COM814: Project 2015/2016

Dissertation

School of Computing & Information Engineering

MSc Professional Software Development

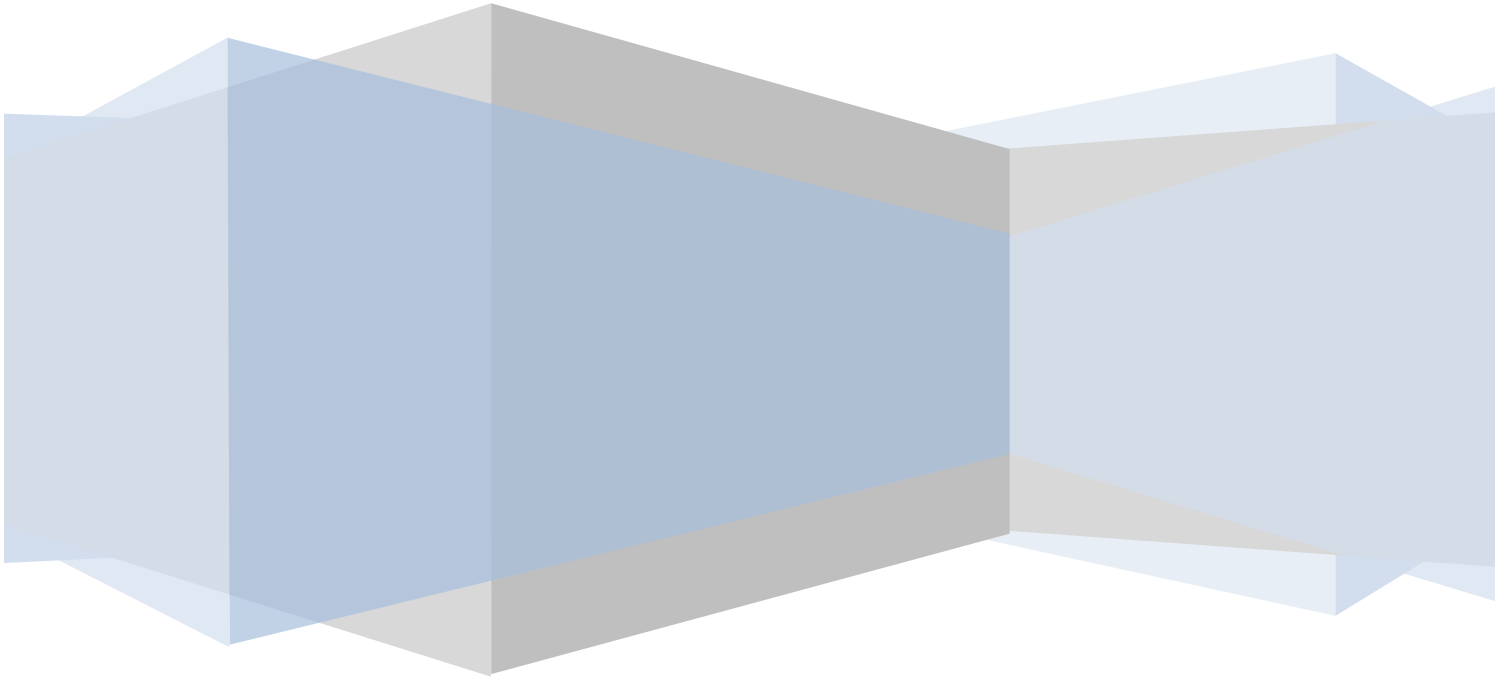
Gareth Leetch B00697573

Mobile Guitar Tutor Application

Supervisor: Dr. Gaye Lightbody

Second Marker: Dr. Shuai Zhang

01/09/2016



Plagiarism Statement

I declare that this is all my own work and that any material I have referred to has been accurately referenced. I have read the University's policy on plagiarism and understand the definition of plagiarism. If it is shown that material has been plagiarised, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded and the reason for that mark will be recorded on my file.

Acknowledgements

I would like to thank Dr. Gaye Lightbody, Dr. Shuai Zhang, Simon Fraser and Dr. Guiseppe Trombino for their help and guidance throughout this project.

I would also like to thank my family for their continuous support and my friends for agreeing to take part in focus groups and providing valuable feedback throughout the project.

Contents

Figures

Tables

Abstract

The guitar is one of the most popularly played and purchased musical instruments in existence, with millions of instruments sold each year to players old and new. Despite this, the number of new players who decide to buy a guitar and to start learning ultimately give up soon after. The “Problem” section of this report aims to investigate the issue of the high quit rate of new guitar players and to highlight a number of key issues which are perceived as influencing factors.

Following this, the “Solution” section of the dissertation looks at how the problem can be addressed by re-engaging a beginner guitar audience through the use of mobile application software. This includes research into existing solutions and how effective each is at providing a solution to the problem. This information was then used to determine which features should be present in the final application which was developed for this project. Based on this information the report goes on to look at the benefits and the requirements of creating such an application.

The next step is the concept of the application design, both in terms of the graphical user interface and the actual architectural design of the system. In this section the basic framework for the user interface and how the on-screen elements should be presented in terms of layout design as well as colour schemes is described. The architectural design outlines how each page in the system is linked from the first page that the user is presented, downwards throughout the pages structural design. As well as this the database design is also described.

Following this, a description of how the development process was implemented over the course of the project is outlined. This is followed ultimately by a description of the testing and evaluation phase at the end of the project.

Introduction

The guitar is a difficult instrument to learn and statistically has an extremely high quit rate among new players. There are a number of factors which have been considered to be a reason for this. One is the assumption that the guitar is easy to learn due to its widespread use in popular culture. For this reason many new users are put off by their perceived lack of fast progress, as well as the physical difficulties of learning a new instrument. Another reason is that, with the guitar and indeed musical instruments in general, due to the learning process, which can usually take the form of one short lesson a week, users become disengaged with learning during their "off time". Due to the proliferation of smartphone technologies in the past decade and the benefits of "m-learning" resources it was decided that the developer would try to address the problem of the high quit rate in new guitar players by providing a guitar tutor application which could be used in collaboration with traditional learning methods in order to make the learning process easier and more engaging.

After identification of the problem the solution aims to identify similar teaching aids which currently exist on the market. The positive and negative aspects of each of these applications is assessed with the aim of informing the design and development of the project application. This is done by creating a list of functional and non-functional requirements which directly influence the applications design.

The next stage of the project deals with the design in terms of aesthetics as well as system and database architecture. The implementation of these designs is then described in the next section.

Following the implementation the application is tested and subjected to evaluation by the developer and focus groups. This allows problems to be addressed and recommendations for future additions to the system to be gathered.

Problem Statement

"To aid beginner guitar players in the initial stages of their learning by providing a mobile application which would provide the user with helpful information and basic tutorials in order to facilitate the learning experience and maintain a level of player engagement."

Project Aims

- The identification of the problem and solution through research into the causes of the problem and existing applications which aim to help beginner musicians learn the guitar.
- Identification of functional and non-functional requirements of the proposed application, as influenced by the analysis stage of the project.
- Produce an effective design of the system's visual and architectural features.
- Implementation of a functional prototype in order to gain feedback from a focus group and the project advisor in order to suggest improvements for the final application design.
- The production of a working application which meets the functional and non-functional requirements as outlined in the project.
- Continuous testing and evaluation of the product in order to avoid system errors and influence features of the application to produce a more effective end product.
- Final testing and evaluation to fix any errors before publication and suggest future improvements to the system.

Analysis

The Problem

The guitar is one of the most popular instruments in the world. According to the digital music distribution website indigoboom.com; in the year 2005, 1.4 billion dollars' worth of guitars were sold in the United States alone, which corresponded to roughly 43 percent of worldwide sales. In that same year the guitar outsold the piano in the US by 0.3 billion. With statistics like this, one might be forgiven for assuming therefore that the guitar is the much more popularly played instrument. This, however, is not the case. In fact, the numbers which are indicated in sales do not accurately represent the number of guitar players at all. According to an estimation by Fender, one of the world's leading guitar manufacturers, 90 percent of people who decide to learn to play guitar give up playing within the first year (Constine, J. 2015). While it's fairly obvious to assume that a number of new players will give up soon after starting, the quoted figure of 90 percent seems shockingly high.

In order to try and address the problem of such a high quit rate for new users it is first necessary to try and understand why so many people decide to give up. As with learning any new skill, the beginning is always the hardest part. Playing the guitar can be especially hard for new users as it can be both physically and mentally demanding. With the guitar being such a major influence within pop and rock music since the mid to late twentieth century, many people make the mistake in thinking that learning the guitar doesn't take as much time or effort to learn as other "more classical" instruments. For this reason they decide to learn guitar with the expectation that it will be easy and that it won't be long before they can play like their favourite guitarist. The reality however, is that like with any instrument, learning the guitar takes time, practice and dedication in order to become a skilled player. This expectation versus the reality can make new players feel like they aren't learning fast enough and that perhaps they don't have what it takes to be musical.

Another prime example of an imbalance between expectation versus reality, and one of the most common complaints which is heard by new players, is that they quit because they didn't realise it would be so painful on their hands. Of course, this is something which is only

really experience at the very beginning of learning, but unless you have tried playing guitar before, it is something which is not really expected and something which often is not explained to beginners before they start (*Painful Fingertips in New Guitarists – A Temporary Thing* 2013). This can be off putting to the beginner because they are discouraged from playing because of the pain.

In an article for the National Association for Musical Education it is stated that one of the factors in relation to school children learning a musical instrument is that students may practice less during school breaks and become disengaged with learning the instrument (Mazzocchi, A. 2015). Thus making the learning process frustrating when they have to pick up the instrument again after a long break. Considering this, since most guitar lessons take place once a week, usually for half an hour or an hour, the same disengagement with the learning experience may apply during the time between lessons each week. Without a tutors immediate help or feedback a learner may become frustrated and less interested in the instruments if they feel that they are struggling to achieve something. This hypothesis seems to be backed up by an article by guitar tutor Aaron Matthies on his blog, where he states that students of his who have discontinued lessons usually call or email between sessions to say they won't be coming back, either because they found it "too hard" or "they did not think they were improving" (Matthies, A. 2009).

Another factor which may be linked to a student quitting the guitar is improper maintenance or lack of any maintenance on the instrument itself. In most guitar lessons, whether it is face to face with a tutor or from a book, what is taught is generally only how to play the instrument. Few students are taught how to perform routine maintenance on their guitar, whether this is cleaning, how and when to change strings or what kind of strings to use. Without proper maintenance, like with any instrument, a guitar will not function as well as it should. This can make the instrument harder to play and ultimately more frustrating for the beginner to learn.

There are resources available to the beginner guitarists outside of face to face tutorials. If the guitarists wishes to find out information, whether it is playing tips, queries about their

instrument or maintenance information, they may wish to look it up online or in a tutorial book. While these resources can indeed be useful many can take for granted that the user already knows a certain level of information which is often not the case, especially in reference to terminology. Another problem which may occur when using these resources is that there can often be a high degree of information, which for a beginner can seem daunting and can put players off.

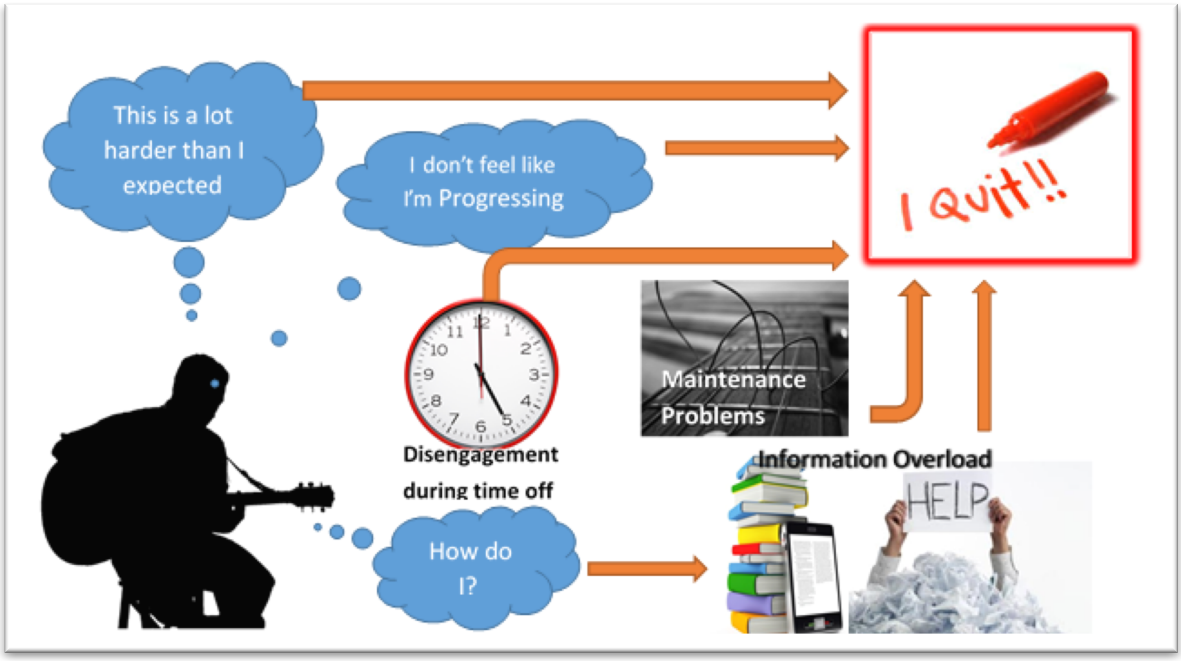


Figure 1. A rich picture depicting the problem

The rich picture diagram above (Figure 1.) aims to visually represent the problems faced by the beginner guitarist as discussed in the preceding paragraphs.

