University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Android app for cost-effective rich multimedia notifications for social media applications

Bachelor's Thesis in Computer Science
by

Andreas Østhus Saltveit

Internal Supervisors

Vinay Jayarama Setty

May 31, 2018

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

Rick Cook [1]

Abstract

Getting the most useful notifications from your available data and battery life is difficult problem. This thesis describes the development of a application which provides constraints to use in a algorithm solving this problem. Using a combination of workmanager, telephonymanager, settingsactivity, JSONObject, SharedPreferences, Sticky intents and more from the android API, the constraints are regularly sent to the algorithm running on a backend. Notifications are generated and received using Firebase Cloud Messaging and displayed as rich notifications on the device.

Acknowledgements

I would like to thank Vinay Setty for beeing a great supervisor during my thesis.

Contents

Abstract **vi**

Acknowledgements **viii**

Abbreviations **xi**

Symbols **xiii**

1 Introduction **1**
- 1.1 Problem Definition . . . 1
- 1.2 Challenges . . . 1
- 1.3 Contributions . . . 2
- 1.4 Outline . . . 2

2 Related Work **3**
- 2.1 Richnote . . . 3
- 2.2 The Backend Server . . . 3

3 Construction **5**
- 3.1 Introduction . . . 5
- 3.2 Tools . . . 5
  - 3.2.1 Version Control . . . 5
  - 3.2.2 The Android Operating System . . . 5
  - 3.2.3 Android Studio . . . 8
  - 3.2.4 Gradle . . . 9
  - 3.2.5 Firebase Cloud Messaging . . . 9
- 3.3 Code . . . 9
  - 3.3.1 The App Manifest [2] . . . 9
  - 3.3.2 MainActivity . . . 14
  - 3.3.3 UpdateBackend . . . 16
  - 3.3.4 SettingsActivity . . . 16
  - 3.3.5 FCM . . . 16

4 Conclusion and Future Work **17**

List of Figures **17**

List of Tables **21**

Bibliography **23**

Abbreviations

Acronym	**W**hat (it) **S**tands **F**or
LAH	**L**ist **A**bbreviations **H**ere
OS	**O**perating **S**ystem

Symbols

symbol	name	unit
a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

Chapter 1

Introduction

1.1 Problem Definition

The object of this thesis is to develop a app for devices running the android operating system. The app needs to get constraints from the device and send them to backend server. The constraints are:

- remaining battery percentage of the device
- Internet connection type
- mobile data remaining in current month
- whether the device is currently charging or not

The app developed in this thesis also needs to receive and display rich notifications that are generated and sent from the backend and to provide it with Spotify login credentials securely. Furthermore the backend needs information from the app, that enables unique identification of a specific installation and correlation of login information previously mentioned.

1.2 Challenges

Starting developing android for the first time on the Android platform require extensive reading and practice. Several parts of the code in official guides also do not compile, adding difficulty, as the guides have not been updated quickly enough compared to the development of the Android OS.

1.3 Contributions

The app developed in this thesis can:

- get remaining battery percentage of the device
- get Internet connection type
- get mobile data remaining in current month
- get whether the device is currently charging or not
- display rich notifications to user

1.4 Outline

The goal of this thesis is to develop an android app to track the smartphone status such as battery, bandwidth availability and mobile data quota etc. The measurements are then sent to a backend service which intelligently adapts the quality of multimedia notifications according to the device status.

The app will be designed as a client for notifications from Spotify as well as social networks such as Facebook. The app authenticates users with the help of APIs from these services and receives notifications from the backend. From the device side it collects and sends the device status to the backend and receives and displays the notifications.

The backend service acts as a forwarding agent for notifications from the original services such as Spotify and Facebook. The backend generates the notifications for the users using the authentication tokens obtained via user credentials. The backend service, according to the device status, transforms the notifications (for e.g, lower resolution images in Facebook) and forwards it to the client app. The forwarding of notifications is done using Google's Firebase Cloud Messaging service.

