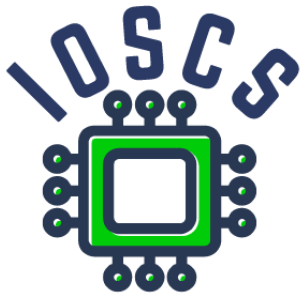


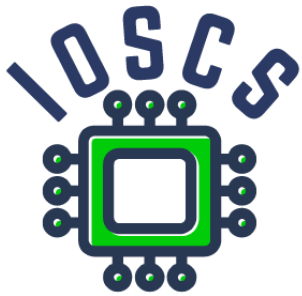

**Project: Innovative Open Source Courses for Computer Science**

# Vývoj mobilních aplikací
# Materiál pro laboratoře

**Radosław Maciaszczyk**
**West Pomeranian University of Technology in Szczecin**

**30.05.2021**

## Innovative Open Source Courses for Computer Science




This material teaching 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

# První aplikace– Hello World

Tato laboratoř obsahuje:

- Instalace prostředí Android Studio
- Vytvoření prvního projektu
- Provádění životního cyklu činnosti

## Úkol 1. Stažení a instalace aplikace Android Studio

Stáhněte si a nainstalujte nejnovější verzi ze stránek https://developer.android.com/studio Android Studio

Wymagania systemowe [1]:

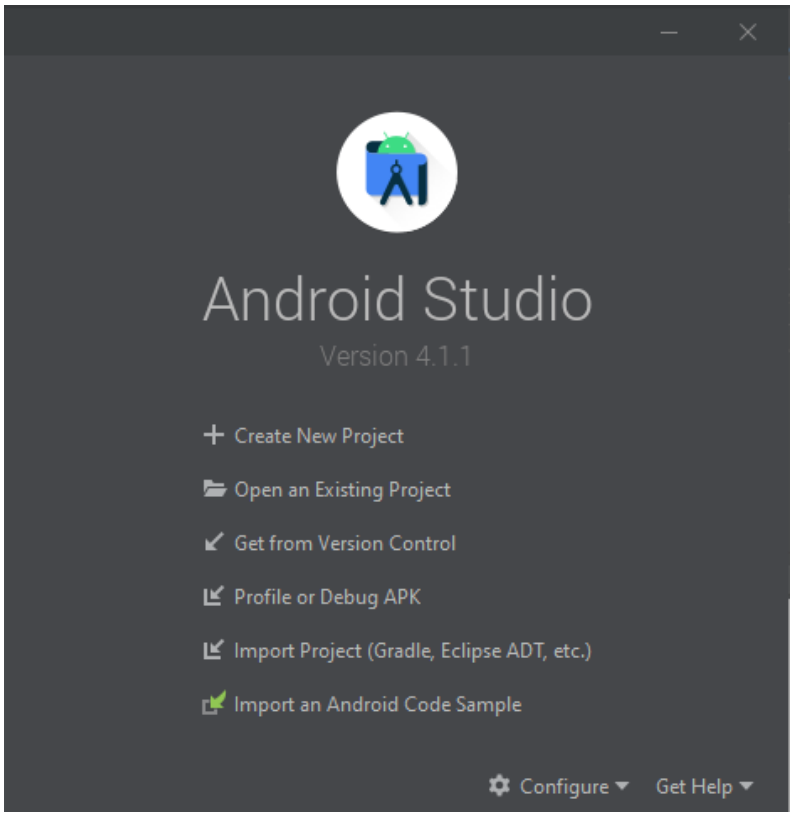
| **Windows** | **Mac** | **Linux** |
|---|---|---|
| • 64-bit Microsoft® Windows® 8/10<br>• x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor<br>• 8 GB RAM or more<br>• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)<br>• 1280 x 800 minimum screen resolution | • MacOS® 10.14 (Mojave) or higher<br>• ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework<br>• 8 GB RAM or more<br>• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)<br>• 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.<br>• x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3<br>• 8 GB RAM or more<br>• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)<br>• 1280 x 800 minimum screen resolution |

## Úkol 2 Vytvoření projektu Hello World

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

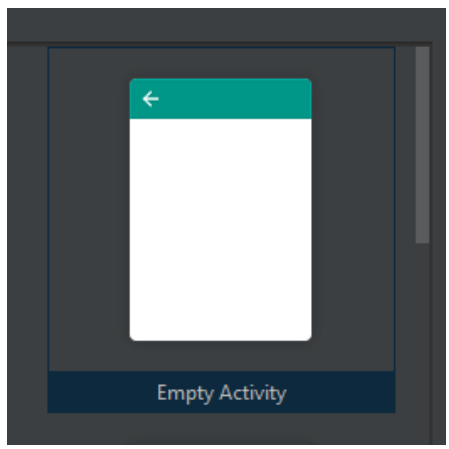
# I. Vytvoření projektu

1. Create New Project

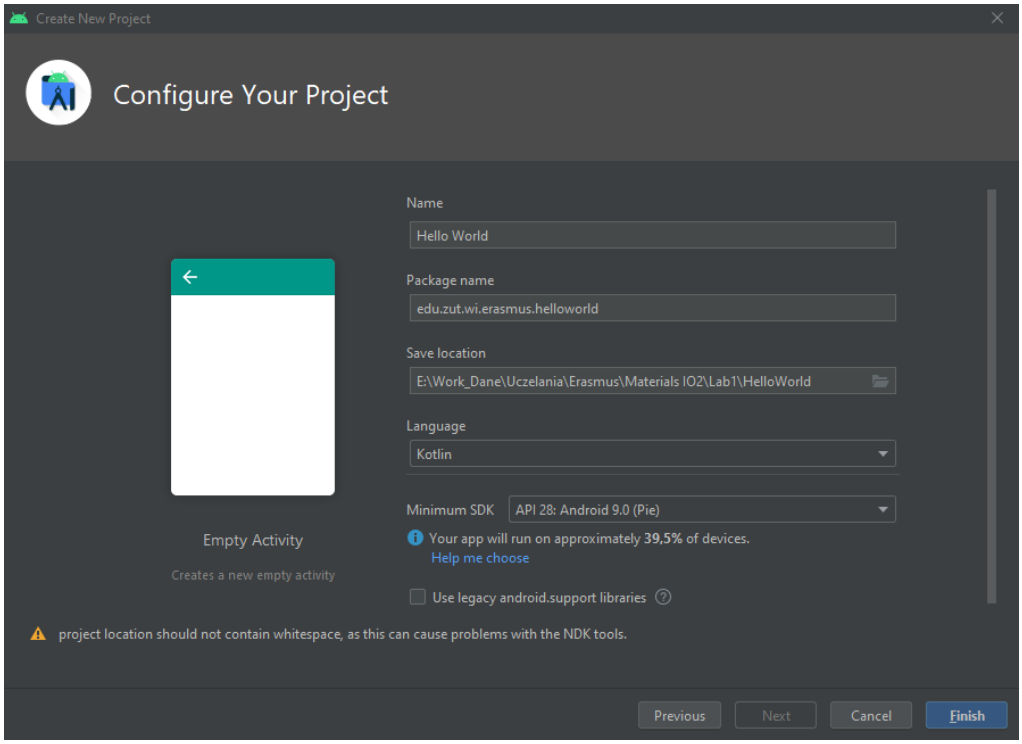



2. Vyberte „Empty Activity" -> next
3. Konfigurace projektu:
   Konfigurace:
   - Name
   - Package name
   - Location project
   - Chose Language
   - Choose SDK

   Finish




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




II. Vytvoření virtuálního zařízení Android

Prostředí Android Studio umožňuje emulaci zařízení se systémem Android. To umožňuje bezproblémové testování vytvořených aplikací.

1. Vyberte z nabídky: Tools->AVD Manager
2. Pokud neexistuje žádný AVD, vytvořte jej.
    a. Vyberte zařízení, např. Pixel 2
    b. Vyberte obraz systému, např. Android 9
    c. Definujte název
    d. Přejděte do pokročilých nastavení a zaškrtněte je.
    e. Finish

III. Spuštění aplikace

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

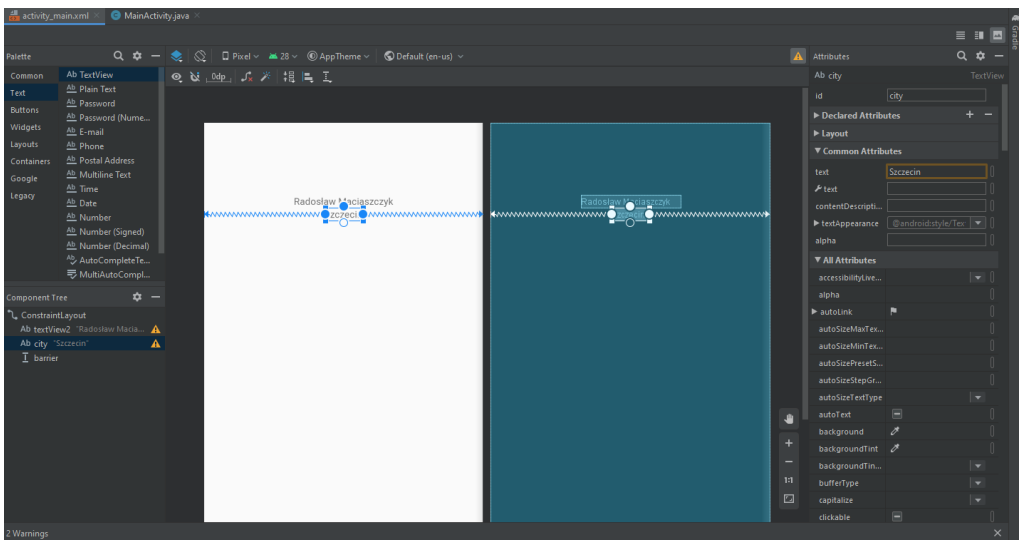
IV. Změna textu na obrazovce a přidání nového objektu TextView

1. Vyhledejte soubor v projektu ***R.layout.activity_main.xml*** a zobrazení (double click)
2. Vyberte text “Hello word” a změnit ji
3. Pro TextView zmień **andoriod:id ->** android:id="@+id/name"
4. Z palety widgetów přidat nový widget: TextView
5. Pro TextView definovat: **id** a **text**

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

Fragmentace 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" />
```




## Úkol 3 Životní cyklus činnosti (Acivity)

Tato úloha ukazuje celý životní cyklus aktivity, jak implementovat záměr a jak přidat tlačítko. Kromě toho ukazuje, jak používat protokolování pomocí **logcat**.

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

Znalost životního cyklu aktivity je klíčem k vytvoření správně fungující aplikace, která dokáže ukládat stavy aplikace mezi jednotlivými spuštěními.

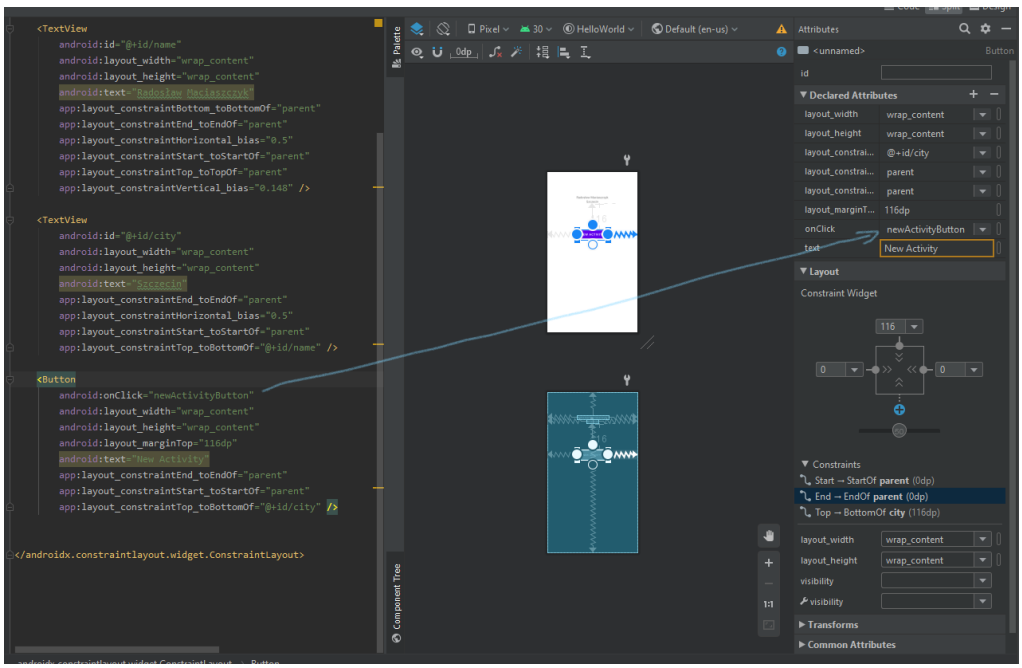
I. Přidání tlačítka na obrazovku

1. V souboru ***R.layout.activity_main.xml*** přidání tlačítka z palety widgetů.
2. Definice pro tlačítko:
   id: newActivity
   text: New Activity

3. Přejděte na ***MainActivity.kt*** a definujte metodu, která bude událost zpracovávat onClick.

