Розробка на Android

Весняний семестр 2024

МП-31 Захаров Дмитро *Викладач: Величко О.М.*

Підсумкова Робота

§ Підсумкова робота §

(1) Вступ Цей документ дещо неформально описує процес розробки підсумкового міні-проекту, які інструменти було використано та короткий аналіз коду. Також тут ми наводимо майже повний перелік коду. Щоб погратися з самим додатком, можна перейти за наступним посиланням на *GitHub* репозиторій (також там є скріншоти додатку):

`https://github.com/ZamDimon/android-course`

Умова завдання. Побудувати графік функції

$$y(t) = \sin \omega t + \cos \omega t,$$

де користувач може налаштовувати параметри циклічної частоти ω, а також відрізок $t \in [t_0, t_1]$, на якому малювати графік.

(2) Реалізація базової задачі Як і було запропоновано в основному курсі лекцій, ми використали пакет `androidplot`. Також, щоб було трошки легше жити, ми додали пакет `androidx.databinding:databinding-compiler`, котрий дозволяє викликати компоненти layout'а не через `findViewById`, а напряму, використовуючи `binding`. Далі опишемо основні частини додатку.

1. Layout. Наш layout складається з наступних 6 основних компонент:

- `topAppBar` – бар, на якому знаходиться назва додатку, а також кнопки зберегти, імпортувати та поставити дефолтні значення.
- `plot` – власне наш графік, на якому ми будемо малювати синусоїду.
- `frequency_slider_label` та `frequency_slider` – слайдер з підписом зверху, на якому ми будемо обирати циклічну частоту.
- `x_limits_slider_label` та `x_limits_slider` – слайдер з межами по вісі x (або часу).

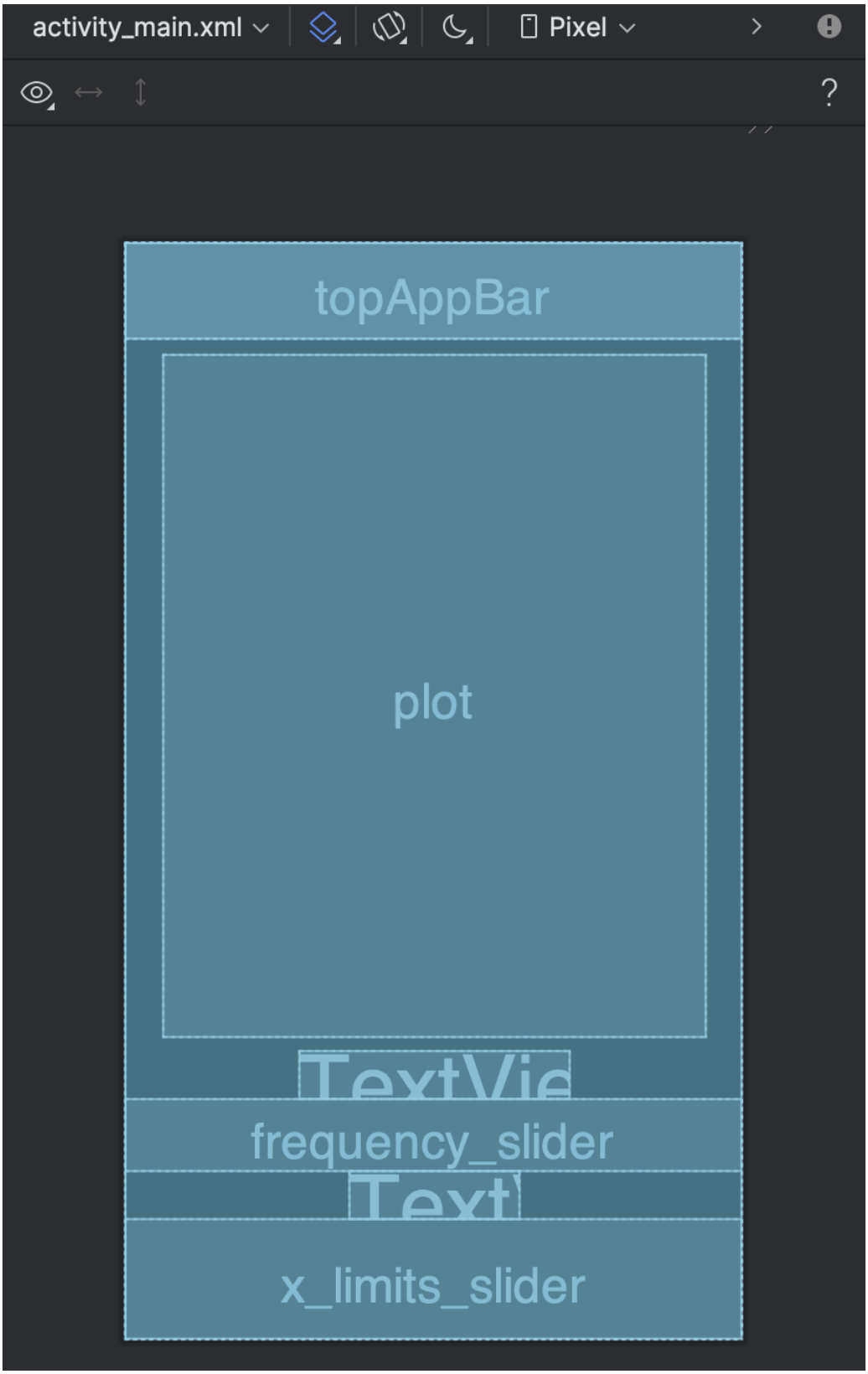

Рис. 1: Layout нашого додатку з *Android Studio*.

