Development of an Android Guitar Tuner Application

Zoe Wall
40182161@live.napier.ac.uk
Edinburgh Napier University
Mobile Applications Development (SET08114)

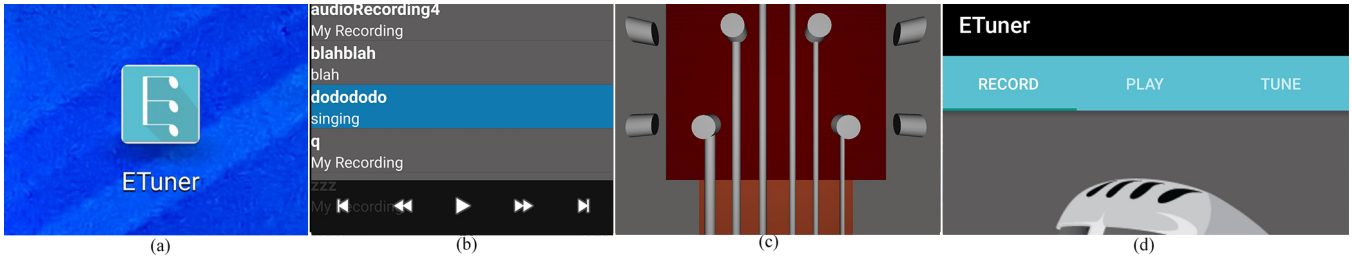

Figure 1: *Project screenshots. (a) Launch icon of Application, (b) Audio controller and clip selection, (c) 3D rendered guitar and (d) navigation tab bar.*

Abstract

This project aims to implement a mobile application prototype using the Android API. The application was intended to be a fully functional guitar tuner and audio recorder, also making use of the OpenGL API for embedded systems, to display 3D graphics. This report details the techniques used, and design decisions made within the development of the application.

Keywords: Android, fragments, accelerometer, file I/O, OpenGL

1 Introduction

The requirements of this project were to develop a mobile application for the android platform. The initial ideas were that of a chromatic guitar tuner app. On the market at the moment there are hundreds of guitar tuner applications however the successful ones all seem quite similar. Reading an article about the best guitar tuner applications, they all seem to have the same design feel towards them. [Hindy 2015] Most are chromatic, meaning it 'recognizes each of the twelve chromatic (semitone) steps of the equal-tempered scale (C, C#, D, D#, E, F, F#, G, G# A, A#, B)' which is useful as it can tune any instrument. [Roland]

Most of them are very complex, with many different features such as chord libraries and metronomes. Hardly any application within the top ten has a reference pitch section, and if they do, it's is almost always an extra feature. Making a decision to use a reference tuner as a main feature of the project would make the application unique compared to the hundreds of tuner applications already on the App Store.

Another unique feature that was planned to implement would be that of a voice recorder and play back function. Taking inspiration from these successful applications, for this project it was decided to incorporate an extra feature, that would use more elements of the API to fit with the project's specifications. A voice recorder is a tool used by many singer-songwriters to quickly sing or strum down a tune or melody for later reference. However, none of the researched guitar applications included this as a feature. Most devices come with their own voice recorder applications built-in however from experience, the recordings are usually difficult to edit, find or delete. Which inspired the project to include a very simple user interface where the user can in one tap discard the recording immediately, or if they wish give it a more meaningful name than "Voice0001" for example.

1.1 Related Work

DaTuner One app that is very simple and particularly useful is the DaTuner. The free version of the application is very quick and simple to use, however if a user was looking for more, they will also find some advanced features within the pro version (see figure 2). DaTuner is a good application as it is very quick to use, it launches straight into a tuner which you can use automatically without any set-up or navigation.

Figure 2: ***Screenshot of a simple chromatic tuner - DaTuner. Note how clear the complex information is displayed through a simple two-colour display.*** *- [Applaud Apps 2016]*

