A Live Streaming App for Android devices

Abdul-Rahman Mawlood-Yunis
Physics and Computer Science
Department
Wilfrid Laurier Univeristy
Waterloo, On, Canada
amawloodyunis@wlu.ca

Abstract— In this paper, we present a live streaming app for Android devices using URL. The app is useful for two reasons; first, all your favorite radio stations will be grouped together in one place, and hence, you can easily play and switch from one station to another without any hassle. Second, it is an app that turns your device into a radio and lets you listen to live streaming stations while in the office, on the road, or in any other setting. The app is similar to Spotify but on a smaller scale (that's probably all you need; your favorite station and not all the stations that come with Spotify). The app has two main contributions: 1) we describe all the steps and components needed to develop such an app. We also discuss the functionality and the trade-offs using different components and approaches. 2) The app's source code and complete documentation can be used by instructors to teach various Android topics.

Keywords—Mobile Application, Android, MediaPlayer, Digital Streaming

I. Introduction

In this paper, we present a live streaming app for Android devices using URL. The app links one's favorite online radio stations and plays them on the user's devices. It is similar to Spotify but on a smaller scale (that's probably all you need; just your favorite station and not all the stations that come with Spotify). The app is useful for two reasons; first, all your favorite radio stations will be grouped together in one place, and hence, you can play and easily switch from one station to another without any hassle. Second, it is an app that turns your device into a radio and lets you to listen to live streaming stations from anywhere in the world and in any setting. For example, if you connect your phone with an audio cable or Bluetooth to the media player of your car, you can readily live-stream your favorite online radio channels and listen to them with multiple speakers.

The app has two main contributions: 1) we describe all the steps and components needed to develop such an app. We also discuss the functionality and the trade-offs using different components and approaches. 2) the app is open source and its complete documentation and source code can be used as classroom material in an Android class for teaching multiple topics: Android Service, broadcasting and receiving messages, using MediaPlayer object to stream radio stations and mange Android power and WIFI connection programmatically.

The current version of the app presents the author's favorite radio stations. Developers can easily refactor the app and replace the existing station with the ones they prefer. In the feature version of the app, we enable not only the developers but also end users to be able to customize the stations they listen too through the app's interface.

This paper is organized as follow: in Section 2, we study the app's components and architecture, in Section 3, we provide instructions on how to check out the app and run it, and in Section 4, we conclude the paper and describe some future works.

II. App components and Architecture

In the app, we use Service and MediaPlayer components to play live streaming radio stations from URL. We also use BraodcastReceiver to notify users of events. The MainActivity maintains a reference to the Service, thus it can make calls on the Service just as any other class and can directly access members and methods of the Service. It starts the Service which in turn gets radio URL form the MainActivity and starts the MediaPlayer. Below we describe each of these components. We discuss the functionality and the trade-offs using different components and approaches. We also present the class structure, i.e., the architecture, of the app.

A. Activity

The app's user interface and Service setup all done at the MainActivity class. MainActivity can communicate with the background Service to start and stop the MediaPlayer and it listens to the message broadcasts from the Service.

B. Service

Service is an app component that performs long-running tasks in the background with no graphical user interface. Once Service starts, it continues to run even if the original application is ended or the user moves to a different application. Service is the right choice to use when an activity does not interact with the user, i.e., is not the forefront activity.
Service can be private or public. When private, it is usable only by the app it belongs to; however, when public, it is usable by apps other than the app it belongs to, i.e., another app component can start Service using a call to the API. In current app the MainActivity compoent starts Service with the method call startService(). Once started, Service can run in the background indefinitely.

C. BroadcastReceiver

We use Broadcastreceiver to receive and handle broadcast Intents sent from Service by the sendBroadcast(Intent) method. The Broadcast class doesn't have user interface, but it can create a status bar notification to alert the user when a broadcast event occurs. Android system delivers a broadcast Intent to all interested (registered) broadcast receivers. Applications can initiate broadcast messages to let other applications know for example that some data has been downloaded to the device and is available for them to use.

