Development of an Android Guitar Tuner Application

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

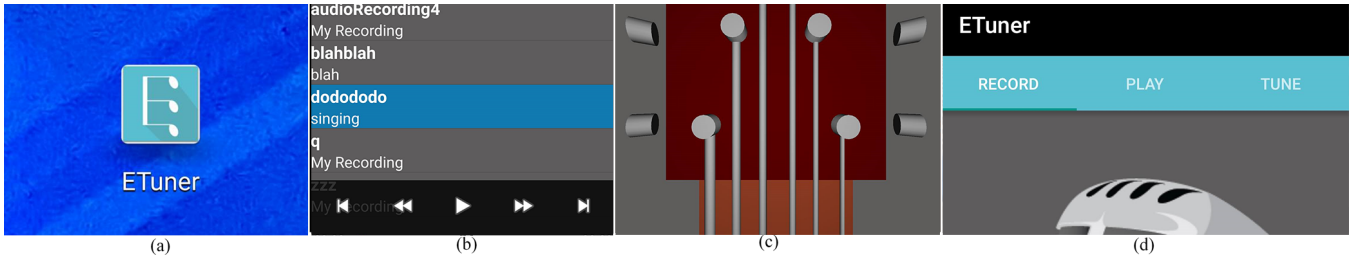

Figure 1: *Project screenshots. (a) Launch icon of Application, (b) Audio controller and clip selection, (c) 3D rendered guitar and (d) navigation tab bar.*

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 also 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 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 be implemented was 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 application. The free version of this 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 due to it's minimal layout and automatic functionality without the need for 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]*

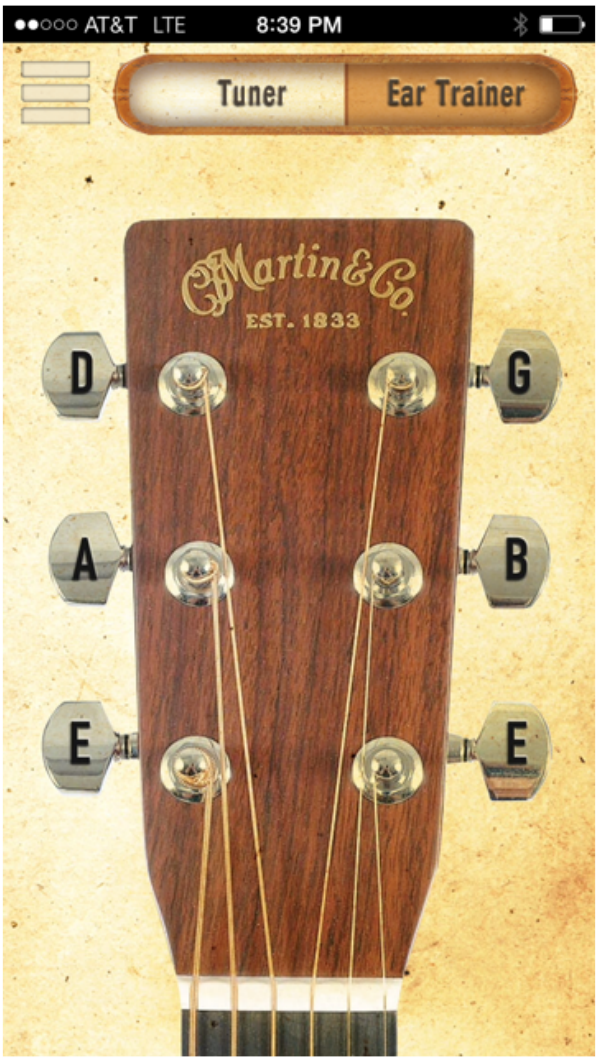

Figure 3: ***Screenshot of the Martin Guitar Tuner. Note the inclusion of a reference note tuner.*** *- [C.F Martin & Co 2015]*

Martin Guitar Tuner Another application that is inspirational is the C.F. Martin & Co. Guitar Tuner Application (see Figure 3). Overall it is not a very good application, it 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.

Samsung Galaxy S6 Voice Recorder Taking inspiration from the pre-installed Voice Recorder application from the Samsung Galaxy S6 device, an aim for the voice recorder feature is to have instant avalible playback functionality. As seen in Figure 4 the application is very quick to use however there is no ability to discard or edit a clip immediately and the voice recordings are not visible to any other application.

