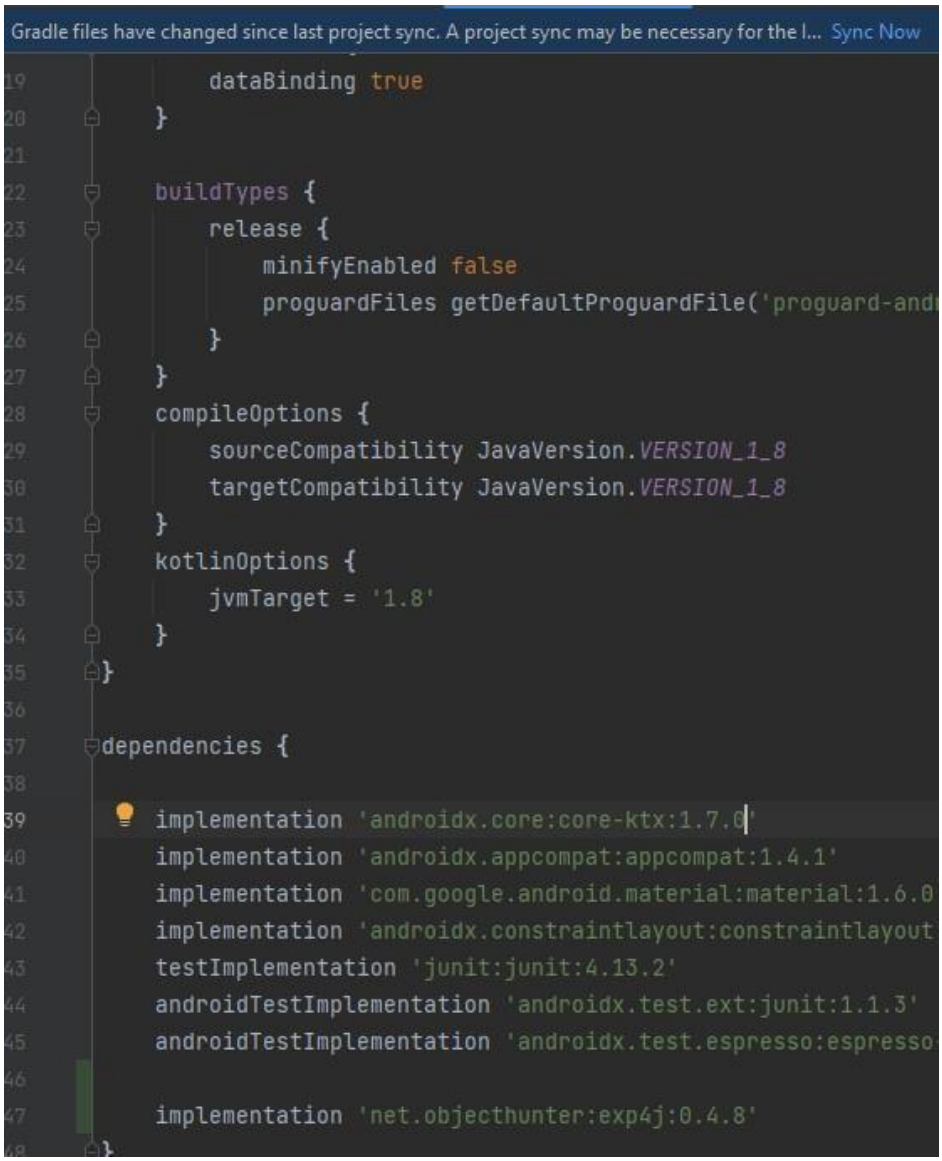
7. Definovanie metódy **evaluateExpression()**,
   a) Pridanie knižnice do aplikácie

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

Prejdite do **build.gradle(app)** a do sekcie závislostí pridajte

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

po ktorej by sa mal projekt synchronizovať. Stlačte tlačidlo: ***Synchronizovať*** teraz



b) Pridať import

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

c) Vytvorenie metódy ***evaluateExpression()*** v súbore MainActivity.kt

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

```
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. Spustenie aplikácie

View Binding

"View Binding" je funkcia, ktorá vám umožňuje jednoduchšie písať kód, ktorý spolupracuje so zobrazeniami. Keď je v module povolená väzba zobrazení, vygeneruje sa trieda väzby pre každý súbor XML s rozložením, ktorý sa v danom module nachádza. Inštancia triedy binding obsahuje priame odkazy na všetky pohľady, ktoré majú ID v príslušnom rozložení.

Vo väčšine prípadov väzba pohľadu nahrádza funkciu ***findViewById()***.

1. Povolenie prepojenia zobrazení

Ak chcete v module povoliť väzbu na **zobrazenie,** nastavte v súbore build.**gradle na** úrovni modulu možnosť build **viewBinding** na hodnotu true, ako je uvedené v nasledujúcom **príklade:**

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

2. Aplikácia

   Ak je pre modul povolená väzba zobrazenia, pre každý súbor rozloženia XML, ktorý obsahuje modul, sa vygeneruje trieda väzby. Každá trieda väzby obsahuje odkazy na hlavné zobrazenie a všetky zobrazenia, ktoré majú ID. Názov triedy väzby sa vygeneruje tak, že sa názov súboru XML prevedie na názov podľa "Pascal Case" a na koniec sa pridá slovo "Binding" (Väzba). Napríklad: calculator.xml -> vygenerovaná trieda CalculatorBinding

3. Používanie väzby zobrazení v aktivitách
   ViewBinding nám umožňuje nahradiť definície všetkých objektov z rozloženia.

   a) Nahradíme ich jedným objektom. V našom kóde vytvoríme objekt (**MainActivity.kt**)

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

Všetky predtým definované objekty súvisiace s obrazovkou by sa mali z kódu odstrániť.

Ak chcete nakonfigurovať inštanciu triedy väzby na použitie s aktivitou, vykonajte nasledujúce kroky v metóde **onCreate():**

b) Volanie statickej metódy **inflate()** obsiahnutej vo vygenerovanej triede väzby. Tým sa vytvorí inštancia triedy väzby, ktorú bude aktivita používať.

c) Získajte odkaz na koreňový pohľad zavolaním metódy **getRoot()** alebo pomocou syntaxe vlastnosti jazyka Kotlin.

d) **Funkcii setContentView()** odovzdajte hlavné zobrazenie, aby sa stalo aktívnym zobrazením na obrazovke.

```
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)
```

e) Teraz môžeme odstrániť aj kód, ktorý viaže prvky rozloženia (to robí kompilátor automaticky). Z kódu odstránime riadok obsahujúci volanie metódy **findViewById()** a všetky deklarované objekty

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

f) Na jednotlivé objekty sa teraz budeme odvolávať pomocou objektu väzby

```
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) Odkazy by sa mali zmeniť v celom kóde

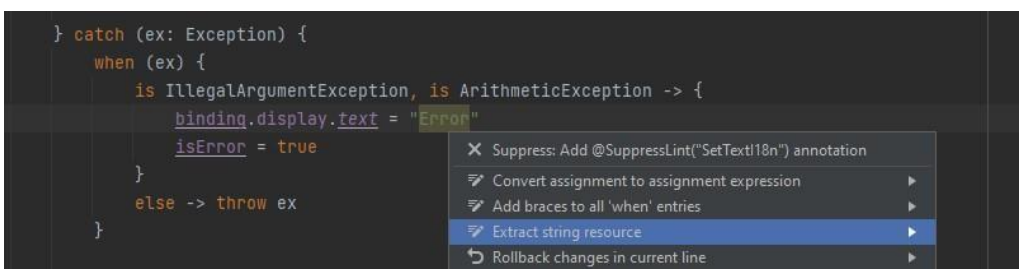
4. Spustenie aplikácie
5. Odstránenie nepotrebných dovozov

Po vykonaní tejto operácie môžete odstrániť nepotrebné importy buď ručne, alebo pomocou klávesovej skratky. Ak chcete optimalizovať importy v súbore, môžete tiež stlačiť klávesovú skratku Ctrl+Alt+Shift+L , vybrať položku "Optimalizovať importy" a kliknúť na tlačidlo Spustiť. Poznámka: skratka sa týka preformátovania kódu a umožňuje aj "Code Cleaup" (Vyčistenie kódu).