```
fun newActivityButton(view: View){

}
```

4. Přejděte na ***activity_main.xml*** a přidejte metodu onClick do tlačítka.



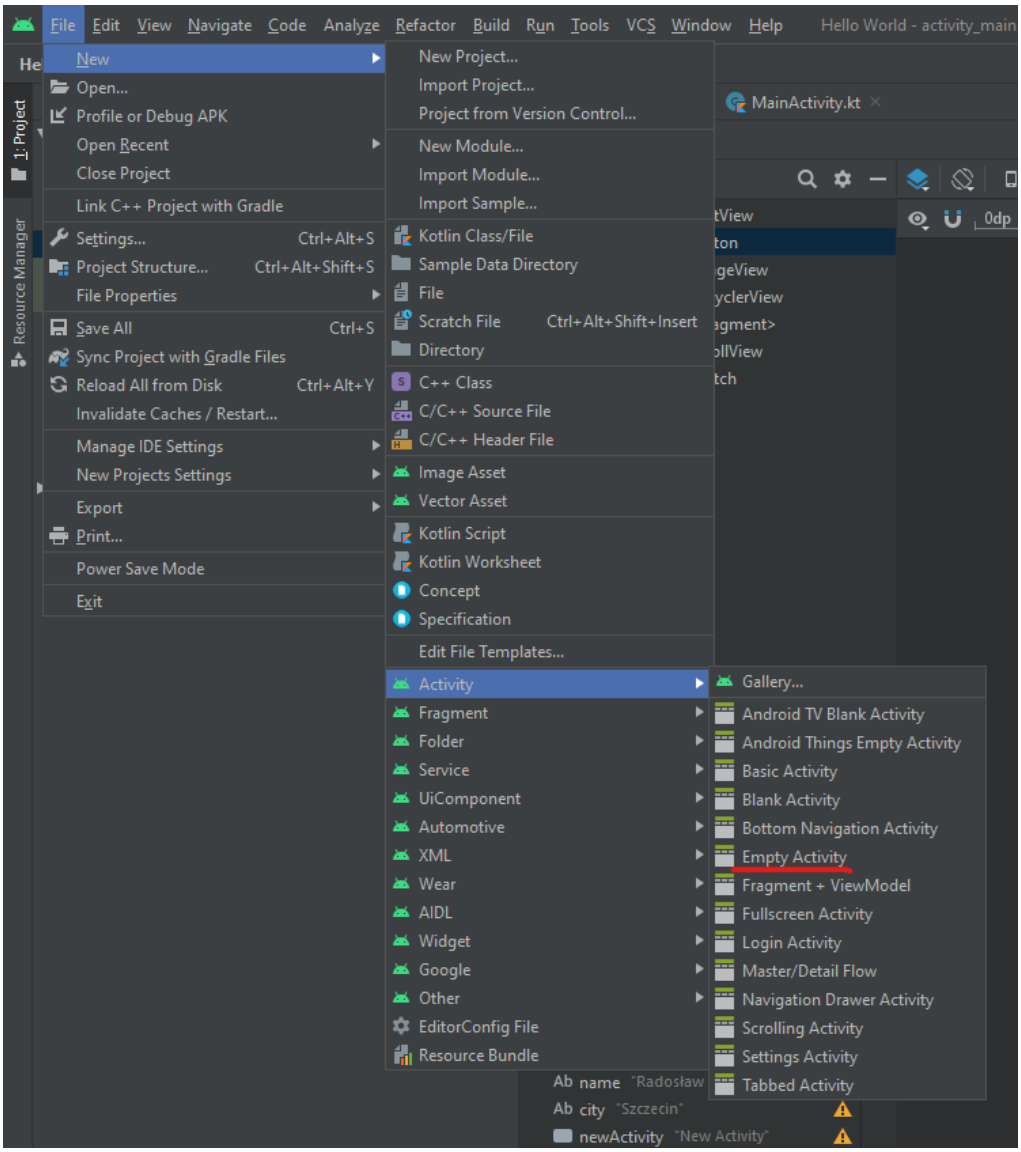

5. Konečná verze xml pro tlačítko

```
<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" />
```

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

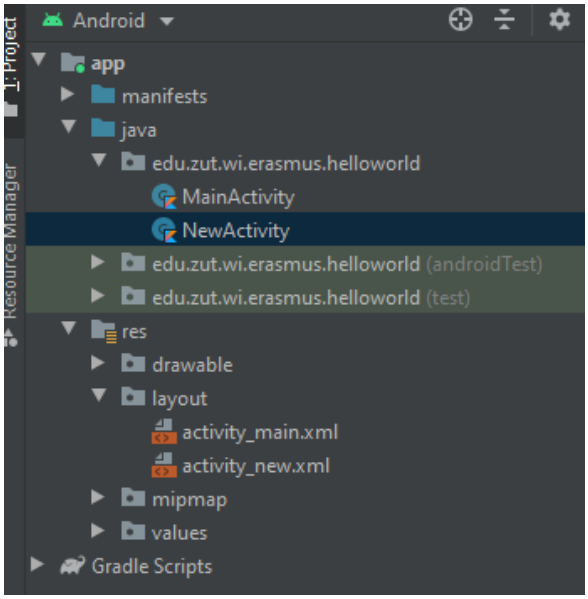
## V. Vytvoření nové aktivity

1. V nabídce vyberte File-> New -> Activity -> Empty Activity
2. Definovat Activity
   Activity name: NewActivity
   Click Finish




Po vytvoření máme následující adresářovou strukturu

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




VI. Přidání metod životního cyklu aktivity

1. Použijte klávesovou zkratku CTRL+O (Show Override Members)
2. Vyberte metody z životního cyklu ***onPause, onStart, onRestart, onStop, onDestory***

VII. Použití třídy **Log**

1. Definujte konstantu TAG – (Dobrá praxe: TAG má stejný název jako třída)

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

2. Přidejte volání metody **Log** na všechny metody životního cyklu
   Např.

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

3. Konečná verze 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. Přidání zpracování návratů pomocí události **UP**

1. Otevřít AndroidManifest.xml a definovat nadřazenou činnost

```
<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. Vrátit se na MainActivity.kt
    1. Přidejte všechny metody životního cyklu do této aktivity, stejně jako v předchozí aktivitě.
    2. Přidejte volání metody ***Log***

3. Uvnitř metody zpracovávající událost onClick() ***newActivityButton*** dodaj

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

Přidejte také přihlašovací údaje

4. Konečná verze MainActivity.kt

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

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
```

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

```
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. Zahájení projektu

Podívejte se na ***logcat*** a sledujte zaznamenané hodnoty.

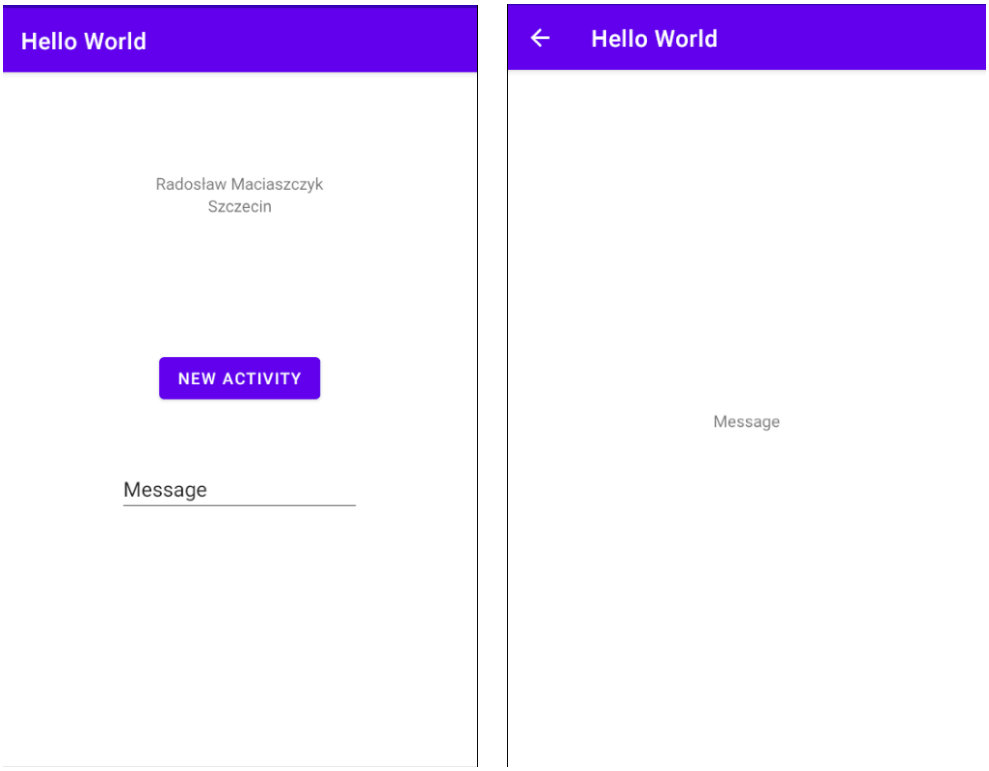
Dodatočná úloha:

K první aktivitě přidejte *Input Text* a použití Intent odeslat zprávu jiné aktivitě a poté ji zobrazit..

Vytvořte pohled, jako je ten níže:

První MainActivity odešle po stisknutí tlačítka zprávu na adresu Second Activity

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




Další informace o odesílání dat pomocí záměru:
https://developer.android.com/training/basics/firstapp/starting-activity

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab1.

Literatura:

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

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

Další informace:

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

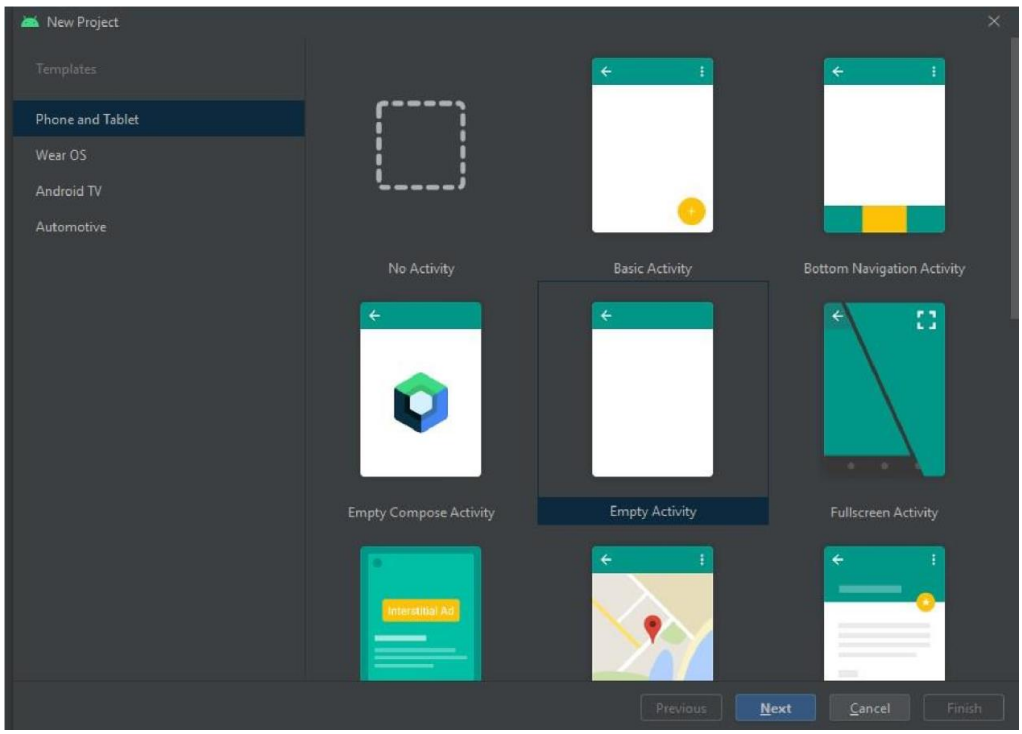
# Uživatelské rozhraní

Uživatelské rozhraní - část programu, aplikace nebo operačního systému, která je zodpovědná za komunikaci s uživatelem. Jeho nejběžnější formou je grafické uživatelské rozhraní (GUI). Skládá se z grafických prvků, které sjednocují vzhled aplikace a prezentují ji v rozpoznatelné a předvídatelné podobě. Při návrhu rozhraní je kromě grafické vrstvy (ikony, nabídky, textová pole, seznamy) důležité pamatovat na vytvoření cest pohybu uživatele, informační architekturu a interakční procesy. Proto je dnes v procesu návrhu aplikací tak důležitá kombinace UI a UX (user experience). V systému Android představuje kompletní systém UI a UX Material Design (https://material.io/ ), který je v současné době ve verzi 3.