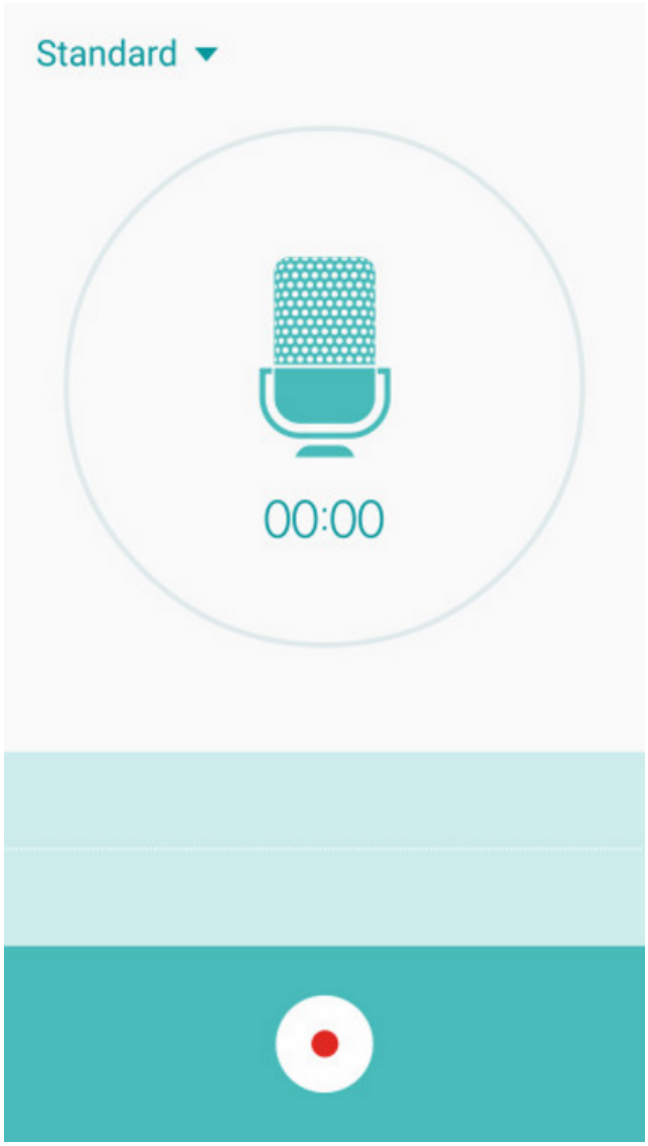

Figure 4: ***Screenshot of the Samsung Galaxy S6 Voice Recorder. Note the simple UI design.*** *- [Samsung Galaxy S6 2015]*

2 Software Design

2.1 Design Decisions

A decision was made to favour a more simple app than some of the more feature-heavy applications researched. 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 also include a recording and playback feature. However, the project aimed to improve upon the Samsung Voice Recorder, by inclusion of a simple way of choosing to delete or rename a recording instantly without the need of re-locating it. Another good feature intended to be included was that the voice recording clips can be found by other applications on the device.

2.2 Experience Story

Envisionment in the form of an experience story for a particular scenario is important for understanding the wants or needs of a particular user for the application. In writing a scenario for a potential user of the application, a persona is created that is different from oneself. This is extremely important as it helps to prioritize the audience and their requirements, it is also important for the developer to realise that they are not designing for themselves. Follows is an experience story created for this project.

"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 from a list of recordings."

From this particular scenario, it is clear that a simple interface for recording quickly should be key to the design of the recording feature of the project. At this stage of the design there is very little specification, however it is part of the idea-generation process by noting down user needs that the application should fulfil.

2.3 User Interface Sketches

After completing a conceptual scenario, design constraints were added in the form of quick user interface sketches. Sketches are useful for prototyping because they are disposable, yet informative. They help to envision a physical design by specifying constraints for example a menu bar or a particular colour scheme. Figures 6, 5 and 7 are the final outcome of the user interface design.

At this stage it was decided that the application be split into three defined parts, the tuner, the recorder, and the voice player. In researching different types of navigation a horizontal swipe view seemed the most appropriate for this project as it allows the user "to efficiently move from item to item using a simple gesture", which in turn makes access to information or features faster and more enjoyable. [Android 2011]

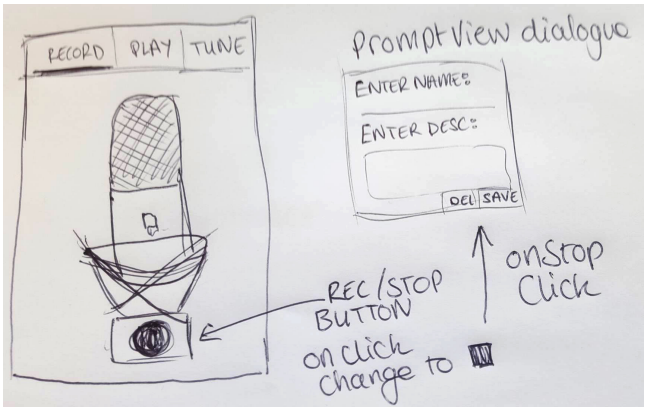

Figure 5: ***A quick sketch of a user interface design for the record tab, including the dialog prompt for user input.***

For the Record tab, it was decided that the easiest way to allow instant editing capabilities for the user was to include a dialog prompt appearing when the recording was stopped. This way, the user is given the option of renaming their file, and the input validation checks can be within this prompt.