The BroadcastReceiver needs to be instantiated and registered to process the broadcasted messages on arrival. The four steps involved in Message Broadcast and Receive are:

- Create the BroadcastReceiver Object
- Register BroadcastReceiver object to receive messages
- Message Broadcasting
- Actions performed upon receiving the Broadcasted message

D. *MediaPlayer*

We use MediaPlayer class to control playback of radio streams. Media player needs to be prepared, started and ultimately released. The MediaPlayer's lifecycle show that the player must first enter the Prepared state before playback can start and there are two ways that the prepared state can be entered:

1. Synchronous way using the prepare() method.
2. Asynchronous way using the prepareAsync() methods.

The difference between those methods is in what thread they are executed.

The **Prepare ()** method runs in the UI thread and thus it takes a long time. It will block your UI thread and a user might get an ANR (Application Not Responding) message.

The **PrepareAsync ()** method, on the other hand, runs in a background thread and thus your UI thread is not blocked. However, the MediaPlayer object might not prepared instantly so you want to set onPreparedListener in order to know when the MediaPlayer is ready for use.

The prepareAsync() method is generally used for playing the live data over stream and that is why in current app we are using the prepareAsync() method. It allows playing without blocking the main thread.

E. *PowerManager and WakeLock*

If phone goes into a low-power state it will prevent apps from running. To control the power state on device you need to use power management to:

a. keep the CPU running,
b. prevent screen dimming or going off or
c. prevent backlight from turning on

Android *WifiLock* class allows an application to keep the Wi-Fi component awake. Acquiring a WifiLock will keep the Wi-Fi on until the application releases the lock. In the app, we have decided even when the device screen is off to keep the radio stations running; hence in our app we acquire the wifilock. To use a Wifilock we added the below permission to the manifest file.

```
 <uses-permission
android:name="android.permission.WAKE_LOCK" />
```

F. *RADIO STATIONS*

The radio station names, the URLs and Image links are all saved or referenced at this component. This enables easy extension and changes. For example, if you change a streaming URL or an image, you only change it here without need to change any other parts of the code. Similarly, if you want to add a new station to your list, then you simply add the new station to the list of existing stations and there is no need to change other parts of the code.

G. *The manifest file*

To run Service, you need to declare it inside the manifest file as shown below, table 1. Also, proper permissions need to be added to the manifest file.

TABLE I. SNAPSHOT OF APP'S MANIFEST FILE

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/andro
id"

package="code.android.abdulrahman.android.serviceandm
ediaplayer" >

   <uses-permission
android:name="android.permission.INTERNET" />
   <uses-permission
android:name="android.permission.WAKE_LOCK" />
   <uses-permission
android:name="android.permission.ACCESS_WIFI_STAT
E"/>
   <uses-permission
android:name="android.permission.VIBRATE" />

   <application
      ...
      <activity ...
      </activity>

      <service
         android:name=".RadioService"
         android:enabled="true"
         android:description="@string/runningRadio"
         android:exported="false">
      </service>