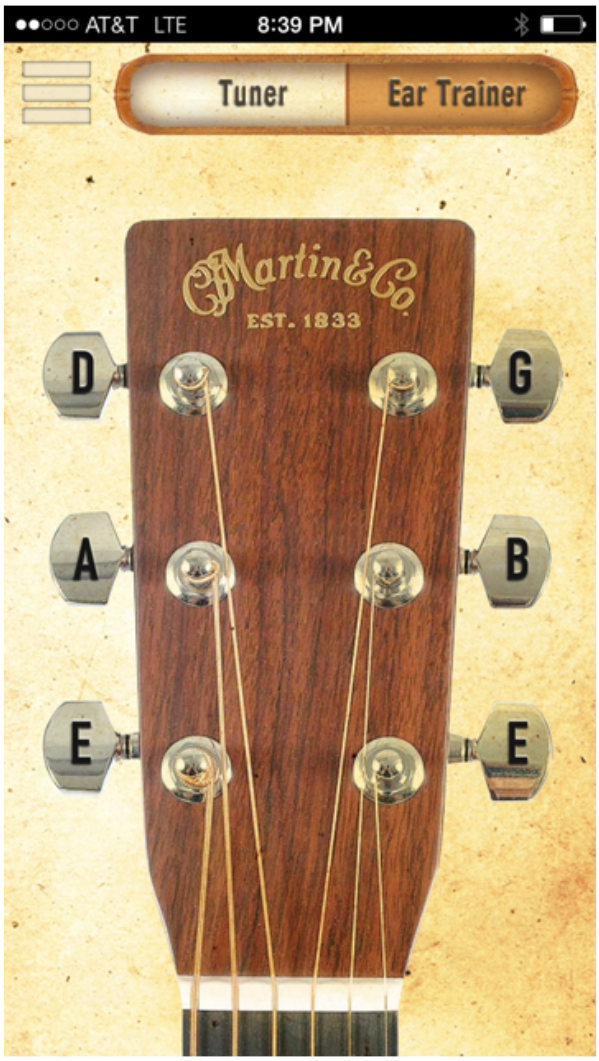

Figure 3: ***Screenshot of the Martin Guitar Tuner. Note the inclusion of a reference note tuner.*** *- [C.F Martin & Co 2015]*

Martin Guitar Tuner Another application that is inspirational is the C.F. Martin & Co. Guitar Tuner App (see Figure 3). Overall it is not a very good application, it's uses a "hamburger" style navigation menu and it's tuner pages are low-quality images with very little user interaction. Hamburger menus, even though widely used, are not a good way of displaying menu items. They are useful in the sense that complex navigation can be hidden behind a button which saves screen space, but they are 'less efficient' due to the need of tapping the button before you can reach any options. They also make the ability to glance at an available interaction more difficult, which can lead to users forgetting that some features exist. [Constine 2014] One good thing about the application is that it includes a very basic approach to an ear tuner. It is simple, you tap the tuning peg and the corresponding note plays, which is why it is inspiring. For this project, the reference tuner will be implemented using a 3D rendered guitar with animation to show the user which string is being played.

2 Software Design

2.1 Design Decisions

A decision was made to favour a more simple app than some of the more feature-heavy application. This was due to the similarities of most applications already on the market, one of the project aims was to produce something different however still useful. The aim was to develop an application to be used as an everyday tool and not as a game or distraction.

When researching implementation of chromatic tuning software, it seemed a little beyond the scope of the assignment. The biggest issue with the implementation was the pitch detection. On first glance, it can seem as easy as putting data into a FFT, Fast-Fourier Transform. A FFT, is an algorithm that can compute compromising frequencies of a given signal. [Roche 2012] However, it also requires some pre-processing of the sampled data. For this project,

it was inappropriate as the objectives of the tasks were to develop an application that used advanced features of the android API, so it was decided that implementing a pitch detection feature wouldn't have met these objectives. Instead, it was decided to include a reference note tuner implemented with 3D graphics.

As the focus of the project was using the android API, it was decided that the app should include a recording and playback feature. The implementation of a simple voice recorder.

EXPLAIN WHY THIS SHOWS OFF THE API

2.2 Experience Story

"A guitarist is walking in the park, an idea occurs to him of a melody. He opens the application, presses record and starts singing. He wants to remember this idea for when he arrives home. He stops the recording and can quickly decide if he wants to discard it or rename the file and save it. He types the name "In the Park" taps save, he swipes right and can playback his recording instantly by tapping on it's name within the view."

In creating an experience story for a particular scenario.