Cílem této laboratoře je ukázat základy tvorby rozhraní a seznámit vás s nástroji, které vám při tvorbě rozhraní pomohou. Nejprve vytvoříme typickou aplikaci "Hello World" a poté aplikaci "Kalkulačka".

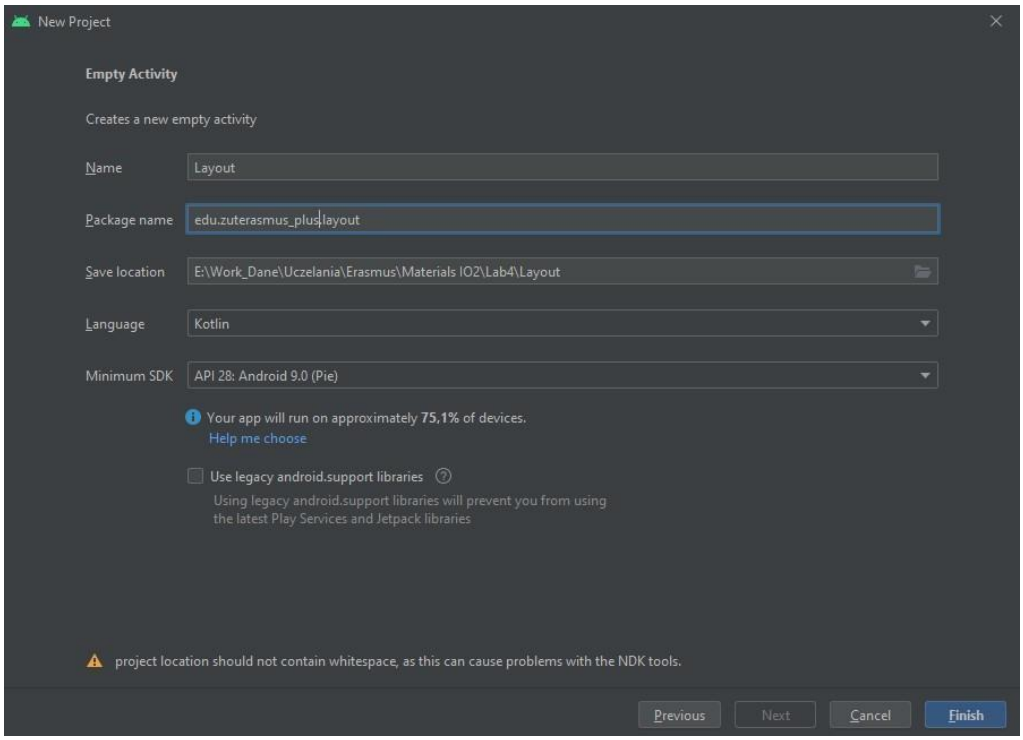
## Projekt Hello World

1. Spuštění aplikace Android Studio
2. Vytvoření nového projektu - vyberte možnost "Empty Activity".




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

3. Zadejte název projektu, název balíčku, vyberte verzi API a cestu.



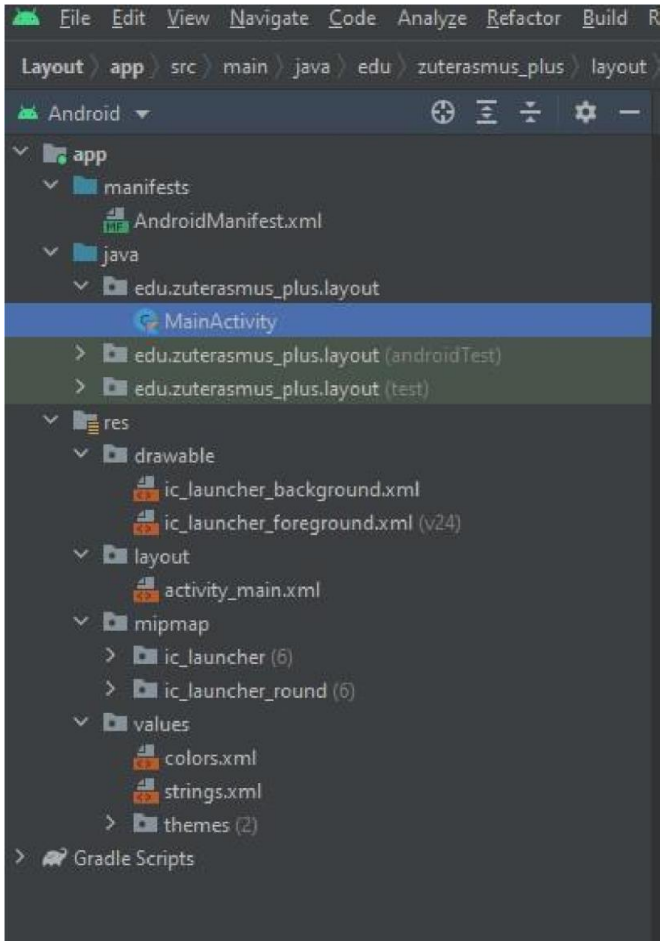

4. Stiskněte tlačítko Dokončit a počkejte na vytvoření kostry projektu.

Níže je zobrazen počáteční pohled na projekt a struktura projektu.

Kód ukládáme do adresáře ***JAVA (***i při psaní v jazyce Kotlin). V adresáři ***manifest*** je uložen soubor **AndroidManifest.xml** obsahující důležité informace pro překladač, včetně definic komponent aplikace nebo oprávnění.

***Adresář res*** slouží k ukládání informací o zdrojích aplikace, včetně adresáře ***layouts,*** kde definujeme vzhled aplikace v souboru xml.

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

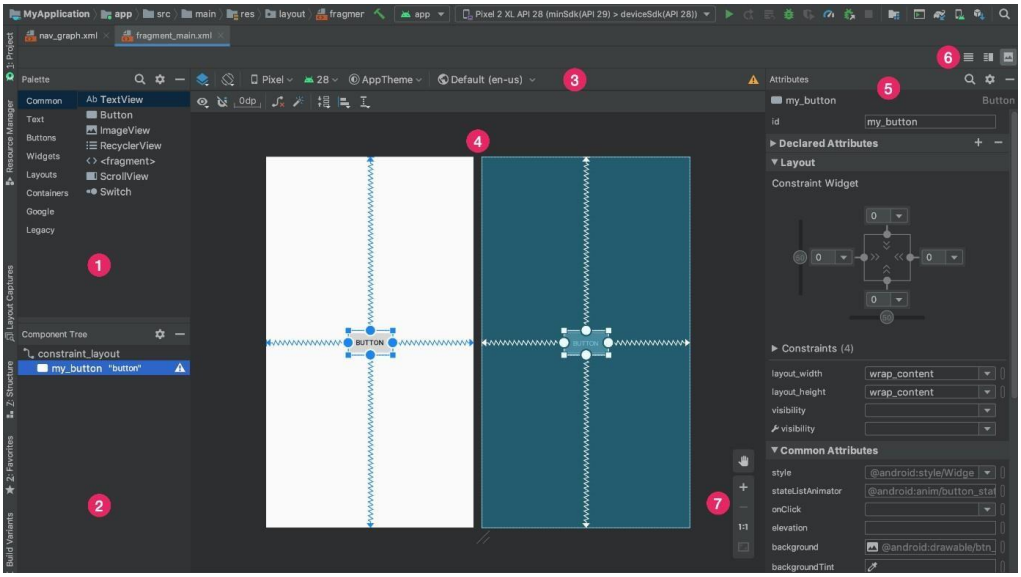



5. Vytvoření uživatelského rozhraní

Rozhraní je vytvořeno v souborech xml. Android Studio umožňuje vytvářet kód přímo nebo pomocí grafického editoru.

Vyberte soubor **activity_main.xml** z adresáře rozvržení. Otevře se nástroj Editor rozvržení (https://developer.android.com/studio/write/layout-editor):

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

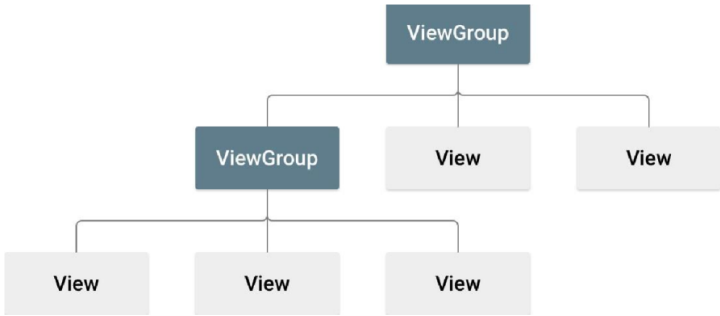



1) **Palette**: Obsahuje různá zobrazení a skupiny zobrazení, která můžete přetáhnout do rozvržení.
2) **Component Tree:** Zobrazuje hierarchii komponent v rozvržení.
3) **Toolbar**: Slouží ke konfiguraci vzhledu rozvržení v editoru a umožňuje měnit atributy rozvržení.
4) **Design editor**: Upravujte rozvržení v zobrazení návrhu, v zobrazení plánu nebo v obou.
5) **Attributes:** Ovládací prvky **atributů** vybraného zobrazení.
6) **View mode**: Zobrazte rozvržení v ikoně režimu kódu, v ikoně režimu návrhu nebo v ikoně režimu rozdělení. Režim Split zobrazuje současně okna Code i Design.
7) **Zoom and pan controls**: Ovládání velikosti a polohy náhledu v editoru.

Další informace o rozhraní naleznete na adrese:
https://developer.android.com/studio/write/layout-editor

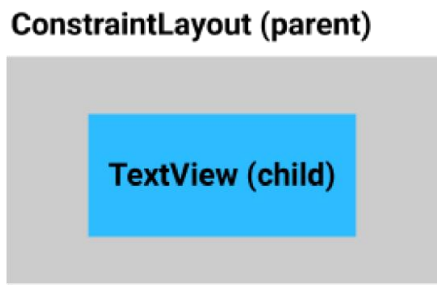
Uživatelské rozhraní je vytvořeno hierarchicky pomocí objektů **ViewGroup** (rozvržení) a **View** (widget).

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




Jak je vidět na obrázku výše, objekty **ViewGroup** umožňují vytvářet hierarchie, v nichž jsou obsaženy jednotlivé objekty.

6. Uživatelské rozhraní vyvíjeného projektu je definováno takto





Soubor xml s definicí rozvržení

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

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

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

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

Pomocí atributů definujeme vzhled objektů. Jednotlivé značky xml (např. TextView) v souboru XML odpovídají názvům tříd, ve kterých jsou tyto objekty definovány.

Podívejte se na značku **ConstraintLayout** a všimněte si, že používá **androidx.constraintlayout.widget.ConstraintLayout** místo pouhého **ConstraintLayout** Je to proto, že je součástí systému Android Jetpack. Android Jetpack má další funkce, které vám usnadní vytváření aplikace. Poznáte, že tato komponenta uživatelského rozhraní je součástí systému Jetpack, protože začíná na "**androidx**".

Cvičení 1

a) Nahraďte "Hello World" svým jménem.
b) Přidejte nový objekt TextView s názvem města, ze kterého pocházíte.
c) Vložte všechny titulky do souboru strings.xml (průvodce - https://developer.android.com/guide/topics/resources/string-resource )

Cvičení 2 (doplňkové) Vyplňte cvičení

na této straně.

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

**Projektová kalkulačka**

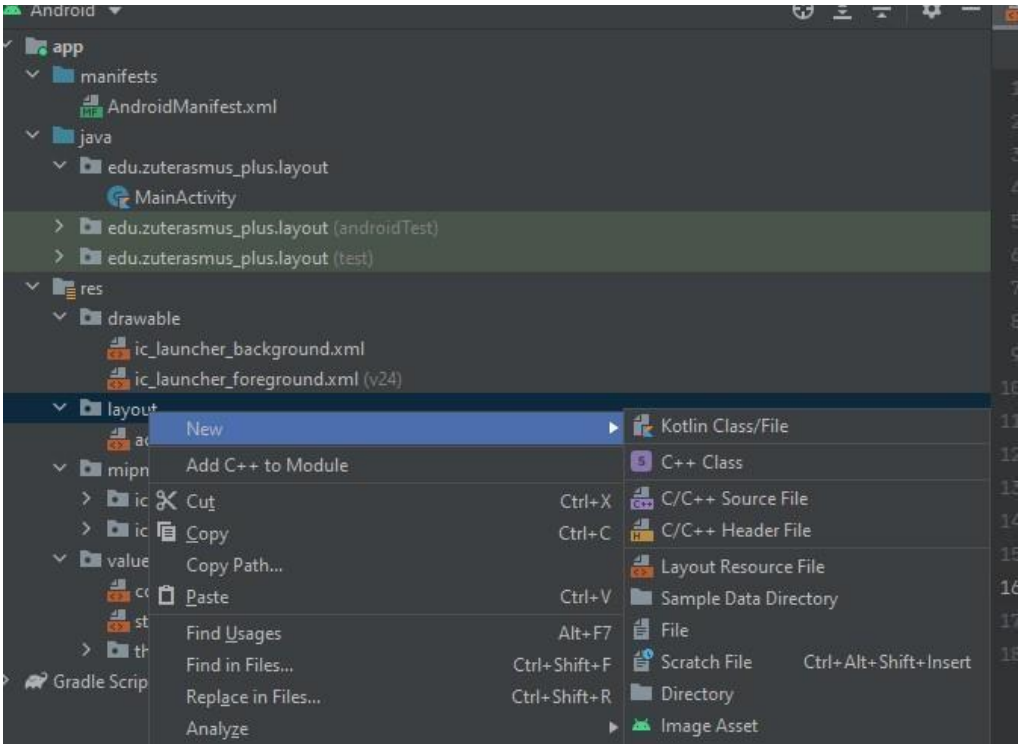
1. vytvořte nové rozložení nebo si stáhněte nový návrh z webu https://github.com/matam/Erasmus_Lab2

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

Pokud projekt stahujete z úložiště, přejděte k bodu č. 2.