Chapter 2

Related Work

2.1 Richnote

In the paper on richnote[3], an algorithm is suggested to best solve the problem of delivering content rich multimedia notifications scaled down based on constraints of the device they are to be displayed on.

2.2 The Backend Server

The backend and the client app is in development at the same time. The algorithm has been tested to be more accurate than the richnote algorithm according to the author Mats Jonassen.

Chapter 3

Construction

3.1 Introduction

[4]

3.2 Tools

This section is about the different tools used during the construction of the app.

3.2.1 Version Control

Git was used as the version control during the development of the app. It was chosen because the developer already knew how to use it. GitHub for windows was used to backup the project. The full GitHub project is available online [5].

3.2.2 The Android Operating System

Android OS is the mobile operating system with the largest market share at the time of this writing[6]. Android will be celebrating its 10 year anniversary on September 23, 2018[7]. Figure 3.1 shows the major components of the Android platform. The two official programming languages of Android applications are Kotlin and Java, the latter being most used today. The OS is open source, and anyone can contribute to the development[8].

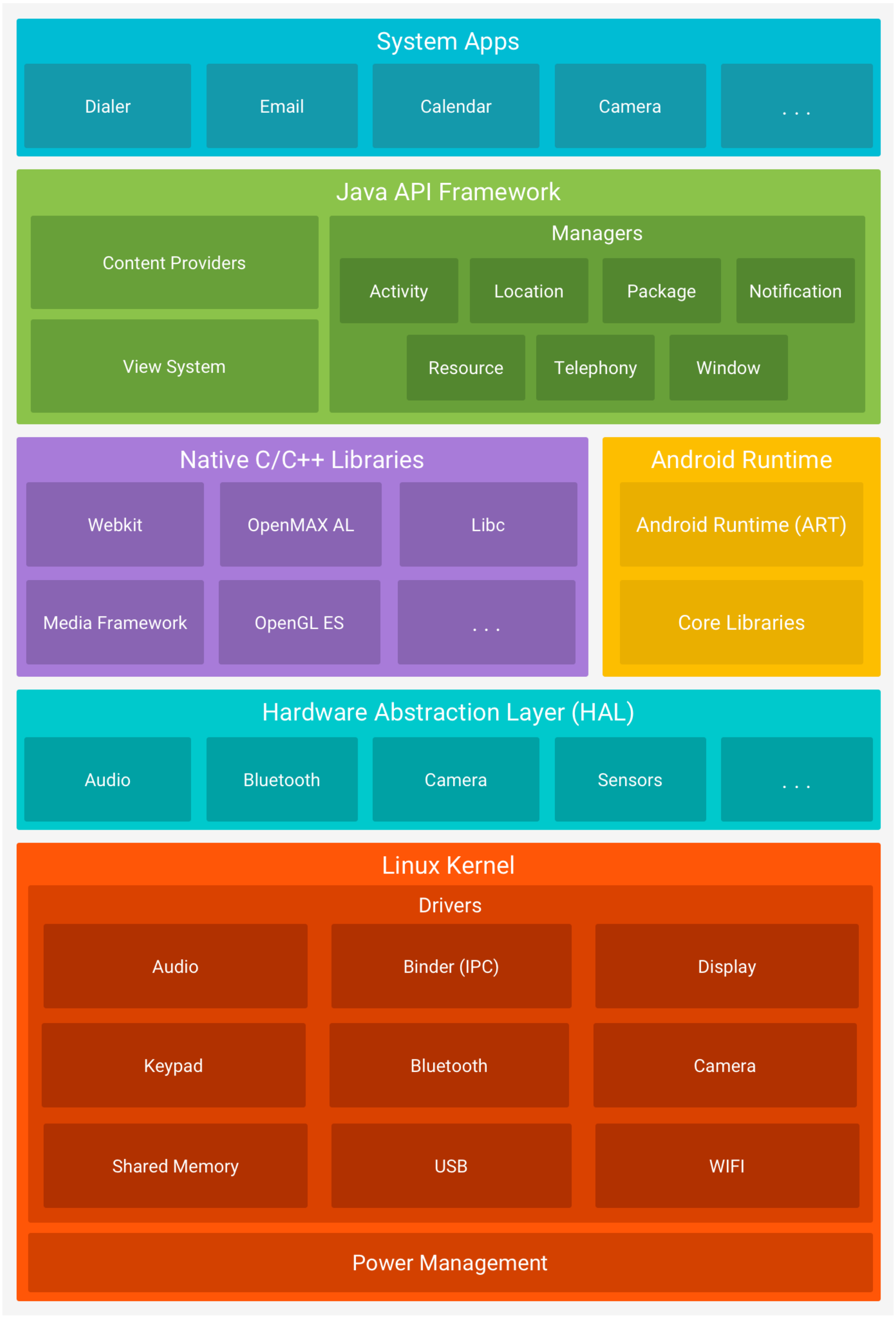


Figure 3.1: The Android software stack [9]

The Linux Kernel

"The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel allows Android to take advantage of key security features and device manufacturers to develop hardware drivers for a well-known kernel." [9]

Android Runtime

"For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields...

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features, that the Java API framework uses." [9]

Java API Framework

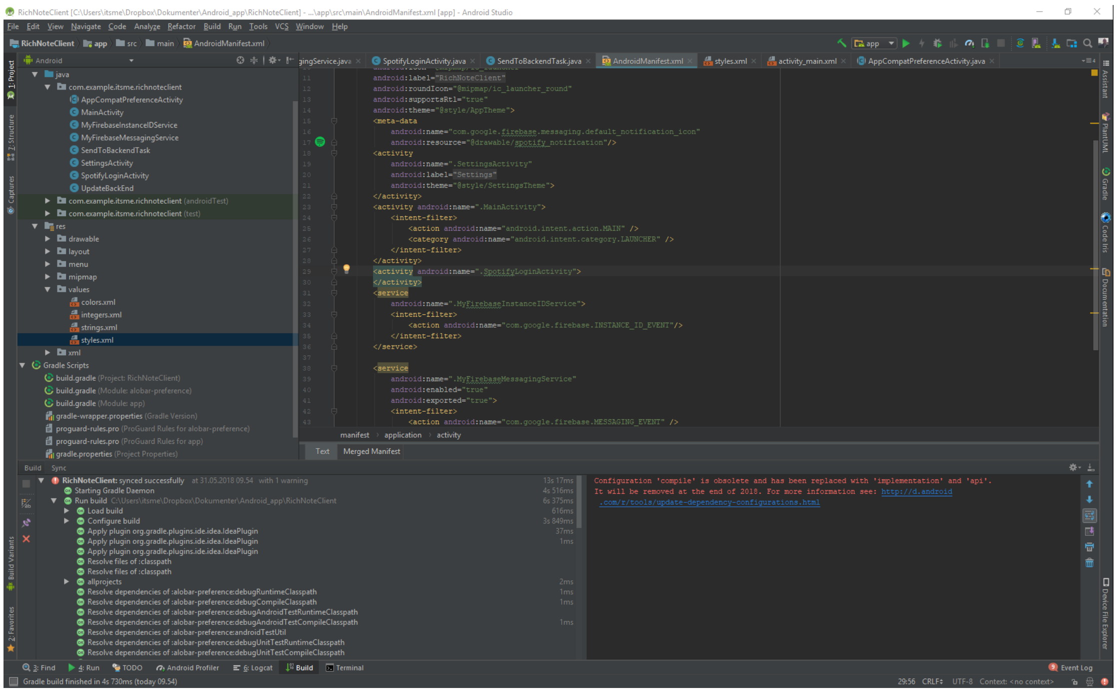
"The entire feature-set of the Android OS is available through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible View System you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack

- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same framework APIs that Android system apps use." [9]