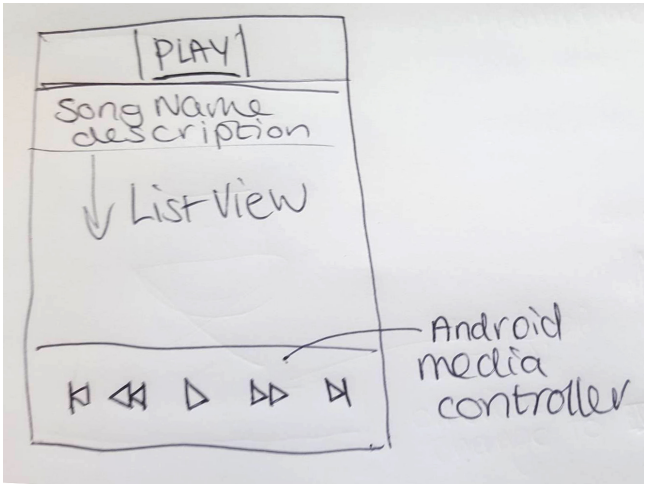

Figure 4: ***A quick sketch of a user interface design for the play tab.***

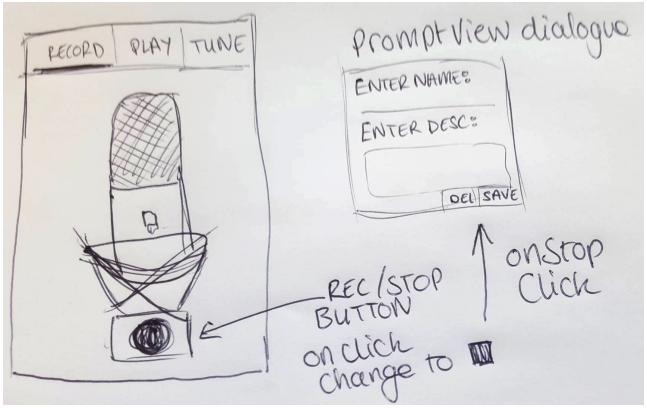

Figure 5: ***A quick sketch of a user interface design for the record tab.***

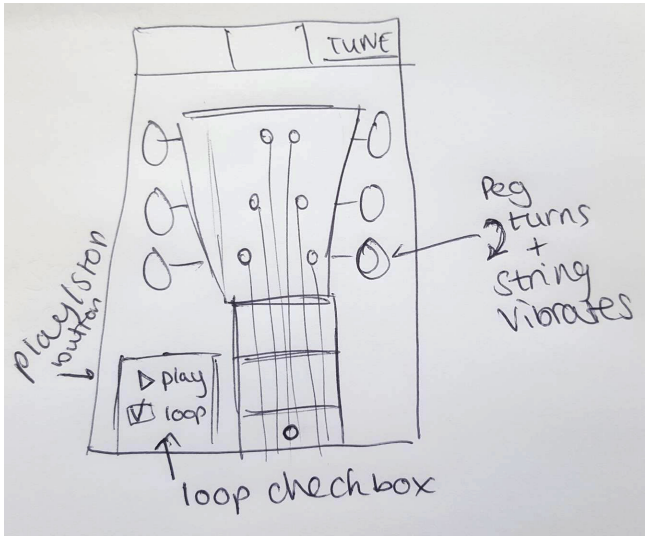

Figure 6: ***A quick sketch of a user interface design for the tune tab.***

SWIPE VIEWS!! http://developer.android.com/design/patterns/swipe-views.html

2.3 Activity Diagrams

After completing rough sketches of the user interface design. The application was split into three tabs.