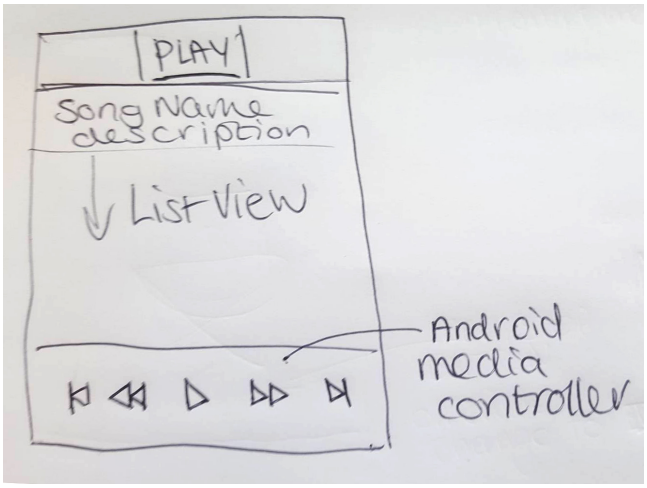

Figure 6: ***A quick sketch of a user interface design for the play tab, including the list of recorded songs and a media controller.***

For the Play tab, the design is quite simple, a list view of every recording made by the application with a floating controller at the bottom of the screen. The list can show any data that goes along with the recording.

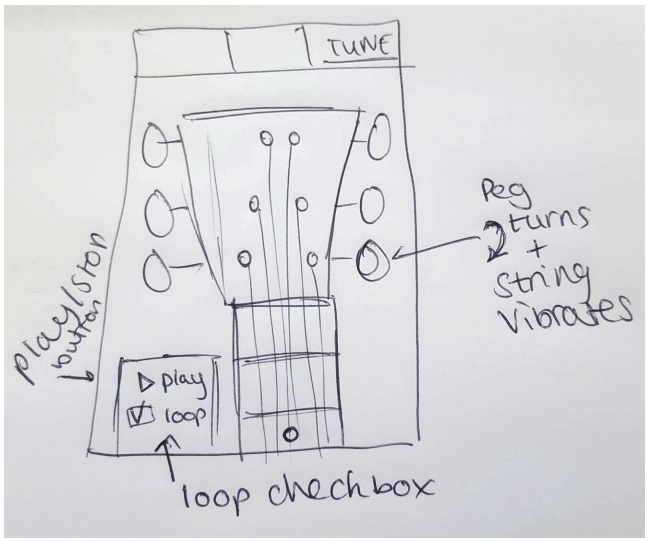

Figure 7: ***A quick sketch of a user interface design for the tune tab, including the options for playing and looping.***

For the Tune tab, the design is inspired by the Martin Guitar tuner, however the idea is to render the headstock in 3D to make it look more realistic and have the tuning peg and string animated to show the user in a unique way which string is vibrating.

2.4 Activity Diagrams

After completing rough sketches of the user interface design. The application was split into three tabs and activity diagrams were drawn to model the software. Figures 8, 9 and 10 show the high-level activity diagrams created for the Recording tab, the Play tab and the Tune tab respectively.

These were key to the design of the implementation as they showed a structured flow of control for the program. Lower-level models could have been drawn with functional decomposition, however these diagrams were sufficient to begin implementation due to the main requirements and logic of the application being included.

