Mobile and Wearable Computing SA 2024-2025
Università della Svizzera italiana

Assignment 2

Paolo Deidda
paolo.deidda@usi.ch
https://github.com/USI-Projects-Collection/MWCTutorial04.git

October 23, 2024

Contents

Important notes

asdfasdf

1 Exercise 1 – Android Basics

1. **What does the `android:minSdkVersion` in an Android project indicate?**

   The `android:minSdkVersion` attribute specifies the minimum Android OS version that the app can run on. It ensures that the app will only be installable on devices with an OS version *equal to or higher* than the declared version. For example, if `minSdkVersion` is set to 21, the app will only work on devices running Android 5.0 (Lollipop) or newer.

2. **Why does Android documentation indicate that declaring the attribute `android:maxSdkVersion` is not recommended?**

   The Android documentation advises against using `maxSdkVersion` because it restricts the app's availability for future Android versions. If this attribute is set, the app won't be available for users running newer versions of Android, even if it could still work. This can cause compatibility and distribution issues as new Android versions are released.

3. **What are the two types of Navigation Drawer? Explain the differences between the two types.**

   There are two types of Navigation Drawer in Android:

   - **Permanent Navigation Drawer**: This type is always visible alongside the app's content, often used in tablet layouts or on large screens. The main content of the app is displayed next to the drawer.
   - **Modal Navigation Drawer**: This type is hidden by default and slides in over the app's content when triggered. It is commonly used in mobile apps where screen space is limited.

   Differences: The permanent drawer is better suited for larger screens where space is not an issue, while the modal drawer is more suitable for smaller screens, as it saves space by presenting the drawer as an overlay.

2 Exercise 2 – Material Design

2.1 Change App Icon

To change our app's icon, I used Android Studio's built-in feature to generate icons (see Figure 1). I navigated to the `res` folder, right-clicked on the `drawable` folder, and selected *New* → *Image Asset*. After selecting the image I had previously imported, Android Studio automatically generated the required icons.

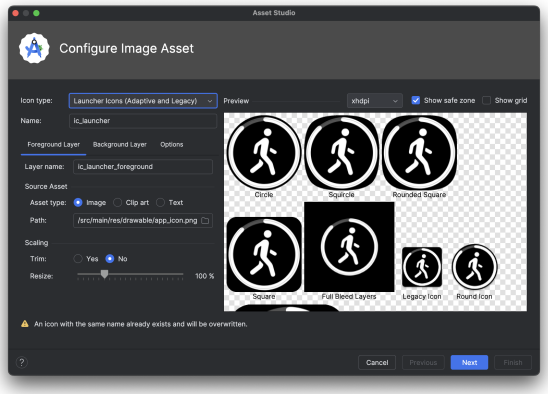

Figure 1: Generating Icons in Android Studio

Android Studio handled the generation of all necessary formats for the image, including various resolution sizes and shapes (see Figure 2).

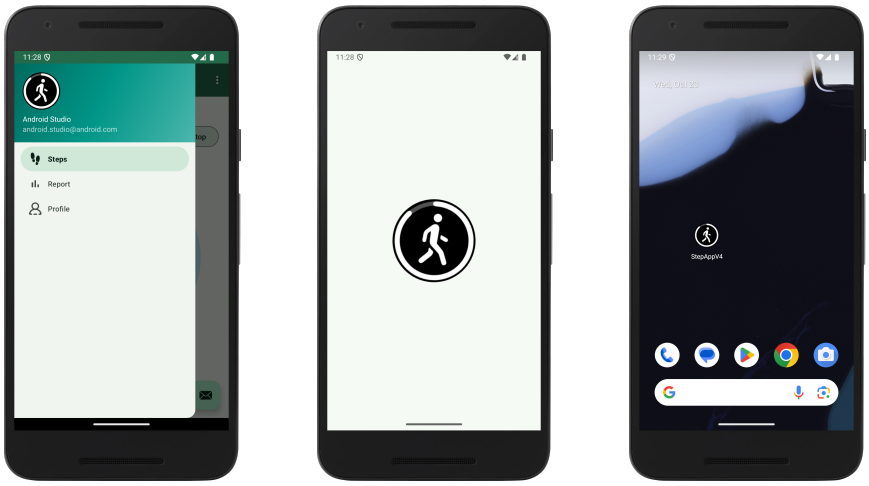

Figure 2: Generated Icons with Different Resolutions and Shapes

2.2 Implement Dark Theme

To implement the dark theme, I utilized the `AppCompatDelegate` class provided by Android's support library. This allows for easy switching between light and dark modes. The dark mode can be toggled by the user via a switch placed in the `Profile` page (see Figure 3).