The Solution

In order to try and tackle the problems which were highlighted in the previous paragraphs it was been decided that a mobile guitar tutor application, would be developed. The working title for which became “Fretboard Buddy”. A number of factors led to this decision. Mobile learning or m-learning is a form of e-learning through apps developed for smartphones and other hand held mobile devices. There are many benefits to using m-learning. The first is flexibility and accessibility; due to the proliferation of the use of hand held smart devices in the last decade or so a large percentage of the world’s population now owns a smartphone. In an Ofcom report published in August 2015 found that in the UK 66% of the population now owns a smart phone. This means that learning applications are ever increasingly available to more and more users and because of the nature of the platform can be accessed anywhere and at any time.

Another benefit of the m-learning platform is that the learning experience can be personalised and can be tailored to the learning progress of individual users which as an effect can “stimulate individual autonomy” (Ferreira et al., Cited in Wankel & Blessinger, pg. 74) creating a more engaged learning experience. The same source also states that m-learning can be used in conjunction with other forms of learning, “enriching them”. This is something which highly influenced the choice of creating an app for this project. The idea for this app is not to be the only method or learning for a new guitarist but rather to act as a tool to be used in order to help new users progress and stay engaged during times when they are outside of other, more traditional, learning environments.

Platform

The app which will be created for this project has been chosen to be developed for the Android platform. Google’s Android operating system is the most adopted platform for smartphones and tablets worldwide, with roughly 1.4 billion active users equating to around 82 percent of the smartphone market (Lomas, 2015). With this in mind, it makes sense that, since the purpose of the application is to try and address the large percentage of people who quit the guitar, the application should be developed for the android platform in order to make it available to a larger number of users. Due to the widespread adoption of Android

smartphones, it has been decided that the application should be developed for mobile sized devices in order to reach as large an audience as possible.

Figure 2. A rich picture depicting the solution

The rich picture above (figure 2.) depicts how by addressing the problems which can lead to a beginner guitar player quitting by using m-learning the intended outcome would be to help the user to maintain the support and development needed in early stages of learning so that they decide to continue playing.

Existing Applications

There are numerous guitar based learning applications available to Android users via the Play Store. Research into a number of these apps was undertaken with the aim of providing some insight into which features are good and beneficial to the user as well as potential problems with each app. The idea being that this information will be used to develop a useful, functional and engaging guitar tutor app, while avoiding any negative features which may be present in existing applications.

In order to review each application it seemed that an analysis of the pros and cons for each of the applications is the best approach, taking into account their design, functionality and how useful they were to a beginner guitar player.

App 1 (All of Chords for Guitar – BRBR):

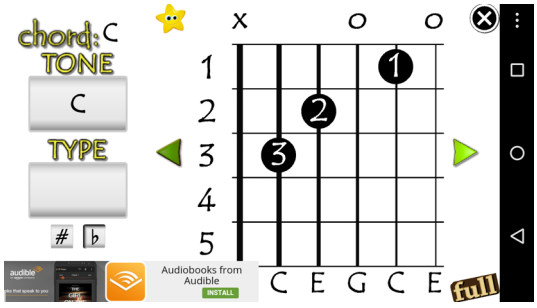


Figure 3. App 1: App's Main Screen

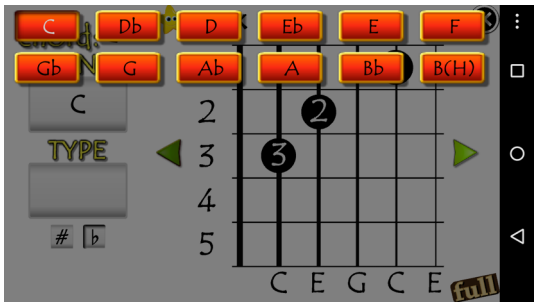


Figure 4. App1: Selection of different chord tone

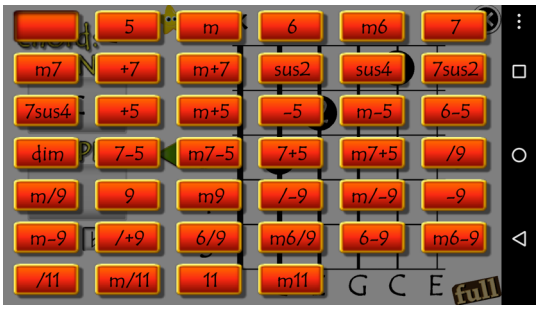


Figure 5.App 1: Selection of type of chord

The first application which was looked at was called "All of Chords for Guitar" which was developed by BRBR.

Pros

The main image for each chord diagram in fairly large and clear as shown in Figure 3. The operation of the application is pretty simple; there are 2 main buttons which allow the user to choose their desired chord and an arrow button on each side of the chord diagram to allow scrolling through chords.

Cons

The App assumes a certain level of knowledge from the user, especially in relation to the types of selectable chord variations which the app refers to as "type" (Figure 5.). As well as this, the app does not explain the meanings of the numbers in the chord shapes or the symbols above the diagrams. Some aspects of the layout are poorly designed. For example, the advertising banner covers the top string's musical note in the diagram. As well as this, the right arrow button is also quite close to the built in Android "Home" button, which may cause the user to accidentally close the application when trying to browse content.

App 2 (Guitar Lessons Beginners Lite – Webrox):

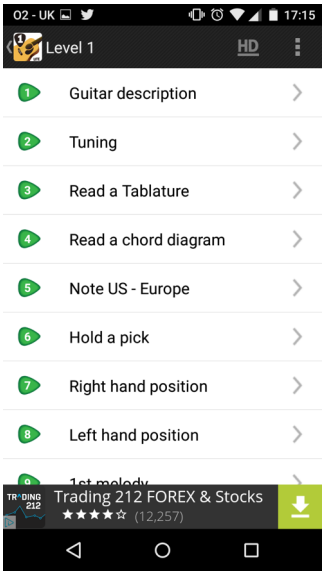


Figure 6. App 2: Lesson Menu

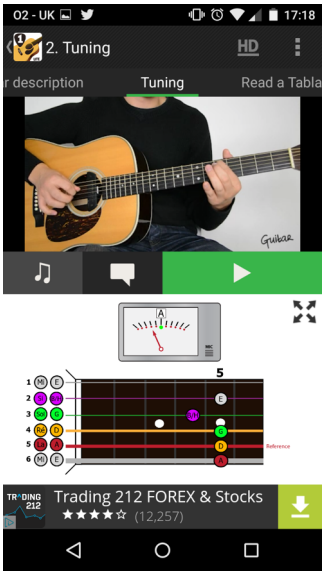


Figure 7. App 2: Diagram Layout

Figure 8. App 2: Lesson Text

The next application which was analysed was “Guitar Lessons Beginners Lite”, developed by Webrox. This version of the app is a free version which offers the “level 1” section for free.

Pros

The app offers quite a sleek, professional looking design with each lesson offering a diagram as well as a text explanation and in some cases a well-produced video presentation. Navigation between lessons is quick and easy, either through side swiping or via the sub-menu as shown in Figure 6. The app offers some good information for a beginner guitarist such as, how to hold the guitar and the plectrum, how to read a chord diagram and a description of the parts of a guitar.

Cons

The app claims to be targeted towards beginners, however some of the content within the first lesson is a little confused. For example, in the level 1 lessons tutorials on finger picking techniques precede the tutorials on basic chord shapes, which are generally the first thing a beginner guitarist learns. As well as this, some of the content featured is quite in-depth and not relevant to someone who is just beginning to learn guitar. Examples of this include, diagrams showing what notes are called in different cultures and languages and complicated scale diagrams. These have the potential to be off putting as the user might assume that they have to know these before they can learn to play.

App 3 (Basic Guitar Lessons – Maddy Apps):

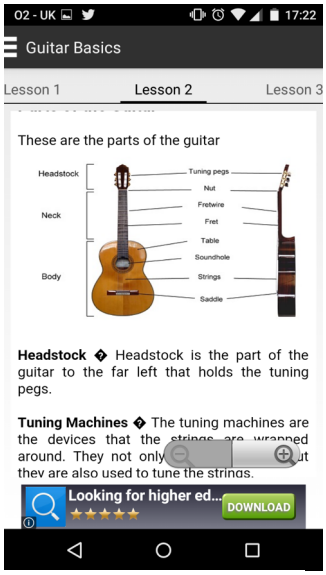


Figure 9. App 3: Guitar Guide

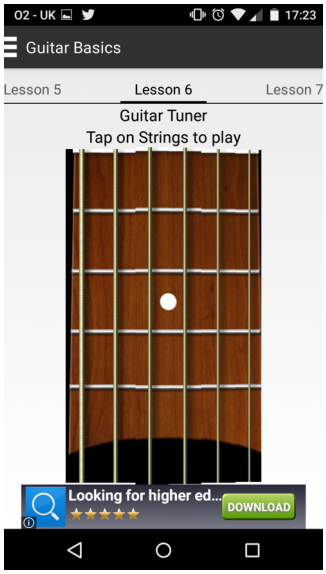


Figure 10. App 3: Tuner

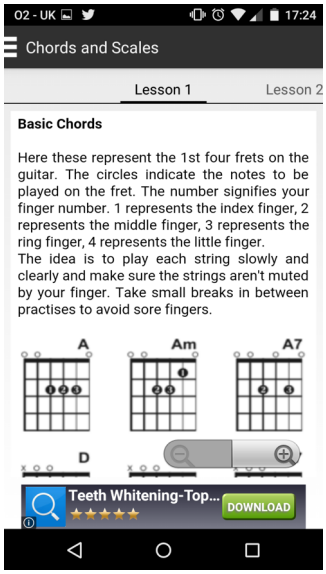


Figure 11. App 3: Chords

The third app which was looked at is “Basic Guitar Lessons” which was developed by Maddy Apps.

Pros

A lot of the information provided is very good and helpful to a beginner player. Most of the information is tailored toward a beginner and as such explains things in a straightforward way. The design of the app is quite simple and easy to navigate, either through the menu or by swiping gestures. A good feature which has been included is the guitar tuner (Figure 10.). To use this the user taps on a string to play a sound file which corresponds to the tone of the chosen string. The user uses these sounds to match the desired string tones on their own guitar.

Cons

With the exception of the guitar tuner, most of the app serves and looks more like a text book rather an interactive application. The chord diagrams, which look like they have originated from print material, are set out on a single scrollable page (Figure 11). This lack of interactivity may not encourage much engagement from the user. While most of the information in the app is good and useful some information is not really important to a beginner player. For example, within the app there are pages explaining sweep picking and pinched harmonic techniques, both of which are quite advanced guitar playing techniques.

App 4 (How to Play Guitar – Dreamer Studio):

Figure 12. App 4: Main Menu

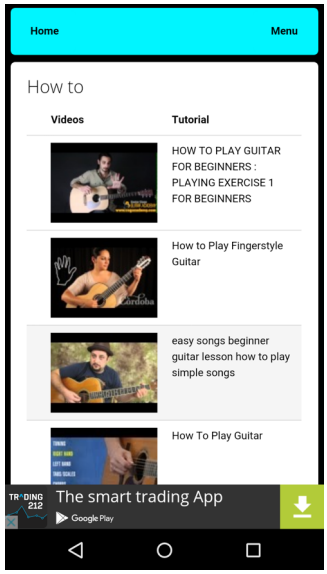


Figure 13. App 4: Content Menu

The fourth app is called “How to Play Guitar” developed by Dreamer Studio. The app itself is very unimpressive. The content within the app is essentially a menu of embedded YouTube videos with “how to play guitar” in the title. The main menu (Figure 12.) is made up of four buttons; the home button does nothing except refresh the homepage, the about button opens an information page which contains no content and the “how to” button being the link to the content menu (Figure 13.)

App 5 (Learn Guitar Chords – Alkaline Labs Apps):

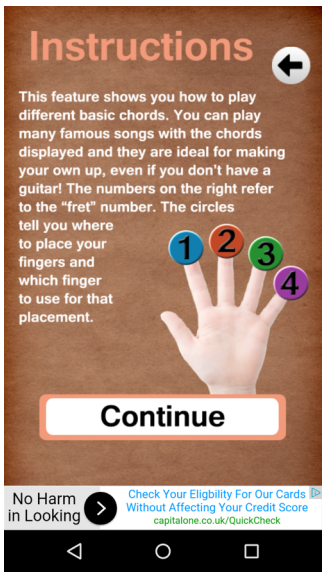


Figure 14. App 5: Instructions

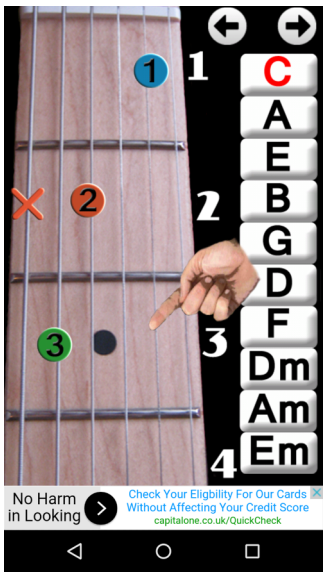


Figure 15. App 5: Chord Diagrams

Another app which formed part of the research was “Learn Guitar Chords” which was developed by Alkaline Labs Apps.

Pros

The app is very straightforward to use, the main feature of the app is that the user chooses a chord by pressing a button and the figure positions appear on the background image of a guitar neck. The way in which the chord diagrams change when buttons are selected feels quite interactive. There is a short description when starting the app which explains what each of the numbered circles on the diagram mean (Figure 14.).