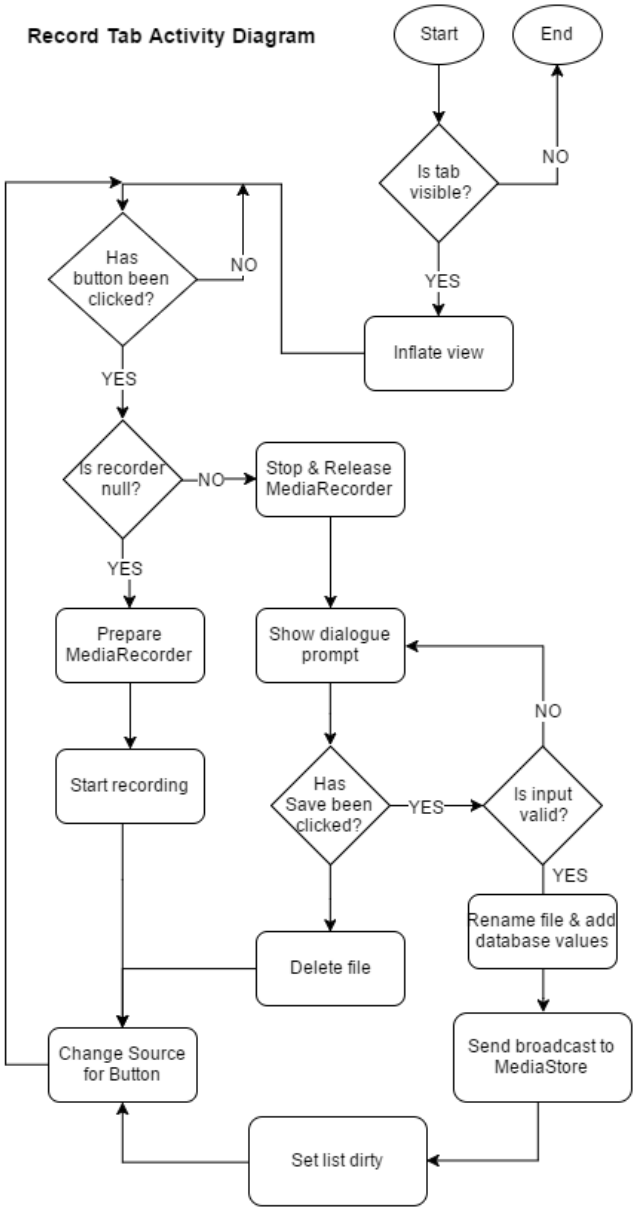

Figure 8: *The activity diagram for the Record Tab*

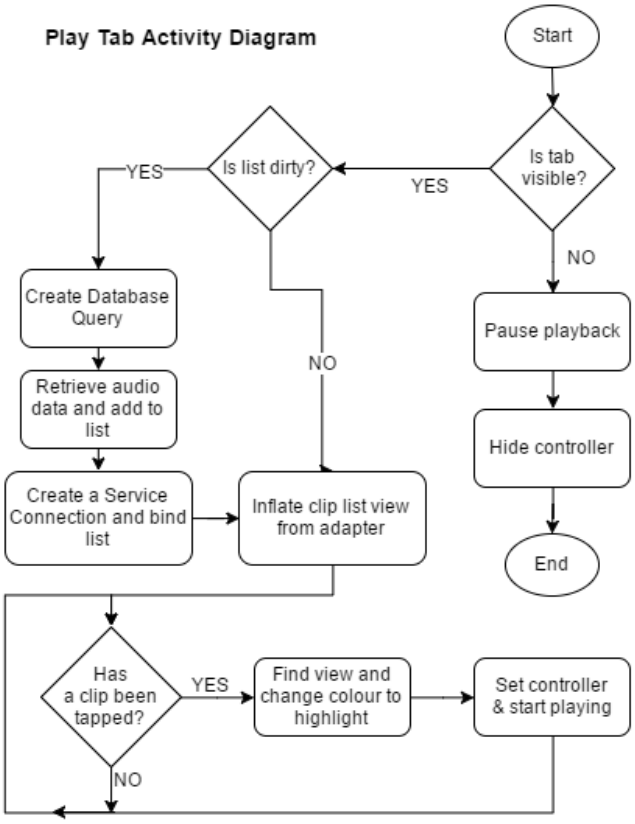

Figure 9: *The activity diagram for the Play Tab*

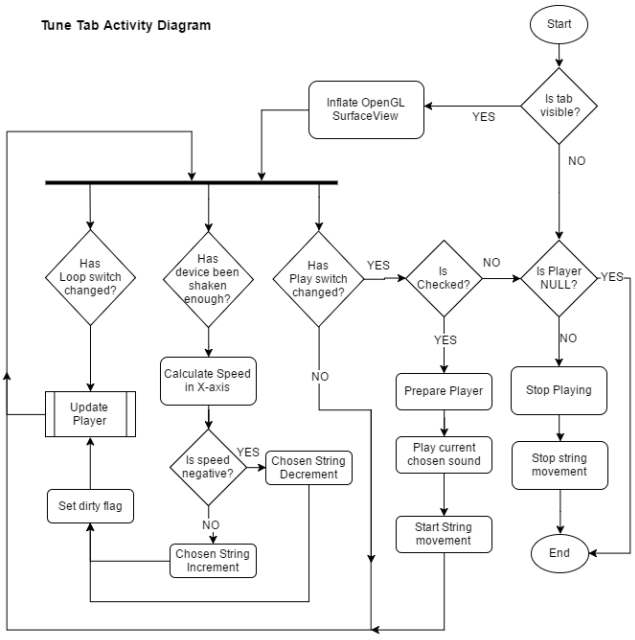

Figure 10: *The activity diagram for the Tune Tab*

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, shown in Figure 11. To implement this, a ViewPager and custom PagerAdapter was used.

