Assignment 2

Mobile and Wearable Computing

Alessandro Gobbetti

October 23, 2024

Contents

1 Introduction 1

2 Exercise 1: Android Basics 1

3 Exercise 2: Material Design 2

4 Exercise 3: Step Counter 3

Università della Svizzera italiana

Faculty of Informatics

1 Introduction

This is the second assignment for the Mobile and Wearable Computing course. The code is available at: https://github.com/Alessandro-Gobbetti/Mobile-and-Wearable-Computing/Assignment2.

2 Exercise 1: Android Basics

Question 1: What does the `android:minSdkVersion` in android project indicate?

Answer: Android applications should declare the API level they are targeting (`android:minSdkVersion` and `android:targetSdkVersion`). The `android:minSdkVersion` attribute indicates the minimum API level required for the application to run. Lowering the `minSdkVersion` allows the app to run on older devices, but it also limits the features that can be used. The `android:targetSdkVersion` attribute indicates the highest API level that the app has been tested on. This allows the app to take advantage of new features and optimizations in newer versions of Android.

If a device's API level is below the `minSdkVersion` , the app will not be installed. On the other hand, if the device's API level is above the `targetSdkVersion` , the app will still run, but it may not take advantage of the latest features and optimizations.

Question 2: Why Android documentation indicates that declaring the attribute `android:maxSdkVersion` is not recommended?

Answer: The `android:maxSdkVersion` indicates the maximum API level that the app can run on. However, the Android documentation recommends against using this attribute because it can lead to compatibility issues. If the `maxSdkVersion` is set to a specific API level, the app will not be available on devices with a higher API level. This can prevent users from installing the app on newer devices, even if the app is fully compatible with the newer API level. By design, newer versions of Android are backward compatible, so apps should be able to run on newer devices without issues. Additionally, in some cases, declaring the `maxSdkVersion` attribute can result in the app being removed from users' devices after system updates to a higher API level. This can lead to a poor user experience and loss of data.

Question 3: What are the two types of Navigation Drawer? Explain the differences between the two types?

Answer: Navigation drawers provide access to destinations and app functionality. There are two types of navigation drawers: the standard drawer and the modal drawer.

The standard drawer is a panel that shares the screen with the main content. It allows users to simultaneously access drawer destinations and app content. The standard drawer can be permanently visible or opened and closed by tapping a navigation menu icon. It is often used on tablets and desktops where there is more screen space.

The modal drawer is more common on mobile devices where screen space is limited. The modal drawer slides in from the edge of the screen and overlays the main content, blocking interaction with the content until the drawer is closed.

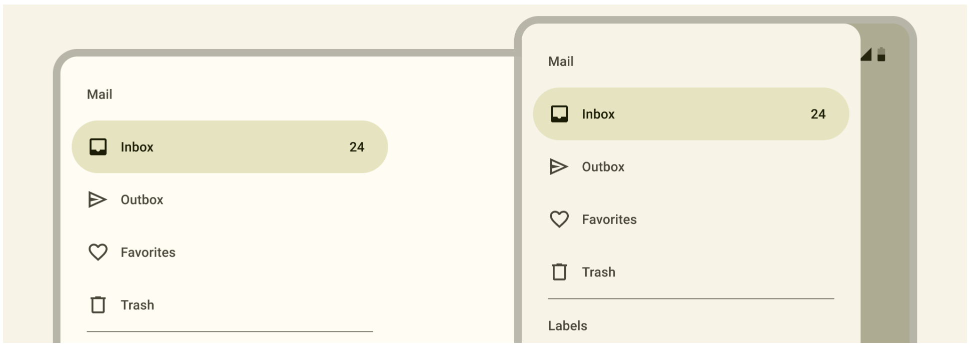


Figure 1: Standard [Left] and Modal [Right] Navigation Drawers

3 Exercise 2: Material Design

Task 1: Change the icon of the app. Choose/create a representative icon for your StepAppV4. Set the default Android icon to the icon you chose/created.

A new icon was created for the StepAppV4 application. The icon was downloaded from SVGRepo and used as the foreground, with the background consisting of a solid color that matches the application's main theme.

