PBD 2023
Android Mini App 2

Summary

Your second assignment for this course consists of creating a music player as an Android app that allows users to: (1) play, stop and pause music (mp3 files) using a simple UI; (2) play, stop, and pause music as a foreground service using notification commands; (3) and, play and pause music using simple gestures (e.g., horizontal and vertical smartphone movements).

NOTE: Please read the instructions carefully before you begin programming!

Deadline: 23:59, April 16th, 2023.

Features and Architecture

Your app must consist of one Activity, one Foreground Service, and one Background Service :

1. **MainActivity** (of type AppCompatActivity), with layout file **activity_main.xml**. It will show six buttons for six commands (start, pause, stop, exit, gesturesOn and gesturesOff), two text views for the track title and the duration information, and a progress bar for visualizing the elapsed/remaining time. It will communicate with the **MediaPlayerService** via intents and MediaPlayerService local Binder.
2. **MediaPlayerService** (extends Service) without a layout. It will implement the four functionalities start, pause, stop and exit. It will communicate both with the **MainActivity** and with the **AccelerationService.** It will implement a MediaPlayerService **local Binder**, to enable binding from the **MainActivity.** It will keep an AccelerationService-local binder to communicate with the **AccelerationService.**
3. **AccelerationService** (extends Service, implements SensorEventListener) without a layout. It will implement the two functionalities gesturesOn and gesturesOff. It will implement an AccelerationService local Binder to enable binding from the **MediaPlayerService.**
4. Communication:
   MainActivity <==> **MediaPlayerService** <==> **AccelerationService** <==> **SensorManager (Android)**

Requirements

1. The app must target API version 31;
2. Use Android X (do not use the support libraries);
3. Do not use findViewById for accessing layout elements. Use Kotlin Synthetics or View Binding instead.
4. Do not use other external libraries or frameworks, you must complete this assignment using only what the Android SDK and Android X offer you.

Implementation details

- Assume that the user has at least three mp3 files placed in the assets directory (please use mp3 files that are smaller than 3MB to allow easier transfer)
- Use android.media.MediaPlayer to implement the media player functionalities
- The commands **Play, Pause, Stop and Exit** are available both through the MainActivity and through foreground notifications
- The commands **GesturesOn** and **GesturesOff,** and the **ProgressBar** are available only through the MainActivity
- Command **Play**
  - starts the **MediaPlayerService** (if not started)
  - plays a random mp3 file
  - updates UI - the track title and the track duration are displayed both in the **MainActivity** (if active) and in the foreground notification; the progress bar is updated accordingly in the **MainActivity**.
- Command **Pause**
  - pauses the music (if playing)
  - updates UI - the track title and the track duration are displayed both in the **MainActivity** (if active) and in the foreground notification; the progress bar is also paused in the **MainActivity**.
- Command **Stop**
  - stops the music (if playing)
  - updates UI - the track title and the track duration are set to default values both in the **MainActivity** (if active) and in the foreground notification; the progress bar returns to its default state in the **MainActivity**.
- Command **Exit**
  - stops the music (if playing)

- ○ stops the **MediaPlayerService** (if running)
- ○ stops the **AccelerationService** (if running)
- ○ exits the application
- ● Command **GesturesOn/GesturesOff**
  - ○ Starts/Stops **AccelerationService** and enables/disables gesture-based commands
  - ○ Displays a pop-up message: "Gestures activated/deactivated" (e.g., using Toast)
- ● **Progress bar**
  - ○ Displaying in **MainActivity** (if active) the elapsed time (played) out of the total song duration
  - ○ Implemented using coroutines
- ● Gesture-based commands
  - ○ When active, the **AccelerationService** can recognize three simple gesture-based commands: IDLE vs. HORIZONTAL vs. VERTICAL
  - ○ The **MediaPlayerService** is informed only when the command HORIZONTAL or the command VERTICAL are recognized.
  - ○ HORIZONTAL calls **MediaPlayerService.Pause()**
  - ○ VERTICAL calls **MediaPlayerService.Play()**
  - ○ IDLE represents no change. Thus, it is not transmitted to the **MediaPlayerService.**
  - ○ Check the Appendix for more details about the gesture recognition algorithm
- ● Other
  - ○ The app should be able to play music both in foreground and in background (when another app is activated or when the screen is locked)
  - ○ The user should be able to use gesture-based commands (when gestures are activated) both in foreground and in background
  - ○ The views that display the track duration (including the progress bar) should be updated every second. This includes both the **MainActivity UI** and the foreground notifications
  - ○ The **MediaPlayerService** updates the **MainActivity UI** every second.
  - ○ The **MainActivity UI** and the foreground notification must always display equal information (track name, duration, and player state)
  - ○ The app should not crash in normal usage (nevertheless, you don't need to implement thorough and exotic testing scenarios)
  - ○ Add meaningful comments to your code
  - ○ Feel free to experiment with layout modifications (colors, structure, fonts): you are free to make the app look as good as you wish, as long as all the required functionality is there