Отже, наведемо сам layout нижче. Також, на Рисунку 1 можна побачити наш Layout візуально.

```
<?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:id="@+id/main"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <com.google.android.material.appbar.AppBarLayout
            android:id="@+id/appBarLayout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@color/material_dynamic_tertiary80"
            app:layout_constraintTop_toTopOf="parent">

            <com.google.android.material.appbar.MaterialToolbar
                android:id="@+id/topAppBar"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:minHeight="?attr/actionBarSize"
                app:title="@string/main_page_title"
                app:subtitle="@string/main_page_subtitle"
                app:menu="@menu/top_app_bar" />

        </com.google.android.material.appbar.AppBarLayout>

        <com.androidplot.xy.XYPlot
            style="@style/CustomPlotStyle.Pinkish"
            android:id="@+id/plot"
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:layout_marginTop="10dp"
```

```
38              android:layout_marginLeft="25dp"
39              android:layout_marginRight="25dp"
40              android:layout_marginBottom="10dp"
41              android:layout_weight="1"
42              app:domainTitle="domain"
43              app:layout_constraintBottom_toTopOf="@+id/frequency_slider_label"
44              app:layout_constraintTop_toBottomOf="@+id/appBarLayout"
45              app:layout_constraintEnd_toEndOf="parent"
46              app:layout_constraintHorizontal_bias="0.5"
47              app:layout_constraintStart_toStartOf="parent"
48              app:lineLabelRotationBottom="-45"
49              app:lineLabels="left|bottom|right"
50              app:rangeTitle="range"
51              app:title="Plot"
52              />
53
54          <TextView
55              android:id="@+id/frequency_slider_label"
56              android:layout_width="wrap_content"
57              android:layout_height="wrap_content"
58              app:layout_constraintTop_toBottomOf="@+id/plot"
59              app:layout_constraintBottom_toTopOf="@+id/frequency_slider"
60              app:layout_constraintEnd_toEndOf="parent"
61              app:layout_constraintStart_toStartOf="parent"
62              android:textSize="24sp"
63              android:text="@string/sliders_frequency_slider_label" />
64
65          <com.google.android.material.slider.RangeSlider
66              android:id="@+id/frequency_slider"
67              android:contentDescription="@string/sliders_frequency_slider_description"
68              android:layout_width="match_parent"
69              android:layout_height="12dp"
70              android:stepSize="0.1"
71              android:valueFrom="-3.0"
72              android:valueTo="3.0"
73              app:layout_constraintBottom_toTopOf="@+id/limits_slider_label"
74              app:layout_constraintEnd_toEndOf="parent"
75              app:layout_constraintStart_toStartOf="parent" />
76
77          <TextView
78              android:id="@+id/limits_slider_label"
79              android:layout_width="wrap_content"
```

```
80              android:layout_height="wrap_content"
81              app:layout_constraintBottom_toTopOf="@+id/x_limits_slider"
82              app:layout_constraintEnd_toEndOf="parent"
83              app:layout_constraintStart_toStartOf="parent"
84              android:textSize="24sp"
85              android:text="@string/sliders_limits_slider_label" />
86
87          <com.google.android.material.slider.RangeSlider
88              android:id="@+id/x_limits_slider"
89              android:contentDescription="@string/sliders_limits_slider_description"
90              android:layout_width="match_parent"
91              android:layout_height="12dp"
92              android:stepSize="0.1"
93              android:valueFrom="-5.0"
94              android:valueTo="5.0"
95              app:labelBehavior="withinBounds"
96              app:layout_constraintBottom_toBottomOf="parent"
97              app:layout_constraintEnd_toEndOf="parent"
98              app:layout_constraintStart_toStartOf="parent" />
99
100     </androidx.constraintlayout.widget.ConstraintLayout>
101 </layout>
```

2. HarmonicConfig. Далі буде дуже зручно зберігати інформацію про графік у вигляді допоміжного класу `HarmonicConfig`. В ньому ми і покладемо наші межі та циклічну частоту. Клас виглядає наступним чином:

```
1  package com.zamdimon.graph_plotting.logic;
2
3  public class HarmonicConfig {
4      /** Represents the minimal left value the plot can start drawing from. */
5      public static final float MIN_LEFT_LIMIT = -5.0f;
6
7      /** Represents the maximal right value the plot can end drawing up. */
8      public static final float MAX_RIGHT_LIMIT = 5.0f;
9
10     /** Represents the default cyclic frequency available by default */
11     public static final float DEFAULT_CYCLIC_FREQUENCY = 1.0f;
12
13     /** Represents the left x limit from which to draw the plot */
14     private float leftLimit;
```

```
15      /** Represents the right x limit until which to draw the plot */
16      private float rightLimit;
17      /** Represents the cyclic frequency of the function */
18      private float cyclicFrequency;
19
20      /**
21       * Constructs the config with default values
22       */
23      public HarmonicConfig() {
24          this.leftLimit = MIN_LEFT_LIMIT;
25          this.rightLimit = MAX_RIGHT_LIMIT;
26          this.cyclicFrequency = DEFAULT_CYCLIC_FREQUENCY;
27      }
28
29      /**
30       * Constructor for the HarmonicPlot class with the specified fields.
31       * @param leftLimit the left x limit from which to draw the plot
32       * @param rightLimit the right x limit until which to draw the plot
33       * @param cyclicFrequency the cyclic frequency of the function
34       */
35      public HarmonicConfig(float leftLimit, float rightLimit, float cyclicFrequency) {
36          this.leftLimit = leftLimit;
37          this.rightLimit = rightLimit;
38          this.cyclicFrequency = cyclicFrequency;
39      }
40
41      /**
42       * Updates the cyclic frequency of the function.
43       * @param cyclicFrequency the new cyclic frequency of the function
44       */
45      public void updateCyclicFrequency(float cyclicFrequency) {
46          this.cyclicFrequency = cyclicFrequency;
47      }
48
49      /**
50       * Updates the left and right limits of the plot.
51       * @param leftLimit the new left x limit from which to draw the plot
52       * @param rightLimit the new right x limit until which to draw the plot
53       */
54      public void updateLimits(float leftLimit, float rightLimit) {
55          assert rightLimit > leftLimit : "Right limit must be greater than the left
             ↪ one";
```

```
56
57          this.leftLimit = leftLimit;
58          this.rightLimit = rightLimit;
59      }
60
61      /**
62       * @return Left x limit from which to draw the plot
63       */
64      public float getLeftLimit() {
65          return leftLimit;
66      }
67
68      /**
69       * @return Right x limit until which to draw the plot
70       */
71      public float getRightLimit() {
72          return rightLimit;
73      }
74
75      /**
76       * @return Length of the plot along the Ox axis
77       */
78      public float getLimitLength() {
79          return rightLimit - leftLimit;
80      }
81
82      /**
83       * @return the cyclic frequency of the function
84       */
85      public float getCyclicFrequency() {
86          return cyclicFrequency;
87      }
88  }
```

Насправді доволі простий клас, в котрому є аксесори до параметрів, а також методи для їх оновлення, ініціалізації, а також підтримка публічних константних значень, що відповідають дефолтним значенням відповідних полів.