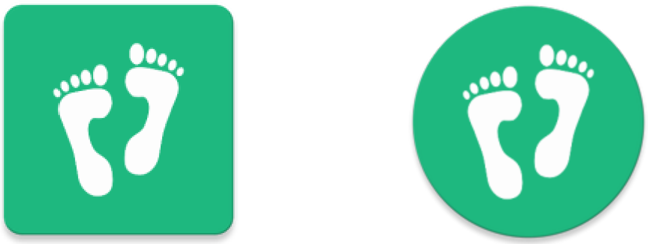
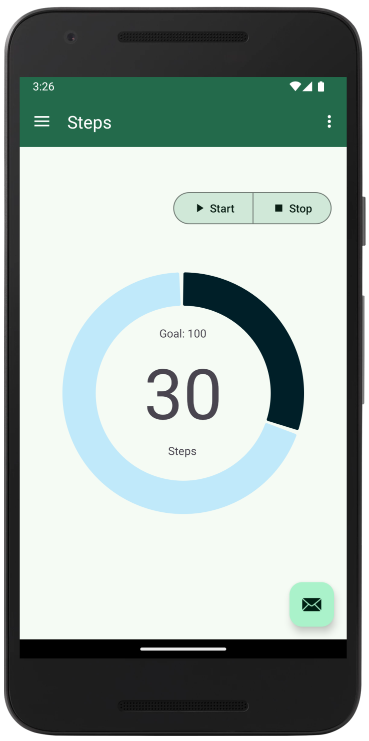


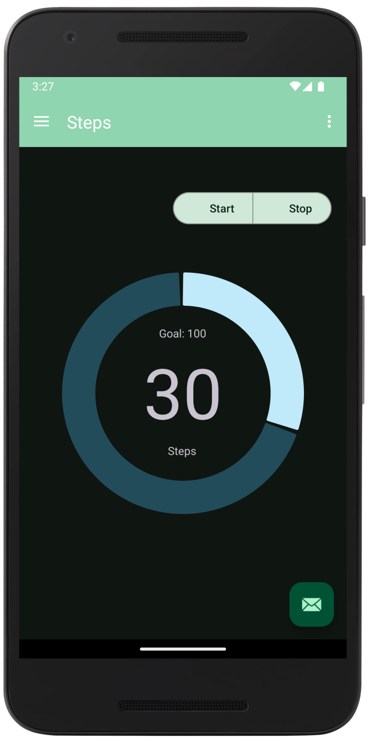
Figure 2: App Icon

Task 2: Implement the dark theme for the layout of your StepAppV4 app.

As done during the tutorial for the light theme, the dark theme was implemented. The new colors and themes (renamed to start with `dark_md_theme`) were created and added to the `colors.xml` and `themes.xml` files.



Light Theme



Dark Theme

Figure 3: Light and Dark Themes

4 Exercise 3: Step Counter

The main goal of this exercise is to use the Android STEP_DETECTOR sensor is used to detect steps. To do this, in the `onSensorChanged()` method of the `class StepCounterListener` we need to handle the case when the sensor type is `Sensor.TYPE_STEP_DETECTOR`. The STEP_DETECTOR is triggered every time a step is detected. So we just have to store the newly detected step in the database.

```java
public class StepCounterListener implements SensorEventListener {
    ...
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        switch (sensorEvent.sensor.getType())
        {
            case Sensor.TYPE_LINEAR_ACCELERATION:
                ...
                break;
            case Sensor.TYPE_STEP_DETECTOR:
                countSteps(sensorEvent.values[0]);
                break;
        }
    }
    ...

    private void countSteps(float step)
    {
        // increment the step counter
        stepDetectorStepCounter += 1;
        // Log the number of steps detected by the step detector
        Log.d("STEP DETECTOR STEPS: ", String.valueOf(stepDetectorStepCounter));
        // Save the number of steps
        saveStepInDatabase();
        // Update the TextView and the progress bar
        stepCountsView.setText(String.valueOf(stepDetectorStepCounter));
        progressBar.setProgress(stepDetectorStepCounter);

    }
    ...
}
```

The `countSteps()` method is used to increment the class variable `stepDetectorStepCounter`, save the step in the database, update the TextView and the progress bar with the new values only detected by the STEP_DETECTOR sensor.

Finally, when the app page is created, the number of steps is retrieved from the database and displayed in the TextView and the progress bar. In the `onCreateView()` method of the `class StepsFragment` the progress bar and the TextView are immediately updated with the number of steps stored in the database.

```java
public class StepsFragment extends Fragment {
    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        ...

        // Load the steps counter from the database
        //Timestamp
        long timeInMillis = System.currentTimeMillis();
        // Convert the timestamp to date
        SimpleDateFormat jdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss:SSS");
        jdf.setTimeZone(TimeZone.getTimeZone("GMT+2"));
        final String dateTimestamp = jdf.format(timeInMillis);
        String currentDay = dateTimestamp.substring(0,10);
        stepsCounter = StepAppOpenHelper.loadSingleRecord(getContext(), currentDay);

        ...

        // Set the progress bar
        CircularProgressIndicator progressBar =
                (CircularProgressIndicator) root.findViewById(R.id.progressBar);
        progressBar.setMax(100);
        progressBar.setProgress(stepsCounter);

        // Set the steps counter
        stepsTextView = (TextView) root.findViewById(R.id.stepsCount_textview);
        stepsTextView.setText(""+stepsCounter);
        ...
    }
}
```