3.2.3 Android Studio



```
        android:label="RichNoteClient"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.google.firebase.messaging.default_notification_icon"
            android:resource="@drawable/spotify_notification"/>
        <activity
            android:name=".SettingsActivity"
            android:label="Settings"
            android:theme="@style/SettingsTheme">
        </activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SpotifyLoginActivity">
        </activity>
        <service
            android:name=".MyFirebaseInstanceIDService">
            <intent-filter>
                <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
            </intent-filter>
        </service>

        <service
            android:name=".MyFirebaseMessagingService"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.google.firebase.MESSAGING_EVENT" />
```

```
RichNoteClient: synced successfully  at 31.05.2018 09.54  with 1 warning      13s 17ms
  Starting Gradle Daemon                                                       4s 516ms
  Run build C:\Users\itsme\Dropbox\Dokumenter\Android_app\RichNoteClient       6s 375ms
    Load build                                                                 616ms
    Configure build                                                            3s 849ms
    Apply plugin org.gradle.plugins.ide.idea.IdeaPlugin                        37ms
    Apply plugin org.gradle.plugins.ide.idea.IdeaPlugin                        1ms
    Apply plugin org.gradle.plugins.ide.idea.IdeaPlugin
    Resolve files of :classpath
    Resolve files of :classpath
    allprojects                                                                2ms
    Resolve dependencies of :alobar-preference:debugRuntimeClasspath           1ms
    Resolve dependencies of :alobar-preference:debugCompileClasspath           1ms
    Resolve dependencies of :alobar-preference:debugAndroidTestRuntimeClasspath
    Resolve dependencies of :alobar-preference:debugAndroidTestCompileClasspath  1ms
    Resolve dependencies of :alobar-preference:androidTestUtil
    Resolve dependencies of :alobar-preference:debugUnitTestRuntimeClasspath
    Resolve dependencies of :alobar-preference:debugUnitTestCompileClasspath
```

```
Configuration 'compile' is obsolete and has been replaced with 'implementation' and 'api'.
It will be removed at the end of 2018. For more information see: http://d.android
 .com/r/tools/update-dependency-configurations.html
```

```
Gradle build finished in 4s 730ms (today 09.54)
```

RichNoteClient [C:\Users\itsme\Dropbox\Dokumenter\Android_app\RichNoteClient] - ...\app\src\main\AndroidManifest.xml [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SpotifyLoginActivity.java
SendToBackendTask.java
AndroidManifest.xml
styles.xml
activity_main.xml
AppCompatPreferenceActivity.java
manifest › application › activity
Text
Merged Manifest
Build
Sync
3: Find
4: Run
TODO
Android Profiler
6: Logcat
Build
Terminal
Event Log
29:56 CRLF UTF-8 Context: <no context>

Figure 3.2: Android Studio

"Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code

- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine"[10]

3.2.4 Gradle

The flexible Gradle-based build system is so well integrated, that you barely have to know anything about it to use Android Studio. However when it fails to parse some of the dependencies of the project, it can get time consuming. During the development of the application the upgrade of android studio would sometimes break the project. A complete removal and reimporting av the project would be needed to continue development. Three files relating to Gradle where edited without automation. The project build.gradle file, the app build.gradle file and the settings.gradle file. They were edited to set minimum android SDK versions, add directory with the alobar-preference[11] library that was used in the SettingsActivity.3.3.4

3.2.5 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) was chosen as the delivery service for the notifications coming from the backend. This was chosen because it is free and also developed by Google[12].

3.3 Code

3.3.1 The App Manifest [2]

Every app project must have an AndroidManifest.xml file (with precisely that name) at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building your project. When packaging the app, the build tools replace this value with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play.

- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Kotlin or Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started.
- The permissions that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.