Cons

Aesthetically the app is not very pleasing, this is mainly down to the use of a poorly taken photograph of a guitar neck being featured as the background image for the chord diagrams. Secondly, the app itself is rendered almost unusable due to the amount of intrusive pop-up advertisements. For instance, during the first 30 seconds of using the app, the session had been interrupted by advertisements at least four times. For someone who wants to be able to study the screen as they are playing the guitar, this would be very off putting. The app also fails to mention what the “X” symbol on the diagram means and features a symbol of a hand which gestures to certain points on the diagram which don’t seem to mean anything (Figure 15.). For someone who is just starting to learn to read chord diagrams this could be quite confusing as it may lead them to believe that they have misunderstood something.

Summary of Existing App Research

Based on the research conducted into existing guitar tutor and guitar chord library applications it is clear to see that there is room for improvement within the genre. Some of the apps are quite good and have some useful features. However, in each of the apps which were researched there is a considerable number of negative points to consider. Knowing these points should allow a better end product for the development of the “Fretboard Buddy” application, as the flaws highlighted as negative in each of the researched applications can be avoided. As well as this researching these products has highlighted a number of useful features from which to draw inspiration from during the development of this project.

Based on the information gathered through identification of the problem and research of existing applications there are a number of features which it was decided should be included in the development of the project application:

- The main feature of the application is the depiction of basic chord diagrams. These should have some level of interactivity in order to make the user feel more engaged with the process of learning these chords. The initial intention is that this would be done by changing the figure position diagrams on selection of a button, in a similar manner to the method which was seen in the application developed by Alkaline Labs Apps. In order to engage the user even more it is felt that the chords should be presented in a number of learning stages with the player's progress being saved through each level. The intention is that this should make the app feel more tailored towards the individual user and as a result give the impression of a higher level of user involvement. The initial idea was that there would be a maximum of around 12 chords, this will cover the basic chords from A to F and their minor variations.
- Outside of the progression "Levels" users should be able to access any of these chord diagrams independently so that they can go back to chords they have already learned or looked at.
- The app should include pages with information about the guitar and basic playing information. These should include a presentation of the different parts of a guitar, playing tips (including how to avoid injury) and how read chord diagrams and tablature.
- The app should also include maintenance information including pages on how to properly and safely clean a guitar in or to avoid damage and degradation. There should also be some information on guitar strings and how to properly re-string the instrument.
- Another useful addition would be a guitar tuner page. A good approach to creating a guitar tuner would be to create one which is similar to the tuner included in the application created by Maddy Apps. In order to provide a bit more information to the user it should be made clear which notes each of the strings are actually playing by displaying the names of the notes for each string.

- All pages should be clear and easy to navigate with the information on each page being as concise as possible. This is to avoid the problem which was encountered in some of the existing apps of having an overabundance of information, more suited to a learning text book rather than an interactive application.

Professional Issues

Business Case

"Fretboard Buddy" was developed as an aid to the traditional method of face to face tuition as opposed to a direct replacement and as a result there are a number of financial costs to consider for the successful use of the application. To start with it is assumed that most beginner guitar players using the app will own a guitar, there is quite a large price range for new guitars ranging from as low as £20 to anywhere in the hundreds of pounds. Since the app is aimed towards beginners the most likely price range is between £20 and £150 as this is the bracket into which most beginner guitar prices fall. Other than the guitar some other equipment is generally needed. Since the app aims to provide information on how to restring a guitar it makes sense to consider the cost associated with a pack of six strings which usually costs anywhere under £10. There is also the price of plectrums, although this is not such an important factor as plectrums can be bought as cheaply as 20 or 30 pence individually. Since the app is intended for use alongside face to face tutorials it is also necessary to consider that the user might also need to pay for lessons, although this is not obligatory and not everyone chooses to learn guitar from a professional tutor. After considering these initial costs it is important to consider the benefits associated with the use of this application:

- Since the app aims to provide a resource for a number or learning materials this should enable the user to reduce the number or learning materials such as books and tutorial CD's allowing them to save on costs.
- Providing a tuner in the app should mean that the beginner guitarist has access to a method of tuning without needing to buy a separate guitar tuning product.
- Due to information in the app about restringing the guitar this can lead to a reduced cost in having to replace strings due to the fact that a proper restringing technique will reduce the amount of strings broken.

- Maintenance information within the app will also allow for less cost associated with fixing or replacing broken parts.

Besides the benefits seen by the guitarist there are also a number of factors which may benefit other stake holders, such as:

- Increased retention in the number of people playing guitar is beneficial to guitar manufacturers because it will allow for a larger number of guitar sales.
- This will also benefit music instrument retailers who sell guitars.
- Guitar tutors will also feel the benefit as the number of students which they lose would be diminished.

Project Risks

Event	Probability (1-5)	Impact (1-5)	Risk Factor (PxI)	Contingency Plan
Physical Risk				
None	-	-	-	n/a
Project Management Risk				
Missed Deadlines	**1**	**5**	**5**	Make sure to employ proper time management when working on the project. This can be done by working to a schedule and setting multiple deadlines
Unforeseen Circumstances	**3**	**2**	**6**	Be on schedule or ahead of schedule so that any unforeseen circumstances such as time off due to illness do not result in a setback in progress
Loss of materials	**1**	**5**	**5**	Make backups on multiple devices so that any loss of materials can be retrieved from another source

Technical Risk				
Loss of data/code	**1**	**5**	**5**	Make sure to make backups on multiple devices as often as possible. This way if data is lost or deleted a recent copy can be retrieved without too much setback.
Hardware failure	**2**	**3**	**6**	Ensure access to multiple computers
Code not working	**2**	**4**	**8**	Make sure to regularly check that code is running properly and there are no errors within the code.

Table 1. Risk Table

A number of perceived risks were drawn up at the beginning of the project as outlined in Table 1. Throughout the project each risk was successfully avoided by adhering to the contingency plans laid out in the table.

System Requirements

User Stories

A number of user stories have been identified in order to create an effectively engaging application. This should allow a clearer, more personal understanding of the functional requirements as they relate to a real user. Methods are listed for each user story as to how each should be catered for.

"I want help with learning to play the guitar."

- Provide information on terminology and parts of the guitar.
- Provide a number of chord diagrams with simple explanations on how they are used.
- Provide chord diagrams in a sectionalised lesson plan beginning with the most basic chords and gradually working towards harder chords.

"I want to see my progress."

- Create a method to save the level to which the user has progressed through the lesson system.

"I want to know how to keep my guitar in good working order."

- Provide short informative and easy to follow guides on how to re-string a guitar.
- Provide information on how to clean and maintain a guitar.

"I want to feel free to learn at my own pace."

- Allow users to access information on any of the chord diagram lessons or other information regardless of whether or not they have completed each of the lessons.

Based on the information gathered in research of the problem and proposed solution sections as well as the user stories a number of requirements for the intended app can be listed. These requirements can be classed as either functional or non-functional. A basic description of a functional requirement is that it is one which specifies "something the system should do". Non-functional requirements on the other hand should describe "how the system should behave" (Eriksson, 2012).

Functional Requirements

- The app should allow tracking of user progression.
- The app must provide lessons and information which are factually correct.
- The app must allow users to navigate to different sections of information within the app as they please instead of being restricted to a lesson plan. This includes allowing users to access chords from the lessons which have been previously completed.
- The app must provide information in a number of different, clearly defined sections.

Non-Functional Requirements

- The interface should be aesthetically pleasing to help encourage continued usage.
- The application should have a level of interactivity outside of just navigation in order to encourage engagement.
- User interactivity with the app should be straightforward and logical.
- User progression should be stored for each individual user with the help of a database.

- The app must be able to run on multiple sized devices while maintaining a consistent layout.
- All buttons and links on screen should have an appropriate related action performed.
- All section headers should be clear and descriptive in order to make navigation as easy as possible
- All diagrams, especially the chord diagrams, should be large enough so that they can be read easily and without confusion.

User Interface Design

To create a functional version of the final application it was first necessary to draw up a framework and storyboard for the exterior design so that there is a clear outline to work from for the development of the user interface which adheres to the functional and non-functional requirements of the application. When designing the features of the application it was necessary to consider good principles of mobile app design. These principles were sourced from an article by Google's UX Research Lead, Jenny Gove. Gove states that "as many as 25% of app users open an app once and never return". For this reason it is important to follow good design principles in order to provide a good user experience so that users' engagement will be maintained.

Main Menu

The first screen which the user will interact with is the main menu. It was considered, that before the main menu page there may be a splash page with a logo for the application. This was deemed unnecessary, however as a splash screen provides no real advantage in the design of the application. The menu will be simplistic in its design and will provide 4 large buttons to allow the user to navigate to all main categories of the application as shown in figure 16. Initially the tuner section was accessible from within the "tips + Info" section. However this was changed due to the fact that the tuner is considered to be a key feature of the application and may require frequent usage and therefore navigation to this section should be much easier. This is in line with the good design principles which state

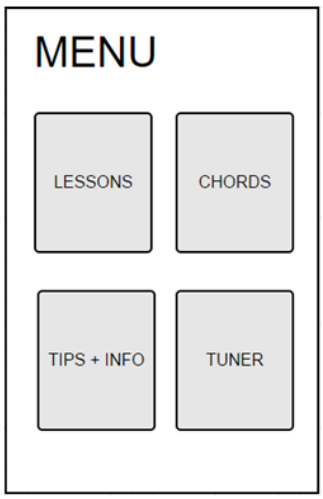


Figure 16. Main Menu

that the developer should “show the value of the app upfront” by displaying the apps key features “in context at the appropriate place in the app”(Gove, J 2016). Each of the buttons in the main menu will navigate to a sub menu for each category, apart from the tuner, which only consists of a single activity.

Lesson Menu

The “Lesson Menu” like the previous menu is fairly straightforward and simplistic in its design, it provides six buttons to each of the separate lessons, a menu title and a “home” button in order to navigate back to the main menu. This button is intended to be represented on all pages which are not the main menu, so that no matter how many activities deep the user has navigated into the application, they should always have an easy way to get straight back to the main menu. This was not present in the initial design, but has been included as a key navigational feature of the application. . Despite this feature the app always allows the user to select the Android systems “Back” button to move back through a single activity instead of only navigating back to the home screen menu. This means that users are not forced to go back to the home screen any time they want to navigate to a previous screen, this “eliminates frustration and the need for any inadequate workarounds” (Gove, 2016). The lesson menu will also represent one of the key features of the app. That is, that user progress through each lesson will be visually represented by showing the buttons for each lesson as a green colour if the lesson has been completed. Initially the button layout for this menu was similar to the main menu, however, it was felt that, while this works well when representing different distinct categories, for this menu, the stacked button style works better as it represents the sequential nature of the lesson structure much better.

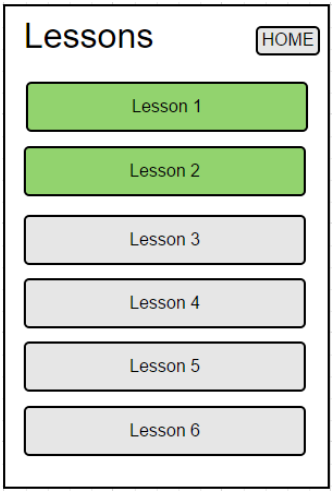


Figure 17. Lesson Menu

Lesson Page

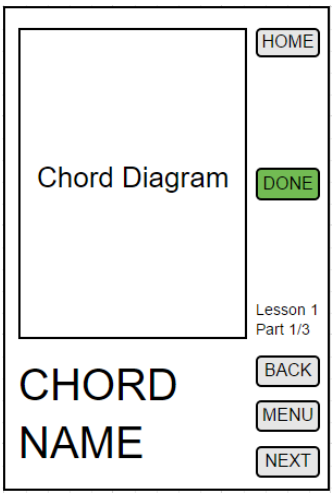


Figure 18. Lessons

Figure 18 shows the activity page to which all buttons of the lesson menu will navigate to. Depending on which lesson is chosen a different layout will be loaded programmatically to the screen by the "Lesson" class. Figure 18 shows the layout in regard to the depiction of chord diagrams, since this is the main focus of the Lesson activity. In this case the chord diagram is displayed on the screen. The corresponding name for this chord will be displayed below the chord. Both the diagram and the name will take up a large portion of the screen as they should both be easily read and due to the fact that they are the main focus of the page. In the event that the contents of the page are not chords, the diagram section is replaced by an appropriate image, while the "chord name" section is replaced by a scrollable text area. The proportions of each are dependent on the importance of the size of the image and the amount of text to display. The "Done" button beside the image will allow the user to choose if they are confident with what they have learned in the lesson, this button is only visible on the last page of each lesson. When the user selects this button they are prompted with a message which asks for their confirmation in order to save progress data. In the event that the user has already saved this data the "Done" button will now be displayed in green each time the user revisits this page as well as mentioned, on the lessons menu. There are three buttons present on the bottom half of the page, two of these, the "Menu" button, and the "Next" button, will be visible at all times during each lesson. The "Menu" button will, as the name suggests, navigate back to the previous menu. The "Next" button will navigate to the next page within that particular lesson. When the user navigates to the last page in each lesson the "Next" button becomes the "Next Lesson" button to allow them to navigate to the next lesson without having to return to the menu. The back button in each activity is not visible on the first page of each lesson due to the fact that it has nowhere to navigate to and therefore is not functional.