Klikněte pravým tlačítkem myši na složku

**rozvržení->Nový->Soubor s rozvržením**




Vytvoření nového souboru rozvržení "**calculator.xml**"


Kopírování obsahu ze souboru:
https://raw.githubusercontent.com/rmaciaszczyk/SummerSchool_Lab1/main/app/src/main/res/layout/calculator.xml

Vytvořte nový soubor styles.xml v adresáři values a zkopírujte obsah z následujícího zdroje

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

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

Nahraďte soubor colors.xml v adresáři values a zkopírujte jeho obsah z následujícího zdroje
https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/colors.xml

Stáhněte ikonu Backspace 24 px a nahrajte ji do adresáře drawable.
https://github.com/rmaciaszczyk/SummerSchool_Lab1/blob/main/app/src/main/res/drawable/ic_baseline_backspace_24.xml
Ikonu můžete importovat pomocí aplikace VectorAsset Studio
https://developer.android.com/studio/write/vector-asset-studio#svg

2. Seznamte se se souborem ***calculator.xml***. Analyzujte jeho strukturu

Cvičení 3

a) Které a kolik objektů **ViewGroup** bylo použito
b) Které a kolik objektů **View** bylo použito

3. Dalším krokem je vytvoření kódu kalkulačky.

Přiřazení rozvržení

a) Přejděte do souboru ***MainActivity.***kt
b) Uvnitř funkce ***onCreate()*** změňte hodnotu

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

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

b) Spuštění aplikace

4. Definice objektů

Nyní je třeba definovat všechna tlačítka, kterým budeme přiřazovat akce.

V jazyce Kotlin musí být všechny proměnné inicializovány nebo je nutné explicitně určit, že mohou nabývat ***nulové hodnoty***. Je také možné určit, že budou inicializovány později před prvním použitím(lateinit).

Například:

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
```

Lokální proměnné určené pouze pro čtení se definují pomocí klíčového slova ***val***. Hodnotu jim lze přiřadit pouze jednou. Proměnné, které lze přiřadit znovu, používají klíčové slovo ***var***.

a) Definice objektů v kódu
Přejděme k definici objektů ve třídě

```
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
```

Pokud jsou názvy tříd podtrženy, měl by být přidán import.
Můžete také použít kombinaci kláves **ALT + ENTER** a automaticky přidat importní

5. Propojení objektů v rozvržení s objekty v kódu

   Nyní je třeba propojit rozvržení objektového formuláře s kódem objektového formuláře.

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

```
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
```

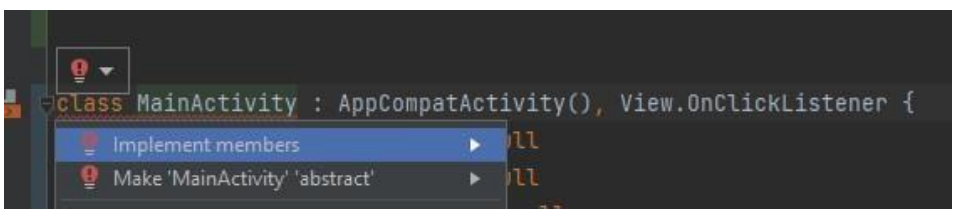
6. Přidání události ***OnClick()*** k tlačítku
   Existuje několik způsobů zpracování události, v tomto případě definujeme globální posluchač události OnClick pro třídu definováním rozhraní.
   a. Přidání rozhraní ***View.OnClickListener*** do definice třídy

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

   b. Definice metody zpětného volání posluchače uvnitř třídy

Přidání rozhraní si vynutí implementaci jeho metod. Klikneme na podtržený název třídy **MainActivity** (červené podtržení znamená chybu), objeví se červená žárovka, po jejímž výběru máme možnost použít rychlé akce. V tomto případě vybereme možnost **Implementovat** členy

Kombinaci kláves **ALT + ENTER** můžete použít také k vyvolání kontextové nabídky.




Po skončení akce dostaneme:

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

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

c. Přiřazení posluchačů tlačítkům (uvnitř funkce **onCreate()** pod **funkcí 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. Dokončení definice posluchače

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
              }
    }
}
```

Výše uvedený kód obsahuje proměnnou, která dosud nebyla definována (**isError)**. Slouží k určení, zda je výraz chybný. Musí být deklarována globálně.

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

Výpočty budou prováděny pomocí knihovny **exp4j** - https://github.com/fasseg/exp4j. Slouží k výpočtu matematických výrazů popsaných jako řetězec. Ve výše uvedeném kódu se výpočet provede, pokud uživatel zvolí tlačítko '=' (R.id.equals), pak se hodnota zadaného řetězce předá metodě ***evaluateExpression(),*** ta využívá zmíněnou knihovnu.

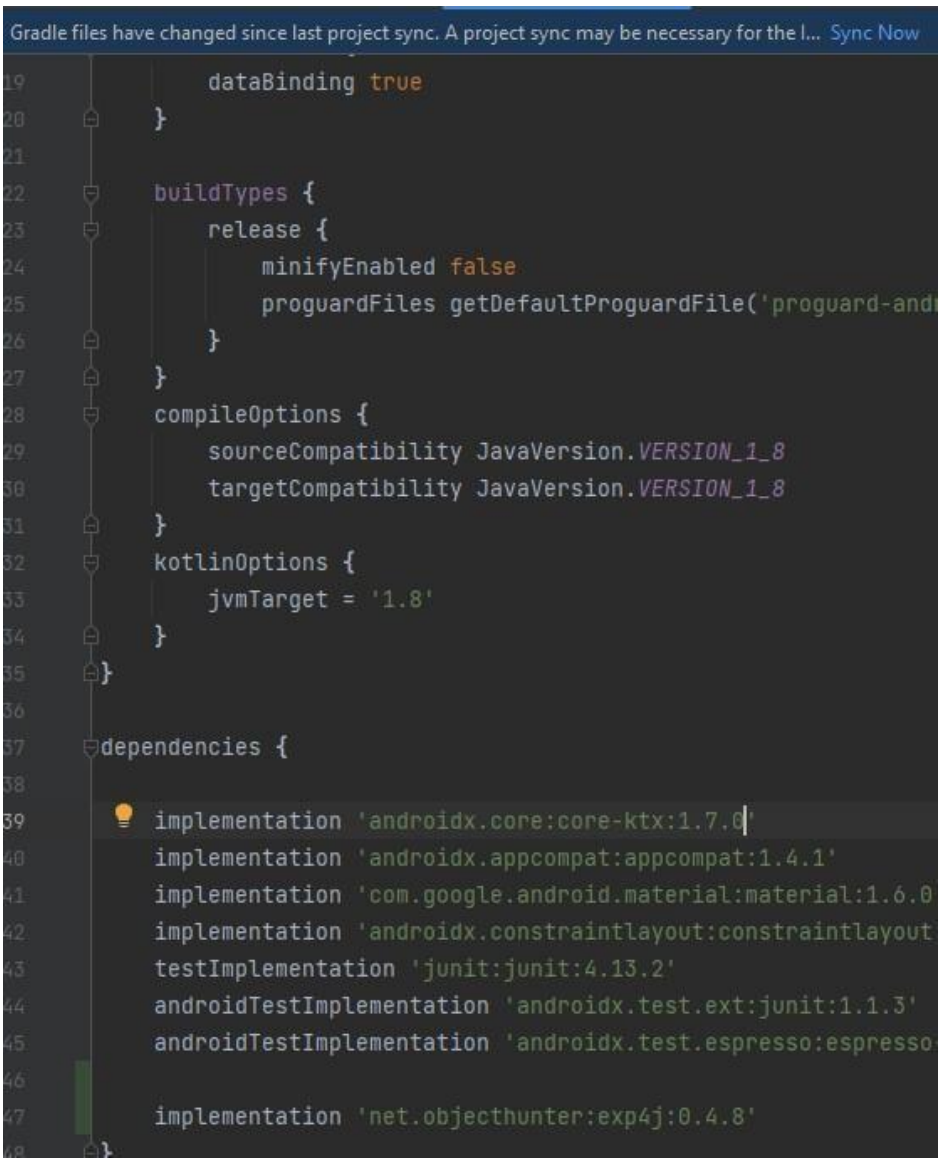
7. Definice metody **evaluateExpression()**,

   a) Přidání knihovny do aplikace

Přejděte do souboru **build.gradle(app)** a do sekce závislostí přidejte položku

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

po kterém by měl být projekt synchronizován. Stiskněte tlačítko : Stiskněte tlačítko ***Synchronizovat*** nyní




b) Přidat import

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

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

c) Vytvoření metody ***evaluateExpression()*** v souboru MainActivity.kt

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

    }

}
```

8. Spuštění aplikace

## View Binding

" View Binding" je funkce, která umožňuje snadněji psát kód, který spolupracuje se zobrazeními. Když je v modulu povolena vazba zobrazení, vygeneruje se pro každý soubor XML s rozvržením, který je v tomto modulu přítomen, třída vazby. Instance třídy binding obsahuje přímé odkazy na všechny pohledy, které mají ID v příslušném rozvržení.

Ve většině případů vazba zobrazení nahrazuje funkci ***findViewById()***.

1. Povolení propojení zobrazení

Chcete-li v modulu povolit vazbu **zobrazení,** nastavte v souboru build**.gradle na** úrovni modulu volbu build **viewBinding na** hodnotu true, jak je uvedeno v **příkladu** níže:

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

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

2. Aplikace

   Pokud je pro modul povolena vazba zobrazení, je pro každý soubor rozvržení XML, který obsahuje modul, vygenerována třída vazby. Každá vazebná třída obsahuje odkazy na hlavní zobrazení a všechna zobrazení, která mají ID. Název vazební třídy se vygeneruje převedením názvu souboru XML na název podle "Pascal Case" a přidáním slova "Binding" na konec.
   Například: calculator.xml -> vygenerovaná třída CalculatorBinding

3. Použití vazby zobrazení v aktivitách
   ViewBinding nám umožňuje nahradit definice všech objektů z rozvržení.

   a) Nahradíme je jedním objektem. V našem kódu vytvoříme objekt (**MainActivity.kt**).

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

Všechny dříve definované objekty související s obrazovkou by měly být z kódu odstraněny.

Chcete-li nakonfigurovat instanci třídy vazeb pro použití s aktivitou, proveďte následující kroky v metodě **onCreate():**

b) Volání statické metody **inflate()** obsažené ve vygenerované třídě vazeb. Tím se vytvoří instance třídy vazeb, kterou bude aktivita používat.

c) Získejte odkaz na kořenový pohled voláním metody **getRoot()** nebo pomocí syntaxe vlastností jazyka Kotlin.

d) Předejte hlavní zobrazení **funkci setContentView(),** aby se stalo aktivním zobrazením na obrazovce.