The app manifest is shown in listing 3.1

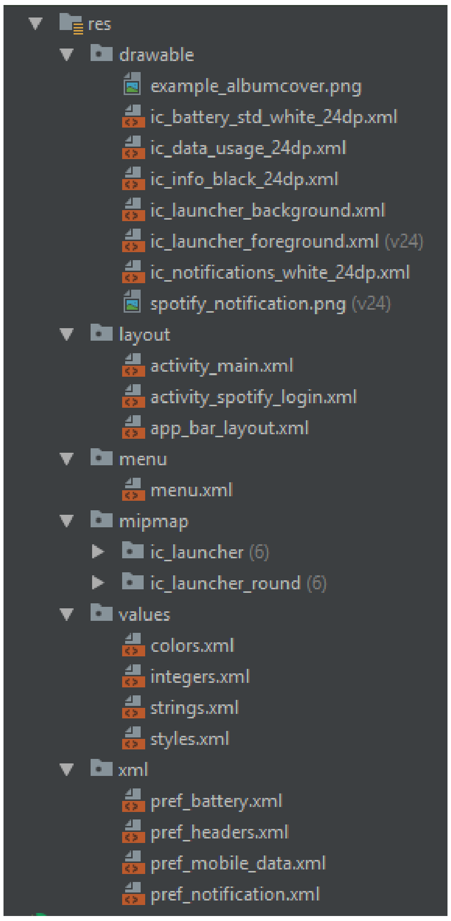
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.itsme.richnoteclient">
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/spotify_notification"/>
    <activity
        android:name=".SettingsActivity"
        android:label="@string/SettingsActivityName"
        android:theme="@style/SettingsTheme">
    </activity>
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".SpotifyLoginActivity">
    </activity>
    <service
        android:name=".MyFirebaseInstanceIDService">
        <intent-filter>
            <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
        </intent-filter>
    </service>
    <service
        android:name=".MyFirebaseMessagingService"
        android:enabled="true"
        android:exported="true">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
        <action android:name="com.firebase.jobdispatcher.ACTION_EXECUTE" />
    </intent-filter>
    </service>
</application>
</manifest>
```

Listing 3.1: app manifest

The first two lines in the app manifest are automatically generated my Android Studio. These lines are used by the Android Runtime Environment(ARE), the first line to uniquely identify elements in the rest of the manifest, the second to uniquely identify classes in the project. The next few lines declares permissions that the app uses in some of its classes.

Permissions [13]

Permissions are part of the android Application Programming Interface (API). The purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request. Some permissions are never granted to 3rd party apps.