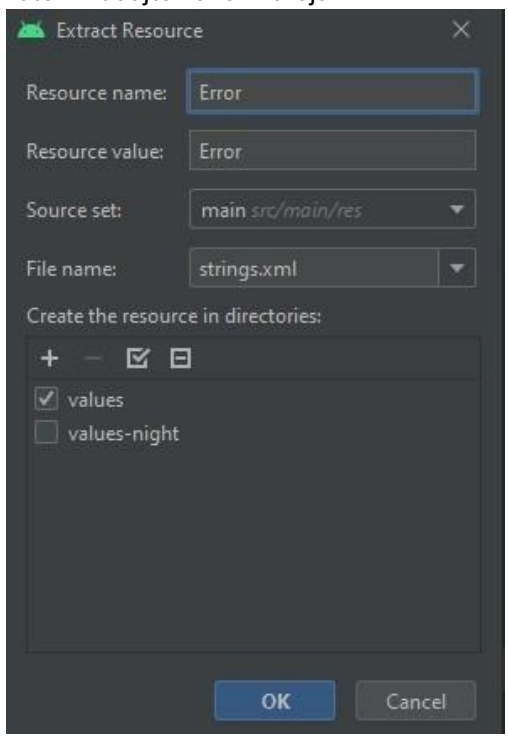
6. Odstránenie varovaní

Kompilátor analyzujúci kód nám hovorí, aby sme dodržiavali osvedčené postupy. Jedným z nich je umiestnenie všetkých reťazcov do osobitného súboru zdrojov. (**res/values/strings.xml)**. Vďaka tomuto riešeniu môžeme našu aplikáciu ľahko preložiť do iného jazyka. Viac informácií (https://developer.android.com/training/basics/supporting devices/languages)

(a) Vytvorenie konštánt pomocou, AndroidStudio vyzve:
V metóde evaluateExpression máme natvrdo zadaný objekt String - "Error". Prejdite kurzorom myši na tento objekt a pomocou **klávesovej skratky ALT+Enter** vyberte možnosť "**Extrahovať reťazcový zdroj".**



Potom zadajte názov zdroja



Kódex v znení neskorších predpisov

```
} catch (ex: Exception) {
    when (ex) {
        is IllegalArgumentException, is ArithmeticException -> {
            binding.display.text = getString(R.string.Error)
            isError = true
        }
        else -> throw ex
```

(b) Vylepšite kód v aplikácii tak, aby neobsahoval žiadne varovania.

Cvičenie 4

Prosím, otestujte aplikáciu. Opravte chybu, keď používateľ v aplikácii vykoná nasledujúce akcie:

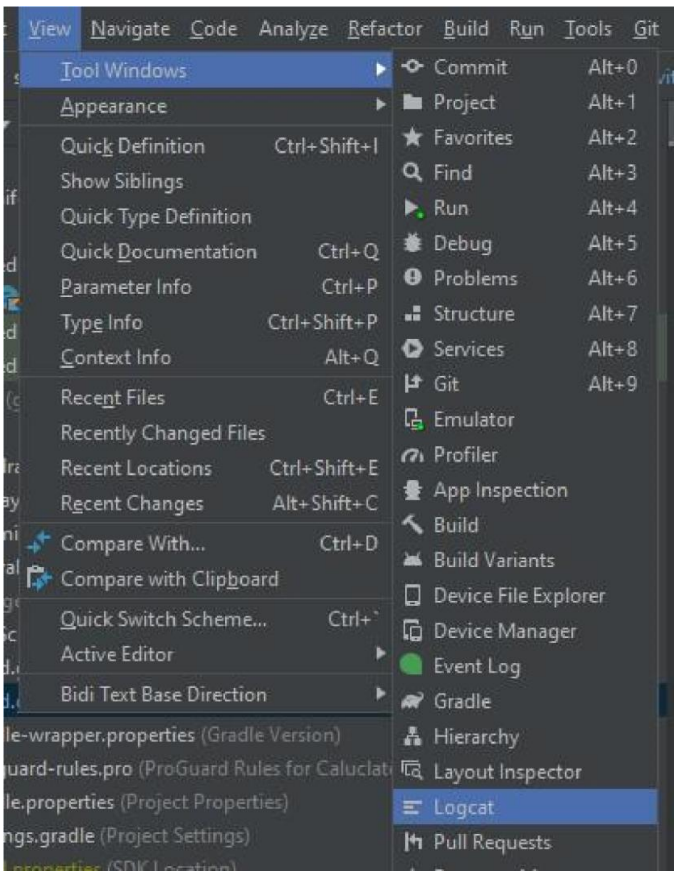
a) Stlačte tlačidlo "2".
b) Stlačte tlačidlo "5".
c) Stlačte tlačidlo "/".
d) Stlačte tlačidlo "=".

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

e) Stlačte tlačidlo "=".

☹ ☹ ☹

Ak chcete zistiť, čo spôsobuje chybu, použite nástroj Logcat



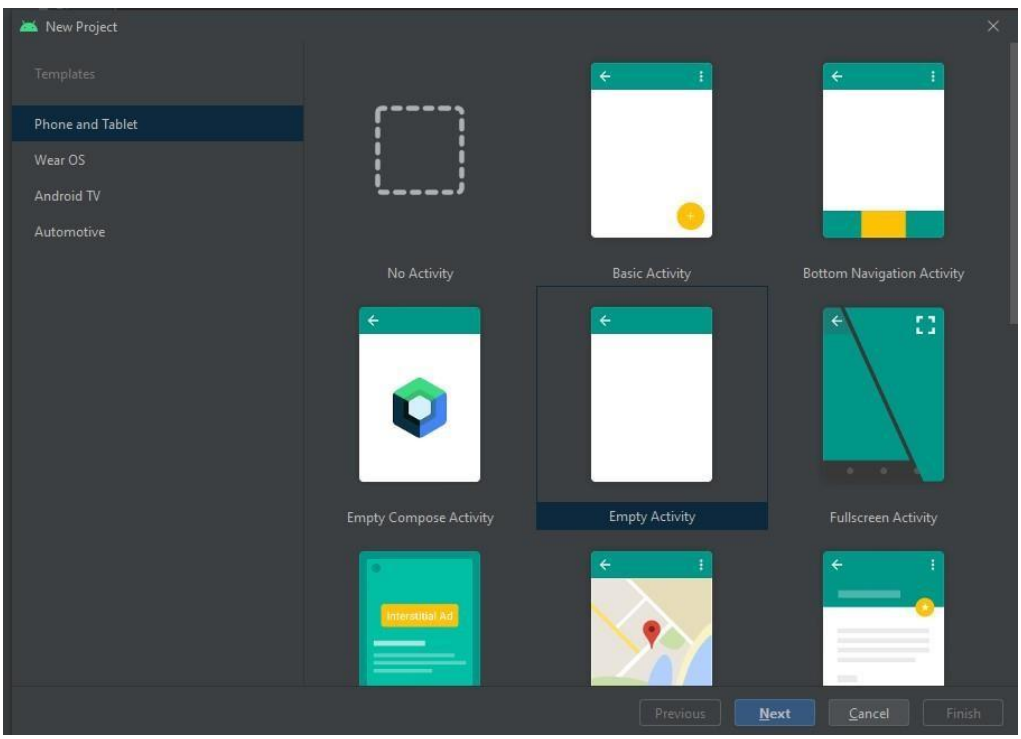
Výsledný kód aplikace najdete na adrese
https://github.com/matam/Erasmus_Lab2 .