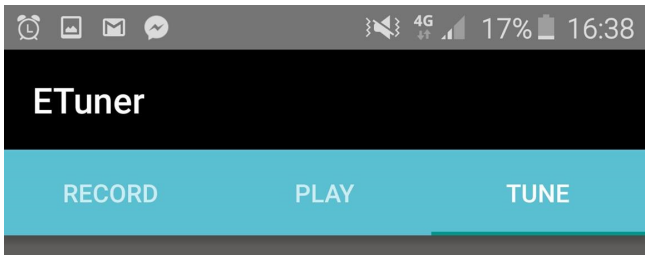

Figure 11: ***An image of the final implementation of the navigation bar. It consists of the title of the application and three tabs which are clickable or swipeable.***

The PagerAdapter created extended the FragmentStatePagerAdapter class of the API. It was chosen due to the fact, "it destroys fragments as the user navigates to other pages, minimizing memory usage." [Android c] This meant that if the record tab was visible, the tune tab would call it's onPause lifecycle method, stopping the continuous call for the renderer's update function. Also the use of the ViewPager does not pause fragments that are next to the visible tab to allow for smoother scrolling.

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 accordingly, see Figure 13 for the implementation of this.

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, it means that they can also be accessed by any other application 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 12: ***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 to allow for validation checks on the user's input. A descriptive toast is displayed if there is a problem, see Figure 13.

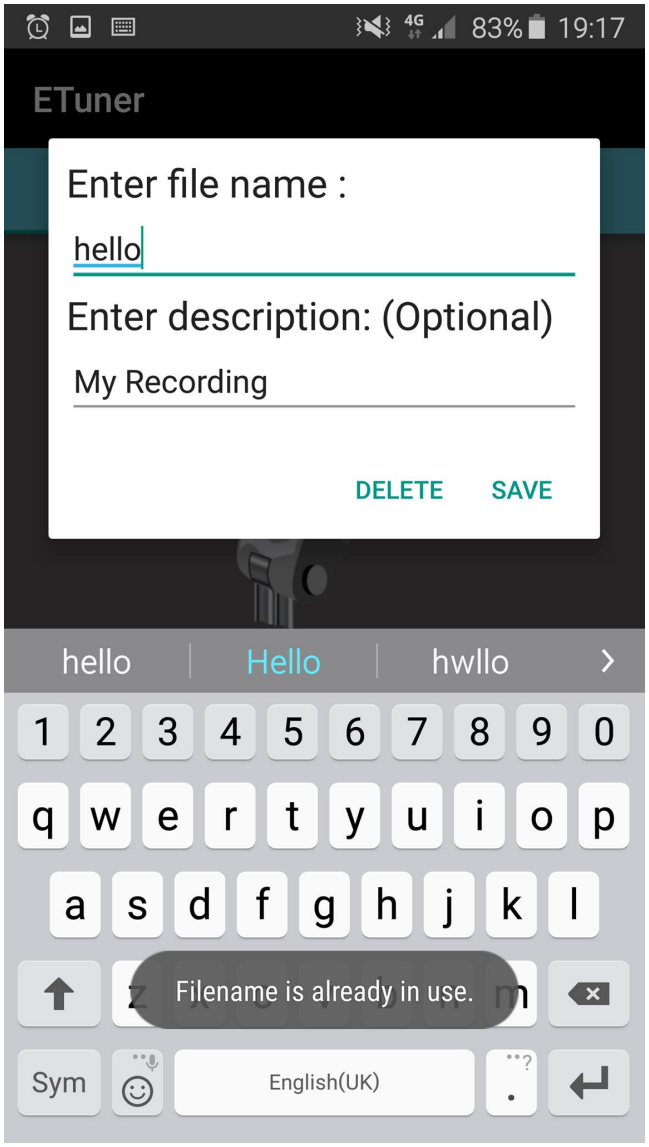

Figure 13: ***A screenshot of the alert dialog that is used when a user stops a recording. Note the toast that appears if the filename is already in use.***

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 were 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. Then, 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.