3. HarmonicPlot. Тут найцікавіше, це власне клас, в якому ми будемо будувати і малювати графік. Виглядає він наступним чином:

```java
package com.zamdimon.graph_plotting.logic;

import android.graphics.Color;
... (other imports)

/**
 * Class responsible for internal logic needed to display a plot.
 */
public class HarmonicPlot {
    /** Function which is being plotted. By default, as the problem suggests,
     * we draw a function sin(omega*x) + cos(omega*x) */
    private Function<Float, Float> function = x -> (float) (Math.sin(x) +
     ↪ Math.cos(x));

    /** Configuration file of our plot. Contains the core data such as frequency or x
     ↪ limits. */
    private final HarmonicConfig config;

    /**
     * Sets the function to be plotted.
     * @param function the new function to be plotted
     */
    public void setFunction(Function<Float, Float> function) {
        this.function = function;
    }

    /**
     * Constructs the HarmonicPlot with default parameters
     */
    public HarmonicPlot() {
        this.config = new HarmonicConfig();
    }

    /**
     * Contructs the HarmonicPlot with specified config
     * @param config Config to initialize the plot with
     */
    public HarmonicPlot(@NonNull HarmonicConfig config) {
        this.config = config;
    }

    /**
```

```
41      * Updates the limit of the internal config
42      * @param leftLimit coordinate from which to start drawing the plot
43      * @param rightLimit coordinate until which to draw the plot
44      */
45     public void updateLimits(float leftLimit, float rightLimit) {
46         config.updateLimits(leftLimit, rightLimit);
47     }
48
49     /**
50      * Updates the cyclic frequency of the internal config
51      * @param frequency the new cyclic frequency of the function
52      */
53     public void updateCyclicFrequency(float frequency) {
54         config.updateCyclicFrequency(frequency);
55     }
56
57     /**
58      * Estimates the number of points needed to draw the plot.
59      * @return the estimated number of points needed to draw the plot
60      */
61     private int estimatePointsNumber() {
62         float period = 2 * (float) Math.PI / config.getCyclicFrequency();
63         // We want approximately 8 points per period. Meaning, the total
64         // number of points is the total number of periods in the specified
65         // interval multiplied by 8.
66         float periods = config.getLimitLength() / period;
67         // We want at least 10 points, otherwise adapt according to the number
68         // of periods.
69         final int POINTS_PER_PERIOD = 35;
70         final int MIN_POINTS = 35;
71         return Math.max((int) (periods * POINTS_PER_PERIOD), MIN_POINTS);
72     }
73
74     /**
75      * Forms the plot points for the plot.
76      * @return the XYSeries representing the plot points
77      */
78     private @NonNull XYSeries formPlotPoints() {
79         // Initializing empty lists
80         List<Number> xNumbers = new ArrayList<>();
81         List<Number> yNumbers = new ArrayList<>();
82
```

```
83          int pointsNumber = this.estimatePointsNumber();
84          for (int i = 0; i < pointsNumber; ++i) {
85              float x = config.getLeftLimit() + config.getLimitLength() * i /
                ↪ pointsNumber;
86              float y = function.apply(x * config.getCyclicFrequency());
87
88              xNumbers.add(x);
89              yNumbers.add(y);
90          }
91
92          final String PLOT_TITLE = "harmonic_plot";
93          return new SimpleXYSeries(xNumbers, yNumbers, PLOT_TITLE);
94      }
95
96      /**
97       * Draws the plot on the given XYPlot.
98       * @param plot the XYPlot to draw the plot on
99       */
100     public void drawPlot(@NonNull XYPlot plot) {
101         XYSeries series = formPlotPoints();
102
103         final int LINE_COLOR = Color.BLUE;
104         final int POINT_COLOR = Color.GREEN;
105         LineAndPointFormatter pointFormatter = new LineAndPointFormatter(LINE_COLOR,
                ↪ POINT_COLOR, null, null);
106
107         // Adding interpolation to make curve a little bit smoother
108         final int POINTS_PER_SEGMENT = 20;
109         pointFormatter.setInterpolationParams(new
                ↪ CatmullRomInterpolator.Params(POINTS_PER_SEGMENT,
                ↪ CatmullRomInterpolator.Type.Centripetal));
110
111         // Change the line width
112         final int STROKE_WIDTH = 10;
113         Paint paint = pointFormatter.getLinePaint();
114         paint.setStrokeWidth(STROKE_WIDTH);
115         pointFormatter.setLinePaint(paint);
116
117         // Clearing and redrawing the plot
118         plot.clear();
119         plot.addSeries(series, pointFormatter);
120         plot.redraw();
```

```
121     }
122 }
```

Тут варто звернути увагу на наступні речі:

- Найголовніша проблема була зробити так, щоб точок на графіку було небагато (бо велику кількість обробляти дуже важко), але при цьому графік виглядав максимально гладко. Для цього був застосований вбудований в пакет інтерполятор `CatmullRomInterpolator`, а також дещо саморобний метод адаптивного підбору оптимальної кількості точок. Його ідея полягає в тому, що чим більше періодів синусоїди намальовано, тим більше точок треба наносити і навпаки. Тому, якщо в середньому потрібно 35 точок на період (емпірично підібране значення), то приблизна загальна кількість точок задається як $\frac{t_1-t_0}{T} \times 35$.
- Параметри, по яким ми будуємо точки, беремо з `HarmonicConfig`.
- Також ми зробили сам графік трошки жирнішим за допомогою методу `.setStrokeWidth`.

4. MainActivity. Оскільки нововедення дуже суттєво змінять логіку, сам код `MainActivity` ми наведемо пізніше після того, як опишемо нові функції.

Отже, ми маємо весь базовий функціонал! Тепер давайте пограємось і додамо щось цікавеньке.

(3) Нові функції.

1. Shared Preferences. По-перше, було б цікаво, якщо б параметри частоти та меж зберігалися при виході з додатку. Саме тому ми додали wrapper навколо `HarmonicConfig`, котрий дозволяє зберігати преференції і підвантажувати при запуску додатку.