   </application>

</manifest>
```

The static class structure of our app is shown in Figure 1 in which essential classes and their relations are presented. Figure 1 reveals that the Service class has other member classes. These include WifiLock, PowerManager and AudioManager, and it implements the OnPreparedListener interface.

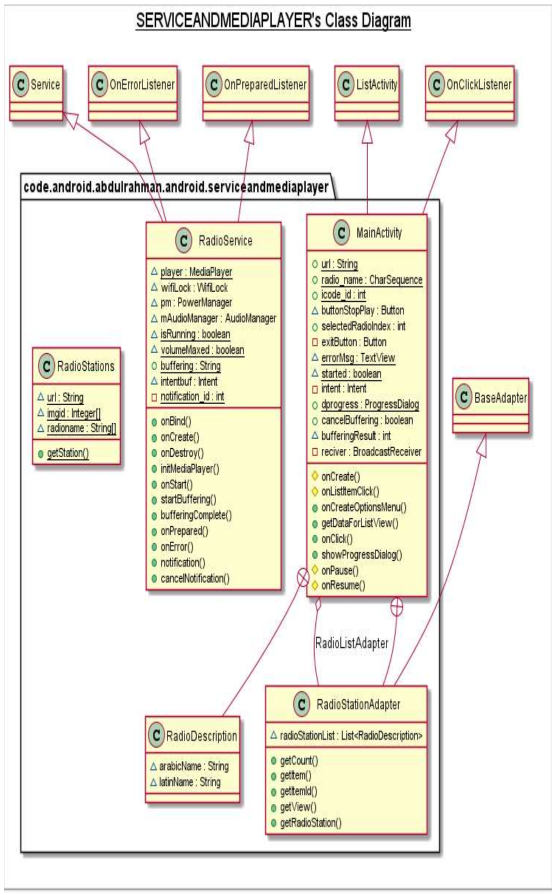

Figure 1, The class structure of the WLURadioStations app

III. Checkout the Source Code of the APP

Download the WLURadioStations.zip[1] app code from Wilfried Laurier University server. Unzip the code in your download folder then use the **File->import->Existing Android code into Android Studio workspace**. The project is ready to run. You can also download .APK (Android Package Kit) to your Android device and run it by tapping on the downloaded file. Figure 2 shows the user interface for the WLU Radio Stations app. The current version of the app present author's favorite radio stations. Developers can easily refactor the app and replace the existing station with the one they prefer.

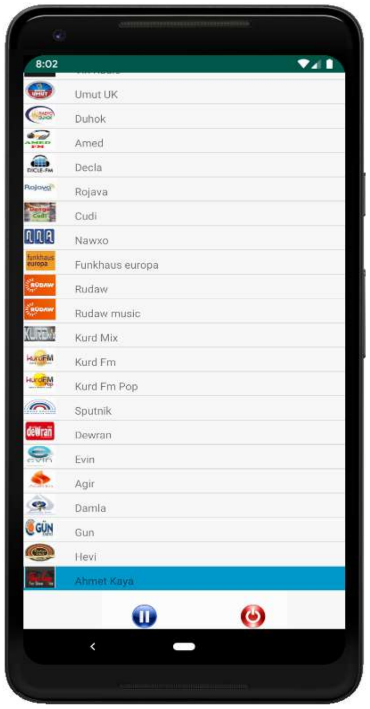

Figure 2, App's user Interface

IV. Conclusion and Future work

In this paper, we have presented a live streaming app for Android devices using URL. The app groups together one's favorite radio stations in one place enabling easy play and switching from one station to another. The app is similar to Spotify but on a smaller scale. We described all the steps and components needed to develop such an app. We also discussed the functionality and the trade-offs using different components and approaches. The app's source code and complete documentation can be used by instructors to teach various Android topics.

V. Future Extension

The current version of the app presents the authors favorite radio stations. Developers can easily refactor the app and replace the existing station with the ones they prefer. In the feature version of this app, we enable not only developers but end users to be able to customize the stations they listen too using the app's user interface.

References

[1] Service Overview, Reterived Oct 27, 2019 from https://developer.android.com/guide/components/services

[2] Broadcasts overview, Reterived Oct 27, 2019 from https://developer.android.com/guide/components/broadcasts

[3] MediaPlayer Overview, Reterived Oct 27, 2019 from https://developer.android.com/guide/topics/media/mediaplayer

[1] http://www.wlu.ca/amawloodyunis/wluRadioStations.zip

[4] Power Management, Reterived Oct 27, 2019 from https://developer.android.com/about/versions/pie/power

[5] WifiManager, Reterived Oct 27, 2019 from https://developer.android.com/reference/kotlin/android/net/wifi/WifiManager

[6] Notification Overview, Reterived Oct 27, 2019 from https://developer.android.com/guide/topics/ui/notifiers/notifications

[7] Audio Focus, Reterived Oct 27, 2019 from https://developer.android.com/guide/topics/media-apps/audio-focus