Senzory

Väčšina mobilných zariadení má zabudované senzory, ktoré merajú pohyb, orientáciu a rôzne podmienky prostredia. Tieto snímače dokážu poskytovať nespracované údaje s vysokou presnosťou a správnosťou. Sú užitočné, ak chcete monitorovať 3D pohyb alebo polohu zariadenia. Umožňujú monitorovať zmeny prostredia v blízkosti zariadenia. Napríklad hra môže sledovať údaje z gravitačného snímača zariadenia a odvodiť tak zložité gestá a pohyby používateľa(nakláňanie, trasenie, otáčanie alebo kolísanie). Podobne by mohla aplikácia o počasí využívať snímač teploty a snímač vlhkosti zariadenia na výpočet a hlásenie rosného bodu alebo aplikácia na cestovanie by mohla využívať snímač geomagnetického poľa alebo akcelerometer na určenie smeru pohybu.

Počas týchto laboratórnych cvičení sa vytvorí aplikácia na čítanie senzorov a potom sa prestaví tak, aby používala návrhový vzor MVVM.

1. Vytvorenie nového projektu -> Prázdna aktivita



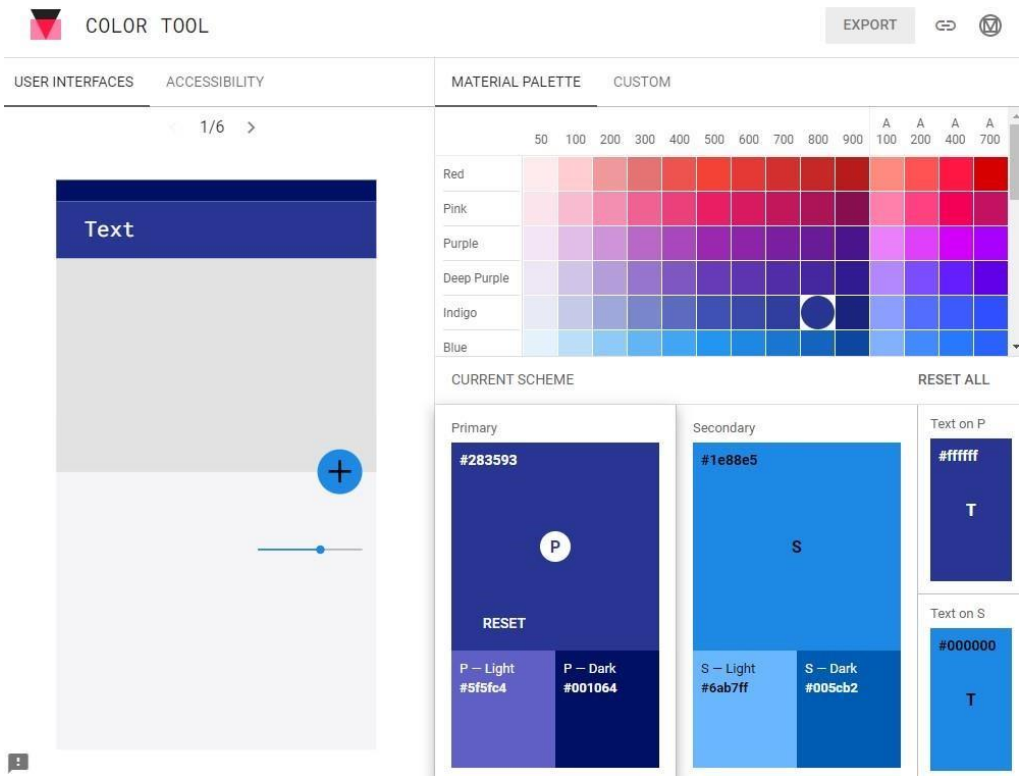
2. ***Definujte*** názov aplikácie (***Lab5)*** a balíka (***edu.zut.erasmus_plus.sensors),*** vyberte minimálne API (***API28***)

3. Prečítajte si kód
    1. nájdite a analyzujte súbory: **MainActivity.kt, activity_main.xml, AndroidManifest.xml**
4. Prejsť do build.gradle -> Aktualizovať všetky závislosti a knižnice pre projekt a modul (môžeme preskočiť)

5. Spustenie aplikácie

6. Vytvorenie farebnej schémy (voliteľné)

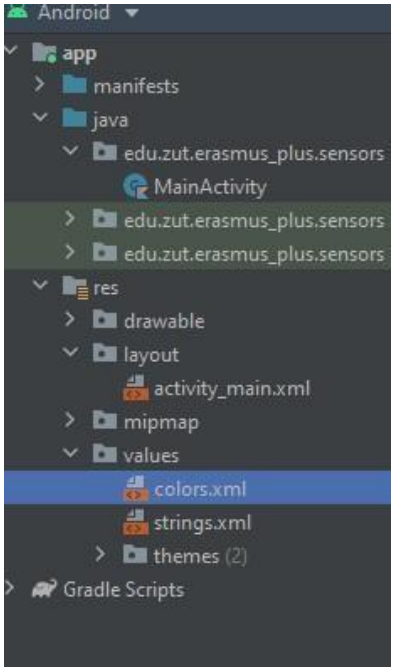
   Pred definovaním rozloženia použite [nástroj Farba - Material Design] a definujte farby aplikácie, potom túto farbu použite pri definovaní prvkov.
   Príklad farby:
   https://material.io/resources/color/#!/?view.left=0&view.right=0&primary.color=283593&secondary.color=1E88E5
   Viac informácií o farbách: https://material.io/design/color/the-color-system.html#color-theme-creation



Po výbere vhodnej farebnej schémy prejdite na kartu DOSTUPNOSŤ a skontrolujte upozornenia. Ďalším krokom je export konfigurácie. Vyberte tlačidlo EXPORT v hornej časti obrazovky a uložte súbor colors.xml.

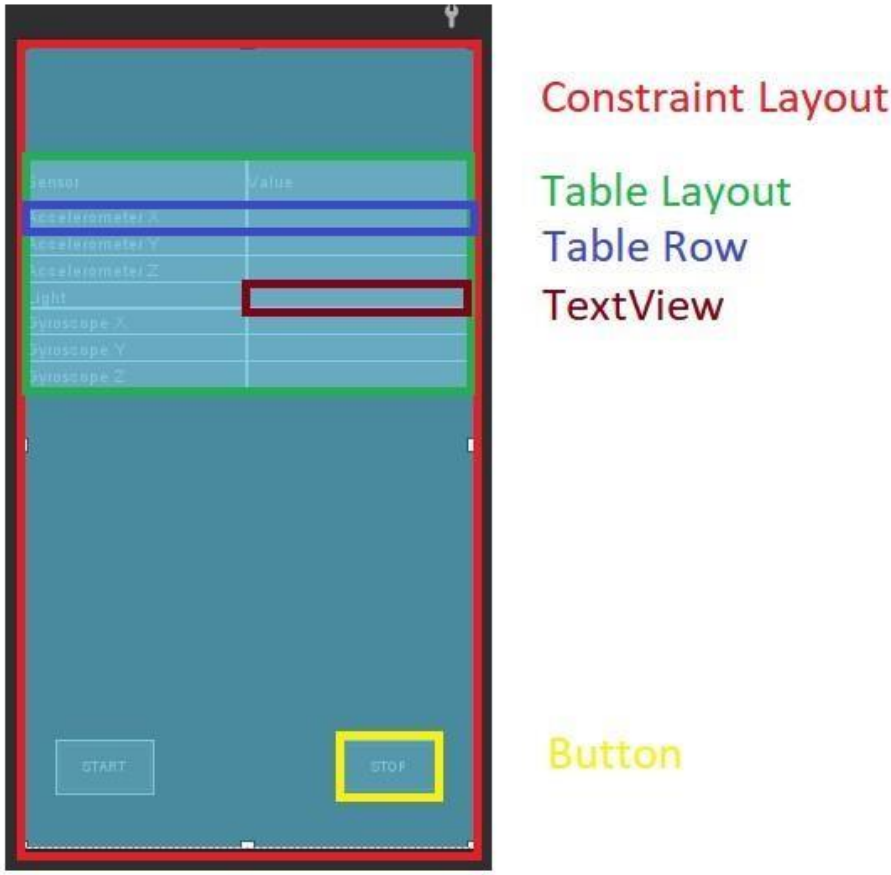
V rámci projektu otvorte súbor colours.xml a vložte hodnotu zo stiahnutého súboru.