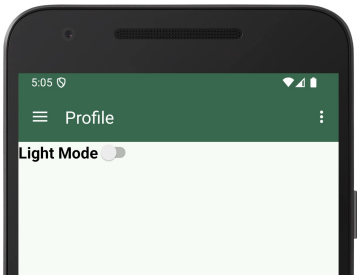

Figure 3: Switch for toggling between Light and Dark Modes

The dark mode implementation involved modifying the `ProfileFragment` and adding an `OnCheckedChangeListener` to the switch. Depending on whether the switch is checked or not, the theme is switched using `AppCompatDelegate.setDefaultNightMode()`. If the switch is turned on, the dark theme is enabled using `AppCompatDelegate.MODE_NIGHT_YES`, and if the switch is off, the light theme is restored using `AppCompatDelegate.MODE_NIGHT_NO`.

The relevant code for the theme switching logic is shown below:

../app/src/main/java/com/example/stepappv4/ui/slideshow/ProfileFragment.java

```
45      private void updateSwitchText() {
46          int currentNightMode = getResources().getConfiguration().uiMode & android.
                content.res.Configuration.UI_MODE_NIGHT_MASK;
47          if (currentNightMode == android.content.res.Configuration.UI_MODE_NIGHT_YES) {
48              // Dark mode is active
49              darkModeSwitch.setChecked(true);
50              darkModeSwitch.setText("Dark Mode");
51          } else {
52              // Light mode is active
53              darkModeSwitch.setChecked(false);
54              darkModeSwitch.setText("Light Mode");
55          }
56      }
```

Additionally, I ensured that the `colors.xml` file contained separate color definitions for both light and dark modes . This file is located in the `res/values` and `res/values-night` directories for light and dark modes, respectively. Same thing I had to do for the Themes, this prevented the app from changing layout changes. The following figures are some screenshots of the app in dark mode (see Figure 4).

Figure 4: App Screenshots in Dark Mode

3 Exercise 3 – Step Counter

3.1 Android STEP_DETECTOR

First of all I modified the variable `accSensor` in the `StepsFragment` class to be of type: `Sensor.TYPE_STEP_DETECTOR`. I could have easly have defined another variable but I wanted to make sure that all the counts were updted thanks to this sensor and not the previsouly used `Sensor.TYP_LINEAR_ACCELERATION`.

```
accSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
```

Then I moved to the `StepCounterListener` class and mofied the `onSensorChanged` method to call the `countSteps` method.

../app/src/main/java/com/example/stepappv4/StepCounterListener.java

```
101             case Sensor.TYPE_STEP_DETECTOR:
102                 countSteps(sensorEvent.values[0]);
103                 break;
```

../app/src/main/java/com/example/stepappv4/StepCounterListener.java

```
150     private void countSteps(float step) {
151         accStepCounter += step;
152
153         Log.d("ACC STEPS: ", String.valueOf(accStepCounter));
154
155         saveStepInDatabase();
```

In this first part of the method I add the steps detected by the sensor to the step count variable, log the number of steps detected and save the current step count to the shared preferences.

3.2 Update Circulat Progress Bar

In the second part of method `countSteps` from the previous exercise I update the step count text view and the step count progress bar.

../app/src/main/java/com/example/stepappv4/StepCounterListener.java

```
150     private void countSteps(float step) {
151         accStepCounter += step;
152
153         Log.d("ACC STEPS: ", String.valueOf(accStepCounter));
154
155         saveStepInDatabase();
156
157         // update View ex 3.2
158         stepCountsView.setText(String.valueOf(accStepCounter));
159         progressBar.setProgress(accStepCounter);
160     }
```

3.3 Persistant Step Count

In the last part of the `onCreate` method, I implemented functionality to load the number of steps from the database. This ensures that the step count persists after the user swtched to another fragment or even after the application is closed and reopened.

First, I retrieve the current timestamp and format it to extract the date. This is important because I want to load the step count corresponding to the current day.

../app/src/main/java/com/example/stepappv4/ui/steps/StepsFragment.java

```
99          long timeInMillis = System.currentTimeMillis();
100         SimpleDateFormat jdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss:SSS");
101         jdf.setTimeZone(TimeZone.getTimeZone("GMT+2"));
102         final String dateTimestamp = jdf.format(timeInMillis);
103         String currentDay = dateTimestamp.substring(0,10);
```

Next, I use the `loadSingleRecord` method from the `StepAppOpenHelper` class to retrieve the number of steps for the current day. This method queries the database and returns the stored step count, which I then display on the `stepsTextView` and update the progress bar accordingly.

../app/src/main/java/com/example/stepappv4/ui/steps/StepsFragment.java

```
105         // Load the number of steps from the database
106         int numberOfSteps = StepAppOpenHelper.loadSingleRecord(getContext(),
                currentDay);
107         stepsTextView.setText(String.valueOf(numberOfSteps));
108         progressBar.setProgress(numberOfSteps);
```