Chords

When the user chooses the "Chords" button they will navigate to a menu of seven buttons, similar in design to the "Lesson Menu" with each button representing a chord from "A" to "G". Once one of these buttons has been selected the user will be directed to the chords screen, as shown in figure 19. The chord page is fairly similar to the layout of the main "Lesson" page, and will operate in a similar fashion, meaning that it will consist of a single page which loads a different image and chord name based on the selection from the previous menu. The main difference is that instead of a "Next" button, there will be a "Variation" button, as represented in the figure by the "Minor" button. The operation of this button is that when the user chooses a chord to view they can press this button which will show a variation of the current chord. For example, if the user chooses to view an "A" chord, the page will open on an "A Major". They can then press the "variation" button which will then change the chord on-screen to an "A Minor", the button will be freely clickable to allow the user to switch back and forward between the two as much as they like. All chord diagrams, as with the Lesson activity will play the sound of a chord when the user presses the image.

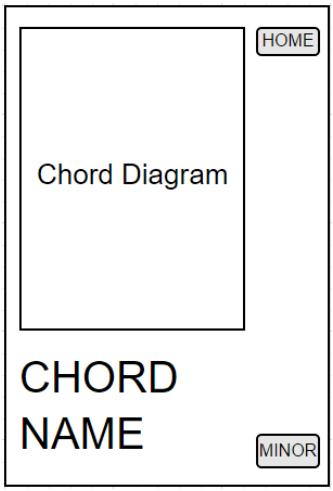


Figure 19. Chords

Tuner

The "Tuner" page, as shown in figure 20, will consist of 6 buttons representing each string of the guitar. When each button is pressed the app will play a sound file of the appropriate guitar string being plucked. The user will then tune their guitar to the sound of the corresponding string on-screen. The layout for the buttons is that they are invisible and will overlay an image of some actual guitar strings in order to give the impression that the user is pressing the strings of the image rather than a number of buttons. Beneath the image there is a short set of instructions which will indicate to the user on how to use the tuner effectively. It is felt that this method of displaying the buttons is slightly

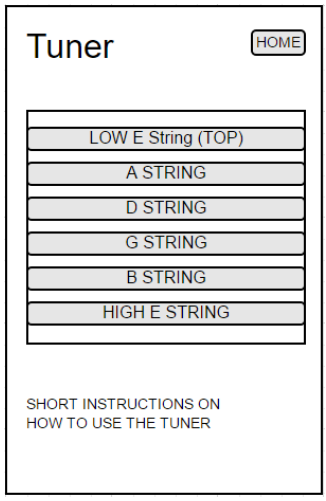


Figure 20. Tuner

more immersive and engaging than simply presented the user with some default buttons. On each press of a string a toast notification appears to indicate the note of the string being played. This was added due to the fact that, since the buttons are invisible they could not show any text. The usage of the toast feature also provides a more interactive feel to the page due to the fact that the toast reacts to the user's selection input.

Tips and Info

The Menu for the tips and Info section is laid out in a similar fashion to the main menu, with the main categories being represented by four large buttons on-screen. Within each category most of the content will be largely image and text based. For this reason each of these pages will be fairly simplistic and will include an image field and a text field which, depending on how much information can be fit on-screen the text field, is scrollable. With a few of the pages where there is a lot of textual information the image was left out due to the fact that an image for these section provided no instructional advantage and would only detract from space which could be occupied by text. As with the previously discussed pages, the info pages remain consistent in including a "Home" button for navigation to the main menu, as well as a making use of the Android "Back" button to return to the "Tips and Info" menu.

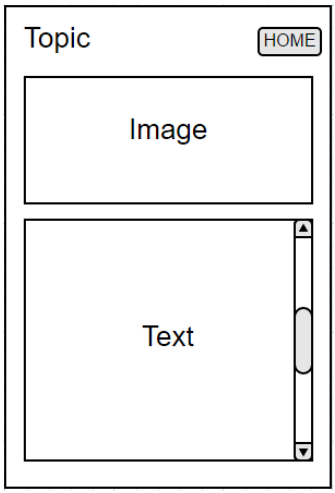


Figure 21. Info pages

Design Principles Throughout Application

As mentioned, it was felt that it was important to design Fretboard Buddy with principles of good application design in mind in order to maintain user engagement. These key principles as are as follows (Gove, 2016):

- **"Show the value of your app up-front"**

  This is achieved by presenting the main menu to the user when the application is opened, which lists all main categories within the application allowing for straightforward and easy navigation.

- **“Organise and label menu categories to be user friendly”**

  Menus and buttons are clearly defined and organised, which should prevent any confusion for the user.

- **“Allow users to “go back” easily in one step”**

  As discussed, the user can navigate directly to the home page via the “home” button or to go back a single step by pressing the Android “Back” button.

- **“Speak the same language as your users”**

  This refers to not using any “technical speak” in the app which may not be familiar with users. For that reason all information within the app is in clear English. Any guitar specific terms are explained within the application.

- **“Be responsive with visual feedback after significant actions”**

  This is especially important in the lesson pages when the user presses the “Done” button. To provide responsive feedback this button will turn green while the user remains on the page to indicate that the user has saved their progress.

- **“Ask for Permissions in context”**

  This is appropriate in the lesson pages where the user is prompted for confirmation on whether they want to save their data when the “Done” or “Next Lesson” buttons are pressed. A differing and appropriate message is displayed for each button due to the fact that their basic functions are not the same.

Layout Consistency

The development of this app is intended, as previously stated for mobile phone sized Android devices. This means that device screen sizes may vary slightly from device to device so the app should be developed so that there should be no negative effects from the difference in screen size between one device and another. To do this the layout is designed using sizing options which are relative to the size and shape of the screen such as, “fill_parent” and “wrap_content”, as well as assigning a layout “weight” parameter to layout objects. Using these options instead of a set image size will avoid the problem of things

becoming too spaced or stretched when using larger devices or elements becoming too cramped on smaller devices. Although the layout is consistent across multiple sized devices, larger devices will provide larger space within textViews, meaning that on larger devices more text can be shown on screen without the need for scrolling.

Theme

In order to provide an aesthetically pleasing and harmonious feel throughout the app, a standard colour scheme is used throughout all areas. The idea for the colour scheme is that the background throughout the whole application should look like the wooden texture of an acoustic guitar. The tone of the wood colour is also quite light as it makes the app look quite clean and neat. As well as this, it also contrasts well with the black and white which will be used for buttons and text features as shown in Figure 22. This combination of colours provides quite a smart and professional looking design. A custom design was chosen for the button style which made buttons rounded and semi-transparent. It was felt that this design was more visually pleasing than a standard black rectangular button since that design looked very basic. Figure 22 also shows an example of how the green buttons will be used to depict the user progress in the “Lesson Menu” and the main “Lesson” Activities. The figure also depicts how the Application title and logo will appear in the Android home screen. The red colour of the logo was chosen because it was felt that it contrasted well with the rest of the colours in the application. The use of this red colour for the logo, inspired the use of the same colour to depict the scroll bars within TextViews and the finger positions on chord diagrams in order to maintain consitency with the app logo.

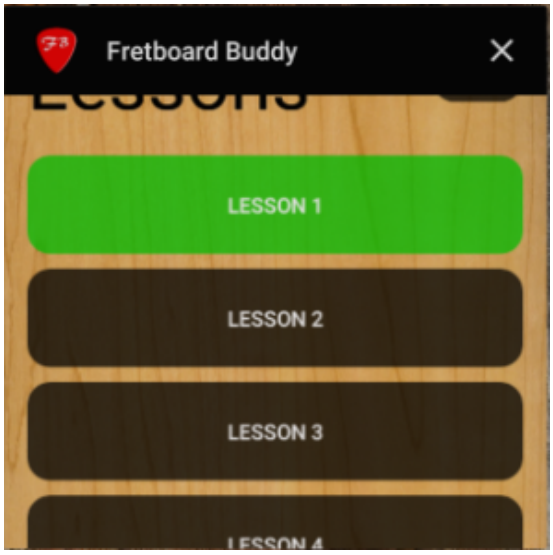


Figure 22. Colour Scheme

Story Board

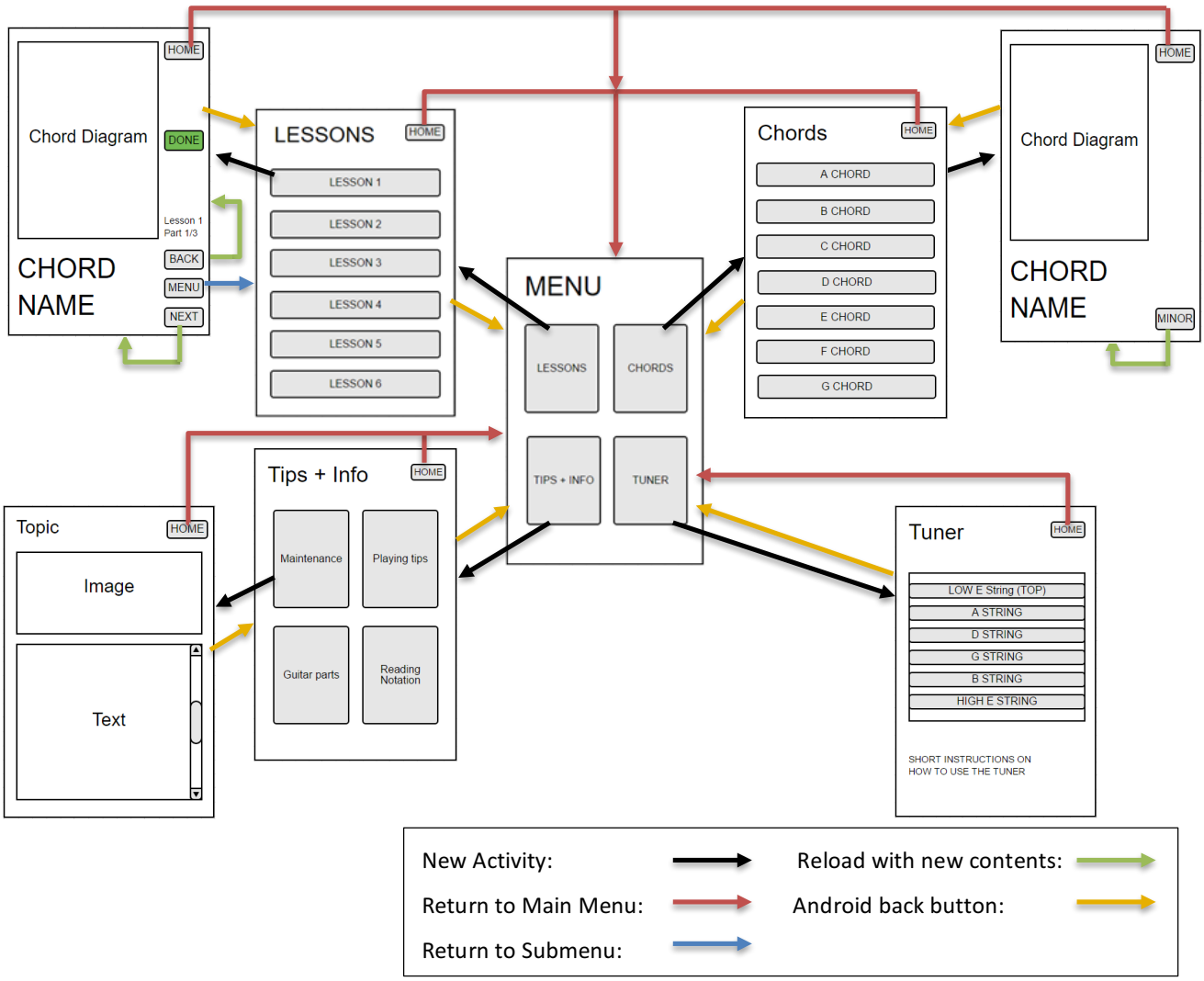


Figure 23. Story Board

Architectural Design

The Architecture of the application has changed somewhat from that of the design in the initial stages of the project. All features of the app remain consistent with the original design, however, due to becoming more experienced in using the Android Studio software package and the process of application development since the Analysis and design stage, a few features of the architectural design have changed. Primarily to become a much more efficient means of development. For instance, in the initial design each chord and lesson was represented by its own separate activity. A much more effective approach and the one which was adopted for the current architectural design was using a single activity for each chord or lesson with the information being depicted programmatically onscreen depending

on switch statement conditions determined by button presses. This method of displaying information is also used for the tips and information pages. This cuts down on the amount of activities which are created within the app and as a result means that the size of the final app will not be unnecessarily large, and won't have to load as much information on start-up. The main lesson section, the chord section and the tuner all include the ability for the user to be able to play sound files. In order to do this the "soundpool" feature built into Android studio will is used. This allows the app to load a pool of sound clips which are stored in a "Raw" folder, within the app, and played on a method call. The outline for the architectural design of the application as a whole is shown in figure 24 below.

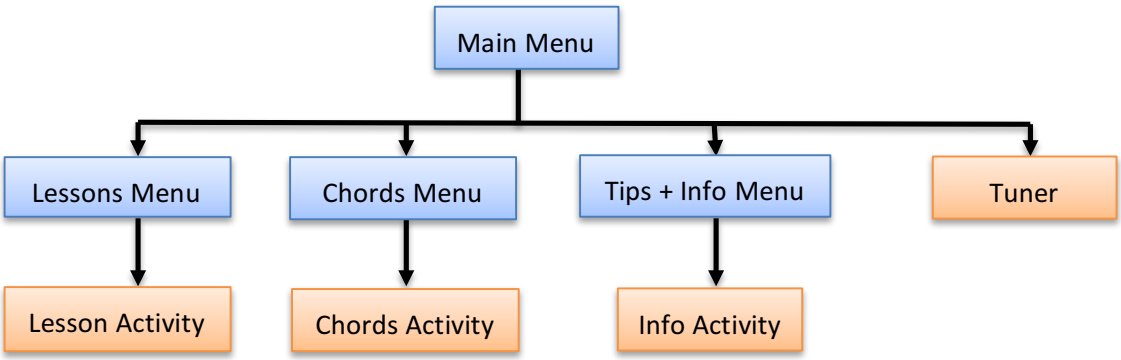


Figure 24. Architecture

To demonstrate how the architecture shown in Figure 24 is a much more effective design. The single activity "Lesson Activity" is used to depict all six lessons, which consists of nineteen pages in total. The "Chords Activity" will also be used to depict twelve different chords and the "Info Activity" will be used to show six separate categories of information, meaning that it is possible to show close to forty pages of information using only three Android activities.