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



7. Návrh grafického rozhrania

Rozloženie navrhnite takto:



Toto rozhranie je vytvorené pomocou dvoch typov "rozloženia", ConstraintLayout a TableLayout

Náčrt rozhrania - bude doplnený

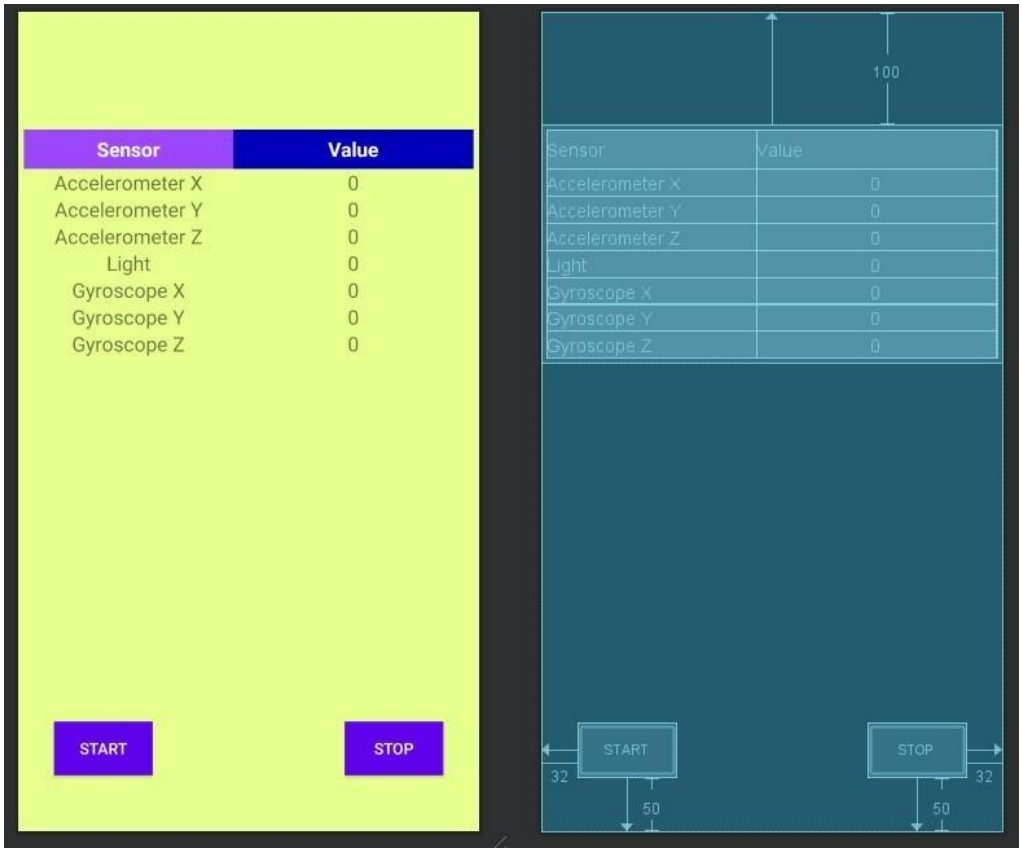
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    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. Prejdite do **MainActivity.kt** a do funkcie **onCreate()** pridajte objekty **SensorManager** a 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)
```

Predtým definujte objekty ako globálne pre triedu, napr.

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

9. Pridanie rozhrania ***SensorEventListener*** do triedy MainActivity na prijímanie udalostí zo snímačov.

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

10. Pridanie metód spätného volania ***onAccuracyChanged,*** **onSensorChanged**

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

```
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. Skontrolujte, či sa názvy objektov v kóde (R.id.XXXX) zhodujú s názvami ID v rozložení.
12. Definícia pomocných metód

```
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
}
```

a tiež pridať premennú **resume**

```
private var resume = false
```

13. Na registráciu a odhlásenie funkcie SensorsListener použite predchádzajúce metódy.

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

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

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

Prečo sa registrujeme a odhlasujeme?

14. Pridanie metódy ***onClick()*** (definujte aj v rozložení)

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

15. Spustenie aplikácie

Dodatočná úloha

16. Definovanie rôznych farieb v závislosti od stavu tlačidla.

Vytvorte background_button.xml a pridajte kód

```
<?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. Zmena definície pozadia pre každé tlačidlo

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

18. Spustenie aplikácie

MVVM, LiveData

Uvedená aplikácia demonštruje základný prístup k programovaniu pre systém Android. V súčasnosti sa odporúča, aby sa aplikácie pre Android vyvíjali pomocou návrhového vzoru MVVM (Model - View - ViewModel).

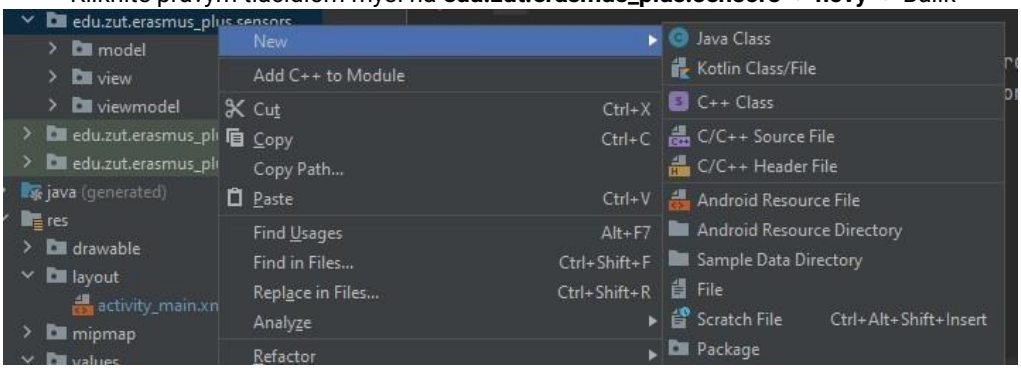
Nasledujúce riešenie využíva komponenty architektúry uvedené v knižnici AndroidX.

Práve vytvorenú aplikáciu chceme premeniť na aplikáciu, ktorá sa riadi návrhovým vzorom MVVM.

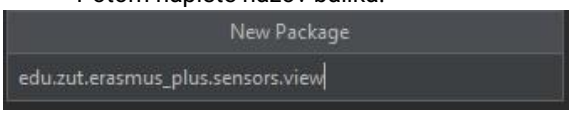
1. Príprava balíkov
   - **edu.zut.erasmus_plus.sensors.viewmodel**
   - **edu.zut.erasmus_plus.sensors.view**
   - **edu.zut.erasmus_plus.sensors.model**

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

Kliknite pravým tlačidlom myši na **edu.zut.erasmus_plus.sensors -> nový ->** Balík



Potom napíšte názov balíka.



2. presunúť súbor **MainActivity.kt** do **edu.zut.erasmus_plus.sensors.**view

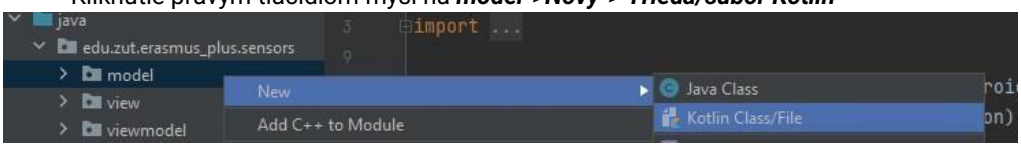
Všetky aktivity a fragmenty sú v modeli MVVM klasifikované ako pohľad.

3. Vytvorenie dátovej triedy https://kotlinlang.org/docs/data-classes.html

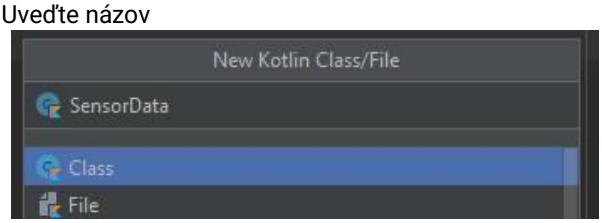
Dátová trieda - je trieda, ktorá obsahuje iba polia a metódy na prístup k nim (gettery a settery). Sú to jednoducho kontajnery pre dáta, ktoré používajú iné triedy. Tieto triedy neobsahujú žiadnu ďalšiu funkcionalitu a nemôžu samostatne pracovať s údajmi, ktoré uchovávajú.

a) Vytvorte súbor SensorData.kt vo vnútri **edu.zut.erasmus_plus.sensors.**model

Kliknutie pravým tlačidlom myši na ***model->Nový-> Trieda/súbor Kotlin***



Uveďte názov



b) Definícia triedy

Vo vnútri súboru SensorData.kt definujte všetky premenné

```
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. Väzba údajov.