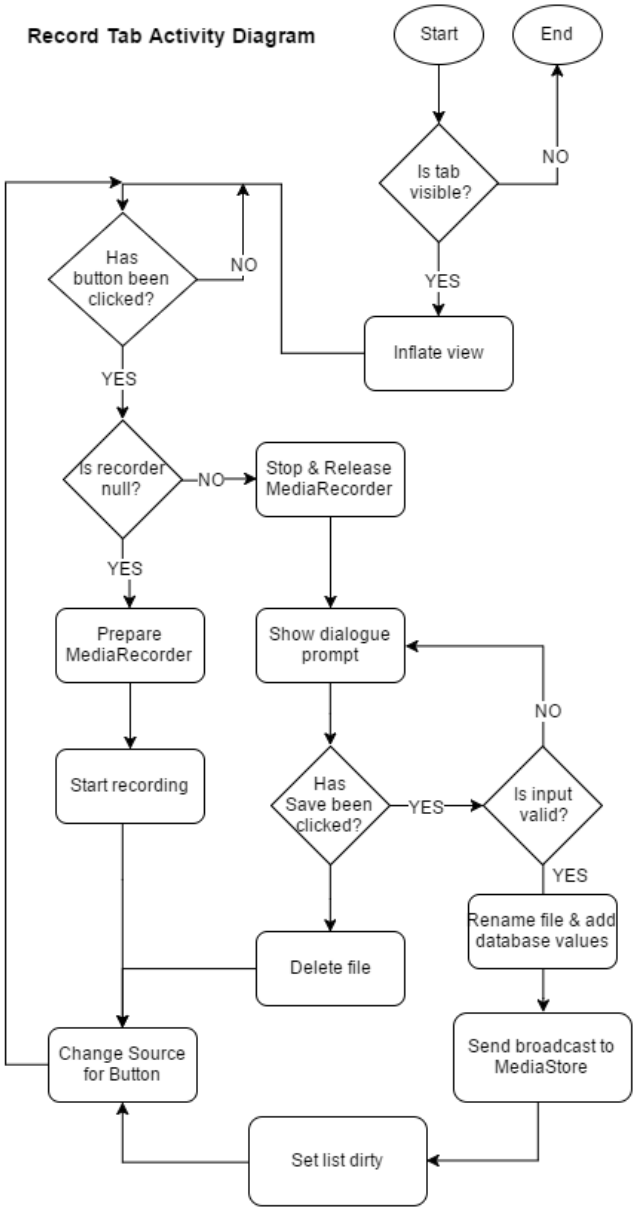

Figure 7: *The activity diagram for the Record Tab*

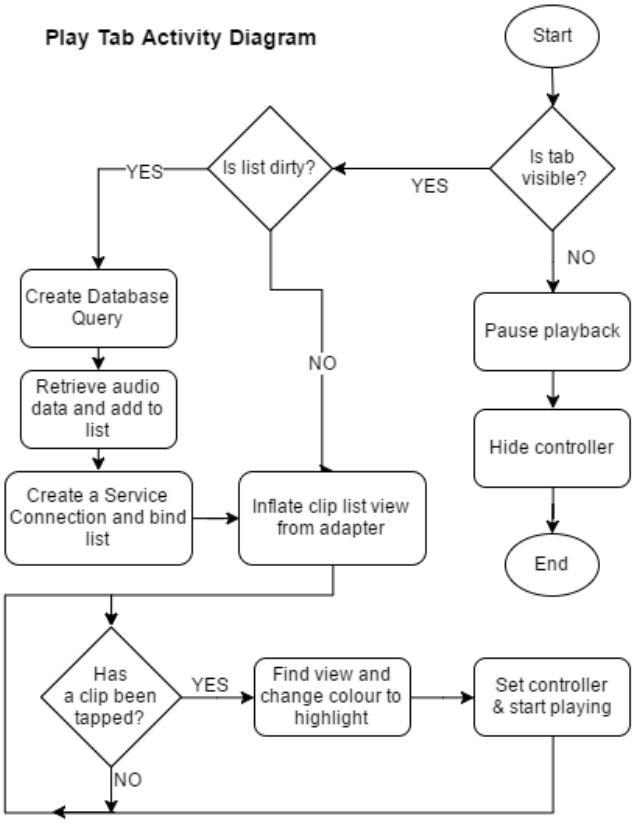

Figure 8: *The activity diagram for the Play Tab*

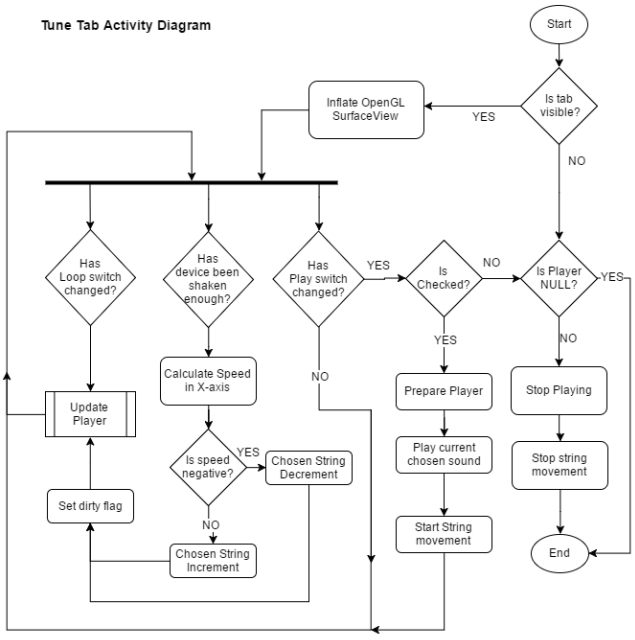

Figure 9: *The activity diagram for the Tune Tab*

3 Implementation

3.1 Navigation

For the main navigation of the app, one main activity was used alongside fragments. In doing so, a tab layout could be used making it easy for the user to see what options are available. (See Figure 10) To implement this, a ViewPager and custom PagerAdapter was used. A ViewPager does not pause fragments that are next to the visible tab to allow for smoother scrolling.