Database Design

In order to keep track of user progression through the lessons portion of the application a database is used. Due to the fact that the application only stores progression information from the lessons, the database being used in the application requires the creation of only a single table of information. In order to implement a database the application makes use of the SQLite database library which is a built in feature of the Android operating system. Since

there is only a small amount of information needed to be stored within the database, and the information only needs to be stored locally, then there is no need to use online servers, so SQLite will suit the needs of the application.

ID	LESSON NO.	COMPLETION
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1

Table 2. Database

The database table which will be created within the app consists of three columns representing the I.D., lesson number and the completion of the lesson, as shown in Table 2. The initial design for the database was that the completion column would initially be created with “0” values for each row, indicating that the corresponding lesson has not been completed. When the user chooses the option at the end of a lesson to indicate that they have completed it, the value for completion, would be updated to the value of “1”. However, during implementation this was seen as an unnecessary process. Instead the creation of the database produces an empty data table. When the user saves their progress this adds the lesson number and a “1” value to indicate completion to the table. If the application reads that this lesson has previously been completed the user cannot save this data again. This avoids having multiple entries in the table with the same information, meaning that the application does not unnecessarily store large amounts of data within the user’s device. If information has been stored in the table the “Lesson Menu” class reads from the database which of the lessons have been completed then the buttons on the menu for that lesson are represented in green so that the user has a visual representation of their

progress each time they enter the menu. The I.D. column is used simply to auto-increment the entry position of new values within the table.

Implementation

Working Methodology

In order to be able to produce and deliver an effective product within the project delivery window it was necessary to choose a software development cycle which would allow continuous development and refinement at all working stages within the development timeline. For this reason it was decided that the development method would be based on the Agile methodology due to some of its key principles which, it was felt would be well suited to the developer as the working methodology for this project. Unlike traditional "waterfall" methods of development, one of the key aspects of Agile development as outlined by the Agile Alliance website, is that it "welcomes changing requirements" at any stage of development. This would allow the application to be added to or refined based on any updated functional requirements during the development stage instead of being strictly fixed to requirements set out at the project planning stage, meaning that if any already implemented features were deemed to be unnecessary or in need of re-evaluation then this could be addressed.

There are other key principles which supported the decision of using an Agile development strategy. These include the fact that agile proposes "small, incremental releases", a "frequent delivery of products" and advises that each feature is completed before moving on to the next (Waters, K 2007). By basing the project delivery timeline on these principles it would allow the developer to complete the project in several incremental stages, gathering feedback on the project at each stage and using this to refine the project.

The outline was to complete the project a number of sprint stages with multiple deadlines and chances for feedback. These were, firstly the Analysis and design stage which included research and an initial design proposal. Secondly the Design and implementation stage which included refinement of the application design and implementation of an application prototype. Based on the feedback gathered from this, stage 2 of the development would then be undertaken to produce and fully functional final design.

Following this stage, evaluation and testing were undertaken which would provide feedback on any design flaws which needed to be fixed before final delivery as well as any improvement proposals for improvements on a future iteration of the product.

Development approach

Development of this application was undertaken in two main stages the first being the design and implementation of a working prototype, and the second being the implementation of a final product based on evaluation of the prototype from the developer, project supervisor and potential users.

Within each stage the development strategy the development approach was to complete each separate activity before moving on to the next, keeping in line with the Agile principles stated earlier. The development strategy for each page was to begin by implementing the activity's XML layout design first and then move on to the functionality of the page within the activity's corresponding java class.

Prototype Development

With that in mind it made sense to start the process by developing the main menu first, as this is the centre point from which all other activities are accessed in the app, as indicated in the story board in figure 23. The next step which was taken was to develop the menu for the lesson section. For the prototype development stage this did not include the use of the database tracking feature as the database would not be implemented until stage two of the development process. The main reason for this is that due to the short development time of the application the developer wanted to have a complete navigational framework for the application prototype before implementing any other features. This was therefore prioritised over the database initially.

After the lesson menu was created the chord menu was then created using the same basic layout. The main difference between the two being that the chord menu had an extra button added since there are seven chord options and six lessons. For the next step the Lesson activity was created. This was given priority over anything else as the lessons are intended to be the main focus of the application. As well as this the layout features created in the lessons could also be used in creating the single chord activity afterwards. In the

prototype application the content of the lessons was simply two chords diagrams for each lesson, with the intention of adding a save progress button once the database had been implemented. The layout of the lesson activity was then used as a base for the next stage of developing the single chord activity.

Following this, it was important to implement the tuner activity as this was deemed to be another key feature of the application which needed to be present in the prototype design. At this stage this was simply a set of six buttons, with each representing a guitar string and playing a sound file of the appropriate string accordingly.

The features that had been implemented at this stage made up the design on the prototype application. The tips and information section of the app was not implemented at this stage as it is comprised mostly of text and images. Since it has less functionality and is mostly content based compared to the rest of the application it was felt that this section was not a priority for the delivery of the prototype. This would be one of the last features to be implemented once the other more interactive elements of the app design had been fully implemented.

Development of the Final Product

After meeting with the project advisor and gaining some comments and suggestions on the prototype, the next stage of development was undertaken with this feedback in mind.

The first step taken at this stage was implementation of the database class named “DBHandler”. Once this had been completed the feature of being able to save progress data to the database was added to the lesson activity. Following this the feature of reading progress data and displaying it to the user was added to the lesson menu activity. The next step was to improve the features of the lessons. To do this more content was added to each lesson with each lesson having three or four pages instead of two and including more varied information and tutorials instead of just chord diagrams. As well as this the level of user interaction was increased by creating the ability for the user to listen to an example of each of the chords onscreen. This feature was then included in the separate single chord activity which would allow the user to access this feature without having to go back through previous lessons.

Once this had been completed the tips and information section was created. This included a tips and information section with the same layout design as the main menu and the main information content activity which consisted of textual information and images.

Once all the design features had been implemented the app as a whole was “polished”. Firstly by updating the tuner activity to include an image of guitar strings which could be pressed individually rather than a set of buttons on screen and instructions on how to use the tuner as well as making the audio in the tuner repeat in a loop five times instead of just once as was the case in the prototype. This means that while the user is tuning their guitar they will have to stop and press the screen a lot less. All buttons within the app were then improved aesthetically to provide a more professional look than the previously used Android default styles. Following this, images were re-edited using Photoshop and updated within the application to make sure they were of good quality, easy to understand and as visually pleasing as possible to the user.

Development Resources

At the outset of this project the developer was relatively inexperienced in Android development. For this reason a number of different resources were used throughout the development of the project in order to learn, and improve any existing knowledge of, Android programming techniques. These included Android programming tutorials from the YouTube series’ “Java Programming with Android Studio” by John Tapley, “Android App Development for Beginners” from the channel “thenewboston”. As well as these video tutorials the developer also made use of the official Android online developers training guide.

Coding Implementation

There were a number of coding practices which were maintained throughout development of the application. When each new activity was created within Android studio, the option to generate a layout file for each class was chosen. With these created, the developer began by creating the base layout of each activity using the layout design editor and it’s component tree in

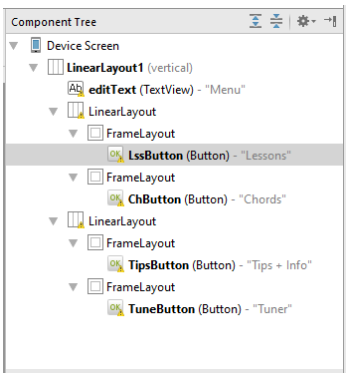


Figure 25. Component tree

order to add and visually organise any components added to the device screen (Figure 25).

Once the layout had been organised in this manner, the text editor was used to set the properties of each of the components by directly editing the XML text for each. This included writing parameters to set the size, weight, id, colour, text appearance and onClick methods. Creating the layouts took some time to get to grips with, especially when it came to creating the more complex layouts due to the need to have multiple “LinearLayout” and “RelativeLayout” components nested within each other. One persisting problem, which took some time to find a solution to, was that when the developer set a weight value for each of the included components it seemed to work in the opposite way to which it was supposed to. According to the Android Developers guide, the “Layout_weight” attribute “assigns an “importance” value to a view” which indicates how much of the available space that view should occupy. This means that if two views are set side by side and both are given an equal weight value, they will occupy an equal amount of screen space and be placed evenly side by side. The problem which occurred was that when the developer set a component’s weight value to “2” for example, and an adjacent component’s weight to “1”, the component with the larger weight, would become smaller, when it should realistically take up more of the space. Eventually the reason for the problem was found to be that overall parent layout on screen had either a “Layout_height” or “Layout_width” attribute of “wrap_content” which, when changed to “match_parent”, solved the problem. It was unclear in the beginning why this problem occurred as this characteristic caused by the main parent layout was not described in the manual. Once this had been addressed it made creating layouts for other activities much easier.

When adding buttons to the layout, the onClick attribute was assigned to a method in the XML editor as shown in Figure 26. While it is possible to manually create an onClickListener to determine the view of button clicks, it was felt that assigning a method to call in the XML editor was a simpler and more straightforward way of assigning button operation. This is because when a method is assigned to the onClick parameter of a button, Android automatically sets up a listener and passes the view as a

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="A Chord"
    android:id="@+id/ChordA"
    android:layout_weight="0.15"
    android:clickable="true"
    android:onClick="ChordNav"
    android:background="@drawable/custombutton"
    android:textColor="#fcfafa"
    android:layout_marginBottom="10dp" />
```

Figure 26. Button layout in xml

parameter to the defined method which is being called. It was felt that this provided a more straightforward and organised way of dealing with the button operations within java code as it meant that each button was assigned its own method to call in order to perform a particular operation.

Database Implementation

The creation of the database is largely based on the tutorials from the YouTube channel "thenewboston" with some changes made in order to more appropriately fit the operation needed for the Fretboard Buddy application. The database was implemented using Androids built in SQLite database functionality. SQLite implements "a self-contained, serverless, zero-configuration" SQL database within the Android architecture (SQLite website). This means that database creation and management is saved locally to the user's device requiring no connection to online servers. SQLite was chosen as the method of database implementation for these reasons. Due to the personal nature of the application and the data being saved the use of servers and an online connection was deemed unnecessary.

The "DBHandler" class was created in order to take care of all database operations. These include, creation of a database, methods to deal with adding information to the database, upgrading the database and reading information from the database. Since each database entry signifies a user's final completion of a lesson, it was felt that it was unnecessary to include the ability for a user to delete information from the database. To further negate the need for this option, unintentional adding of information to the database would be avoided by providing save notifications to the user. This is discussed in more detail later in the "Lessons" section.

The database was implemented by creating a new java class named "DBHandler" that extends Android's built in "SQLiteOpenHelper" class which manages database creation. To start with, a number of variables were declared to hold information on the database properties such as version number, database name, table name, and column names for each individual column.

A class constructor is used with the variables for "DATABASE_NAME" and "DATABASE_VERSION" being passed as parameters in order to create and initialise the

database the first time that a DBHandler object is created. This saves the database file "DATABASE_NAME" which in this case was assigned the value "progressdata.db" to the Android architecture meaning that, once it has been saved, if the user returns to the application after closing, the database and any information saved within will still exist. It was first necessary to override the "onCreate" method in order to add a query which would create an appropriate database table. The "onCreate" method is only called when the database is created for the first time as indicated in the Android developers guide. This means that a table will only be created once and saved to the database file, meaning that all information saved in the data table will not be overwritten each time the user accesses the database.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String query = "CREATE TABLE " + TABLE_NAME + "(" +
            COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT , " +
            COLUMN_LESSON + " INTEGER ," +
            COLUMN_COMPLETE + " INTEGER " + ");";
    //execute the query assigned to the String variable "query"
    db.execSQL(query);
    Log.d("DATABASE OPERATION", "DATABASE CREATED");
}//onCreate
```

Figure 27. DBHandler - OnCreate Method

The SQL query used to create the table (Figure 27) within the database was fairly straightforward due to the developer having some prior experience in SQL database operations. The query which is executed states that a table is created named with the "TABLE_NAME" variable which is "dataTable". The table has 3 columns, the first being an "ID" column, which auto-increments meaning that each time an entry is made to the database this value will increment by 1. This column was added as a solution to the problem of the lesson column not incrementing, which caused entries to get overwritten in the same row each time the user attempted to save their progress. The next column is the lesson column, which would hold information on the lesson number which is completed. The third column is the "complete" column, in which a value of 1 will be entered for any lesson which has been completed by the user. Initially the plan was that a database table would be created with the values for lessons one to six being entered on creation with the "complete" column being initialised with values of "0". Each time the user would complete a lesson then

these “0” value would be changed to a “1”. During the implementation of the database however it was realised that this means of implementation was unnecessary. Instead, a row would only be added to the table for each column as and when each lesson had been completed, this provided the same functionality to the user and made a much more straightforward approach.