```
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) Nyní můžeme také odstranit kód, který váže prvky rozvržení (to provádí překladač automaticky). Z kódu odstraníme řádek obsahující volání metody **findViewById()** a všechny deklarované objekty

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

f) Na jednotlivé objekty se nyní budeme odkazovat pomocí vazebního objektu

```
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 měly být změněny v celém kódu

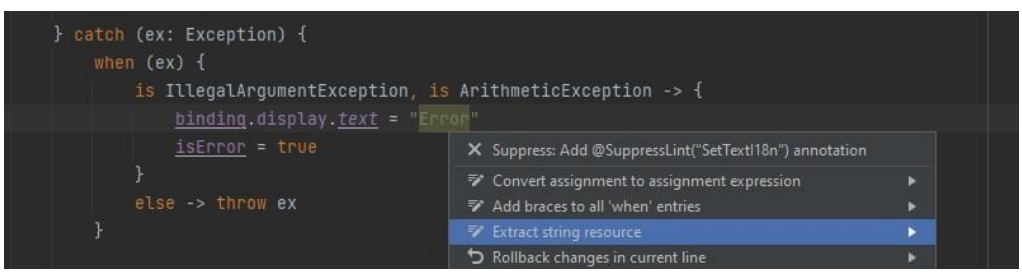
4. Spuštění aplikace
5. Odstranění nepotřebných importů

Po provedení této operace můžete odstranit nepotřebné importy, a to buď ručně, nebo pomocí klávesové zkratky. Chcete-li optimalizovat importy v souboru, můžete také stisknout klávesovou zkratku Ctrl+Alt+Shift+L , vybrat možnost "Optimalizovat importy" a kliknout na tlačítko Spustit. Poznámka: klávesová zkratka se týká přeformátování kódu a umožňuje také "Code Cleaup".

6. Odstranění varování

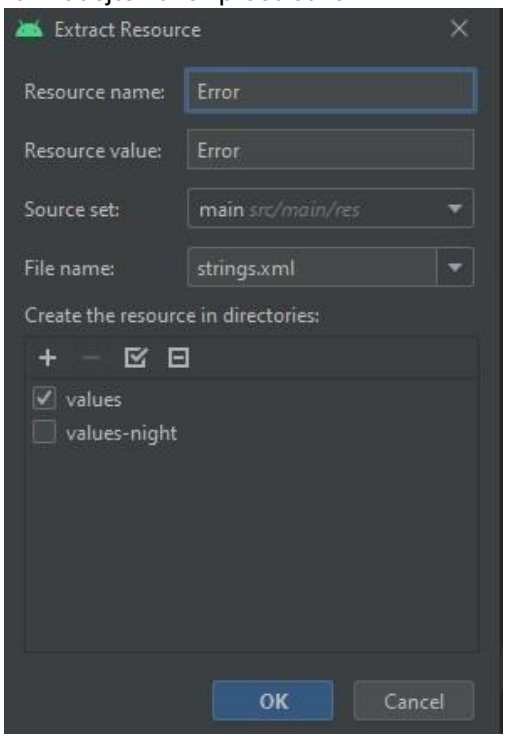
Překladač analyzující kód nám říká, abychom dodržovali správné postupy. Jedním z nich je umístit všechny řetězce do vyhrazeného souboru s prostředky. (**res/values/strings.xml)**. Díky tomuto řešení můžeme naši aplikaci snadno přeložit do jiného jazyka. Další informace (https://developer.android.com/training/basics/supporting devices/languages ).

(a) Vytváření konstant pomocí, AndroidStudio vyzývá:
V metodě evaluateExpression máme natvrdo zadaný objekt String - "Error". Najeďte na tento objekt a pomocí **kláves ALT+Enter** vyberte možnost "**Extract string resource".**




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

Pak zadejte název prostředku




Zákoník v platném znění

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

(b) Vylepšete kód své aplikace tak, aby neobsahoval žádná varování.

Cvičení 4

Otestujte prosím aplikaci. Opravte chybu, když uživatel v aplikaci provede následující akce:

a) Stiskněte tlačítko "2".
b) Stiskněte tlačítko "5".
c) Stiskněte tlačítko "/"

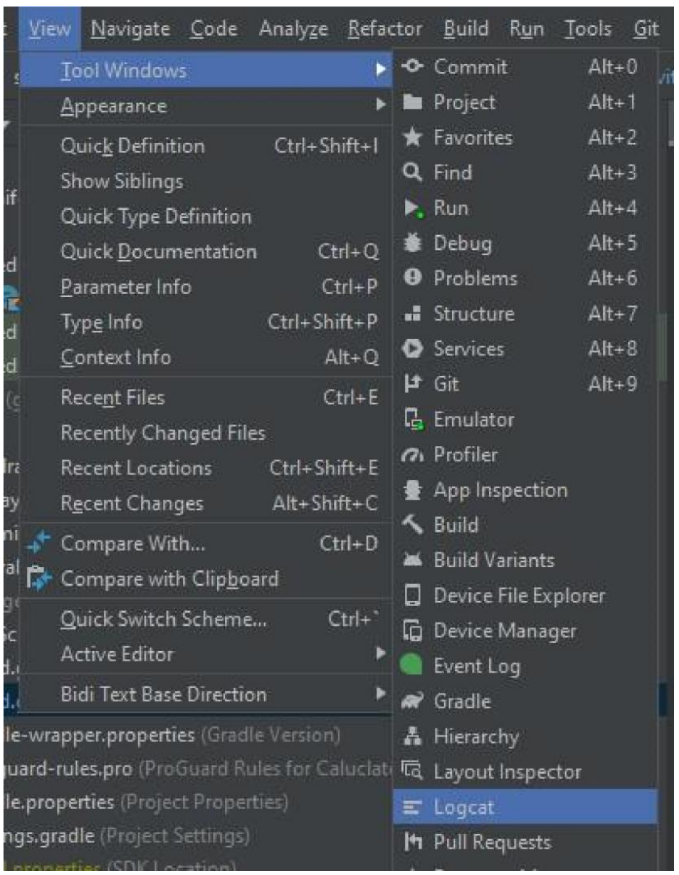
Co-funded by the
Erasmus+ Programme
of the European Union

d) Stiskněte tlačítko "="

e) Stiskněte tlačítko "="

☹ ☹ ☹

Chcete-li zjistit, co je příčinou chyby, použijte nástroj Logcat.



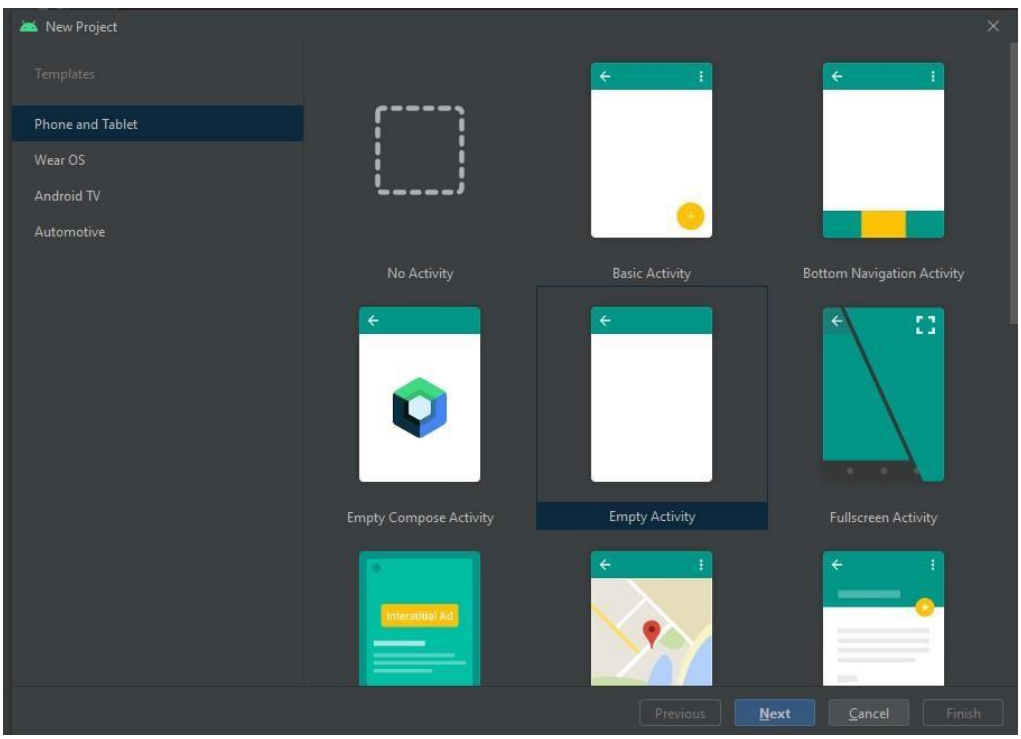

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab2.

# Senzory

Většina mobilních zařízení má zabudované senzory, které měří pohyb, orientaci a různé podmínky prostředí. Tyto senzory jsou schopny poskytovat nezpracovaná data s vysokou přesností a precizností. Jsou užitečné, pokud chcete sledovat 3D pohyb nebo polohu zařízení. Umožňují sledovat změny prostředí v blízkosti zařízení. Například hra může sledovat údaje z gravitačního snímače zařízení a odvodit z nich složitá gesta a pohyby uživatele(naklánění, třesení, otáčení nebo kývání). Podobně může aplikace pro předpověď počasí využívat snímač teploty a snímač vlhkosti zařízení k výpočtu a hlášení rosného bodu nebo aplikace pro cestování může využívat snímač geomagnetického pole nebo akcelerometr k určení směru pohybu.

Během těchto cvičení bude vytvořena aplikace pro čtení senzorů, která bude následně přestavěna tak, aby používala návrhový vzor MVVM.

1. Vytvořit nový projekt -> Prázdná aktivita




2. ***Definujte*** název aplikace (***Lab5)*** a balíček (***edu.zut.erasmus_plus.sensors),*** vyberte minimální API (***API28)***.

3. Přečtěte si kód
    1. vyhledejte a analyzujte soubory: **MainActivity.kt, activity_main.xml, AndroidManifest.**xml**.**
4. Přejděte do souboru build.gradle -> Aktualizace všech závislostí a knihoven pro projekt a modul (můžeme přeskočit).

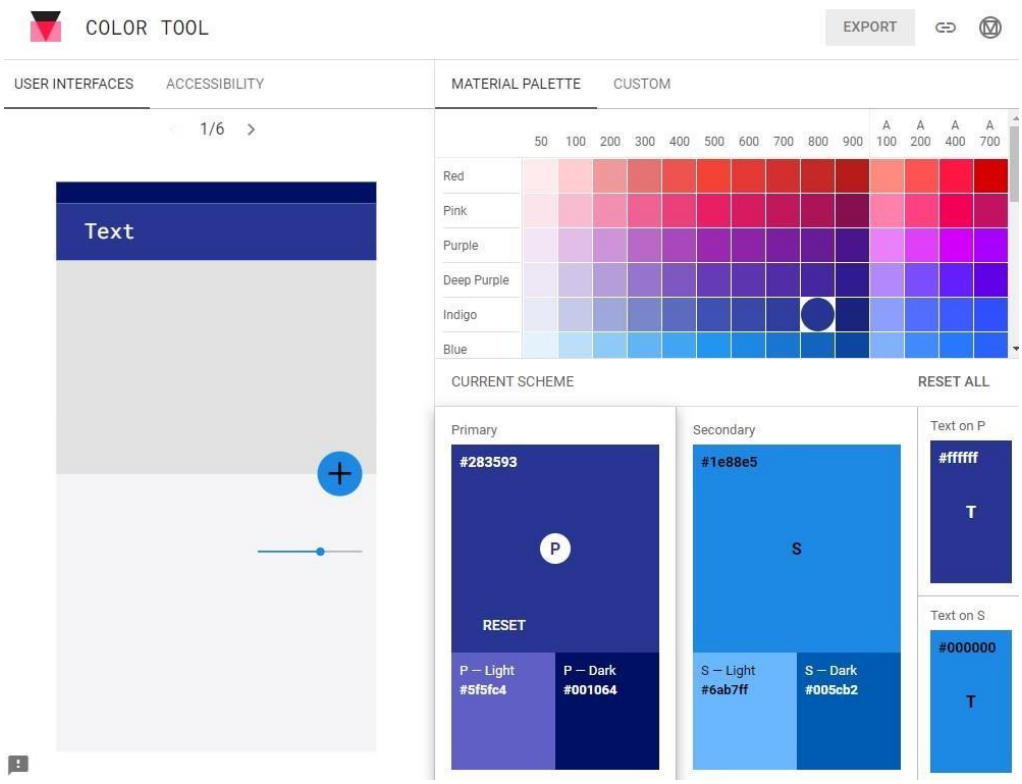
5. Spuštění aplikace

6. Vytvoření barevného schématu (volitelné)

    Před definováním rozvržení použijte nástroj Barva - Material Design a definujte barvy aplikace, které pak použijte při definování prvků.
    Příklad barvy:
    https://material.io/resources/color/#!/?view.left=0&view.right=0&primary.colour=283593&secondary.colour=1E88E5

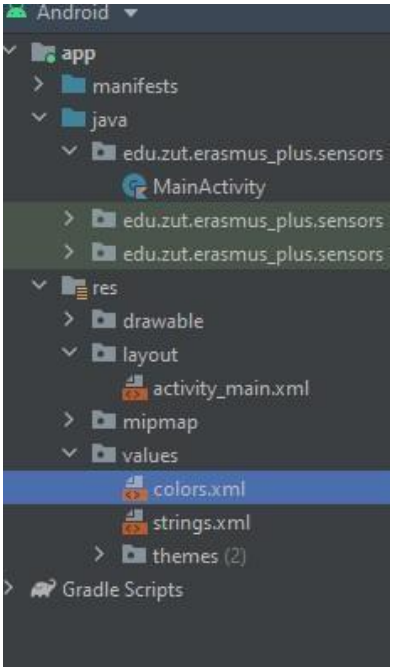
    Další informace o barvách:

    https://material.io/design/color/the-color-system.html#color-theme-creation



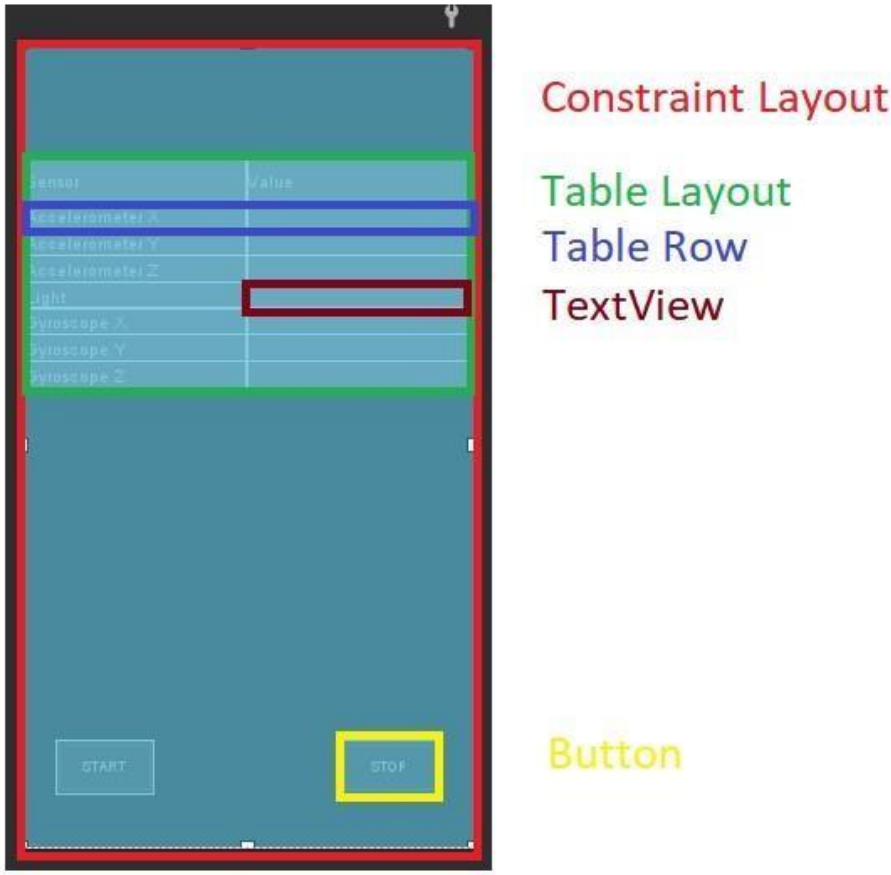

Po výběru vhodného barevného schématu přejděte na kartu DOSTUPNOST a zkontrolujte upozornění. Dalším krokem je export konfigurace. Vyberte tlačítko EXPORT v horní části obrazovky a uložte soubor colors.xml.

V projektu otevřete soubor colours.xml a vložte hodnotu ze staženého souboru.




7. návrh grafického rozhraní

Rozložení navrhněte takto:




Toto rozhraní je vytvořeno pomocí dvou typů rozvržení, ConstraintLayout a TableLayout.

Náčrt rozhraní - bude doplněno

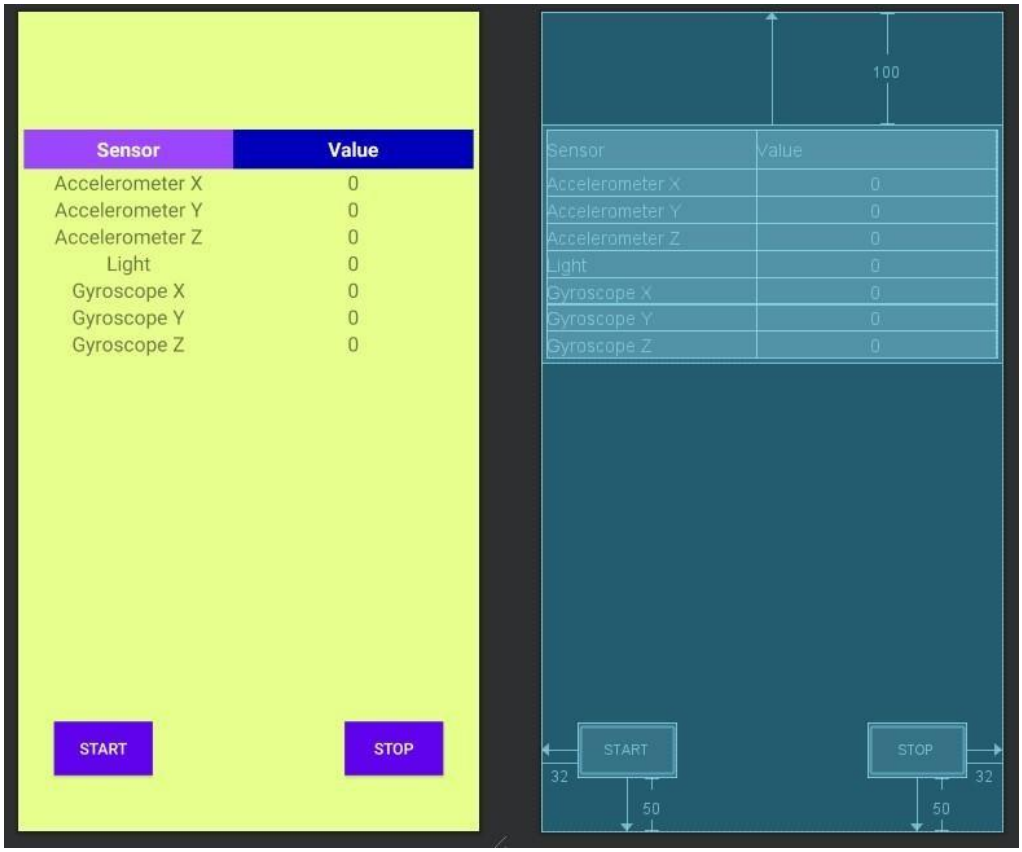
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    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"
        tools:layout_editor_absoluteX="16dp">
```