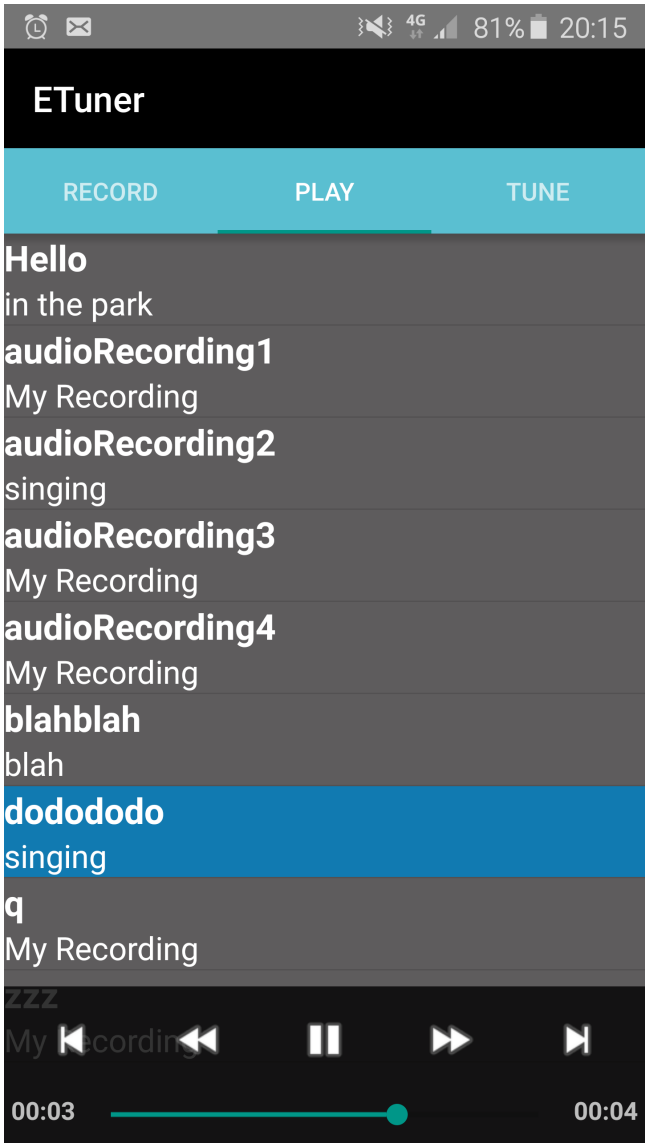

Figure 14: ***An image of the final implementation of the Play tab. It uses Android's own MediaController to control the play functions, and highlights the current chosen track.***

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. Playback was controlled through an instance of a MediaPlayer that was bound to the Service, and a default MediaController for the user interface. As the project focus was more on the guitar tuner and recording aspect, it seemed inappropriate to redesign a media controller just for this purpose, the implementation of this can be seen in Figure 14.

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]