Mechanizmus, ktorý rozširuje väzbu View Binding a zároveň ju nahrádza. Umožňuje obojstrannú väzbu.
Podrobné informácie: https://developer.android.com/topic/libraries/data-binding

Po vytvorení rozhrania je potrebné viazať existujúce objekty na kód. V súčasnosti sa odporúča použiť metódu Data Binding. Okrem automatického vytvorenia kódu umožňuje aj aktualizáciu prepojeného pohľadu pri zmene údajov pomocou **LiveData** (vzor Observer). Knižnica Data Binding bola vytvorená s ohľadom na pozorovateľnosť. Vzor, ktorý sa stal pomerne populárnym vo vývoji mobilných aplikácií. Pozorovateľnosť je doplnkom k data binding, ktorého základný koncept berie do úvahy len objekty pohľadu a údajov. Avšak práve prostredníctvom tohto vzoru môžu údaje automaticky propagovať svoje zmeny do pohľadu. Tým sa eliminuje potreba ručne aktualizovať zobrazenia vždy, keď sú k dispozícii nové údaje, čo zjednodušuje a znižuje počet riadkov kódu.

Naša aplikácia bude používať predtým vytvorenú dátovú triedu na vysielanie informácií o zmenách v senzoroch

5. Povolenie viazania údajov

Otvorte súbor build.gradle aplikácie a pridajte tento riadok do značky android.

```
android {
    compileSdk 31

    dataBinding {
        enabled true
    }
```

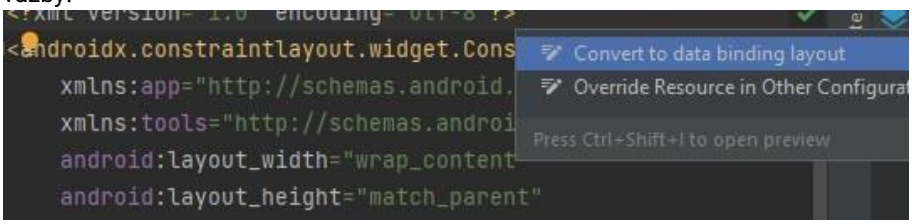
Prepojenie údajov v zobrazení

Konverzia bežného rozloženia na rozloženie pomocou Data Binding:

1. Rozloženie zabalte do značky <layout>
2. Pridanie premenných rozloženia (voliteľné)
3. Pridanie výrazov rozloženia (voliteľné)

Aleb

Android Studio ponúka automatickú konverziu: Kliknite pravým tlačidlom myši na koreňový prvok, vyberte položku Zobraziť kontextové akcie a potom Previesť na rozloženie dátovej väzby:




```
<?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">
```

Značka **<data>** bude obsahovať premenné rozloženia. Hodnoty tam budeme pridávať neskôr.

Premenné rozloženia sa používajú na zápis **výrazov** rozloženia. Výrazy rozloženia sa umiestňujú do hodnôt atribútov prvkov a používajú formát *@{výraz}*. Príklady výrazov sú uvedené nižšie (nepíšte ich do laboratória):

```
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()}">
```

Viac informácií o rozložení výrazov https://developer.android.com/topic/libraries/data binding/expressions#expression_language

6. Pridanie závislostí na LiveData v súbore 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. Vytvorenie triedy na čítanie hodnôt zo senzorov

Všetok kód zodpovedný za zber hodnôt pre senzory teraz presunieme do samostatnej triedy

a) Vytvorte súbor **SensorDataLiveData.kt** vo vnútri ***edu.zut.erasmus_plus.sensors.model***
b) Presun kódu z Activity_Main.kt (všetky funkcie okrem **onCreate()**)
c) Odstráňte nadbytočné objekty z triedy a kódu zodpovedného za vytváranie inštancií Sensor Manager a Sensor.

Po týchto krokoch vyzerá MainActivity.kt takto:

```
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)
    }
   }
```

Prakticky všetok existujúci kód bol odstránený.

Trieda **SensorDataLiveData** vyzerá takto:

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

```
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(
```

Táto trieda je teraz zodpovedná za obsluhu senzorov. Naším cieľom je, aby sa každá zmena oznamovala prostredníctvom pozorovateľov. Na tento účel použijeme triedu **LiveData**. Naša trieda ju rozširuje, a tak bude zmeny pozorovať pomocou modelu **SensorData**. Definícia triedy má nasledujúci tvar.

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

Pomocou metód triedy LiveData (**onActive()** a **onInactive()**) môže začať alebo ukončiť zber udalostí senzora. Metóda **onActive()** sa spustí, keď sa prvý pozorovateľ pripojí k našej triede, a druhá, keď sa posledný pozorovateľ odpojí.
Zvyšok kódu sa riadi tým, čo bolo predtým implementované v metóde **MainActivity(), s** výnimkou toho, že v metóde **onSensorChanged()** objekt ***value*** vracia výsledok. V našom prípade je to objekt triedy **SensorData, ktorý** obsahuje posledné hodnoty zo snímačov.

8. Vytvorenie ViewModelu

Trieda ViewModel je určená na ukladanie a správu údajov súvisiacich s používateľským rozhraním spôsobom zohľadňujúcim životný cyklus. Trieda ViewModel umožňuje, aby údaje prežili zmeny konfigurácie, napríklad otočenie obrazovky.

Vytvorenie súboru SensorViewModel.kt vo vnútri **edu.zut.erasmus_plus.sensors.viewmodel**

Vo vnútri súboru pridáme dve súkromné premenné a ich metódy 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
  }
```

Prvá premenná sa používa na čítanie hodnôt senzorov z predtým vytvorenej triedy SensorDataLiveData, kde je potrebné odovzdať kontext aplikácie. Preto upravíme definíciu triedy, ako je uvedené vyššie. Upozorňujeme, že naša trieda dedí od triedy AndroidViewModel.

Premenná **_pauseReading** určuje, či zastavíme/začneme čítať senzory. Je potrebné ju inicializovať, preto do modelu SensorViewModel pridáme nasledujúci kód:

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

Poslednou metódou, ktorú je potrebné pridať, je správna reakcia na údaje snímača zastavenia/spustenia. Do modelu SensorViewModel pridajte nasledujúcu funkciu:

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. preskupenie

Po vytvorení virtuálneho počítača musíme zmeniť rozloženie a pridať odkaz na model ViewModel

a) Pridanie informácií o premennej

Otvorte activity_mail.xml

Do vlastnosti **<data> </data>** vložte novú premennú

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

Pridanie premennej (**sensorViewModel)** umožní použitie objektov z triedy **SensorViewModel** v súbore rozhrania.

b) Zmena vlastností v activity_main.xml

Nájdite objekt **TextView** zodpovedný za zobrazenie hodnoty **acc_X** (približne na riadku 64) a zmeňte vlastnosť **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" />
```

Zmena vlastnosti **android:text** pre všetky objekty reprezentujúce hodnotu snímania senzora (7-krát)

c) Zmeňte vlastnosti ***onClick*** a ***enable*** pre tlačidlo takto.