Оскільки shared preferences це, по суті, звичайна key-value табличка, то зручно захардкодити ключі у вигляді імпровізованого `enum` типу:

```
1  package com.zamdimon.graph_plotting.storage;
2
3  /**
4   * Represents the fields of the harmonic configuration
5   * in a "sort of" enum-like type.
6   */
7  public final class HarmonicConfigFields {
8      /** Represents the left limit of the plot */
```

```
9       public static final String LEFT_LIMIT = "leftLimit";
10      /** Represents the right limit of the plot */
11      public static final String RIGHT_LIMIT = "rightLimit";
12      /** Represents the cyclic frequency of the function */
13      public static final String CYCLIC_FREQUENCY = "cyclicFrequency";
14  }
```

А далі власне наш wrapper:

```
1   package com.zamdimon.graph_plotting.storage;
2
3   import com.zamdimon.graph_plotting.logic.HarmonicConfig;
4   import android.content.Context;
5   import android.content.SharedPreferences;
6
7   import androidx.annotation.NonNull;
8
9   /**
10   * Class responsible for managing the preferences of the HarmonicConfig object.
11   * Basically, it is a wrapper for HarmonicConfig which allows to save and load the
      ↪ configuration
12   * from the SharedPreferences.
13   */
14  public class HarmonicConfigPreferences {
15      /** Name of the preferences file */
16      public static final String PREFERENCES_NAME = "harmonic_config";
17
18      /** Configuration file which is synced with the current shared preferences */
19      private HarmonicConfig config;
20
21      /**
22       * Constructs the preferences wrapper. It loads the configuration from the shared
          ↪ preferences.
23       * If the preferences are not set, it uses the default values. Updates the inner
          ↪ HarmonicConfig.
24       * @param context the context of the application
25       */
26      public HarmonicConfigPreferences(@NonNull Context context) {
27          SharedPreferences preferences =
              ↪ context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);
28          float leftLimit = preferences.getFloat(HarmonicConfigFields.LEFT_LIMIT,
              ↪ HarmonicConfig.MIN_LEFT_LIMIT);
```

```
29          float rightLimit = preferences.getFloat(HarmonicConfigFields.RIGHT_LIMIT,
             ↪ HarmonicConfig.MAX_RIGHT_LIMIT);
30          float frequency = preferences.getFloat(HarmonicConfigFields.CYCLIC_FREQUENCY,
             ↪ HarmonicConfig.DEFAULT_CYCLIC_FREQUENCY);
31
32          this.config = new HarmonicConfig(leftLimit, rightLimit, frequency);
33      }
34
35      /**
36       * Saves the frequency to the shared preferences and updates the inner
          ↪ configuration.
37       * @param context the context of the application
38       * @param frequency the new frequency to be saved
39       */
40      public void saveFrequency(@NonNull Context context, float frequency) {
41          SharedPreferences preferences =
             ↪ context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);
42          SharedPreferences.Editor editor = preferences.edit();
43
44          editor.putFloat(HarmonicConfigFields.CYCLIC_FREQUENCY, frequency);
45          editor.apply();
46
47          this.config.updateCyclicFrequency(frequency);
48      }
49
50      /**
51       * Saves the limits to the shared preferences and updates the inner configuration.
52       * @param context the context of the application
53       * @param leftLimit the new left limit to be saved
54       * @param rightLimit the new right limit to be saved
55       */
56      public void saveLimits(@NonNull Context context, float leftLimit, float
         ↪ rightLimit) {
57          SharedPreferences preferences =
             ↪ context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);
58          SharedPreferences.Editor editor = preferences.edit();
59
60          editor.putFloat(HarmonicConfigFields.LEFT_LIMIT, leftLimit);
61          editor.putFloat(HarmonicConfigFields.RIGHT_LIMIT, rightLimit);
62          editor.apply();
63
64          this.config.updateLimits(leftLimit, rightLimit);
```

```
65      }
66
67      /**
68       * Saves the configuration to the shared preferences.
69       * @param context the context of the application
70       * @param config the configuration to be saved
71       */
72      public void saveConfig(@NonNull Context context, HarmonicConfig config) {
73          SharedPreferences preferences =
             ↪ context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);
74          SharedPreferences.Editor editor = preferences.edit();
75
76          editor.putFloat(HarmonicConfigFields.LEFT_LIMIT, config.getLeftLimit());
77          editor.putFloat(HarmonicConfigFields.RIGHT_LIMIT, config.getRightLimit());
78          editor.putFloat(HarmonicConfigFields.CYCLIC_FREQUENCY,
             ↪ config.getCyclicFrequency());
79          editor.apply();
80
81          this.config = config;
82      }
83
84      /**
85       * @return the current configuration
86       */
87      public HarmonicConfig getConfig() {
88          return config;
89      }
90  }
```

По суті, `HarmonicConfigPreferences` це і є `HarmonicConfig` з двома ключовими відмінностями:

- В конструкторі підвантажуються значення з `SharedPreferences` або, якщо їх немає, то беруться дефолтні.
- Є методи на оновлення параметрів, котрі одночасно роблять запит на оновлення/створення в `SharedPreferences` і синхронізують внутрішній `this.config` зі значенням в `SharedPreferences`. Це потрібно, щоб при кожному отриманні `SharedPreferences` не потрібно було робити запитів, а просто взяти внутрішній instance класу `HarmonicConfig` (по суті кешування).

2. Зберігання преференцій. А було б ще цікавіше, якщо б параметри не просто зберігались локально, а їх можна було кудись викачати і потім заімпортити. Або, переслати другу файл з преференціями, а друг міг би відкрити цей файл. Тому ми вирішили розробити дві функції:

- Функцію формування `json` файлу з усіма параметрами `HarmonicConfig`.
- Функцію читання `json` файлу, котрий може спарсити данні у `HarmonicConfig`.