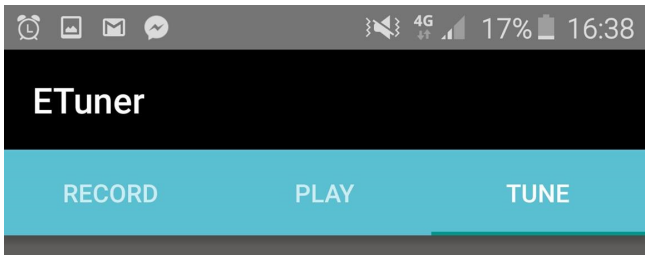

Figure 10: ***An image of the final implementation of the navigation bar. It consists of the title of the application and three tabs which are clickable or swipeable.***

For the implementation, the PagerAdapter created extended the FragmentStatePagerAdapter class of the API. It was chosen due to the fact it "It destroys fragments as the user navigates to other pages, minimizing memory usage." [Android a] This meant that if the record tab was visible, the tune tab would call it's onPause life-cycle method, stopping the continuous call for the renderer's update function.

3.2 Recording

The recording fragment of the app uses a MediaRecorder to save an audio recording in AAC format. Most built in voice recorder applications on mobile phones don't allow for customisation of the media file's meta data. From the experience story, an aim of the project was to implement a function to quickly change the file name, add a description and be able to playback showing these values. Without using an external library, editing meta data for recorded files was difficult. It could have been done using ID3Tags however the android library does not support encoding for MP3. [Android b] To work around this, android's MediaStore was used.

The MediaStore is a content provider that contains meta data for all available media on both external and internal storage of the device, similar to an SQL database. When adding or removing media from a device, the meta data stored remains until Android scans the system for new media. Typically this happens when the device is booted, however it can be called explicitly by broadcasting an Intent. [Denardi 2013] Therefore when an audio file was recorded and saved, a function was used to insert this into the database, see Listing 1.

Listing 1: *Inserting File into MediaStore Database*

```
1  public void insertFileIntoDatabase(String fileName, String fileDesc)
2  {
3      File mySound = new File(recDir, fileName + fileExt);
4
5      // rename file
6      boolean rename = tempFile.renameTo(mySound);
7
8      // add recording to media database
9      ContentValues values = new ContentValues(4);
10     long current = System.currentTimeMillis();
11     values.put(MediaStore.Audio.Media.TITLE, fileName);
12     values.put(MediaStore.Audio.Media.ARTIST, fileDesc);
13     values.put(MediaStore.Audio.Media.ALBUM, albumName);
14     values.put(MediaStore.Audio.Media.DATE_ADDED, (int) (
           current / 1000));
15     values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/AAC
           ");
16     values.put(MediaStore.Audio.Media.DATA, mySound.
           getAbsolutePath());
17
18     ContentResolver contentResolver = TabSwitcher.getmContext().
           getContentResolver();
19
20     Uri base = MediaStore.Audio.Media.
           EXTERNAL_CONTENT_URI;
21     Uri newUri = contentResolver.insert(base, values);
22
23     TabSwitcher.getmContext().sendBroadcast(new Intent(Intent.
           ACTION_MEDIA_SCANNER_SCAN_FILE, newUri));
24
25     Toast.makeText(TabSwitcher.getmContext(), "Added File " + 
           fileName, Toast.LENGTH_LONG).show();
26
27     // sets dirty flag for updating list from database
28     TabSwitcher.setListDirty(true);
29 }
```

To save resources, a recording is initially stored with a temporary file name so if the user decides not to keep the recording, they can easily just click delete and try again. Once a user clicks save, the file is renamed.