Vyššie sú uvedené len zmenené a doplnené výpisy.

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()}"
. . .
 />
```

10. Zmeny aktivity - MainActivity.kt
    a) Otvorte MainActivity.kt
    b) Vo vnútri onCreate zmeňte kód takto:

```
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
    }
}
```

Všimnite si, že sme pridali dva objekty, **binding** a **sensorViewModel**. Objekt **binding** používa **DataBinding** a poskytuje nám automatický prístup k premenným v rozložení bez toho, aby sme museli používať funkciu **findViewById()**.

Objekt **sensorViewModel** viaže vytvorenú triedu ViewModel na pohľad.

11. Spustenie aplikácie

Vyššie uvedené kroky nám umožnili postupovať podľa teraz odporúčaného modelu vývoja aplikácií pomocou komponentov Androidu definovaných v knižnici androidX.

Výsledný kód aplikácie nájdete na adrese https://github.com/matam/Erasmus_Lab3-4.

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

Používanie databázy a aplikácie RecyclerView

Ukladanie údajov je jednou z najčastejšie používaných funkcií v aplikácii. Ak majú údaje štruktúru, na ich ukladanie sa používajú relačné databázy. V prostredí systému Android je databáza implementovaná pomocou databázy SQLite, ale jej priame použitie je pomerne komplikované. Knižnica AndroidX zaviedla framework Room, ktorý používanie databázy výrazne zjednodušil.

Na zobrazenie údajov, ktorých počet prvkov je premenlivý, sa používajú dynamické objekty View, pričom jedným z najčastejšie používaných je RecyclerView.

Tento dokument predstavuje databázovú aplikáciu na ukladanie informácií o knihách.

Tieto laboratóriá obsahujú nasledujúce prvky:

1. Vytvorte novú aplikáciu a nastavte build.grade
2. Vytvorenie dátového modelu
3. Vytvorenie inštancie databázy
4. Vytvorenie zobrazenia recyklátora
5. Súvisiace údaje databázy s RecyclerView

1. Vytvoriť New project -> Empty Activity

2. Definovať App Name (***Lab4***) a názov balíka (***edu.zut.wi.erasmus.lab4***), vyberte minimum API (***API28***)

3. Pridanie závislostí 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")
```

a na plugin

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

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

4. Definovanie nového balíka: database(***edu.zut.wi.erasmus.lab4.database***)
5. Pridať novú triedu do tohto balíka ***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. Vytvoriť 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. Realizovať 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. Implementácia odkladacieho priestoru – 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. Stna svete nový balík: viewmodel(***edu.zut.wi.erasmus.lab4.viewmodel***)
10. Realizovať ViewModel a 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)
    }
}
```

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

```
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 . Realizácia RecyclerView

11. Vytvorte nové rozloženie pre 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"
            android:layout_height="wrap_content"
            style="@style/book_publisher"
            android:text="TextView" />
```

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

```
    </LinearLayout>

</LinearLayout>
```

12. Pridať do *activity_main_xml* kód pre 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. Vytvorenie nového balíka: adapter(***edu.zut.wi.erasmus.lab4.adapter***), vytvoriť novú triedu 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.publisher)


    }

    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. Pridanie hovoru adaptéra MainActivity.kt
    Wstaw kod poniżej ***setContentView***

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

15. Vytvorenie inštancie ***Application*** a v ňom prípad databázy a registra, ***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. Zmeniť ***AndroidManifest.xml*** a pridať vnútornú značku <application … kód uvedený nižšie

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

17. Spustenie aplikácie
18. Pridanie novej aktivity na pridanie nových kníh– NewBookActivity.kt – vytvorenie pomocou sprievodcu

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

```
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. prispôsobiť 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"
        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>
```

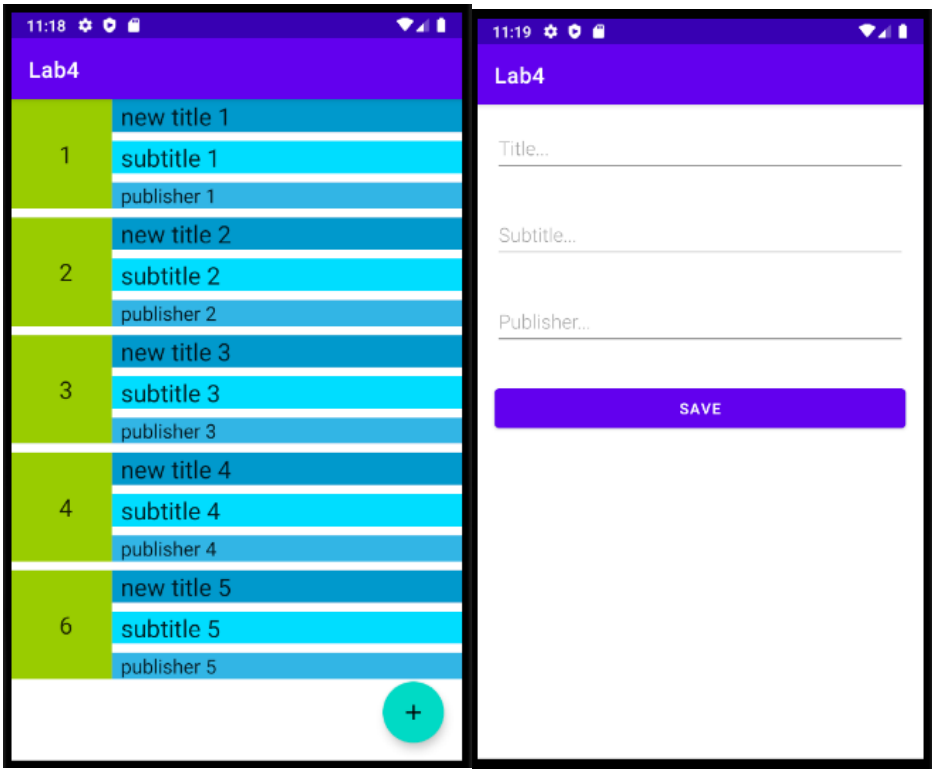
20. Pridať kód pre FloatingActionButton – MainActivity.kt

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

21. Spustenie aplikácie

Konečný pohľad na žiadosť

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



Main Activity New Book Activity

Ďalšie informácie: https://developer.android.com/codelabs/android-room-with-a-view-kotlin

Výsledný kód aplikácie nájdete na adrese https://github.com/matam/Erasmus_Lab5.

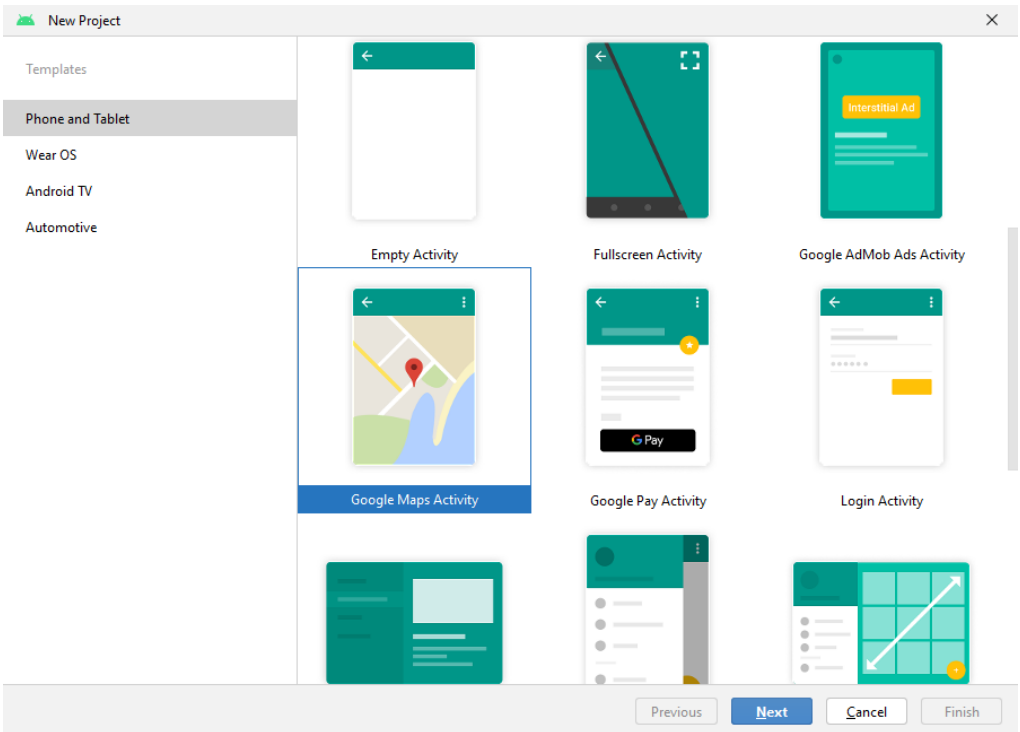
Co-funded by the
Erasmus+ Programme
of the European Union

Lokalizace

Lokalizácia poskytuje možnosť vytvoriť aplikáciu, ktorá zohľadňuje kontext polohy používateľa. Umožňuje vytvárať obsah, ktorý sa bude dynamicky meniť v závislosti od polohy zariadenia. Lokalizácia tiež umožňuje vývojárom zhromažďovať štatistiky, uľahčuje profilovanie, ale existuje určité riziko ohrozenia súkromia. Prístup k polohe si vyžaduje, aby používateľ udelil aplikácii povolenia.

Tento dokument opisuje aplikáciu, ktorá priebežne alebo jednorazovo zobrazuje polohu používateľa. Okrem toho bude opísaný proces udeľovania a overovania oprávnení. Aplikácia používa hotový dizajn, ktorý zobrazuje mapu.

1. Vytvorenie projektu pomocou sprievodcu

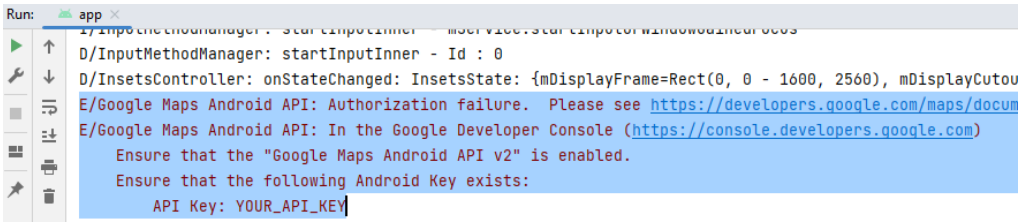


Vyberte aktivitu Mapy Google.

2. Zoznámte sa s kódexom. Všimnite si, aké rozhrania sú implementované, analyzujte vytvorené metódy
3. Run the application.
   Po spustení by sa mala zobraziť mapa so značkou v Sydney. Nezobrazí sa však z dôvodu chýbajúceho platného kľúča API.

V systéme Run máme informáciu o neprítomnosti platného kľúča

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



4. Pridanie správneho kľúča k projektu: Presné informácie nájdete na tejto adrese

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

   a. Vložte svoj API_KEY do súboru AndroidManifest.xml
   b. Konfigurujte API_KEY - pridajte Android Maps
5. Teraz prejdeme k pridaniu zobrazenia polohy zariadenia do aplikácie.
6. Na určenie polohy budeme používať služby Google Play odporúčané spoločnosťou Google.

   Ako pridať "Služby Google Play" - https://developers.google.com/android/guides/setup

Pridanie závislostí do súboru build.gradle (vyberte najnovšiu verziu)

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

...

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

7. Zmeníme grafický návrh **activity_main.xml** a pridáme do neho dve tlačidlá.

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>
```