Following the “onCreate” method another method override was implemented on the “onUpgrade” method. This method is called any time the database needs to be upgraded. This is not used within the version 1 of the application which is being submitted for the project but was added to the code in order to allow for any future update. The operation of this method is simply an SQL query which deletes any older version of the database and replaces it with a new one, which would include any new features added during the update.

The next major aspect of “DBHandler” class which was implemented was the “addInfo” method, which, as the name suggests, allows information to be added to the database (Figure 28).

```
public void addInfo(int lesson, int complete){
    ContentValues values = new ContentValues();
    values.put(COLUMN_LESSON, lesson);//assign value for lesson to the lesson column
    values.put(COLUMN_COMPLETE, complete);//assign value for complete to complete column
    SQLiteDatabase db = getWritableDatabase();//initiate writing to the database
    db.insert(TABLE_NAME, null, values);//insert column values into the table
    db.close();
}//addInfo
```

Figure 28. DBHandler - addInfo Method

When the user chooses to save their progress during a lesson the “addInfo” method is called from the “Lesson” class where values for the current lesson number and a completion value are passed as parameters to the method. Within the method itself a “ContentValues” object is created which creates an empty set of values. The “put()” method of “ContentValues” class is then used to assign the “lesson” and “complete” parameters to the appropriate columns. A “getWriteableDatabase()” method is then called. This opens the database with the intention of writing data into the table. Following this, the database “Insert()” method is then used to pass the “ContentValues” to the database table. The

database is then closed in order to finalise writing to the data table, and to make sure that it is not needlessly running when the user does not need to make use of it.

```
public boolean findProgress(int lesson, int complete){
    int lessonNo = lesson;
    int completed = complete;
    SQLiteDatabase db = getReadableDatabase();
    // String variable to query database
    String query = "SELECT * FROM " + TABLE_NAME + " WHERE " + COLUMN_LESSON +
            "= '" + lessonNo + "' AND "
            + COLUMN_COMPLETE + "= '" + completed + "' ";
    Cursor cursor = db.rawQuery(query, null);

    //if curser moves to first then a value exists and returns true, else return false
    if (cursor.moveToFirst()){
        cursor.getInt(cursor.getColumnIndex(COLUMN_ID));
        entryFound = true;
        //return true;
    } else {
        entryFound = false;
        //return false;
    }//else
    cursor.close();
    db.close();
    return entryFound;
}//findProgress
```

Figure 29. DBHandler - findProgress Method

In order to implement the “findProgress” method (Figure 29) the database is opened as a readable database and an SQL query is created. The query selects all entries in the database table where the value in the “lesson” and “complete” columns is the same as the “lesson” and “complete” parameters passed in the call to the “findProgress” method. This query is then applied to a cursor object. An “if” statement is then used to determine whether or not the value being searched for exists in the table. It states that, if the cursor object moves to the first row in the table which fulfils the “SELECT” statement, then the appropriate entry exists and the method returns that an entry has been found. If the cursor does not “moveToFirst()” then a value for the search parameters does not exists and the return value will be false.

Lessons

The Lessons section of the application is the main focus and therefore the most substantial part of the project code. Since the first point of contact with the lesson section is the lesson menu, the implementation of this section will be discussed first.

LessonMenu Class

The Layout design for the Lesson Menu consists of six main selection buttons and a "Home" button to navigate back to the main menu. The "onCreate" method of the class loads the corresponding layout XML file and creates a new "DBHandler" object named "db". Once this is created the final operation in the "onCreate" is a call to a method called "showProgress()". This is the method which is used to display the user's lesson completion progress.

```
public void showProgress(){
    first = db.findProgress(1,1); //if "lesson=1" and "complete=1" exists "first=true"
    second = db.findProgress(2,1);
    third = db.findProgress(3,1);
    fourth = db.findProgress(4,1);
    fifth = db.findProgress(5,1);
    sixth = db.findProgress(6,1);
```

Figure 30. LessonMenu – read from the database

The "showProgress()" method firstly queries the database by assigning the true or false return value from the "findProgress" method in the "DBHandler" class to a set of Boolean variables representing each of the lessons. For example, as shown in Figure 30, the Boolean variable named "first", is assigned the return value of the "db.findProgress" method, where "lesson" is "1" and "complete" is "1". If, as stated previously, the value is found in the database "first" will be assigned the value "true".

```
if(first==true){ //if entry for lesson1 is found
    butt1.setBackgroundResource(R.drawable.trackedbutton);
    Log.d("Database Check","Lesson1 entry found" );
}//if
```

Figure 31. LessonMenu – showing progression

Using a set of "if" statements the method then sets the colour of the button to green if an entry has been found in the database. This was made possible by creating a Button object

for each of the buttons on screen. Following this, if the appropriate Boolean is found to be true, then the background resource is assigned to a custom XML layout in the "drawable" folder. The custom button background is the same shape and transparency as the default custom button design except that it is green (Figure 31). The result of this is that each time the user navigates to the lesson menu they will have a visual reminder of how far they have progressed through the lesson plan.

The "onClick" attribute for each of the buttons was assigned to the method called "chordsLessons()" which contained a switch statement, that depending on which button was pressed assigned a value to a String variable called "lessNum". For instance if the button to open lesson one was pressed the switch statement would take this as an argument and assign "lessNum" the value "Less1". The "Lesson" activity would then be started with the "lessNum" variable being passed as an Intent. This value would be used to generate an appropriate layout for the "Lesson" activity.

One problem which was encountered during testing was that if the user navigated back to the menu from the Lesson activity after saving their progress, the progress would not show straight away. Instead progress would only show after either leaving the app or going back to the main menu and re-entering the lesson menu. For this reason it was necessary to add an "onRestart" method to "refresh the page" each time the user navigated back to the lesson menu. This problem occurred as a result of the "launchMode" which was defined in the "AndroidManifest". This is discussed further in the "Manifest" section.

Lesson Class

The class' "onCreate" method first generates the default layout, as described in the design section. It then retrieves the Intent "lessonNum" which was passed from the lesson menu and assigns it to a String Variable named "lessonNumber". Following this, A soundPool object is created by calling a method called "createSoundPool()".

```
public void createSoundPool(){
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP){
        attributesBuild = new AudioAttributes.Builder();
        attributesBuild.setUsage(AudioAttributes.USAGE_MEDIA);
        attributesBuild.setContentType(AudioAttributes.CONTENT_TYPE_MUSIC);
        attributes = attributesBuild.build();

        soundPoolBuilder = new SoundPool.Builder();
        soundPoolBuilder.setAudioAttributes(attributes);
        soundPool = soundPoolBuilder.build();
    }else{
        soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
    }
}//createSoundPool method
```

Figure 32. Lesson – createSoundPool Method

Initially a soundPool was created simply by using a constructor which was a single line of code. Android Studio, however, indicated that the soundPool constructor is deprecated for devices running API level 21. Despite the fact that during testing the soundPool constructor ran properly on a device running API level 22, it was decided that it would be safer to properly anticipate any problems with devices running newer API versions by creating using a soundPool Builder class, as advised by the Android developers guide. According to the guide the Builder should be used for API 21 and above, while older versions should continue to use the constructor technique. For this reason it was necessary for the application to be able to determine which version of Android was running on a device and to create the soundPool based on that. To do this, the developer followed a Tutorial video on YouTube by the channel “clientuser.net” which indicated how to create a soundPool object, while accommodating for backwards compatibility. As shown in Figure 32, the method consists of an “if” statement which determines that if the current device’s build version is more than or equal to “Lollipop” which begins at API 21, then use the Builder method. Otherwise use the simple constructor to create the soundPool. With the Builder method it is necessary to first create an “AudioAttributes” object, as this is used to define the attributes of the soundPool.

Once the soundPool is created the next step of the “onCreate” method is a call to the method “LessonNav()”. This method simply consists of a switch statement based on the parameter “lessonNumber” which was defined by the Intent from the lesson menu. Using this information, the switch statement is used to set the value of a String variable named “picID”. This “picID” variable is a static variable and is used as an indicator of the current lesson page which the user is on. For example, when this method is called. If the user chose

to navigate to “lesson 1” then the switch statement defines “picID” as “Hand” due to the fact that the image used on the first page of lesson 1 is the image of a hand diagram.

The next step of the “onCreate” method is to call a method named “setDisplay()” which consists of a number of objects for each of the onscreen elements, meaning the buttons, imageViews, frameLayouts and textViews. This is then followed by a switch statement which consists of a case for each page of the lesson plan. The page is defined by the switch statement with the case being the “picID” variable. Using this, the layout elements are set programmatically for each page using the layout objects which were created in the method.

```
switch (picID){
    case "Hand": //picID determines which page
        chordDiagram.setImageResource(R.drawable.hand); //set image reseource
        chordDiagram.setClickable(false); //make image non clickable
        description.setText("Throughout these lessons you will be shown a number of ...");
        description.setTextSize(18);
        description.setScrollY(0); //make sure scrollable text always starts at the top
        description.setTextColor(Color.parseColor("#fcfafa"));
        description.setBackgroundResource(R.drawable.textbackground); //set text background
        backButton.setVisibility(View.INVISIBLE); //make back button invisible
        backButton.setClickable(false); //back button is not clickable
        saveBut.setVisibility(View.INVISIBLE); //make save button invisible
        tracker.setText("Lesson 1\nPart 1/4"); //set small textview to show lesson progression
        picLayout.setLayoutParams(new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
                0, 4F)); //set the layout weight of the image to 4
        textLayout.setLayoutParams(new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
                0, 6F)); //set the layout weight of the tex to 6
        break;
```

Figure 33. Lesson – setDisplay() Method example

Figure 33 for example shows how the layout for the first of page of lesson one is defined programmatically. As discussed previously, when the user chooses to start the first lesson the “LessonNav()” method assigns the value “Hand” to the variable “picID”. The switch statement in the “setDisplay()” method then uses this value as the case argument in order to determine the image and text resources, the size, colour and background, as well the characteristics of each of the buttons and layout parameters. In the example shown the back button and save button are invisible to the user as the save button only appears on the last page of each lesson, and the back button has no purpose as the user is on the first page of the lesson. If the page which is displayed has an image of a chord diagram, then the Mp3 file of that chord which is stored the “Raw” folder is loaded into the soundPool. Initially all sounds where loaded at the same time during the “onCreate” process. However, this slowed down the app slightly when the activity was opened. Therefore the sound file loading

process was divided across the switch statement cases in the “setDisplay()” method as described, which fixed the issue of any loading lag.

The last two operations of the “onCreate” method are the creation of a “DBHandler” object and a call to a “checkProgress()” method. The “checkProgress()” method operates in a similar way to the “showProgress()” method of the “lessonMenu” class by assigning the return value from the database’s “findProgress()” method to a Boolean value. This information is then employed when the user is on the last page of each lesson, by indicating whether or not the user has completed the lesson already. If they have, then they can navigate freely to the next lesson. However, if they have not, then they will receive a notification, in the form of a dialog object, to ask them if they want to save their progress or not. This notification is also used on the save button, which is only clickable if the lesson has not yet been passed. Otherwise, if it has, the “Done” button will be highlighted as green to provide a visual indication to the user that they have already passed the lesson. The notification dialog is implemented by making a call to a “showAlert()” class, when the user either chooses to press the “Done” button or the “Next Lesson” button.

```
if (view.equals(saveBut)) {
    confirmation.setMessage("This will mark the current lesson as completed\n" +
            "Are you sure you want to do this?")
            .setPositiveButton("YES", (dialog, which) → {
                    saveData(view); //if yes call saveData method
                })
            .setNegativeButton("NO", (dialog, which) → {
                    dialog.dismiss(); //if no close dialog
                })
            .setTitle("Lesson Complete!")
            .setIcon(R.drawable.pickicon)
            .create();
    confirmation.show();
}//if
```

Figure 34. Lesson – showAlert() Method example

As shown in Figure 34, when the user presses the “saveBut”, which on screen is the “Done” button, a dialogue is built. The dialogue in this case includes two options to the question as to whether the user is sure they want to save their progress, either yes or no. If the user chooses no then the dialog is simply closed. If they choose yes, then a method called “saveData()” is called in order to save to the database. The “Next lesson” button

works in a similar way expect that it has three options to choose from. The first is “Yes” which calls the “saveData()” method and uses a switch statement to assign the value of “picID” to the intended next page before calling the “setDisplay()” method. This means that they will save their progress as well as navigating to the next lesson. The second option is “Later” which allows the user to progress to the next lesson without saving their progress. The last is “Cancel” which simply closes the dialog and stays on the same page.

The “saveData()” method called by the dialog works by using a switch statement with the “picID” value as an argument to determine which lesson the user is currently on. Based on this value, “int” variables called “lesson” and “complete” will be assigned appropriate values and passed as parameters in a call to the “addInfo()” method of the “DBHandler” class, resulting in this information being saved to the database.

The class includes navigational methods for the back, next, home and menu buttons. The home and menu buttons are simply the operation of starting the appropriate activity which is intended to be navigated to, using Intents. The “Next” and “Back” buttons are assigned to methods “nextNav()” and “backNav()” with both operating in a similar method to the “lessonNav()” method. Meaning that they use switch statements to set the “picID” variable to the intended page that they want to navigate to, as will be determined by the “setDisplay()” method.