To insert meta data into the database, see the method used on line 21 of Listing 1. This inserts a row into a table at the given URL, with the relevant ContentValues. For the description of the file, the Artist's column was used, as there was no option, without creating a separate database to customise the columns. In using the MediaStore as opposed to a 3rd party library to create meta data for the recordings, the recordings can also be accessed by any other app on the system.

Finally, the intent was broadcast to the MediaScanner and a static flag was set, so that the play tab can update the list of files and show the new recording when navigated to.

In keeping with the proposed graphic design, the layout of the tab was filled with a vector graphic of a microphone with a free to use license. [Vecto2000 2011] It also included an image button with a record icon created for the project.

Figure 11: *A screenshot of the final implementation of the record tab. The button at the bottom switches from a record icon to a stop icon when pressed to indicate the app is recording.*

Once the recording is stopped, a custom alert dialog was used to allow the user to delete the file, or save their own details. To stop the dialog closing if an invalid file name was entered, the click event handlers were created and then overridden, see Figure 12

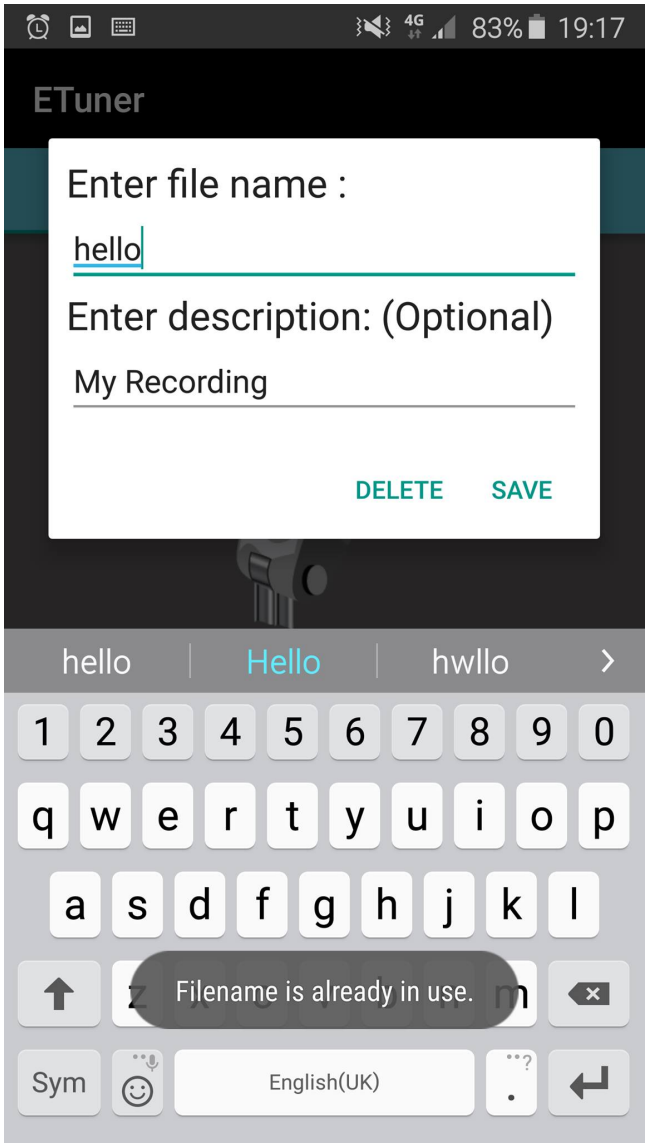

Figure 12: *A screenshot of the alert dialog that is used when a user stops a recording. Note the toast that appears if the filename is already in use.*

3.3 Playback

For the playback implementation, a class was created to store audio clip objects. Each contained an ID, a title and an artist. This meant that when the fragment is created, the device is queried for all audio content stored. Due to the fact, the album was set as constant for every recorded file meant that only files with the exact album name are initialised as Audio Clip objects, see Listing 2.

Listing 2: *Conditional Statement for Creating Audio Clips*

```
1 if (soundCursor.getString(albumColumn).equals(Recorder.↩
      albumName))
2 {
3    clipList.add(new AudioClip(thisId, thisTitle, thisArtist));
4 }
```

By using an ArrayList container to store these AudioClip objects,

the List View within the Fragment's layout can be populated by using a custom Adapter class. This means that for every object in the list, the adapter can easily iterate through the list retrieving each of the instance variables and setting the appropriate text views to display the information. Rather than having to query the database each time, when the play tab becomes visible, it checks whether the flag set in the record tab is true, if so, it will clear the list and re-populate it.