Oprava chýb pridaním príslušných definícií do súboru **strings.xml** (get_position-”Get Position”, start_loop -> „Start Loop”), tiež pridať novú definíciu stop_loop ->”Stop_Loop”

8. Získanie poslednej známej polohy - https://developer.android.com/training/location/retrieve-current
   Prvým krokom bude pridanie kódu na určenie poslednej známej polohy. Túto funkciu pripojíme k tlačidlu "Get Position", pričom metóda spracovania sa bude volať **getLastPos()**.
   Túto metódu pridáme do hlavného súboru aktivity.

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))
            }
        }

}
```

a pridať definíciu objektu v triede ***MapsActivity***

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

Objekt by mal byť inicializovaný. Do metódy ***onCreate()*** pridajte nasledujúci kód.

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

Vyššie uvedený kód neobsahuje kontrolu oprávnení. Podľa pravidiel sa pred každým použitím funkcie, ktorá musí mať povolenie od používateľa, vyžaduje kontrola oprávnenia.

9. Zadanie povolenia aplikácie
   Prvým krokom je pridať do súboru manifest informácie o tom, aké oprávnenia sú potrebné na spustenie aplikácie. V našom prípade pridáme hrubé aj jemné oprávnenia. Všimnite si, v ktorej časti manifestu sú oprávnenia pridané.

```
<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 môžete prejsť k vytvoreniu funkcie, ktorá bude overovať oprávnenia.

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")
        }
    }
```

Vyššie uvedená metóda je najjednoduchšia a kontroluje oprávnenia iba raz, v prípade, že používateľ odmietne natrvalo, túto situáciu neriešte. Ako ďalšiu úlohu vykonajte úplný cyklus dotazov na povolenia podľa: https://developer.android.com/guide/topics/permissions/overview#workflow

11. Teraz je možné upraviť metódu ***getLastPos()*** a zakomentovať kontrolu oprávnenia.
12. Spustenie aplikácie
13. Teraz pridáme podporu pre priebežné aktualizácie položiek. Budeme tiež používať služby Google Play. Úlohou tejto funkcie bude priebežne aktualizovať značku polohy na mape. Na tento účel sa vytvorí návratová metóda, ktorej úlohou bude aktualizovať polohu po prijatí novej polohy. V metóde **MapsActivity()** musíme definovať nasledujúce objekty, ktoré majú byť dostupné pre všetky metódy triedy.

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

14. V metóde ***onCreate()*** teraz inicializujeme metódy pre umiestnenie.
    Prvý objekt sa používa na špecifikáciu parametrov umiestnenia, zatiaľ čo druhý definuje návratové metódy.
    Všimnite si frekvenciu aktualizácie.

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 môžeme zavolať požiadavku na polohu, urobíme to v metóde ***startStopRequestLocation()***, ktorá bola vytvorená na spracovanie druhého tlačidla
    Najprv skontrolujeme oprávnenia a potom na základe stavu skontrolujeme, či sa má úloha aktualizácie polohy spustiť alebo zastaviť. V našom prípade len pridáme bod do mapy. Nasledujúci kód možno optimalizovať vytvorením spoločných metód pridávania

```
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. Spusťte aplikaci. Pokud aplikaci spouštíte v emulátoru, můžete pozici změnit v nastavení virtuálního počítače.
17. Počáteční i finální projekt je umístěn v úložišti.
    https://github.com/matam/Erasmus_Lab6

Úloha:

1. Vykonajte proces priraďovania podľa odporúčaného pracovného postupu
2. Zobrazenie iba posledných 5 značiek na mape.

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

Sieťové prepojenie

Pri vývoji aplikácií je často potrebné používať sieťové pripojenia. Sieťové požiadavky sa používajú na sťahovanie alebo aktualizáciu údajov. Používanie siete môže generovať značné náklady, ako aj predstavovať určitú hrozbu pre súkromie používateľa, preto je potrebné povolenie na používanie siete a vykonávanie sieťových operácií a čítanie stavu siete vo vašej aplikácii, váš manifest musí obsahovať oprávnenia. Vykonávanie sieťových operácií si vyžaduje, aby boli dokončené v samostatnom vlákne a nezaťažovali hlavné vlákno. Pred pripojením je vhodné skontrolovať, či je zariadenie pripojené k internetu. Kontrola siete sa musí vykonať pred každou operáciou sťahovania.