One final method within the class is the “imagePressed()” method. Within the lessons the user has the ability to press certain images, mostly the chord diagrams. This will then play the corresponding sound of the chord, or in the case of the final page of lesson 6, will enlarge the image and play the guitar scale from the image. This is done by using the “play()” method of the soundPool class (Figure 35), which plays the chosen sound file that has previously been loaded to the soundPool during the operation of the “setDisplay()” method.

```
switch (picID){
    case "AMaj"://current page
        soundPool.play(aMajor, 1, 1, 1, 0, 1);//sound played from soundpool
        break;
```

Figure 35. Lesson – play from soundPool

During testing it was noticed that, if the user navigated to another activity while a sound was playing, the sound continued until it was finished. It was felt that this gave an unprofessional feel to the application, and that all sounds should be contained to their appropriate activity. In order to fix this the “autoPause()” method of the soundPool class is called in any method which navigates away from the activity in order to stop any playing sound.

The sounds in this section were created in the Digital Audio Workstation program “Cubase” by creating a virtual MIDI instrument and adding an effect to replicate the sound of an acoustic guitar. This was then recorded as an Mp3 file in order to make the size of the file relatively small, in order to reduce loading time to the soundPool. These files were added to the projects “Raw” folder.

Tips and Info

The “tips and info” section consists of three activities, these are the TipsMenu, “TipsSubMenu” and the main activity called “InfoContent”. The first point of contact for the user is the main “TipsMenu”, the layout and operation of which is the same as the apps main menu. The difference being that the buttons have different text and link to different activities. The “Guitar Parts” and “Reading Tabs” buttons navigate straight to the “InfoContent” activity. The developer decided that the remaining two options would include a few different options within each which fall under the corresponding category. For this reason the “Care Guide” and “Playing Tips” buttons link to the “TipsSubMenu” activity which includes two subcategories. For the “Care Guide” section these categories were “Cleaning” and “Re-Stringing”. The “Playing Tips” category included “Playing” and “Avoiding Injury”. Each of these choices then start the “InfoContent” activity and pass the information on the chosen subject as an intent, in the form of a String.

The main focus of the “Tips and Info” is the “InfoContent” activity which opens and, using the Intent gathered from the previous menus, programmatically defines the content and layout of the page using a switch statement.

Tuner

The soundPool for the tuner was created in the same way as the soundPool in the "Lesson" class. Once the "onCreate" method calls the "createSoundPool()" method it moves on to another method called "loadSounds()" as the name suggests this loads the sound of each string from the "Raw" folder to the soundPool. The string samples which were used in this section were taken from free samples on "SoundCloud" from the user Ben Hillman. The original samples however were too long and faded out too much to be used in the tuner. Therefore the developer edited these sounds in order to make them last only two seconds each and have a very short fade out to avoid "cut-off". The developer chose to use real guitar sounds in the uncompressed WAV format as it was felt that it was important for these sounds to be as clear and as high quality as possible due to the nature of their use.

Each "string" button on screen calls the "stringPressed()" method which, using a switch statement, plays the appropriate sound file from the soundPool and assigns the name of the note being played to a String variable named "text". After the switch statement this String variable is then set as the text of a toast object, which is then shown using the "toast.show()" method.

Similarly to the "Lesson" activity it was found that both the appearance of the toast and the playing of the sound files would "bleed" over into other activities if the user navigated away. To avoid this "soundPool.autopause()" and "toast.cancel()" were added to the "homeNav()" method which navigates to the main menu and the Android back button method called "onBackPressed()".

Chords

The implementation of the chords section works similarly to the lessons section. A switch statement determines which button has been pressed in the menu and passes an Intent to the "SingleChord" activity when it is started. The "SingleChord" class then assigns this Intent to a String variable called "chordName" with which a method called "setDisplay()" is used to set the appropriate visual contents and load the correct sound file to the SoundPool. Within the activity, if a variation of a chord is available there will be a variation button on screen. Pressing this button calls the "variation()" the operation of which uses the current value of the "chordName" variable to define defines the current page. A switch statement is then

used to reassign the value of “chordName” to the chord for the new page which the user wants to navigate to. The “setDisplay()” method is then called, which updates the page layout now that “chordName” has been changed. As with previous activities the developer made sure that any playing files were stopped when navigating away from the “SingleChord” activity.

Manifest

During the development process, testing revealed that navigation and creating new activities in the app did not function as was desired. What is meant by this is that, each time the user would press a button to navigate to a particular activity, an activity task was created. This particular aspect was not the problem. The problem occurred when the user would enter the same activity several times in one app session, in this scenario if the user then pressed the Android back button, they would be taken back through the same activity multiple times. As an example of this, if the user was on the lesson menu and chose lesson one, then pressed the button to return to the menu, then entered lesson one again. When they pressed the Android back button they would move back from Lesson one to the lesson menu and again to lesson one on to the lesson menu. Investigation of the issue using the Android developers guide indicated that this was due to the fact that Android stores all activities which are opened in what they refer to as a “Back Stack” in the order in which they are opened. Figure 36 which is from the developer’s guide indicates why the problem occurred. The way in which the developer preferred the back stack to work is that each time the user opens an activity which is already in the stack, the pre existing instance of the activity should be pulled from the bottom to the top of the stack instead of being duplicated.

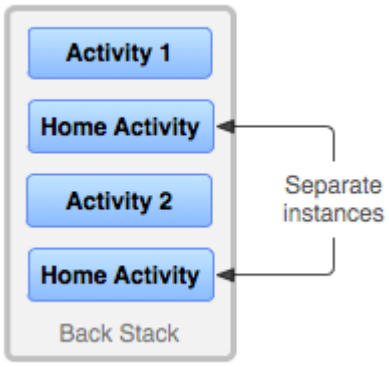


Figure 36. Back Stack

The developer’s guide indicated that this could be achieved by defining the launch mode of each activity in Manifest file. Following this advice the developer defined each of the activities launch modes as “singleTask” due to this operation being the mode which fitted the desired organisation style of the back stack. As discussed earlier, one problem which occurred as a result of this change was that the “lessonMenu” activity did not show progress straight away if the user saved to the database before moving back to the menu. This was

because the previous instance of the lesson menu moves to the top of the stack and does not go through the “onCreate” method which would call the “showProgress()” method. To fix this the “onRestart()” method was added, which is automatically called when the existing instance of the activity is called to the top of the stack. This method operates with the same arguments as the “onCreate()” method, in order to “refresh” the activity.

Due to the layout being designed specifically for a portrait view, the developer did not want the app to be able to turned into landscape view, as in some cases this may interfere with elements of the layout. For this reason each activity’s “screenOrientation” attribute is defined in the manifest as “portrait”.

Resources

The application makes use of a number of text, sound and image resources. These are stored within the resources folder of the application with images in the “drawable” folder, the audio being stored in the “Raw” folder and a String resource XML being stored in the “values folder.

The resources in the drawable folder mostly consist of images, the majority of which are chord diagrams. These chord diagrams and several other image resources were created using the image editing software “Photoshop CS2”. As well as these images there are also some custom layout XML documents inside. These include a custom button layout for both the default button state and the “progress completed” state. As well as these, background styles for textView objects and a custom style for a textView scrollbar are also included. The textView background and scrollbar colour are simple and consists solely of a colour and corner radius statement. The buttons are similar except that it was deemed necessary, for user feedback purposes, to provide two layouts within each button. One for when the button is pressed and one for its default state. Since the colour chosen for the default style was semi-transparent, it was felt that a good way to indicate the pressed state of the button was to make the colour solid, and slightly darker.

As discussed earlier the Raw folder includes sound files which represent each chord within the app and each guitar string. All of these were created and/or edited using the “Cubase LE 5” digital audio workstation program.

The “strings.xml” document was used to provide the text resources for large bodies of text. This was done simply so that long paragraphs of text were not hard coded within the class, which would provide a much more unorganised appearance, making testing and debugging more difficult.

As well as the above mentioned, a set of small app logos was created by the developer and stored within the “mipmap” folder. This image is then set as the resource for the attribute “icon” in the Android manifest. This means that this image is used to represent the app on the user’s Android app menu as well as the preview window of the app in the “recently used” menu in Android. It is felt that this gives the app a much more polished and professional look, especially since the image is an original creation instead of a stock image.

Testing and Evaluation

Testing

Functionality testing was carried out by the developer on each feature as they were implemented before moving on to the next. This meant that any problems which occurred could be located within the code much more easily and addressed before moving forward. Development of the app layout was created using the device preview mode on a virtual “Nexus 5” device with a 5 inch display within Android studio. The developer, however continuously previewed the application in different sized devices throughout the project to make sure that layout consistency was maintained over several devices. The main testing was carried out on a device with a 4.5 inch display, running API 22, which translates as Android version 5.1 “Lollipop”.

As well as the continuous functionality testing as described above, focus group testing was also undertaken at key stages within the development process. The main reason for this is that, due to the developer’s continuous contact with the project some problems may get overlooked, it was therefore important to get feedback from an a focus group. This is because they will have had little to no previous interaction with the product and could provide more objective feedback. The focus group consisted of people who are experienced guitar players, people who have given up playing guitar, beginner guitarists and some who

have no experience in playing guitar at all. This range of users, it was felt, gave the widest perspective on the effectiveness of the application.

An example of this is from the development of the tuner activity. The developer tested the functionality of this page by pressing the buttons in a number of different ways. These included letting the sound files and toast notifications play out completely, cutting sound files off by interrupting them with back navigation and pressing other buttons and finally by pressing buttons multiple times in rapid succession. However, since the developers focus during testing of this activity was directed toward the functionality of the tuner, the “home” buttons functionality was overlooked. This was something that was highlighted by a focus group member and was due to the fact that the “onClick” attribute of the button had not been assigned to the “homeNav()” method. User testing also highlighted a few minor problems such as spelling errors in the text section on one of the activities. Since most of the testing was carried out and addressed during implementation by the developer, there were few functionality problems which were discovered by the focus group.

As mentioned previously, during the development of the application testing took place after each feature was implemented. This was done by running a debuggable version of the application from android studio on a physical device. Usually if any features did not perform as desired the developer would focus on the appropriate section of the code to pinpoint and fix the problem. On some occasions where more serious errors occurred, such as application crashes, the logcat monitoring screen in Android Studio was used to identify the reason for the failure and the particular line of code which caused the crash. In most cases, since the logcat identified the problematic line of code these problems were easily identifiable and fixed. However on one or two occasions where it was not obvious to the developer as to why the particular section of code was causing problems, it was necessary to debug the application. What this meant was that the developer ran the app in “debug” mode using breakpoints in the code. This method of testing means that the application runs until it hits one of these developer defined breakpoints. Once this happens the debug screen provides information on the values of variables at that stage of the code.

For example, using this method of testing was helpful when it came to first implementing the soundPool object in the lessons section. When the chord diagrams were pressed in the

lesson the “Lesson” activity would crash with a nullpointerexception error, if the first page of the lesson included a chord diagram. This means that the application was attempting to use an object which had a null value. Due to the failure point, which was the attempt to load sound files to the soundPool, it was evident that the soundPool object had not been initialised. However, since it appeared that the soundPool was initialised in the “onCreate” method it was not evident to the developer why this problem occurred. Using multiple breakpoints and debugging, it was possible to see the value of the soundPool and other variable at different stages of the code. The result of this is that the developer discovered that the “onCreate” method called the “lessonNav()” and setDisplay()” method before initialising the soundPool object. Since the setDisplay() method loads the sound files of each chord, depending on the page, the code was attempting to load files to the soundPool before the soundPool had been created. This problem was not at first understood due to the fact that in lesson one the first page does not load a sound file but instead sets the display and carries out the rest of the “onCreate” method, meaning that, as long as the first page which is opened did not include a chord diagram, then all sounds would load and play as intended.

Once all features of an activity had been implemented the developer would then test the activity as a whole by using all features and pressing different combinations of navigational buttons, adding database entries and checking that the app added, and read from the database for each attempted entry. As well viewing this on the physical device, the developer also added several “log.d()” methods which allowed the developer to provide feedback in the logcat screen at several key stages of runtime. For example, each time progress was saved to the database, a message would be displayed onscreen to indicate that an entry for the completion of a particular lesson had been added to the database. This was particularly useful for “background components” such as the database and the soundPool, as evidence of their correct execution may were not always clear from the perspective of the device screen. This allowed the developer to keep better track of their operation.

Following the completion of all features of the application it was important to test the application again, this time as a whole, using the same methods as before. Once the

application was considered to be without problems and was in good working order it was then finalised by generating a final build APK.

Evaluation

On completion of the final application it is important to assess how effectively the application adheres to the functional and non-functional requirements which were outlined in the beginning stages of the project.

Adherence to Functional Requirements

- **The app should allow tracking of user progression.**

The feature of user progression has been implemented in the app by displaying each of the buttons in the lesson menu in green if the corresponding lesson has been completed. Within the last page of each lesson the "done" button also appears green and is no longer selectable if the user has previously completed the lesson.

- **The app must provide lessons and information which are factually correct**.

The developer of the application has extensive guitar playing experience and can testify that the information provided within the app is factually correct and at an appropriate level for beginner guitar players. Despite this, the information was further confirmed to be factually correct by consulting the book "Totally Guitar, A Definitive Guide" (Bacon and Hunter, 2004).

- **The app must allow users to navigate to different sections of information within the app as they please instead of being restricted to a lesson plan. This includes allowing users to access chords from the lessons which have been previously completed.**

The app includes a separate section specifically for chords and information section and is not solely reliant on the lesson plan. Since the lessons are made of a number of chords ordered by their difficulty to play, it may be irritating to the user to try and find a particular chord within a lesson that they have completed before. For this reason the chord section allows the user to choose the chord they want to review directly from the menu, with each page including major and minor versions of most of the chords.

- **The app must provide information in a number of different, clearly defined sections.**

As stated in the previous point, the application consists of a number of different sections. These are easily accessible from the main menu which consists of four main buttons, from which the user navigates to sub menus and onwards to their desired page. Each menu is titled with buttons which have their destination clearly outlined in white text on a translucent black background.