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

```
    <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. Přejděte do **souboru MainActivity.kt** a do funkce **onCreate()** přidejte 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)
```

Předtím definujte objekty jako globální pro danou třídu, např.

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

9. Přidání rozhraní ***SensorEventListener*** do třídy MainActivity pro příjem událostí ze senzorů.

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

10. Přidání metod zpětného volání ***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. Zkontrolujte, zda se názvy objektů v kódu (R.id.XXXX) shodují s názvy ID v rozložení.
12. Definice pomocných metod

```
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 přidejte také proměnnou **resume**

```
private var resume = false
```

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

13. Pomocí předchozích metod lze zaregistrovat a odregistrovat funkci SensorsListener.

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

    registerListener()
}

override fun onPause() {
    super.onPause()

    unRegisterListener()
}
```

Proč se registrujeme a odhlašujeme?

14. Přidání metody ***onClick()*** (definované také v rozvržení)

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

15. Spuštění aplikace

Další úkol

16. Definování různých barev v závislosti na stavu tlačítka.

Vytvořte soubor background_button.xml a přidejte 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. Změna definice pozadí pro každé tlačítko

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

18. Spuštění aplikace

## MVVM, LiveData

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

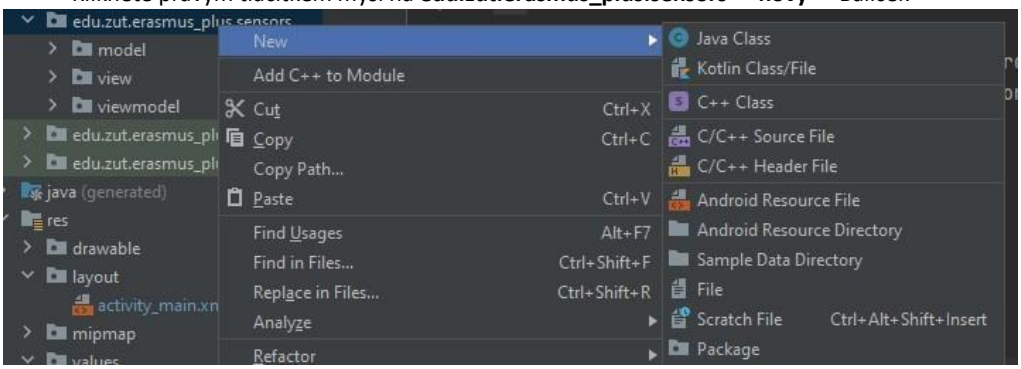
Výše uvedená aplikace demonstruje základní přístup k programování pro Android. V současné době se doporučuje vyvíjet aplikace pro Android pomocí návrhového vzoru MVVM (Model - View - ViewModel).

Následuje řešení využívající komponenty architektury představené v knihovně AndroidX.

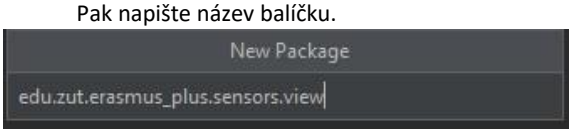
Z aplikace, kterou jsme právě vytvořili, chceme vytvořit aplikaci podle návrhového vzoru MVVM.

1. Příprava balíčků
   - **edu.zut.erasmus_plus.sensors.viewmodel**
   - **edu.zut.erasmus_plus.sensors.view**
   - **edu.zut.erasmus_plus.sensors.model**

   Klikněte pravým tlačítkem myši na **edu.zut.erasmus_plus.sensors -> nový ->** Balíček




   Pak napište název balíčku.




2. přesuňte soubor **MainActivity.kt** do souboru **edu.zut.erasmus_plus.sensors.**view

Všechny aktivity a fragmenty jsou v modelu MVVM klasifikovány jako pohled.

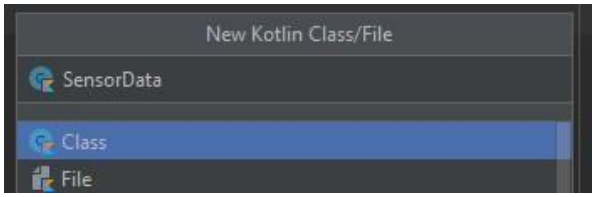
3. Vytvoření datové třídy https://kotlinlang.org/docs/data-classes.html

Datová třída - je třída, která obsahuje pouze pole a hrubé metody pro přístup k nim (gettery a settery). Jsou to prostě kontejnery pro data používaná jinými třídami. Tyto třídy neobsahují žádné další funkce a nemohou samostatně pracovat s daty, která uchovávají.

   a) Vytvořte soubor SensorData.kt uvnitř **edu.zut.erasmus_plus.sensors.**model.

   Klikněte pravým tlačítkem myši na ***model->Nový-> Třída/soubor jazyka Kotlin***


   Uveďte jméno




b) Definice třídy

Uvnitř souboru SensorData.kt definujte všechny proměnné

```
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. Vazba dat.

Mechanismus, který rozšiřuje a zároveň nahrazuje funkci View Binding. Umožňuje oboustrannou vazbu. Podrobné informace: https://developer.android.com/topic/libraries/data-binding

Po vytvoření rozhraní je třeba s kódem svázat existující objekty. V současné době se doporučuje použít metodu Data Binding. Kromě automatického vytvoření kódu umožňuje také aktualizovat propojený pohled při změně dat pomocí **LiveData** (vzor Observer). Knihovna Data Binding byla vytvořena s ohledem na pozorovatelnost. Vzor, který se stal ve vývoji mobilních aplikací poměrně populární. Pozorovatelnost je doplněk datové vazby, jejíž základní koncept zohledňuje pouze objekty pohledu a dat. Nicméně právě prostřednictvím tohoto vzoru mohou data automaticky propagovat své změny do zobrazení. Tím odpadá nutnost ručně aktualizovat zobrazení pokaždé, když jsou k dispozici nová data, což zjednodušuje a snižuje počet řádků kódu.
Naše aplikace bude používat dříve vytvořenou datovou třídu pro vysílání informací o změnách do senzorů.

5. Povolení datové vazby

Otevřete soubor build.gradle aplikace a přidejte tento řádek do značky android.

```
android {
    compileSdk 31

    dataBinding {
        enabled true
    }
```

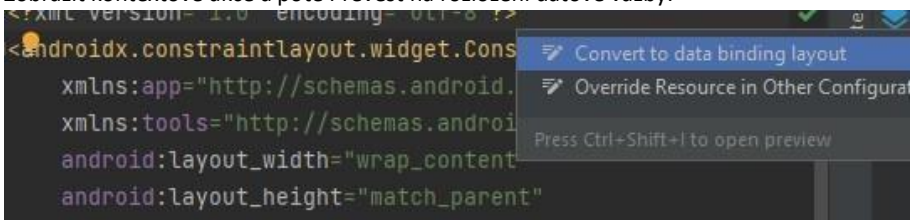
Co-funded by the
Erasmus+ Programme
of the European Union

Propojení dat v zobrazení

Převod běžného rozvržení na rozvržení pomocí funkce Data Binding:

1. Rozložení zabalte do značky <layout>
2. Přidání proměnných rozvržení (nepovinné)
3. Přidání výrazů rozvržení (nepovinné)

Neb
o
Android Studio nabízí automatický převod: Klepněte pravým tlačítkem myši na kořenový prvek, vyberte Zobrazit kontextové akce a poté Převést na rozložení datové vazby:




```
<?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 obsahovat proměnné rozvržení. Hodnoty do nich budeme přidávat později.