```
▼ res
    ▼ drawable
          example_albumcover.png
          ic_battery_std_white_24dp.xml
          ic_data_usage_24dp.xml
          ic_info_black_24dp.xml
          ic_launcher_background.xml
          ic_launcher_foreground.xml (v24)
          ic_notifications_white_24dp.xml
          spotify_notification.png (v24)
    ▼ layout
          activity_main.xml
          activity_spotify_login.xml
          app_bar_layout.xml
    ▼ menu
          menu.xml
    ▼ mipmap
        ▶ ic_launcher (6)
        ▶ ic_launcher_round (6)
    ▼ values
          colors.xml
          integers.xml
          strings.xml
          styles.xml
    ▼ xml
          pref_battery.xml
          pref_headers.xml
          pref_mobile_data.xml
          pref_notification.xml
```

Figure 3.3: The App Res folder

The first two permissions are self-explanatory. The app needs internet to communicate with the backend, and access of the devices network state to monitor the type of connectivity currently in use. The permission in line 6 is a in a group that is labeled by

Google as dangerous. This means that the app has to ask the user for this permission using a dialog at runtime. More about this in the MainActivity class 3.3.2. This permission is used to acquire a identification(ID) that comes from the physical sim-card in the phone. This ID is used in the UpdateBackend class 3.3.3. From line 7 onward, the lines are inside the application tag. This means we only affect the application with the next lines. The preceding lines affected the package, which could have several applications in it. Since the app is not published on the Google Play store, line 8 has no effect on the application. In lines 9 through 13, resources from the Res directory of the project is referenced. In the resources directory there are things that are used depending on the characteristics of the device the application is deployed on. IE there would be different sizes of the images used for the launcher icons that are displayed on the home screen of the device, depending on the size of the screen. The app's resource folder is pictured in figure 3.3. Further in the manifest we assign the launcher and round icon of the app. One of these are used depending on the version of android running on the device. The services and meta data tags later are added by requirements to use the FCM service.

AppTheme

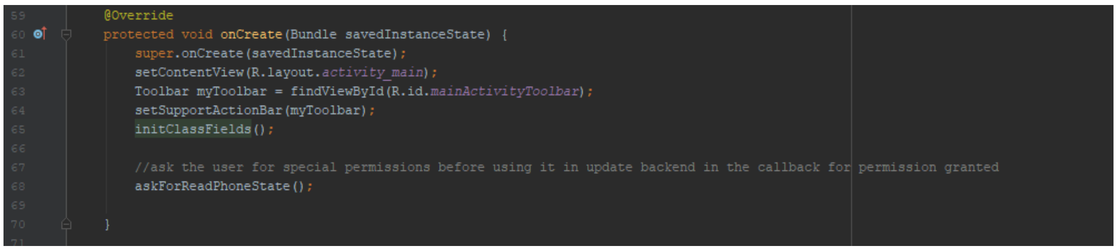
```xml
<resources>
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.NoActionBar">
<!-- Customize your theme here. -->
<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
<style name="SettingsTheme" parent="Theme.AppCompat">
<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
</resources>
```

Listing 3.2: styles.xml

We assign the colors and the app class in line 12. We get it from the styles XML file. The theme is set to AppCompat's subclass NoActionBar which is a application class that has backwards compatibility and no actionbar. This is done to enable the addition of the appCompatActionbar in the MainActivity class which also has backwards compatibility. The settings theme in the styles.xml file is used for the SettingsActivity. You can view the styles.xml in listing 3.2. The remaining lines in the Manifest declare the MainActivity Intent-filters and the SpotifyLoginActivity.

3.3.2 MainActivity

the MainActivity is the first class where code gets executed on android applications. It is analogues to the main method in Java. The reason for this is the registered intent-filter from the manifest.



```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar myToolbar = findViewById(R.id.mainActivityToolbar);
        setSupportActionBar(myToolbar);
        initClassFields();

        //ask the user for special permissions before using it in update backend in the callback for permission granted
        askForReadPhoneState();