Important grading notes

- You must submit your code in a repository titled PBD2023-MA-2 in your Bitbucket account. The repository must be private and the user "pbdfrita" (pbdfrita@gmail.com) must be added as a read-only member;
- Your project title must be: **MiniApp2**
- Your classes must be named: **MainActivity.kt**, **MediaPlayerService.kt** and **AccelerationService.kt**
- Your layout must be named: **activity_main.xml**
- Your project package must be named: si.uni_lj.fri.pbd.miniapp2
- Your code must be fully anonymous - your email, name, or bitbucket user name must not be shown anywhere in the code/comments;
- Your comments, variable names, etc., must be in English
- The bonus task can bring you up to 5 additional points

Screenshots
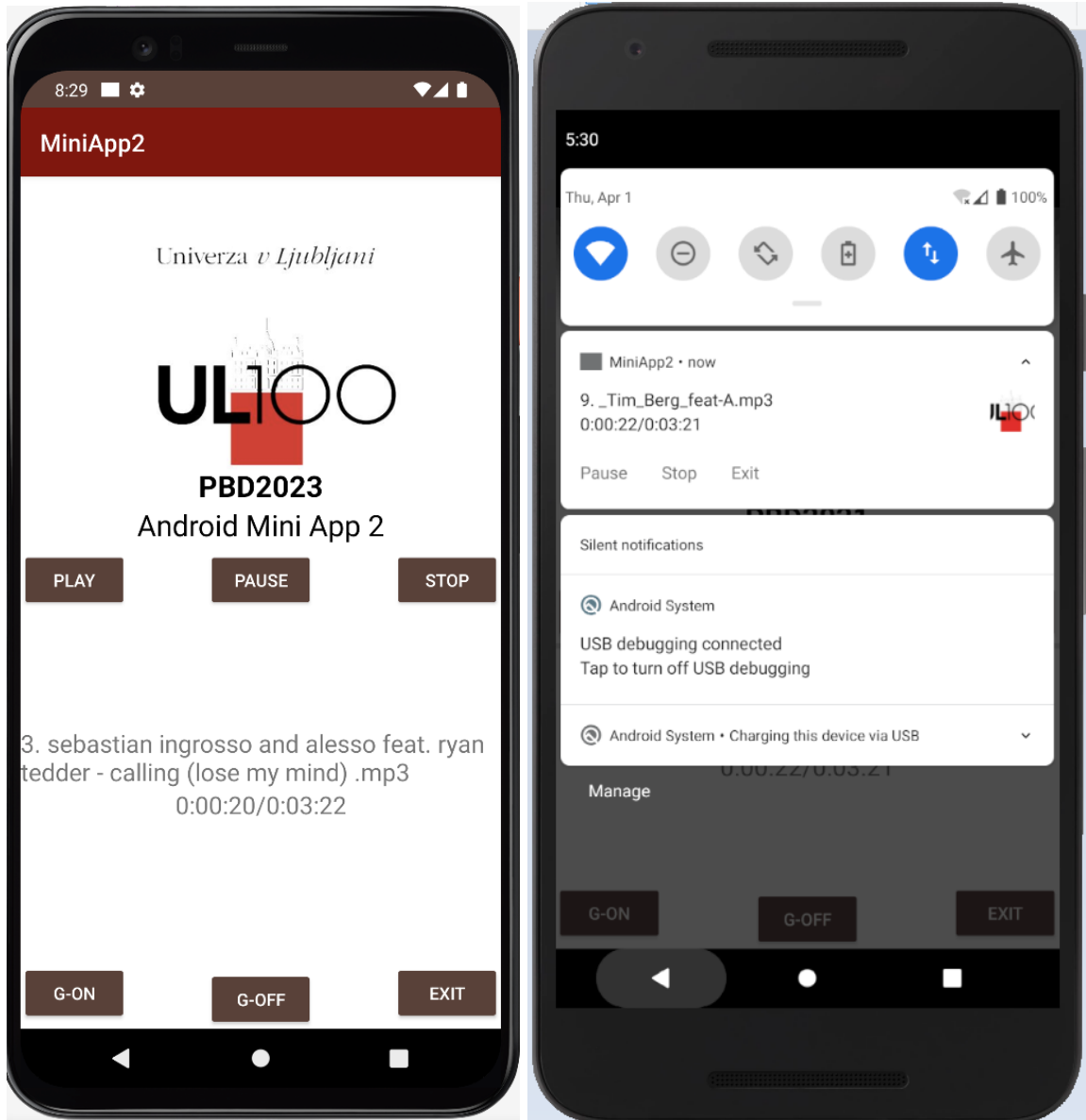
Appendix - Gesture recognition details

To simplify the use-case, we assume that the user will use gesture-based commands only when the smartphone is static. For example, the user listens to an audiobook before going to sleep (in her bed). After listening for some amount of time the user just shakes the smartphone and the smartphone stops playing.