Ці функції ми вирішили помістити на тулбар зверху у вигляді кнопок з іконками. Для цього ми додали наступне меню (тут є ще кнопка сбросу налаштунків, але вона не дуже цікава):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/reset"
        android:title="@string/toolbar_reset"
        android:icon="@drawable/reset"
        app:showAsAction="ifRoom"
        />
    <item
        android:id="@+id/save"
        android:title="@string/toolbar_save"
        android:icon="@drawable/save"
        app:showAsAction="ifRoom"
        />

    <item
        android:id="@+id/upload"
        android:icon="@drawable/upload"
        android:title="@string/toolbar_upload"
        app:showAsAction="always"
        />
</menu>
```

Іконки ми взяли з сайту https://fonts.google.com/icons.

Тепер, для зберігання ми створили наступний клас, котрий зчитує усі kv значення з `SharedPreferences` та формує з них `.json` файл:

```
package com.zamdimon.graph_plotting.utils;

import android.content.Context;
```

```
4  import android.content.SharedPreferences;
5  import android.net.Uri;
6  import android.util.Log;
7
8  import org.json.JSONException;
9  import org.json.JSONObject;
10
11 import java.io.File;
12 import java.io.FileWriter;
13 import java.io.IOException;
14 import java.io.OutputStream;
15 import java.io.OutputStreamWriter;
16 import java.util.Map;
17
18 /**
19  * Utility class for handling SharedPreferences
20  */
21 public class SharedPreferencesUtil {
22     /**
23      * Converts SharedPreferences to JSON Object
24      * @param context the context of the application
25      * @param prefsName the name of the SharedPreferences file
26      * @return JSON Object containing the SharedPreferences
27      */
28     public static JSONObject getSharedPreferencesAsJson(Context context, String
       ↪ prefsName) {
29         // Retrieve SharedPreferences
30         SharedPreferences sharedPreferences = context.getSharedPreferences(prefsName,
           ↪ Context.MODE_PRIVATE);
31         Map<String, ?> allEntries = sharedPreferences.getAll();
32
33         // Convert to JSON Object
34         JSONObject jsonObject = new JSONObject();
35         for (Map.Entry<String, ?> entry : allEntries.entrySet()) {
36             try {
37                 jsonObject.put(entry.getKey(), entry.getValue());
38             } catch (JSONException e) {
39                 Log.e("SharedPreferencesUtil", "Error putting key-value into json
                   ↪ object", e);
40             }
41         }
42         return jsonObject;
```

```
43      }
44
45      /**
46       * Saves JSON Object to file
47       * @param context the context of the application
48       * @param uri the URI of the file to save
49       * @param jsonObject the JSON Object to save
50       */
51      public static void saveJsonToFile(Context context, Uri uri, JSONObject
         ↪ jsonObject) {
52          try {
53              OutputStream outputStream =
                 ↪ context.getContentResolver().openOutputStream(uri);
54              if (outputStream != null) {
55                  OutputStreamWriter outputStreamWriter = new
                     ↪ OutputStreamWriter(outputStream);
56                  outputStreamWriter.write(jsonObject.toString());
57                  outputStreamWriter.close();
58                  outputStream.close();
59              }
60          } catch (IOException e) {
61              Log.e("SharedPreferencesUtil", "Error saving JSON to file", e);
62          }
63      }
64  }
```

І далі створили наступний клас, котрий ми потім вставимо в `MainActivity`. Він містить callback та intent, що потрібні для запуску меню юзера з вибором де зберігти файл, а також хендлінгу його вибору з нашої сторони.

```
1   package com.zamdimon.graph_plotting.storage;
2
3   import android.app.Activity;
4   import android.content.Context;
5   import android.content.Intent;
6   import android.net.Uri;
7   import android.widget.Toast;
8
9   import androidx.activity.result.ActivityResult;
10
11  import com.zamdimon.graph_plotting.utils.SharedPreferencesUtil;
12
```

```
13 import org.json.JSONObject;
14
15 /**
16  * Helper class for managing the saving of HarmonicConfig
17  */
18 public class HarmonicConfigSave {
19     /**
20      * Handles the result of the save activity
21      * @param context the context of the application
22      * @param result the result of the activity
23      */
24     public static void onSaveActivityResult(Context context, ActivityResult result) {
25         if (result.getResultCode() != Activity.RESULT_OK) {
26             return;
27         }
28
29         Intent data = result.getData();
30         if (data == null) {
31             return;
32         }
33         Uri createFileUri = data.getData();
34         if (createFileUri == null) {
35             return;
36         }
37         JSONObject jsonObject =
            ↪ SharedPreferencesUtil.getSharedPreferencesAsJson(context,
            ↪ HarmonicConfigPreferences.PREFERENCES_NAME);
38         SharedPreferencesUtil.saveJsonToFile(context, createFileUri, jsonObject);
39
40         // Displaying a toast
41         CharSequence text = "File saved successfully!";
42         int duration = Toast.LENGTH_LONG;
43
44         Toast toast = Toast.makeText(context, text, duration);
45         toast.show();
46     }
47
48     /**
49      * Creates an intent for saving the .json file with preferences
50      * @return the intent for saving the .json file
51      */
52     public static Intent formSaveIntent() {
```

```
53          Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
54          intent.addCategory(Intent.CATEGORY_OPENABLE);
55          final String MIME_TYPE = "application/json";
56          intent.setType(MIME_TYPE);
57          final String EXTRA_FILE_NAME = "shared_prefs.json";
58          intent.putExtra(Intent.EXTRA_TITLE, EXTRA_FILE_NAME);
59
60          return intent;
61      }
62  }
```

Тут цікаво наступне: для того, щоб створити вікно з вибором, де зберігати файл, потрібно створити `Intent`. Далі ми його запустимо через `ResultLauncher` разом з callback у `MainActivity`.

3. Зчитування преференцій з файлу. Створимо клас, майже аналогічний вище наведеному. Тут багато роботи з файлової системою, що дуже нудно, тому не будемо сильно зосереджувати увагу. Єдине, тут ми використовували пакет `Gson` для десеріалізації сирих данних у класові поля – доволі зручна річ.