    }
```

Figure 3.4: MainActivity's onCreate method

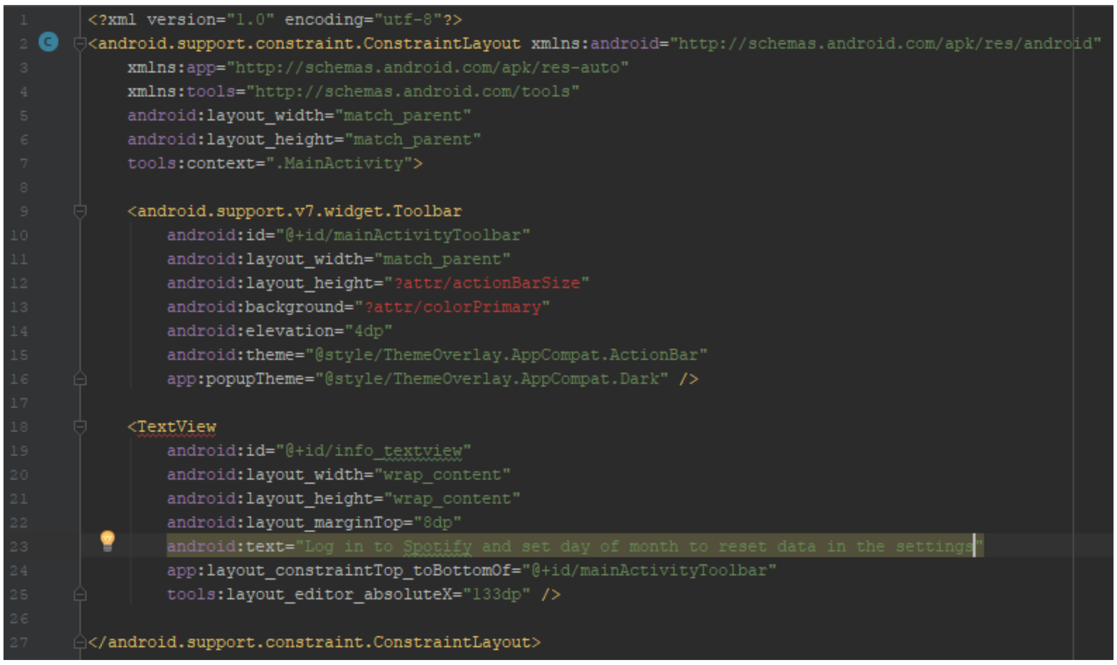
Intents and Intent-filters

Intents are basically a way to send data too, and receive data from, other activities or services. The filters are used to filter which intents actually gets through to the activity[14].

Activity

"The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle. Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states."[15]

You must implement the onCreate callback in every activity. In MainActivity's creation we set the layout to the activity_main XML file. In this layout we have added the toolbar to show the app name and menu items. There is also a Text View used to give a new user a brief setup manual. You can read the manual in figure 3.5. Then the toolbar is added and this triggers the onCreateOptionsMenu callback. In the override of this method we use a API call to inflate the menu declared in the menu XML file. We also make sure to remove the Spotify log in option if the user has previously logged inn. After menu has been added, the initClassFields method is called. In the initialization method the get the text view by its id, which we access from the strings XML file. We also make



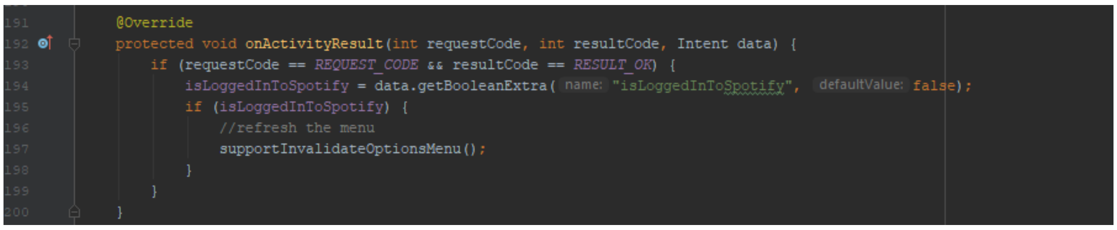
```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
   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

    <android.support.v7.widget.Toolbar
        android:id="@+id/mainActivityToolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:elevation="4dp"
        android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Dark" />

    <TextView
        android:id="@+id/info_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="Log in to Spotify and set day of month to reset data in the settings"
        app:layout_constraintTop_toBottomOf="@+id/mainActivityToolbar"
        tools:layout_editor_absoluteX="133dp" />

</android.support.constraint.ConstraintLayout>
```

Figure 3.5: Main Activity Layout file