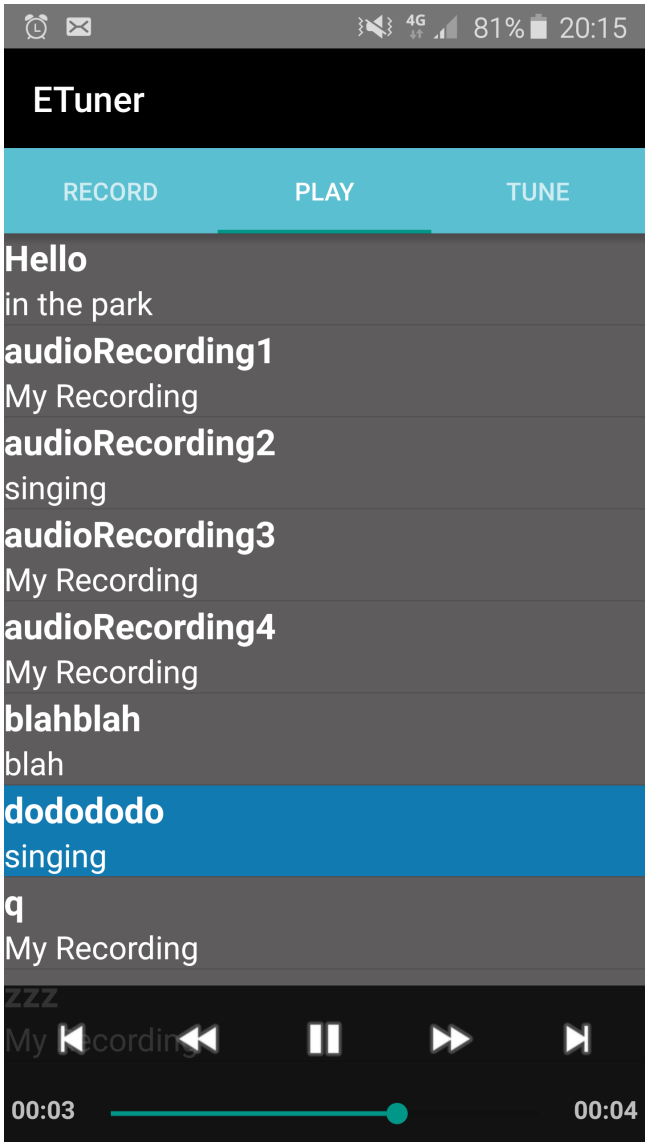

Figure 13: ***An image of the final implementation of the Play tab. It uses Android's own MediaController to control the play functions, and highlights the current chosen track.***

For the playback of these audio clips, a sound Service was implemented meaning that if the user wanted, they could continue listening to their audio clips if they have navigated away from the application, or if they lock their device.

3.4 Tuning

For implementation of the tuning tab, a Frame Layout containing a SurfaceView was used. This SurfaceView was inflated using a custom OpenGL context which rendered to the screen. [Brothaler 2011]

4 Evaluation

The original concept of the app was very similar to the

4.1 Further Work

In improving the application, the next steps

5 Summary

References

ANDROID. Creating swipe views with tabs. *Accessed: March 2016*. www.developer.android.com.

ANDROID. Supported media formats. *Accessed: March 2016*. www.developer.android.com.

APPLAUD APPS. 2016. Tuner - datuner (lite!). www.play.google.com/store/apps.

BROTHALER, K. 2011. Android lesson one: Getting started. *Accessed: Feb 2016*. www.learnopengles.com.

C.F MARTIN & CO. 2015. Martin guitar tuner. www.play.google.com/store/apps.

CONSTINE, J. 2014. Kill the hamburger button. *Accessed: March 2016*. www.techcrunch.com.

DENARDI, S. 2013. Querying and removing media from android mediastore. *Accessed: March 2016*. www.sandersdenardi.com.

HINDY, J. 2015. 10 best guitar tuner apps for android. *Accessed: March 2016*. www.androidauthority.com.

ROCHE, B. 2012. Frequency detection using the fft (aka pitch tracking) with source code. *Accessed: Feb 2016*. www.blog.bjornroche.com.

ROLAND. Roland tu-2, chromatic tuner. *Accessed: March 2016*. www.roland.com.

VECTO2000. 2011. Free vector - vector microphone. www.vectorspedia.com.