```
1  package com.zamdimon.graph_plotting.storage;
2
3  import android.app.Activity;
4  import android.content.ContentResolver;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.net.Uri;
8
9  import androidx.activity.result.ActivityResult;
10 import androidx.fragment.app.FragmentManager;
11
12 import com.google.gson.Gson;
13 import com.zamdimon.graph_plotting.dialogs.HarmonicConfigConfirmDialog;
14 import com.zamdimon.graph_plotting.dialogs.HarmonicConfigErrorDialog;
15 import com.zamdimon.graph_plotting.logic.HarmonicConfig;
16
17 import java.io.BufferedReader;
18 import java.io.InputStream;
19 import java.io.InputStreamReader;
20 import java.util.function.Consumer;
21
```

```java
/**
 * Helper class for managing the upload of HarmonicConfig
 */
public class HarmonicConfigUpload {
    /**
     * Handles the result of the upload activity
     * @param context the context of the application
     * @param result the result of the activity
     * @param fragmentManager the fragment manager
     * @param acceptCallback the callback being called when the user accepts the upload
     */
    public static void onUploadActivityResult(Context context, ActivityResult result, FragmentManager fragmentManager, Consumer<HarmonicConfig> acceptCallback) {
        if (result.getResultCode() != Activity.RESULT_OK) {
            return;
        }

        Intent data = result.getData();
        if (data == null) {
            return;
        }
        Uri chooseFileUri = data.getData();
        if (chooseFileUri == null) {
            return;
        }
        HarmonicConfig config = getFileContent(chooseFileUri, context.getContentResolver());
        if (config == null) {
            // Displaying an error
            HarmonicConfigErrorDialog errorDialog = new HarmonicConfigErrorDialog();
            final String UPLOAD_ERROR_DIALOG_TAG = "error_dialog";
            errorDialog.show(fragmentManager, UPLOAD_ERROR_DIALOG_TAG);
            return;
        }

        // Displaying a success button
        HarmonicConfigConfirmDialog dialog = new HarmonicConfigConfirmDialog(config, () -> acceptCallback.accept(config), () -> {});
        final String UPLOAD_DIALOG_TAG = "upload_dialog";
        dialog.show(fragmentManager, UPLOAD_DIALOG_TAG);
    }
```

```
60
61      /**
62       * Retrieves the content of the file in the form of HarmonicConfig
63       * @param contentUri the URI of the content
64       * @param contentResolver the content resolver
65       * @return the HarmonicConfig object
66       */
67      public static HarmonicConfig getFileContent(Uri contentUri, ContentResolver
         ↪ contentResolver) {
68          try {
69              InputStream inputStream = contentResolver.openInputStream(contentUri);
70              BufferedReader reader = new BufferedReader(new
                 ↪ InputStreamReader(inputStream));
71              StringBuilder content = new StringBuilder();
72              String line;
73
74              while((line = reader.readLine()) != null) {
75                  content.append(line);
76              }
77
78              reader.close();
79
80              Gson gson = new Gson();
81              return gson.fromJson(content.toString(), HarmonicConfig.class);
82          } catch(Exception e) {
83              return null;
84          }
85      }
86
87      /**
88       * Forms the upload intent
89       * @return the upload intent
90       */
91      public static Intent formUploadIntent() {
92          Intent chooseFile;
93          Intent intent;
94          chooseFile = new Intent(Intent.ACTION_GET_CONTENT);
95          chooseFile.addCategory(Intent.CATEGORY_OPENABLE);
96          final String MIME_TYPE = "application/json";
97          chooseFile.setType(MIME_TYPE);
98
99          final String CHOOSER_TITLE = "Choose a file";
```

```
100         intent = Intent.createChooser(chooseFile, CHOOSER_TITLE);
101         return intent;
102     }
103 }
```

Проте, дещо цікаве таки є! Ми додали два діалога: `HarmonicConfigConfirmDialog` та `HarmonicConfigErrorDialog`, котрі потрібні для інформування юзера про статус обробки обраного документа. Якщо виникла помилка, то викликається діалог, що реалізований наступним чином:

```
1  package com.zamdimon.graph_plotting.dialogs;
2
3  import android.app.Dialog;
4  import android.content.Context;
5  import android.os.Bundle;
6
7  import androidx.annotation.NonNull;
8  import androidx.fragment.app.DialogFragment;
9  import androidx.fragment.app.FragmentActivity;
10
11 import com.google.android.material.dialog.MaterialAlertDialogBuilder;
12 import com.zamdimon.graph_plotting.R;
13
14 /**
15  * Dialog that is displayed when the configuration upload fails.
16  */
17 public class HarmonicConfigErrorDialog extends DialogFragment {
18     /**
19      * Creates the dialog.
20      * @param savedInstanceState the saved instance state
21      * @return the dialog
22      */
23     @NonNull
24     @Override
25     public Dialog onCreateDialog(Bundle savedInstanceState) {
26         super.onCreateDialog(savedInstanceState);
27
28         FragmentActivity activity = getActivity();
29         assert activity != null : "Activity must not be null";
30
31         Context context = getContext();
32         assert context != null : "Context must not be null";
```

```
33
34          MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(activity);
35          builder.setTitle(R.string.config_upload_error_title)
36                  .setMessage(R.string.config_upload_error_message);
37
38          return builder.create();
39      }
40  }
```

Якщо ж все нормально, то в діалогу з'являються кнопки:

```
1   package com.zamdimon.graph_plotting.dialogs;
2
3   import android.app.Dialog;
4   import android.content.Context;
5   import android.os.Bundle;
6   import android.widget.Toast;
7
8   import androidx.annotation.NonNull;
9   import androidx.fragment.app.DialogFragment;
10  import androidx.fragment.app.FragmentActivity;
11
12  import com.google.android.material.dialog.MaterialAlertDialogBuilder;
13  import com.zamdimon.graph_plotting.R;
14  import com.zamdimon.graph_plotting.logic.HarmonicConfig;
15
16  /**
17   * Dialog that is displayed when the configuration upload is successful.
18   */
19  public class HarmonicConfigConfirmDialog extends DialogFragment {
20      private final HarmonicConfig config;
21
22      private final Runnable acceptCallback;
23
24      private final Runnable cancelCallback;
25
26      /**
27       * Creates the dialog.
28       * @param config the configuration to be displayed
29       * @param acceptCallback the callback being called when the user accepts the
          ↪ upload
30       * @param cancelCallback the callback being called when the user cancels the
```