Please note that this is one possible solution for recognizing IDLE vs. HORIZONTAL vs. VERTICAL gestures. You are free to implement your own solution.

First, **RegisterListener** to read acceleration data from Andorid's Sensor.*TYPE_ACCELEROMETER.*
Next, implement **onSensorChanged()** and use inside the following algorithm:

1. **Noise_threshold** = 5 *//values smaller than 5 consider as sensor noise*
2. X_t, Y_t, Z_t = accelerometer.X, accelerometer.Y, accelerometer.Z 5 *// get acceleration sensor readings for each of the three axis*
3. **dX,dY,dZ** = abs(X_{t-1} - X_t), abs(Y_{t-1} - Y_t), abs(Z_{t-1} - Z_t) *// calculate relative change for each axis*
4. If (**dX** <= **noise_threshold**) **dX** = 0, same for **dY**,**dZ** *// if the change is smaller than the* Noise_threshold, count it as a noise (no change)
5. **Command** = **IDLE**
6. if (**dX** > **dZ**) **command** = **HORIZONTAL** // if the x-axis change is bigger than the z-axis change, count it as a horizontal movement
7. if (**deltaZ** > **deltaX**) **command** = **VERTICAL** // if the z-axis change is bigger than the z-axis change, count it as a horizontal movement
8. If (**command** != **IDLE)** update **MediaPlayerService**

Note* To avoid sending commands every 50 milliseconds to the **MediaPlayerService** you should also implement a time constraint. For example, there have to be at least 500 milliseconds between the last and the current update.

Bonus task - Machine learning (ML) for recognizing more than two gesture-based commands

The algorithm described in the Appendix can recognize only two simple gestures. Your task here will be to create an ML pipeline that recognizes any two gestures. Thus, the application will keep the two gesture-based commands:

1. User-defined gesture GESTURE-1 calls **MediaPlayerService.Pause()**
2. User-defined gesture GESTURE-2 calls **MediaPlayerService.Play()**

For this task, you can use any of the sensors available in Android (even audio), but the acceleration sensor should be enough. The following description presents one possible approach for the ML pipeline, however, you are welcome to implement your own approaches.

- **Label data.** Since GESTURE-1 and GESTURE-2 are user-defined, we have to ask the user to label some data for the ML pipeline. Thus, in MainActivity implement an additional "Train" button that when clicked, asks the user to perform each user-defined gesture repeatedly for a number of seconds (e.g. 5 seconds). Label the recorded data accordingly. In addition to the two gestures, you will also need to record random data that you will label as OTHER.
- **Windowing:** Most of the ML pipelines work with instances (windows) of equal length. For that reason, find the minimum repetition duration (e.g., 100 milliseconds) and using that duration, split the overall labeled data into segments with equal duration. Thus, if one gesture was recorded for 5 seconds, you will get 50 segments with 100ms out of it.
- **Extract features.** Extract features from each window. Make sure that you have at least 3 different features. The features can be statistical descriptors (e.g., mean, variance skewness, kurtosis, percentiles of each axis and/or their magnitude), frequency-based features (e.g., the three largest magnitudes of the power spectral density-PSD, skewness and kurtosis of the PSD, etc.).
- **Train a model.** After the feature extraction, you will have at least 50 instances each with at least 3 features and labeled with one of the three class labels GESTURE-1 vs. GESTURE-2 vs. OTHER. To avoid imbalances, make sure that the frequency (count) of the class OTHER is not greater than the joint frequency of the GESTURE-1 and GESTURE-2. Use any Android ML toolkit to train a model. One example is this toolkit. Once trained, the model is stored on the phone.
- **Use the model.** Once the model is trained, the user can start using the ML gesture recognition (e.g., you can ask the user to activate the gesture ML recognition via a button).

You should use the same pipeline that you used to train the model (windowing and feature extraction). After the feature extraction, the instance is fed into the trained model which outputs a prediction GESTURE-1 vs. GESTURE-2 vs. OTHER. Based on that prediction, you should update the **MediaPlayerService.**

● **Re-train the model.** Each time the user clicks the "Train" button in MainActivity he should be able to delete already labeled data and re-train the model. This functionality is required as the user may mistakenly label some data (and we want to avoid noisy labels) or in case he/ she wants to change the gestures.

If you decide to implement this Bonus task, you are free to add new classes and layout files to your App. When uploading your solution, make sure the trained model is also included in your app. In the root folder of your project, include a file "gestures.txt" in which you describe the two gestures you trained your model to recognize, e.g. GESTURE-1 = spinning the phone around his x-axis, so that your app can be properly evaluated.