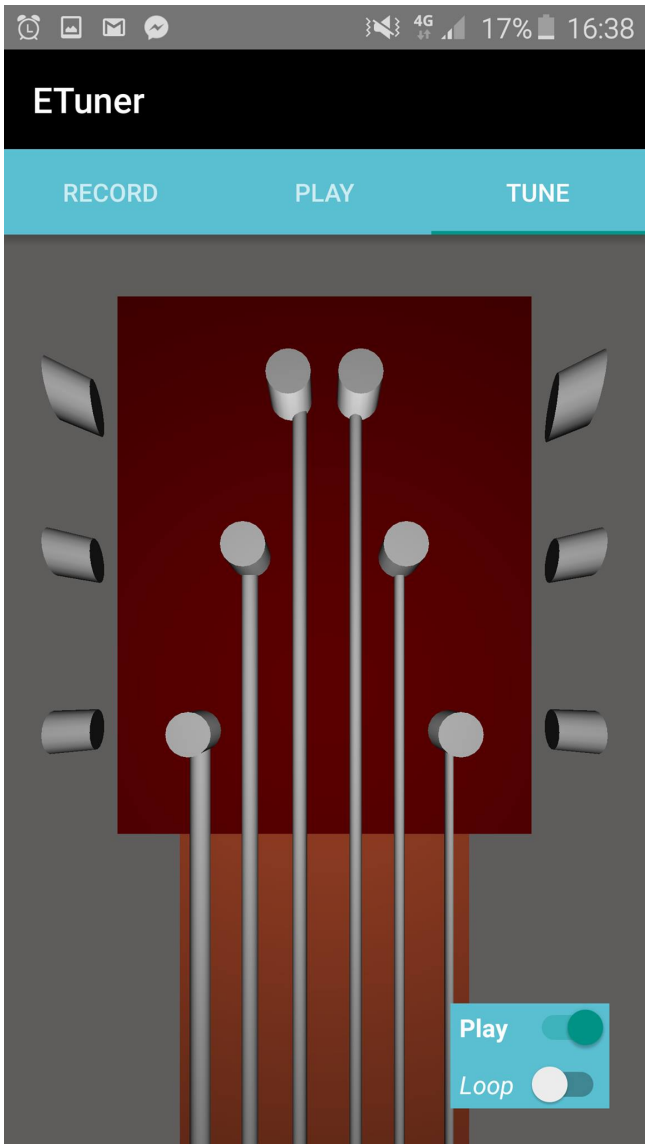

Figure 15: ***An image of the final implementation of the Tune tab. It includes a 3D rendered guitar headstock and two control switches: play and loop.***

When rendering an object to the screen, the object's geometry first needs to be initialised. For simplicity, two shapes were generated from triangle shapes, a cube and a cylinder, both using maths to calculate the position of each vertex of the geometry. These positions were then translated into the screen position using the MVP, Model-View-Projection Matrix - a concatenation of three matrices. The Model matrix, being what translates the original geometry's vertices into world-space, a position in the 3D world. This includes scaling and rotating of the object as both of the two shapes are created at the same size around the origin. The View Matrix, a translation of these positions into the camera's eye space, and the Projec-

tion matrix, which finally then projects the view onto a screen - in this case the dimensions of the device. Each vertex is sent through a program on the devices GPU which consists of two "shaders", first a vertex shader, and then a fragment shader. These shaders are written and compiled in GLSL, the OpenGL Shading Language, and their function is to colour or shade the objects passed into them. The first is the Vertex shader, which transforms every vertex rendered into it's correct position according to it's MVP-Matrix, and the second is the Fragment shader which in this case performs lighting calculations and colours each pixel according to the lighting model.

To display to the user which string sound was currently playing, the project needed a way for the user to choose and change strings. For the animation, this was a case of changing the current string's translation matrix on every render frame. However, due to the complexity of mapping the mouse coordinates to a position within the 3D scene, an accelerometer sensor listener was used to change the chosen string. Every time the user shakes the device the speed is calculated along the x-axis and if it is positive, the direction is right, and therefore the chosen string index is incremented, if the speed is negative, the chosen string is decremented.

This tab also contains two switches for user interaction. The first switch controls the main play function. If this is turned on, a sound from the chosen string will play and every time the device is shaken to change the string the corresponding note will also play. If the looping switch is on with the play switch, the sound will play multiple times until either switch has been turned off.

Listing 3: *Method to stop playing gracefully when looping or shake has occured.*

```
1  private void updatePlay()
2  {
3      // if player is playing, stop and start with new looping set
4      if (myPlayer != null)
5      {
6          while (myPlayer.isPlaying())
7          {
8              // stop playback once the loop has completed (current ↩
           position of track is at zero)
9              if (myPlayer.getCurrentPosition() == 0)
10             {
11                 stopPlay();
12                 break;
13             }
14         }
15
16         // if stopped because shake has occured, start playing with ↩
           new sound
17         if (shakeDirty)
18         {
19             playSound(TabSwitcher.getmContext(), ↩
           currentSoundIndex);
20         }
21
22     }
23
24     // if it IS null and been checked start playing again
25     if (playChecked && loopingChecked || playChecked && ↩
           shakeDirty)
26     {
27         playSound(TabSwitcher.getmContext(), currentSoundIndex);
28     }
29
30 }
```

To allow for a smooth transition between notes, if the sound source changes by shaking, or if the loop function is toggled an update method is called, see Listing 3. Due to the fact the playback may or may not be looping, the Android MediaPlayer's OnCompletionListener is never triggered as the completion event never occurs if a track is set to loop. Therefore to calculate when the note played had finished, while the instance of the MediaPlayer was playing, check the current position of the track, if this was zero, then the sound had completed and was about to repeat meaning that the audio could be stopped without clipping. Once stopped, the method to play a sound again can be called if necessary.

4 Evaluation

4.1 Comparisons

The original concept as outlined in my introduction was to develop a guitar tuner application that was unique compared to the ones already available on the market. This was to include a reference tuner and a voice recorder function. The final application developed implements both these features with full functionality.

Similarly to the DaTuner application, a simple colour scheme was used, and remains consistent throughout. However, processing sound input through a chromatic tuner was too advanced for the project and was not implemented. The developed application is also very quick and simple to use, much like this app.