```
        ↪ upload
31      */
32      public HarmonicConfigConfirmDialog(@NonNull HarmonicConfig config, Runnable
        ↪ acceptCallback, Runnable cancelCallback) {
33          this.config = config;
34          this.acceptCallback = acceptCallback;
35          this.cancelCallback = cancelCallback;
36      }
37
38      /**
39       * Forms the message of the dialog.
40       * @return the message
41       */
42      private String formMessage() {
43          return "Left limit: " + config.getLeftLimit() + "\n" +
44                  "Right limit: " + config.getRightLimit() + "\n" +
45                  "Cyclic frequency: " + config.getCyclicFrequency();
46      }
47
48      /**
49       * Creates the dialog.
50       * @param savedInstanceState the saved instance state
51       * @return the dialog
52       */
53      @NonNull
54      @Override
55      public Dialog onCreateDialog(Bundle savedInstanceState) {
56          super.onCreateDialog(savedInstanceState);
57
58          FragmentActivity activity = getActivity();
59          assert activity != null : "Activity must not be null";
60
61          Context context = getContext();
62          assert context != null : "Context must not be null";
63
64          MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(activity);
65          builder.setTitle(R.string.config_upload_title)
66                  .setMessage(formMessage())
67                  .setPositiveButton(R.string.config_upload_confirm, (dialog, id) -> {
68                      acceptCallback.run();
69
70                      // Displaying a toast
```

```
71                      CharSequence text = "Upload successful!";
72                      int duration = Toast.LENGTH_LONG;
73
74                      Toast toast = Toast.makeText(context, text, duration);
75                      toast.show();
76                  })
77                  .setNegativeButton(R.string.config_upload_cancel, (dialog, id) ->
                     ↪ cancelCallback.run());
78
79          return builder.create();
80      }
81  }
```

Також, ми використали `Toast`, щоб показати юзеру повідомлення, що все пройшло успішно. Нарешті, ітоговий **MainActivity** виглядає наступним чином:

```
1  package com.zamdimon.graph_plotting;
2  import android.content.Context;
3  import android.content.Intent;
4  import android.content.SharedPreferences;
5  import android.os.Bundle;
6  import android.widget.Toast;
7
8  import androidx.activity.result.ActivityResultLauncher;
9  import androidx.activity.result.contract.ActivityResultContracts;
10 import androidx.appcompat.app.AppCompatActivity;
11 import androidx.databinding.DataBindingUtil;
12 import com.zamdimon.graph_plotting.databinding.ActivityMainBinding;
13 import com.zamdimon.graph_plotting.logic.HarmonicConfig;
14 import com.zamdimon.graph_plotting.logic.HarmonicPlot;
15 import com.zamdimon.graph_plotting.storage.HarmonicConfigPreferences;
16 import com.zamdimon.graph_plotting.storage.HarmonicConfigSave;
17 import com.zamdimon.graph_plotting.storage.HarmonicConfigUpload;
18
19 /**
20  * Main activity of the application. It is responsible for managing the UI and the
     ↪ logic of the application.
21  */
22 public class MainActivity extends AppCompatActivity {
23     /** Represents the binding of the activity which makes
24      * it easier to manage elements of the activity */
```

```
25      private ActivityMainBinding binding;
26
27      /** Represents the preferences of the harmonic plot - basically, the settings */
28      private HarmonicConfigPreferences plotPreferences;
29
30      /**
31       * Called when the activity is starting.
32       * @param savedInstanceState If the activity is being re-initialized after
33       *     previously being shut down then this Bundle contains the data it most
34       *     recently supplied in {@link #onSaveInstanceState}.  <b><i>Note: Otherwise
          ↪ it is null.</i></b>
35       */
36      @Override
37      protected void onCreate(Bundle savedInstanceState) {
38          super.onCreate(savedInstanceState);
39          binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
40
41          Context context = getApplicationContext();
42
43          // Setting the harmonic plot
44          plotPreferences = new HarmonicConfigPreferences(context);
45          displayPlot();
46
47          // Setting the top bar
48          initializeTopBarMenu();
49
50          // Setting sliders
51          initializeFrequencySlider();
52          initializeLimitsSlider();
53
54          // Setting updating plot when shared preferences get modified
55          subscribeOnSharedPreferenceChanges();
56      }
57
58      /**
59       * Initializes the limits slider and subscribes to its changes.
60       */
61      private void initializeLimitsSlider() {
62          // Setting limits from preferences
63          float initialLeftLimit = plotPreferences.getConfig().getLeftLimit();
64          float initialRightLimit = plotPreferences.getConfig().getRightLimit();
65          binding.xLimitsSlider.setValues(initialLeftLimit, initialRightLimit);
```

```
66
67          // Setting the minimum separation between the limits to avoid cases
68          // when left limit is exactly the right limit
69          final float MIN_SEPARATION = 0.3f;
70          binding.xLimitsSlider.setMinSeparationValue(MIN_SEPARATION);
71
72          // Subscribing to updating the limits
73          binding.xLimitsSlider.addOnChangeListener((slider, value, fromUser) -> {
74              Context context = getApplicationContext();
75              float leftLimit = binding.xLimitsSlider.getValues().get(0);
76              float rightLimit = binding.xLimitsSlider.getValues().get(1);
77              plotPreferences.saveLimits(context, leftLimit, rightLimit);
78          });
79      }
80
81      /**
82       * Refreshes the sliders with the values from the preferences.
83       */
84      public void refreshSliders() {
85          float initialLeftLimit = plotPreferences.getConfig().getLeftLimit();
86          float initialRightLimit = plotPreferences.getConfig().getRightLimit();
87          binding.xLimitsSlider.setValues(initialLeftLimit, initialRightLimit);
88
89          float initialCyclicFrequency =
             ↪ plotPreferences.getConfig().getCyclicFrequency();
90          binding.frequencySlider.setValues(initialCyclicFrequency);
91      }
92
93      /**
94       * Initializes the frequency slider and subscribes to its changes.
95       */
96      private void initializeFrequencySlider() {
97          // Setting frequency from preferences
98          float initialCyclicFrequency =
             ↪ plotPreferences.getConfig().getCyclicFrequency();
99          binding.frequencySlider.setValues(initialCyclicFrequency);
100
101         // Subscribing to updating the frequency
102         binding.frequencySlider.addOnChangeListener((rangeSlider, frequency, b) -> {
103             Context context = getApplicationContext();
104             plotPreferences.saveFrequency(context, frequency);
105         });
```