Adherence to Non-Functional Requirements

- **The interface should be aesthetically pleasing to help encourage continued usage**.

The application design was created with the intention of being aesthetically pleasing, while retaining its functionality and clearly defined sections. The developer feels like the theme of the application looks good and is appropriate to the content of the application due to the fact that the background and colour schemes are based on the colours and styles of an acoustic guitar. The developer consulted with members of a focus group who agreed that the application interface design was aesthetically pleasing and the colour scheme looked quite professional.

- **The application should have a level of interactivity outside of just navigation in order to encourage engagement**.

In order to adhere to this requirement the developer created the lesson section with a number of chords, which when a button was pressed changed the finger positions of a chord diagram onscreen. As well as this the tuner component of the app requires user interaction to play each of the strings. At the prototype stage, consultation with the project supervisor and a focus group suggestion that there should be more user interactivity within the lesson section of the application. To address this the developer added more content to the lessons, with text sections being scrollable, the ability to listen to chord examples by pressing the diagrams, and the ability to zoom in and listen to the scale diagram on the final page of lesson 6.

- **User interactivity with the app should be straightforward and logical**.
- **All buttons and links on screen should have an appropriate related action performed.**
- **All section headers should be clear and descriptive in order to make navigation as easy as possible**

Buttons and layouts are easy to navigate and clearly defined. Each page of the application is included within its appropriate section with all navigational buttons operating as defined by the button text. The effectiveness of these elements was observed through focus group testing, during which no users had any problems in understanding how to navigate throughout the application.

- **User progression should be stored for each individual user with the help of a database**.

The DBHandler class which was created within the app has allows the user to store their progression for each individual lesson, while also allowing the database to be read in order to provide feedback to the user on their progression.

- **The app must be able to run on multiple sized devices while maintaining a consistent layout.**

The app was developed with and for mobile sized devices of varying sizes. The intended usage of the application is on mobile devices with screen display sizes between four and six inches. The consistency of the layout between these sized devices is maintained. On larger devices the layout is consistent in each element's proportions on the screen, however, the display differs with regard to textView elements, since there is more room on screen more text is shown without the need to scroll.

- **All diagrams, especially the chord diagrams, should be large enough so that they can be read easily and without confusion.**

The diagrams on the screen take up quite a large proportion of the display. In regards to the chord diagrams, this proportion of the layout is enough to be able to read the diagrams clearly, helped by the fact that the diagrams are fairly simple to understand, with the finger position elements being a bright red colour, making them stand out. The only diagram which hit was felt needed to be given a larger proportion of screen space was the scale diagram.

The reason being, that the user is required to read the tablature on the screen. For this reason the user can press the image to reveal a zoomed view of the diagram which fills the screen. Focus group testing revealed that users also believed that the diagrams wear easy to read and simple enough so that they were not confusing.

Focus Group Analysis

As indicated in the previous section, as well the developer own personal analysis of the application the focus group was also consulted with the intention of providing feedback on the effectiveness of the final application. The focus group analysis consisted of face to face discussions with potential users while an application demonstration was presented to them. Following this the users were given a hands on look at the app. A summary of the feedback and comments received while using the application are displayed in Table 3.

Area of analysis	General Response	Additional Comments
Is the Aesthetic Design of the application attractive?	Yes	None
Are sections within the application clearly defined and understandable?	Yes	None
Did you have any problems understanding how to navigate to particular sections?	No	None
Do the lessons seem like an effective resource for a beginner guitarist?	Yes	It would be good to provide more advanced lessons for slightly more experienced guitarists.
Do the chords section and the chord diagrams throughout the application provide an effective way to help learn chords?	Yes	It would be good to provide a lot more variations of each chord as well as just the major and minor variations.
Does the tuner provide a good tool for the user to be able to tune their guitar?	Yes	1) It would be good to be able to have multiple tuning combinations other than standard tuning 2) It would be useful for the application to be able to listen to the user and provide real time feedback during tuning.

Does the level progression indication feature seem useful?	Yes	The use could be seen in this feature particularly if a user spends some time away from the app, in order to remember were they left off in the lessons.
Is any of the information within the application wrong or misleading?	No	There are one or two grammatical errors, but the content of the information seems to be correct.
If you were/are a beginner guitarists do you think that using this application would help with the learning process?	Yes	None

Table 3. User Feedback Summary

The feedback gained from the user interaction was positive and the general consensus is that the application in its current form is an effective tool for a beginner guitarist. Since this is the target audience this is a pleasing result. In addition to the consensus there were some additional comment about possible improvements to the system. Apart from some small grammatical text errors, which were then addressed, it was agreed that the suggestions are more focused toward users who would be a bit more experienced in playing guitar. Another was the inclusion of a tuner which listens and provides frequency analysis and feedback to the user during tuning. While this is a feature that the developer would have liked to include from the start, the implementation of such a feature is quite complicated. For this reason the implementation of these suggested features would be included in possible future updates to the application.

Due to the results obtained from user feedback and the analysis of the application's adherence to functional and non-functional requirements the developer feels that the application is successful in delivering upon its intended purpose as outlined in the beginning of the project.

Conclusions

At the outset of this project the developer chose to create a guitar tutor application. In the beginning it was not known what form this would take. In order to come up with an initial idea for the application the developer carried out research into the area of existing guitar tutor applications. This process allowed the developer to see the pros and cons of

existing applications, and helped with the ability to make informed decisions on the design of the proposed application. This research process also confirmed the choice of using the Android platform due to its dominance in the smartphone market. This was important due to the fact that an app like this should theoretically try to target as many users as possible.

Based on the research conducted the developer was able to draw up a list of requirements which the application needed to include. Based on these an initial design for the prototype version of the application was created. Using this outline the developer started implementation of the prototype.

Due to the relatively little amount of experience in Android app programming at the beginning of the project a lot of the implementation stage was spent learning how to use the Android studio program as well as the methods used in order to create the features within the application. The first stage of implementation was the layout for most of the activities. This was very much a trial and error process at the start due to the fact that many of the layout features added did not appear as they should have, this meant that further time had to be spend researching as to why these inconsistencies were occurring, which usually resulted in some small layout feature being overlooked due to the inexperience of the developer.

Once the activity layouts had been created the developer focused on the java class files for each, this process was relatively easier to get to grips with due to the fact that the developer has had prior experience in java coding. Although on occasion the development of these classes ran in to speed bumps, particularly with imported objects like the soundPool and implementation of the database. Features like these also needed some time to research and learn their implementation process through tutorials and official documentation.

At the midpoint of the implementation stage the developer produced the prototype version of the application. This was a framework of what would become the final application and helped with the next stage of development by allowing the developer to gain feedback from the project supervisor and potential users, providing the opportunity to create a much more effective end product.

Following the prototype development stage, it was possible to progress towards the final application build by adding to or fixing existing features of the application based on the feedback gained. Some aspects of the code during the prototype stage, while they performing as intended were created in an overcomplicated or “roundabout” way. This was due to the fact that the main focus during development of the prototype was on creating a working framework. Once all main features of the application had been developed, this gave the developer the opportunity to re-assess aspects of the code and rewrite them in a more “streamlined” fashion.

With consideration of the overall development process it is felt that if the developer were to begin the project with the skills and experience which have now been gained that development may have been a lot faster and therefore more efficient. This was particularly evident due to the fact that the implementation of the application became a lot easier and faster in the later stages of development. This could possibly leave open opportunities with regards to the implementation. Meaning that additional content within the lessons and information sections as well as some of the features which were suggested as possible future improvements to the existing application could possibly be added. These include the addition of a large amount of additional chords, a larger number of lesson plans for more practiced guitarists and a more advanced tuner feature which could respond to frequency analysis of audio input from the user. As it stands however, these features, are considered to be additions which could be implemented in potential future software updates.

Despite these elements not being included in the current build, the developer considers the application to be a success in terms of providing content which is appropriate and helpful to a beginner guitarist and, due to the development platform, is readily available to users at any time. As well as this the application adheres to the functional and non-functional requirements which were determined before the design of application. This was possible by keeping these requirements in mind throughout the development process. This sentiment seems to have been confirmed in the analysis stage by feedback gained from the focus group discussions.

References

About SQLite

Retrieved 17th June 2016, from

https://www.sqlite.org/about.html

Acoustic Guitar [Image]

Retrieved July 21st 2016, from

http://pngimg.com/download/3361

Agile Alliance 2015 *12 Principles behind the Agile Manifesto*

Retrieved on 13th June 2016, from

https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Android app development for beginners 2015, YouTube Playlist, thenewboston

Retrieved on April 30th 2016, from

https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl

Android developers guide

Retrieved 25th May 2016, from

https://developer.android.com/training/index.html

Android Tutorial #23: SoundPool (2/3); Backwards compatibility API pre & post lollipop,

2015, YouTube Video, clientuser.net

Retrieved July 16th 2016, from

https://www.youtube.com/watch?v=byNOLwmzNz0

Bacon, T. & Hunter, D. 2004, *Totally Guitar – A Definitive Guide*, Backbeat Books, London

Black and White Guitar [Image]

Retrieved on 7th August, from

https://pixabay.com/en/black-and-white-guitar-playing-1010818/

Blank Chord Chart [Image]

Retrieved Jan 26th, 2016, from

http://frasermurray.org/wp-content/uploads/2014/04/Blank-Chord-Chart.pdf

Constine, J. 2015, *Fender Goes Digital So You Don't Quit Guitar.*

Retrieved on Jan 4th, 2016, from

http://techcrunch.com/2015/09/10/software-is-eating-rocknroll/

Eriksson, U. 2012, *Functional vs Non Function Requirements*.

Retrieved Jan 17th, 2016, from

http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

Guitar Pick [Image].

Retrieved Jan 26th 2016, from

https://thenounproject.com/term/guitar-pick/17204/

Hand [Image]

Retrieved July 10th 2016, from

http://newpng.com/download/13644

Gove, J 2016, *Principles of Mobile App Design: Engage Users and Drive Conversation*

Retrieved on April 16th, 2016, from

https://www.thinkwithgoogle.com/articles/principles-of-mobile-app-design-introduction.html

Guitar Player [Image]

Retrieved August 7th 2016, from

https://www.flickr.com/photos/rockmixer/3108333037

Guitar Sounds 2015, Audio Samples, slackerstuff, Ben Hillman

Retrieved June 9th 2016, from

https://soundcloud.com/slackerstuff

Lomas, N. 2015, *Android Now Has 1.4 Billion 30-Day Active Users Globally.*

Retrieved Jan 14th, 2016, from

http://techcrunch.com/2015/09/29/android-now-has-1-4bn-30-day-active-devices-globally/

James Tyler Variax [Image]

Retrieved on July 21st 2016, from

http://line6.com/guitars/

Java Programming with Android Studio 2014, YouTube Playlist, John Tapley

Retrieved on May 4th 2016, from

https://www.youtube.com/playlist?list=PLB--808VLJNYLPy8CHXiEXrRTLiL3B0l*1*

Matthies, A. 2009, *Staying motivated and dedicated when learning guitar.*

Retrieved Jan 9th, 2016, from

https://aaronmatthies.wordpress.com/2009/03/28/staying-motivated-and-dedicated-when-learning-guitar/

Mazzocchi, A. 2015, Why *Students Really Quit Their Music Instruments (and How Parents Can Prevent It).*
Retrieved Jan 9th, 2016, from
http://www.nafme.org/why-students-really-quit-their-musical-instrument-and-how-parents-can-prevent-it/

The Most Popular Musical Instruments.
Retrieved on Jan 4th, 2016, from
https://indigoboom.com/most-popular-musical-instruments/

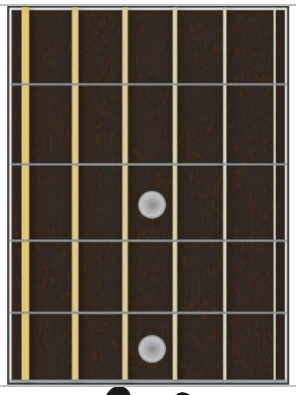
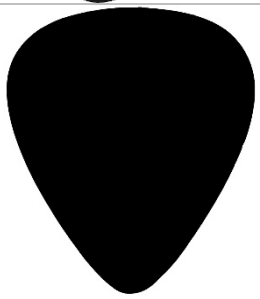
Painful Fingertips in New Guitarists – A Temporary Thing, 2013
Retrieved on Jan 9th 2016, from
http://www.guitarsavvy.co.uk/index.php/guitar-basics/painful-fingertips-in-new-guitarists/

The UK is now a smartphone society, 2015.
Retrieved Jan 12th, 2016, from
http://media.ofcom.org.uk/news/2015/cmr-uk-2015/

Wankel, L. & Blessinger, P. 2013 *Increasing Student Engagement and Retention Using Mobile Applications: Smartphones, Skype and Texting Technologies*,
Emerald Group, Bingley

Waters, Kelly 2007*, 10 key principles of agile development*
Retrieved 10th May 2016, from
http://www.allaboutagile.com/what-is-agile-10-key-principles/

Appendix 1 – In-App Sourced Images

IMAGE	REFERENCE	USAGE
	(*Guitar Player*[Image])	Tips + Info: Playing
	(*Blank Chord Chart* [Image])	Lessons, Chords and Tuner Section
	*(Hand* [Image]*)*	Lessons: Lesson 1, Part1
	(*Guitar Pick*[Image])	Application Logo

	(*Black and White Guitar* [Image])	Lessons: Lesson 1 Part 4
	(*Acoustic Guitar*[Image])	Tips + Info: Guitar Parts
	(*James Tyler Variax* [Image])	Tips + Info: Guitar Parts