```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        isLoggedInToSpotify = data.getBooleanExtra( name: "isLoggedInToSpotify",  defaultValue: false);
        if (isLoggedInToSpotify) {
            //refresh the menu
            supportInvalidateOptionsMenu();
        }
    }
}
```

Figure 3.6: on activity result callback method in main activity

use of the Android API to get access to a file where we can store and retrieve key-value pairs on persistent storage. Using the file we check if the user has logged in to Spotify.

The items in the menu have an event listener that gets triggered should the user click them. This method is called on option item selected. In the method we launch the Spotify log in activity and register the main activity for a to a callback function when the spotify activity has finished.

The callback uses the intent parameter and checks if the user is logged in. If the user is logged in, the menu gets created again so that the menu item for logging in is set to invisible. See the complete code in figure 3.6.

The final lines in main activity prompts the user for the permission to read the phone state. This permission is needed to retrieve mobile data usage of the application. When the dialog is closed, a check for permission granted is performed. If we have the permission we schedule to update the backend periodically. This is done with the newly released WorkManager API. It guaranties code execution that was planned, energy efficiently, in the background on another thread. The User Interface is not stalled by this, since it runs on a different thread. The update is scheduled to run once every 30-60 minutes. It only runs if the device has a Internet connection available. It passes the UdateBackend worker class to a work request, which is enqueued with the Workmanager system service.

3.3.3 UpdateBackend

UpdateBackend is an extension of the Worker class from the Workmanager API. The algoriths in it refreshes the constraints on the backend. It puts the data in a json string and sends it to the backend via http POST request.

3.3.4 SettingsActivity

The settings activity is mostly generated by the IDE using a creation dialog. A number picker preference is added from the alobar library imported with gradle. It has eventlisteners for preference changes, and stores values in the same file retrieved in the main activity class in the initialize method.

3.3.5 FCM

Firebase messaging is implemented with the extension of the FirebaseInstanceIdService, to listen for unique identifiers of the installation of the app. This is sent to the backend and a response with a unique spotify url is given in response from the server. This is used in a webview to open a dialog with Spotify so the user can log in. This is leaves the security concerns to the Spotify oauth API implemetation.

The extension of the Firebasemessagingservice is used to receive messages that can wake up the device and execute code in the background in spite of new restrictions imposed by the latest android version: OREO. Here notifications are prepared and then sent to the system notification service provided by android. Standard layouts are used.

Chapter 4

Conclusion and Future Work

In myFirebaseInstanceId we get a unique identifier and send it to backend, we receive a unique link to login to spotify. This is done in Spotifyactivity. MyFirebaseMassagingService receives and pressents rich notifications to the user. UpdateBackend worker retrieves constraints and send them to the backen.

List of Figures

3.1 The Android software stack [9] 6
3.2 Android Studio 8
3.3 The App Res folder 12
3.4 MainActivity's onCreate method 14
3.5 Main Activity Layout file 15
3.6 on activity result callback method in main activity 15

List of Tables

Bibliography

[1] Rick Cook. *The wizardry compiled.* Baen, 1990.

[2] App manifest overview | android developers, May 2018. URL https://developer.android.com/guide/topics/manifest/manifest-intro.

[3] M. Y. S. Uddin, V. Setty, Y. Zhao, R. Vitenberg, and N. Venkatasubramanian. Richnote: Adaptive selection and delivery of rich media notifications to mobile users. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 159–168, June 2016. doi: 10.1109/ICDCS.2016.107.

[4] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Higher Education, seventh international edition, 2010. ISBN 0071267824, 9780071267823.

[5] Detemegandy. detemegandy/android-app-bachelor. URL https://github.com/detemegandy/Android-app-bachelor/.

[6] Gartner says worldwide sales of smartphones returned to growth in first quarter of 2018, May 2018. URL https://www.gartner.com/newsroom/id/3876865.

[7] Announcing the android 1.0 sdk, release 1, Sep 2008. URL https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html.

[8] the android open source project, . URL https://source.android.com/.

[9] Platform architecture | android developers, Apr 2018. URL https://developer.android.com/guide/platform/.

[10] Meet android studio | android developers, . URL https://developer.android.com/studio/intro/.

[11] Alobar. Alobar/androidpreferencetest. URL https://github.com/Alobar/AndroidPreferenceTest/tree/master/alobar-preference.

[12] Firebase cloud messaging | firebase. URL https://firebase.google.com/docs/cloud-messaging/.

[13] Permissions overview | android developers, May 2018. URL https://developer.android.com/guide/topics/permissions/overview.

[14] Intents and intent filters | android developers. URL https://developer.android.com/guide/components/intents-filters.

[15] Introduction to activities | android developers. URL https://developer.android.com/guide/components/activities/intro-activities.