V súčasnosti väčšina aplikácií používa na načítanie údajov rozhranie API REST danej služby. Niekedy je však potrebné získať súbor zo servera tradičným spôsobom. Vtedy sa uprednostňuje protokol HTTP/HTTPS.

V tomto laboratóriu vyvíjame program, ktorý sťahuje obrázky pomocou API NASA (Astronomy Picture of the day) - https://api.nasa.gov/#browseAPI .

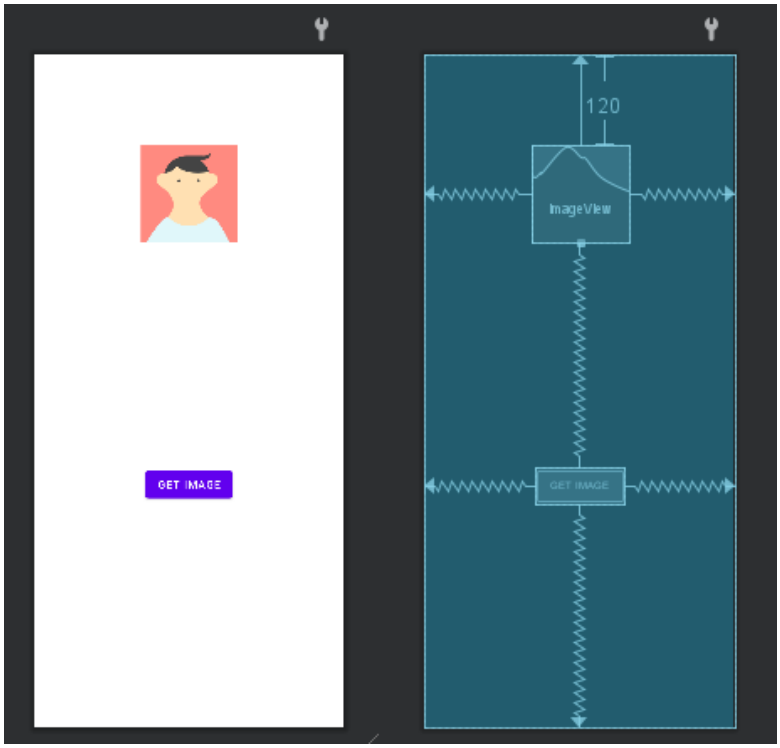
1. Na realizáciu tohto laboratória potrebujeme Generovať API KEY - https://api.nasa.gov/#signUp .
   Zaregistrujte sa a uložte prijaté **API_KEY**.

2. Vytvorenie nového projektu -> “Empty Activity”

3. Definujte názov aplikácie (***Lab7***) a názov balíka (***edu.zut.erasmus_plus.networking***), vybrať minimálne API (***API28***)
4. Preskúmajte kód
   - Nájsť súbor MainActivity.kt, activity_main.xml, AndroidManifest.xml
5. Prejdite do súboru build.gradle -> “Upgrade all dependencies and libraries for Project and Module” (môžeme preskočiť)

6. Navrhnite takéto rozloženie:

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



Obrazovka aplikácie bude obsahovať dva prvky, tlačidlo na stiahnutie údajov a fotografiu stiahnutú z internetu.

7. Spustiť aplikáciu

8. Definovanie oprávnení v súbore AndroidManifest.xml

Ak chcete používať internet, musíte definovať príslušné oprávnenie; okrem toho musíte definovať oprávnenie aj na kontrolu stavu siete.

```
<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. Vytvorenie metódy na kontrolu sieťového pripojenia

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

Nasledujúci kód bude užitočný na overenie, či je k dispozícii internetové pripojenie. Túto metódu použite pred získaním súboru 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 je knižnica pre Android a Javu, ktorá je vynikajúca pri načítavaní a odosielaní štruktúrovaných údajov, ako sú JSON a XML. Táto knižnica vykonáva požiadavky HTTP pomocou ***OkHttp***, ďalšej knižnice od spoločnosti Square. Pomocou tejto knižnice definujeme službu, triedu údajov a úložiskois

Použitie knižnice pozostáva z definovania údajov (trieda údajov), služby definujúcej dotazy (rozhranie) a metódy volania. Najprv definujeme závislosti.

11. Definujte závislosti

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

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

12. Definovanie triedy údajov

Údaje z rozhrania API budú štruktúrované a vrátené ako JSON. Vytvorenie triedy údajov uľahčí používanie týchto údajov. Opis jednotlivých polí je uvedený v časti

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

```
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?
)
```

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

13. Definovať servis

Stiahnutie obrázka bude prebiehať v dvoch fázach, najprv sa stiahne objekt s popisom obrázka dňa a adresa, odkiaľ je možné obrázok stiahnuť. Druhá fáza zahŕňa stiahnutie obrázka. Preto služba obsahuje dve definované metódy sťahovania.

```
//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. Používanie Retrofitu

```
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!!!!")
        }

    })
}
```

Použitie vytvorenej služby spočíva v definovaní vlastnej inštancie služby a odkazovaní na definovanú metódu v tomto prípade ***getApodItem***

Potom nasleduje volanie metódy ***enqueue()***, čím sa odošle sieťový dotaz. Výsledok dotazu sa prijíma v spätných volaniach metód ***onResponse()*** alebo ***onFailure()*** v závislosti od výsledku. Vo vyššie uvedenom kóde sú dve metódy, jedna načíta metadáta fotografie a druhá načíta fotografiu.

15. Pridajme kód na obsluhu tlačidla a zobrazenie obrázka (**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()
    }
}
```

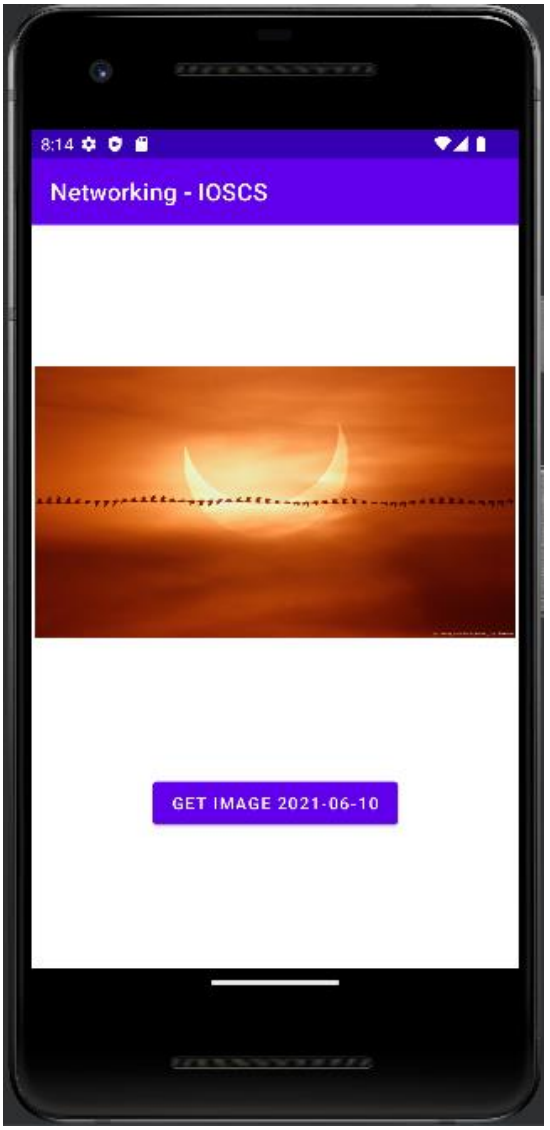
16. A metóda, ktorá konvertuje výsledný bitový tok na obrázok

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

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

17. Spustiť aplikáciu

V dôsledku toho môže aplikácia vyzerať nasledovne:

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



Otázka

- Je potrebné požiadať používateľa o povolenie používať internet?
- Sú definované povolenia typu "nebezpečné"

Výsledný kód aplikácie nájdete na adrese https://github.com/matam/Erasmus_Lab7.

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