Proměnné rozvržení se používají k zápisu **výrazů** rozvržení. Výrazy rozvržení se umísťují do hodnot atributů prvků a používají formát *@{výraz}*. Níže jsou uvedeny příklady výrazů (nepište je prosím v laboratoři):

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

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

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

Další informace o rozložení výrazů
https://developer.android.com/topic/libraries/databinding/expressions#expression_language

Přidání závislostí na LiveData v souboru 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"
```

6. Vytvoření třídy pro čtení hodnot ze senzorů

Nyní přesuneme veškerý kód zodpovědný za sběr hodnot pro senzory do samostatné třídy.

a) Vytvořte soubor **SensorDataLiveData.kt** uvnitř ***edu.zut.erasmus_plus.sensors.model.***
b) Přesun kódu z Activity_Main.kt (všechny funkce kromě **onCreate()**)
c) Odstranění nadbytečných objektů ze třídy a kódu odpovědného za vytváření instancí Správce senzorů a Senzorů.

Po těchto krocích vypadá soubor 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 veškerý stávající kód byl odstraněn.

Třída **SensorDataLiveData** vypadá následovně:

```
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(
```

Tato třída je nyní zodpovědná za obsluhu senzorů. Naším cílem je, aby každá změna byla komunikována prostřednictvím pozorovatelů. K tomuto účelu použijeme třídu **LiveData**. Naše třída ji rozšiřuje, a bude tedy pozorovat změny pomocí modelu **SensorData**. Definice třídy má následující podobu.

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

Pomocí metod třídy LiveData (**onActive()** a **onInactive()**) může zahájit nebo ukončit sběr událostí senzoru. Metoda **onActive() se** spustí, když se první pozorovatel připojí k naší třídě, a druhá, když se poslední pozorovatel odpojí.
Zbytek kódu odpovídá tomu, co bylo dříve implementováno v metodě **MainActivity(), s** tím rozdílem, že v metodě **onSensorChanged()** vrací objekt ***value*** výsledek. V našem případě je to objekt třídy **SensorData** obsahující poslední hodnoty z odečtů ze snímačů.

8. Vytvoření ViewModelu

Třída ViewModel je určena k ukládání a správě dat souvisejících s uživatelským rozhraním způsobem zohledňujícím životní cyklus. Třída ViewModel umožňuje, aby data přežila změny konfigurace, například otočení obrazovky.

Vytvoření souboru SensorViewModel.kt uvnitř **edu.zut.erasmus_plus.sensors.viewmodel**

Uvnitř souboru přidáme dvě soukromé proměnné a jejich metody get**.**

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

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

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

První proměnná slouží ke čtení hodnot senzorů z dříve vytvořené třídy SensorDataLiveData, kde je nutné předat kontext aplikace. Proto upravíme definici třídy, jak je uvedeno výše. Všimněte si, že naše třída dědí od třídy AndroidViewModel.

Proměnná **_pauseReading** určuje, zda přestaneme/začneme číst senzory. Je třeba ji inicializovat, proto do modelu SensorViewModel přidáme následující kód:

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

Poslední metodou, kterou je třeba doplnit, je správná reakce na údaje snímače stop/start. Do modelu SensorViewModel přidejte následující funkci:

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

9. přeskupení

Po vytvoření virtuálního počítače musíme změnit rozložení a přidat odkaz na ViewModel.

a) Přidání informací o proměnné

Otevřete soubor activity_mail.xml

Uvnitř vlastnosti **<data> </data>** vložte novou proměnnou

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

Přidání proměnné (**sensorViewModel)** umožní použití objektů třídy **SensorViewModel** v souboru rozhraní.

b) Změna vlastností v souboru activity_main.xml

Najděte objekt **TextView, který** je zodpovědný za zobrazení hodnoty **acc_X** (kolem řádku 64), a změňte vlastnost **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" />
```

Změna vlastnosti **android:text pro** všechny objekty představující hodnotu snímání senzoru (7krát)

c) Změňte vlastnosti ***onClick*** a ***enable*** pro tlačítko takto.

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

Výše jsou uvedeny pouze změněné výpisy.

10. Změny aktivity - MainActivity.kt
    a) Otevřete soubor MainActivity.kt
    b) Uvnitř onCreate změňte 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
    }
}
```

Všimněte si, že jsme přidali dva objekty, **binding** a **sensorViewModel**. Objekt **binding** používá **DataBinding** a poskytuje nám automatický přístup k proměnným v rozvržení, aniž bychom museli používat funkci **findViewById()**.

Objekt **sensorViewModel** váže vytvořenou třídu ViewModel k zobrazení.

11. Spuštění aplikace

Výše uvedené kroky nám umožnily dodržet nyní doporučovaný model vývoje aplikací pomocí komponent systému Android definovaných v knihovně androidX.

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab3-4.

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

# Použití databáze a 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 laboratória obsahujú nasledujúce prvky:

1. Vytvořte novou aplikaci a nastavte build.grade
2. Vytvoření datového modelu
3. Vytvoření instance databáze
4. Vytvoření ovládacího prvku RecyclerView
5. Související danyvh z databáze s RecyclerView

1. Create New project -> Empty Activity

2. Definovat App Name (***Lab4***) a název balíčku (***edu.zut.wi.erasmus.lab4***), vybrat minimum API (***API28***)

3. Přidání 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 do pluginuu

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

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

4. Definování nového balíčku: database(***edu.zut.wi.erasmus.lab4.database***)
5. Přidání nové třídy do tohoto balíčku ***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. Vytvářet 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. Implementovat 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
```

```
            // 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. Implementovat Repository – 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. Stnasvětě nový balíček: viewmodel(***edu.zut.wi.erasmus.lab4.viewmodel***)
10. Implementovat 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)
    }
}
```

```
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 Implementace RecyclerView

11. Vytvoření nového rozložení pro 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. Zvýšit *activity_main_xml* kód pro RecyclerView a 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. Vytvoření nového balíčku : adapter(***edu.zut.wi.erasmus.lab4.adapter***), Vytvořte novou třídu 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. Přidání volání adaptéru MainActivity.kt
    Níže vložte kód ***setContentView***

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

15. Vytvoření instance ***Application*** a v tomto případě instanci databáze a úložiště, třída ***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. vyměnit ***AndroidManifest.xml*** a přidat vnitřní značku <application … kód níže

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

17. Spuštění aplikace
18. Přidání nové aktivity pro přidání nových knih– NewBookActivity.kt – vytvořit pomocí průvodce

```
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. Přizpůsobení rozložení – 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"
```

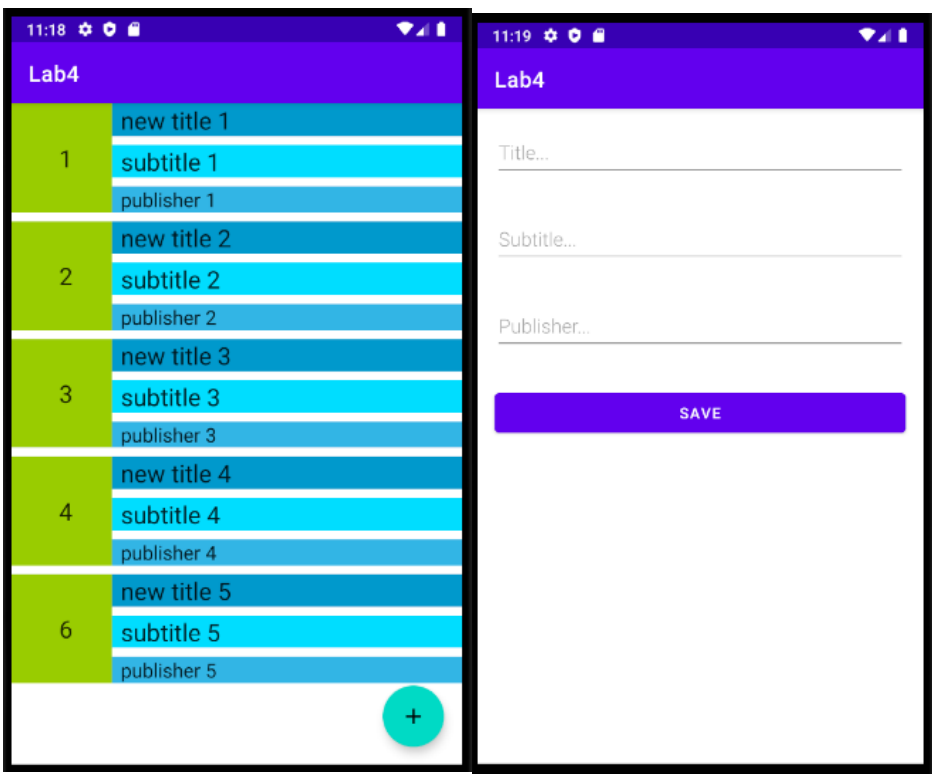
```
        android:layout_margin="@dimen/big_padding"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_publisher"
        android:inputType="textAutoComplete"
        android:minHeight="@dimen/min_height"
        android:textSize="18sp"
        android:autofillHints="@string/hint_publisher"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

    <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. Přidat kód pro FloatingActionButton – MainActivity.kt

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

21. Spuštění aplikace

Konečný pohled na žádost




Main Activity New Book Activity

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

Další informace: https://developer.android.com/codelabs/android-room-with-a-view-kotlin

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab5.

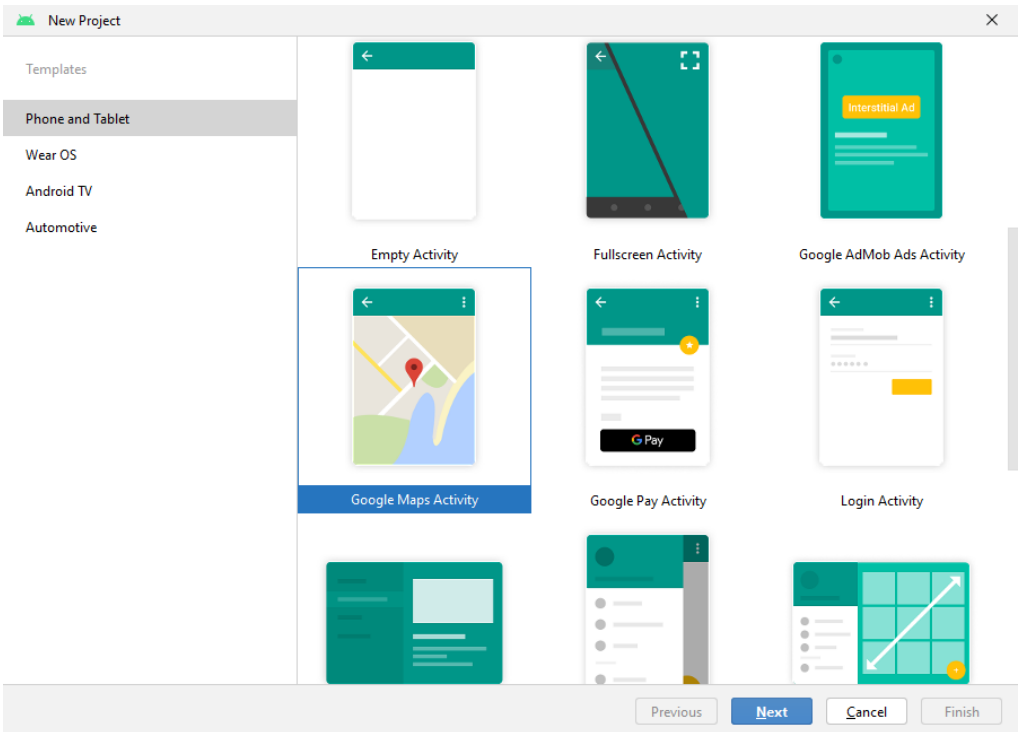
Co-funded by the
Erasmus+ Programme
of the European Union

# Localization

Lokalizace poskytuje možnost vytvořit aplikaci, která zohledňuje kontext polohy uživatele. Umožňuje vytvářet obsah, který se bude dynamicky měnit v závislosti na poloze zařízení. Lokalizace také umožňuje vývojářům shromažďovat statistiky, usnadňuje profilování, ale existuje určité riziko ohrožení soukromí. Přístup k poloze vyžaduje, aby uživatel udělil aplikaci oprávnění.

Tento dokument popisuje aplikaci, která průběžně nebo jednorázově zobrazuje polohu uživatele. Dále bude popsán proces udělování a ověřování oprávnění. Aplikace používá hotový návrh, který zobrazuje mapu.