The application developed encompasses the reference tuner inspired by the Martin Guitar Tuner, however it improves a lot on the user experience design by incorporating simple lateral navigation using a swipe-able tab view. This by far is easier to use than the "hamburger" style menu of the Martin Guitar Tuner, because it allows the user to see every option available to them instantly, and to navigate between them within one movement. Even though, the ability to tap on a sound and have it play is a good feature of the Martin app that is not implemented within this project, the developed application improves on the feedback of this particular reference tuner as of the added functionality of the looping play-back. When tuning a guitar by ear, looping play back is often essential to ensure the tone is correct. The inclusion of this feature allows the user to continuously play the tone whilst attempting to tune their guitar - without having to repeatedly interact with the device.

Compared to Samsung's Voice Recorder, the record feature of the developed application is very similar in its simple yet effective user interaction. Although, there was improvements made on the user friendliness of the application by making it easer to use without the need to navigate and edit files manually.

4.2 User Feedback

The application was given to a professional guitarist to give some qualitative feedback on the overall functionality and usability of the application. When asked about the tuning section of the application, the results were that he "[loved] the idea of the ear tuner, the fact [he] can loop the sounds is a really useful function, sometimes it's a little awkward to make it go in the right direction but the shaking is fun." This shows that the application definitely needs some improvement in functionality, the inclusion of tapping could be an option to use alongside the sensor. Another feature that he would have liked to see within the application is a "larger library of sounds for different tunings, being able to customise this is important for [him]."

As for the voice recorder "being able to give my recordings a name and a description is such a good feature. [He] spend[s] half [his] time searching through numbered recordings making it hard to show other people [his] ideas. Putting both a recorder an a tuner in one application means [he doesn't] have to waste time switching between two."

From this feedback gained, the next step is to create a new iteration of the application and evaluate further by possibly creating a questionnaire for a larger audience.

4.3 Further Work

In improving the application, the next steps are to build more functionality in the app by including a chromatic tuner, as this was one of the more important features of most guitar applications on the market currently.

One improvement to be made to the application, gained through user feedback, is refinement of the user interaction for the Tune tab. Currently, the only way to change the chosen string is to shake the device, this is a unique feature that shows of the use of the Android API however it is limited. Implementing functionality where a user can tap on one of the strings or tuning pegs, similar to the Martin Guitar Tuner, would be a nice addition to the applications user interface. However, due to the fact the surface view is implemented using OpenGL ES, it means that to make this possible calculations would have to be made on the exact position of the object within the 3D space. This is called picking, which is a form of ray casting. [Gerdelan] A bounding volume would have to be calculated for each clickable object within the scene, and each time the user would tap on the screen calculations must be performed to see if the ray, from the screen coordinates, intersects with the bounding volume of the shape. This was definitely beyond the scope of the project's requirements, as it ventures on more computer graphics than mobile app development, however if implemented would improve the application significantly. As for the current shaking implementation, it is not very accurate and could be improved.

Aesthetics wise, the tune tab could also be improved by loading in a model of a guitar headstock to make it look more realistic. This was originally going to be the case, however a suitable model could not be found, with separate strings. Another way of making the guitar more realistic would be to change the string vibration animation to use vertex displacement within the shader program. Currently the shader program only shades each pixel according to the lighting model. However, a separate shader program could be written to change the actual geometry of the strings and displace it's vertices along a sine wave, modelling an actual guitar string's movement.

5 Summary

Overall the application prototype was a success. There is a definite improvement in functionality from the inspirational applications, and some positive feedback was gathered.

The only resources used that were not created for this project include the vector graphic of the microphone [Vecto2000 2011], and the reference notes [Harri Bionic 2006].

Any online articles and tutorials used for reference for parts of the Android API that were not covered in the module are below:

- Creating an OpenGL surface: "ANDROID LESSON ONE: GETTING STARTED" [Brothaler 2011]
- Using Fragments for tab navigation: "Creating Swipe Views with Tabs" [**?**]
- Using a Service for the Media Player and using a Content Resolver: "Media Playback" [Android a]
- Querying Android's Media Store: "Querying and Removing Media From Android MediaStore" [Denardi 2013]
- Using a MediaController: "MediaController Reference" [Android d]
- media player
- recorder

References

ANDROID. Android develop: Media playback. *Accessed: March 2016*. www.developer.android.com.

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

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

ANDROID. Mediacontroller. *Accessed: March 2016*. www.developer.android.com.

ANDROID. 2011. Android design: Swipe views. *Accessed: Feb 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.

GERDELAN, D. A. Mouse picking with ray casting. *Accessed: March 2016*. www.antongerdelan.net.

HARRI BIONIC. 2006. Acoustic guitar pack. www.freesound.org.

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.

SAMSUNG GALAXY S6. 2015. Voice recorder.

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