```
106     }
107
108     /**
109      * Subscribes to shared preferences changes and updates the plot when they change.
110      */
111     private void subscribeOnSharedPreferenceChanges() {
112         SharedPreferences preferences =
             ↪ getSharedPreferences(HarmonicConfigPreferences.PREFERENCES_NAME,
             ↪ Context.MODE_PRIVATE);
113         preferences.registerOnSharedPreferenceChangeListener((sharedPreferences, key)
             ↪ -> displayPlot());
114     }
115
116     /**
117      * Displays the plot on the screen.
118      */
119     private void displayPlot() {
120         HarmonicConfig config = plotPreferences.getConfig();
121         HarmonicPlot plot = new HarmonicPlot(config);
122         plot.drawPlot(binding.plot);
123     }
124
125     /**
126      * Registers the activity result launcher for saving the file.
127      */
128     private final ActivityResultLauncher<Intent> saveFileResultLauncher =
         ↪ registerForActivityResult(
129             new ActivityResultContracts.StartActivityForResult(),
130             result ->
                 ↪ HarmonicConfigSave.onSaveActivityResult(getApplicationContext(),
                 ↪ result)
131     );
132
133     /**
134      * Registers the activity result launcher for uploading the file.
135      */
136     private final ActivityResultLauncher<Intent> uploadFileResultLauncher =
         ↪ registerForActivityResult(
137             new ActivityResultContracts.StartActivityForResult(),
138             result ->
                 ↪ HarmonicConfigUpload.onUploadActivityResult(getApplicationContext(),
                 ↪ result, getSupportFragmentManager(), (HarmonicConfig config) -> {
```

```
139                 // NOTE: it is quite painful to pass all the necessary parameters
                     ↪ inside
140                 // the HarmonicConfigUpload class, so we just pass the Consumer to
                     ↪ the method
141                 plotPreferences.saveConfig(getApplicationContext(), config);
142                 displayPlot();
143                 refreshSliders();
144             }));
145
146     /**
147      * Initializes the top bar menu.
148      */
149     private void initializeTopBarMenu() {
150         binding.topAppBar.setOnMenuItemClickListener(item -> {
151             if (item.getItemId() == R.id.reset) {
152                 HarmonicConfig defaultConfig = new HarmonicConfig();
153                 plotPreferences.saveConfig(getApplicationContext(), defaultConfig);
154                 displayPlot();
155                 refreshSliders();
156             }
157             if (item.getItemId() == R.id.save) {
158                 Intent intent = HarmonicConfigSave.formSaveIntent();
159                 saveFileResultLauncher.launch(intent);
160                 return true;
161             }
162             if (item.getItemId() == R.id.upload) {
163                 Intent intent = HarmonicConfigUpload.formUploadIntent();
164                 uploadFileResultLauncher.launch(intent);
165                 return true;
166             }
167
168             return false;
169         });
170     }
171 }
```

4. GitHub CI. Коли ми викладали репозиторій на GitHub, було просто цікаво зробити CI, котра перевіряла б коректність білда (тобто, запускала юніт тести та перевіряла, чи білдиться проект взагалі). Не те, щоб ця функція була для нас дуже практичною в рамках такого малого проєкту, але вона таки

запрацювала. Знизу наведений GitHub Workflow:

```
name: Generated APK AAB (Upload - Create Artifact To Github Action)

env:
  # The name of the main module repository
  main_project_module: app

  # The name of the Play Store
  playstore_name: Frogobox ID

on:

  push:
    branches:
      - '**'

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      # Set Current Date As Env Variable
      - name: Set current date as env variable
        run: echo "date_today=$(date +'%Y-%m-%d')" >> $GITHUB_ENV

      # Set Repository Name As Env Variable
      - name: Set repository name as env variable
        run: echo "repository_name=$(echo '${{ github.repository }}' | awk -F '/'
          ↪ '{print $2}')" >> $GITHUB_ENV

      - name: Set Up JDK
        uses: actions/setup-java@v3
        with:
          distribution: 'zulu' # See 'Supported distributions' for available options
          java-version: '17'
```

```
40          cache: 'gradle'
41
42      - name: Change wrapper permissions
43        run: chmod +x ./gradlew
44
45      # Run Tests Build
46      - name: Run gradle tests
47        run: ./gradlew test
48
49      # Run Build Project
50      - name: Build gradle project
51        run: ./gradlew build
52
53      # Create APK Debug
54      - name: Build apk debug project (APK) - ${{ env.main_project_module }} module
55        run: ./gradlew assembleDebug
56
57      # Create APK Release
58      - name: Build apk release project (APK) - ${{ env.main_project_module }} module
59        run: ./gradlew assemble
60
61      # Create Bundle AAB Release
62      # Noted for main module build [main_project_module]:bundleRelease
63      - name: Build app bundle release (AAB) - ${{ env.main_project_module }} module
64        run: ./gradlew ${{ env.main_project_module }}:bundleRelease
65
66      # Upload Artifact Build
67      # Noted For Output [main_project_module]/build/outputs/apk/debug/
68      - name: Upload APK Debug - ${{ env.repository_name }}
69        uses: actions/upload-artifact@v3
70        with:
71          name: ${{ env.date_today }} - ${{ env.playstore_name }} - ${{
            ↪ env.repository_name }} - APK(s) debug generated
72          path: ${{ env.main_project_module }}/build/outputs/apk/debug/
73
74      # Noted For Output [main_project_module]/build/outputs/apk/release/
75      - name: Upload APK Release - ${{ env.repository_name }}
76        uses: actions/upload-artifact@v3
77        with:
78          name: ${{ env.date_today }} - ${{ env.playstore_name }} - ${{
            ↪ env.repository_name }} - APK(s) release generated
79          path: ${{ env.main_project_module }}/build/outputs/apk/release/
```

```
80
81        # Noted For Output [main_project_module]/build/outputs/bundle/release/
82        - name: Upload AAB (App Bundle) Release - ${{ env.repository_name }}
83          uses: actions/upload-artifact@v3
84          with:
85            name: ${{ env.date_today }} - ${{ env.playstore_name }} - ${{
              ↪ env.repository_name }} - App bundle(s) AAB release generated
86            path: ${{ env.main_project_module }}/build/outputs/bundle/release/
```

5. Різні стилістичні правки. Також в ході виконання ми трошки експериментували з різними стилями: наприклад, змінювали колір фону графіку, щоб він краще поєднувався з фоном додатку, змінили логотип додатку тощо.