1. Vytvoření projektu pomocí průvodce

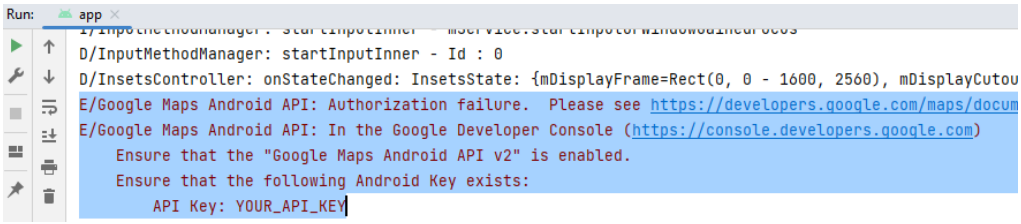



Vyberte aktivitu Mapy Google.

2. Seznamte se s kódem. Všimněte si, jaká rozhraní jsou implementována, a analyzujte vytvořené metody.
3. Run the application.
   Po spuštění by se měla zobrazit mapa se značkou v Sydney. Nezobrazí se však kvůli chybějícímu platnému klíči API.

V aplikaci Run máme informaci o absenci platného klíče.

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




4. Přidání správného klíče k projektu: Přesné informace naleznete na této adrese

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

   a. Vložte svůj API_KEY do souboru AndroidManifest.xml.
   b. Konfigurace API_KEY - přidejte Android Maps

5. Teraz prejdeme k pridaniu zobrazenia polohy zariadenia do aplikácie.
6. Na určenie polohy budeme používať služby Google Play odporúčané spoločnosťou Google.

   Jak přidat "Google Play Services" - https://developers.google.com/android/guides/setup

Přidání závislostí do souboru build.gradle (vyberte nejnovější verzi)

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

...

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

7. Změníme grafický návrh souboru **activity_main.xml** a přidáme do něj dvě tlačítka.

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 chyb přidáním příslušných definic do souboru **strings.xml** (get_position-”Get Position”, start_loop -> „Start Loop”), také přidat novou definicji stop_loop ->”Stop_Loop”

8. Získání poslední známé polohy - https://developer.android.com/training/location/retrieve-current
   Prvním krokem bude přidání kódu pro určení poslední známé polohy. Tuto funkci připojíme k tlačítku " Get Position ", metoda zpracování se bude jmenovat **getLastPos()**.
   Tuto metodu přidáme do hlavního souboru 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 přidejte definici objektu ve třídě ***MapsActivity***

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

Objekt by měl být inicializován. Do metody ***onCreate()*** přidejte následující kód.

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

Výše uvedený kód neobsahuje kontrolu oprávnění. Podle pravidel je před každým použitím funkce, která musí mít oprávnění od uživatele, vyžadována kontrola oprávnění.

9. Zadání oprávnění aplikace
   Prvním krokem je přidat do souboru manifest informace o tom, jaká oprávnění jsou nutná pro spuštění aplikace. V našem případě přidáme hrubá i jemná oprávnění. Všimněte si, do které části manifestu jsou oprávnění přidána.

```
<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. Nyní můžete přejít k vytvoření funkce, která bude ověřovat nároky.

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")
        }
    }
```

Výše uvedená metoda je nejjednodušší a kontroluje oprávnění pouze jednou, v případě trvalého odmítnutí uživatele tuto situaci neřešte. Jako další úkol proveďte úplný cyklus dotazů na povolení podle: https://developer.android.com/guide/topics/permissions/overview#workflow

11. Nyní je možné upravit metodu ***getLastPos()*** a zakomentovat kontrolu oprávnění.
12. Spuštění aplikace
13. Nyní přidáme podporu pro průběžné aktualizace položek. Budeme také používat služby Google Play. Úkolem této funkce bude průběžně aktualizovat značku polohy na mapě. Za tímto účelem bude vytvořena návratová metoda, jejímž úkolem bude aktualizovat pozici při přijetí nové pozice. V ***MapsActivity()*** musíme definovat následující objekty, které budou k dispozici všem metodám třídy.

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

14. V metodě ***onCreate()*** nyní inicializujeme metody pro umístění.
    První objekt slouží k určení parametrů umístění, zatímco druhý definuje návratové metody.
    Všimněte si frekvence aktualizací.

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. Nyní můžeme zavolat požadavek na polohu, což provedeme v metodě ***startStopRequestLocation()***, která byla vytvořena pro obsluhu druhého tlačítka.
    Nejprve zkontrolujeme oprávnění a poté na základě stavu ověříme, zda spustit úlohu aktualizace polohy, nebo ji zastavit. V našem případě pouze přidáme bod do mapy. Následující kód lze optimalizovat vytvořením společných metod přidávání

```
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

Úkol:

1. Proveďte proces přiřazování podle doporučeného pracovního postupu.
2. Zobrazení pouze posledních 5 značek na mapě.

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

# Síťování

Při vývoji aplikací je často potřeba používat síťová připojení. Síťové požadavky se používají ke stahování nebo aktualizaci dat. Používání sítě může generovat značné náklady a je také určitou hrozbou pro soukromí uživatele, proto je nutné povolení k používání sítě a provádění síťových operací a čtení stavu sítě v aplikaci musí váš manifest obsahovat oprávnění. Provádění síťových operací vyžaduje, aby byly dokončeny v samostatném vlákně a nezatěžovaly hlavní vlákno. Před připojením je vhodné zkontrolovat, zda je zařízení připojeno k internetu. Kontrola sítě musí být provedena před každou operací stahování.

V současné době většina aplikací používá k načítání dat rozhraní REST API dané služby. Někdy je však nutné načíst soubor ze serveru tradičním způsobem. Pak se dává přednost protokolu HTTP/HTTPS.

V této laboratoři vyvíjíme program, který stahuje obrázky pomocí rozhraní API NASA. (Astronomy Picture of the day) - https://api.nasa.gov/#browseAPI .

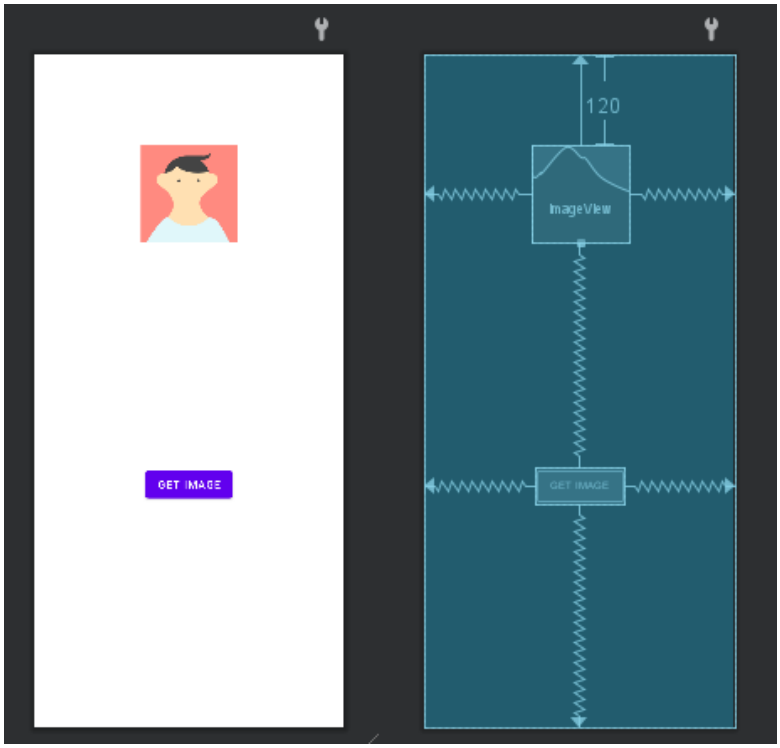
1. K realizaci této laboratoře potřebujeme generovat API KEY https://api.nasa.gov/#signUp .
   Zaregistrujte se a uložte přijaté **API_KEY**.

2. Vytvoření nového projektu -> “Empty Activity”

3. Definujte název aplikace (***Lab7***) a balíčku (***edu.zut.erasmus_plus.networking***), vybrat minimální API (***API28***)
4. Prozkoumejte kód
   - Vyhledat MainActivity.kt, activity_main.xml, AndroidManifest.xml
5. Přejděte na “build.gradle -> Upgrade all dependencies and libraries for Project and Module” (můžeme přeskočit)

6. Takovéto rozvržení návrhu:

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




Aplikace bude obsahovat dva prvky, tlačítko pro stažení dat a fotografii staženou z internetu.

7. Spustit aplikaci

8. Definice oprávnění v souboru AndroidManifest.xml

Pokud chcete používat Internet, musíte definovat příslušné oprávnění; kromě toho musíte pro kontrolu stavu sítě také definovat oprávnění.

```
<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>
```

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

9. Vytvoření metody pro kontrolu síťového připojení

Následující kód bude užitečný pro kontrolu, zda je k dispozici připojení k internetu. Tuto metodu použijte před získáním souboru 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 knihovna pro Android a Javu, která vyniká při načítání a odesílání strukturovaných dat, například JSON a XML. Tato knihovna provádí požadavky HTTP pomocí ***OkHttp***, další knihovny od společnosti Square. Pomocí této knihovny definujeme službu, třídu dat a úložiště.

Použití knihovny se skládá z definice dat (datová třída), služby definující dotazy (rozhraní) a metody volání. Nejprve 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. Definice datové třídy

Data z rozhraní API budou strukturovaná a vrácena ve formátu JSON. Vytvoření datové třídy usnadní používání těchto dat. Popis jednotlivých polí je uveden v části

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

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

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

13. Definujte servis

Stažení obrázku proběhne ve dvou fázích, nejprve se stáhne objekt popisující obrázek dne a adresa, odkud lze obrázek stáhnout. Druhá fáze zahrnuje stažení obrázku. Proto služba obsahuje dvě definované metody stahování.

```
//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. Používání Retrofit

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

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

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

    })
}
```

Použití vytvořené služby spočívá v definování vlastní instance služby a odkazu na definovanou metodu v tomto případě ***getApodItem***

Poté následuje volání metody ***enqueue()***, čímž se odešle síťový dotaz. Výsledek dotazu je přijat ve zpětných voláních metod ***onResponse()*** nebo ***onFailure()***, v závislosti na výsledku. Ve výše uvedeném kódu jsou dvě metody, jedna načítá metadata fotografie a druhá načítá fotografii

.

15. Přidáme kód pro obsluhu tlačítka a zobrazení obrázku (**onCreate**())

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

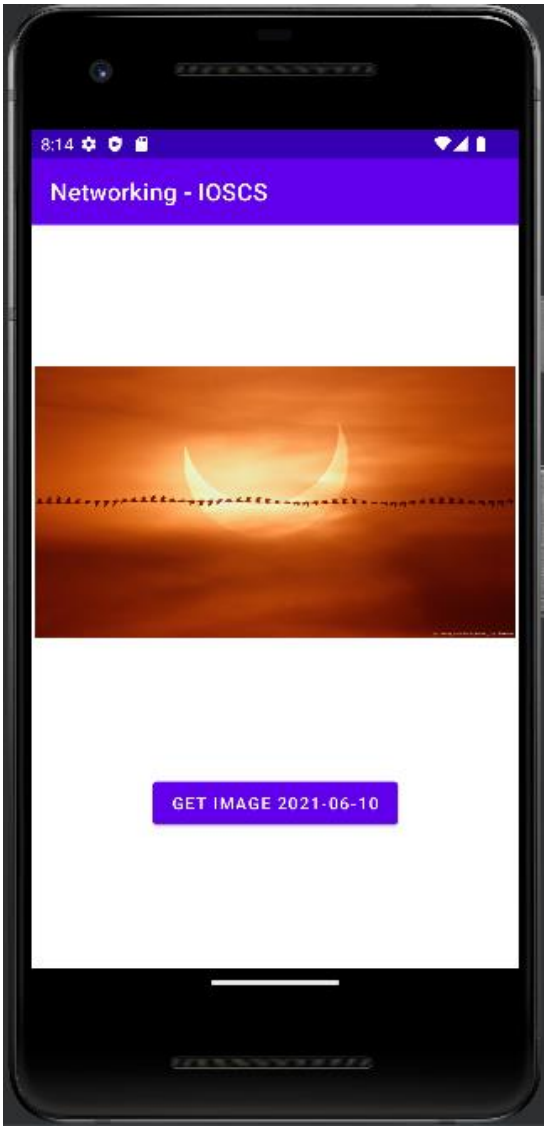
```
button = findViewById(R.id.button)
button.setText("Get Image " + formattedDate)

imageView = findViewById(R.id.imageView)

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

16. A metoda, která převede výsledný datový tok na obraz.

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

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

17. Aplikace Run

Aplikace tak může vypadat následovně:




Otázka

- Je nutné žádat uživatele o povolení používat internet?
- Jsou definovaná oprávnění typu "nebezpečné"?

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab7.

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