

Guitar Tutor Application

A dissertation submitted in partial fulfilment of The requirement for the
degree of MASTER OF SCIENCE in Software Development in The
Queen's University of Belfast

By 'Colm Maguire '

13/09/17

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER
SCIENCE**

CSC7057 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook?
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via Subversion.
7. Journal has been submitted.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism -

<http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/> - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used. I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS) _____

Student Number: _____

Student's signature _____

Date of submission _____

Abstract

Name: Colm Maguire



This document is a report of my final project for a Master's degree in Computer Science. The project I undertook was to create an Android application that would be helpful to user's learning to play the guitar for the first time. The skills I utilised during this project were, self-determination, time management, independent study and resourcefulness. The project also tested my organisational skills and discipline as well as my ability to deal with setbacks and problems over the course of the project.

Contents page	Page number
Introduction	1
Chapter 1: Problem Specification	2
1.1 Problem Specification	2
- 1.1.1 General problems with learning guitar	2
- 1.1.2 Identification of problems and analysis of current solutions	2
- 1.1.3 Problems identified with current systems	5
- 1.1.4 Conclusion	5
Chapter 2: Proposed Solution & Development Strategy	6
2.1 Proposed Solution	6
- 2.1.1 Sources of information	6
- 2.1.2 Chord library	6
- 2.1.3 Guitar Tuner	7
- 2.2.4 Chord changes	7
- 2.2.5 Finger placements	8
- 2.2.6 Feedback	8
- 2.2.7 User account	9
- 2.1.8 Features of proposed solution	9
2.2 Development strategy	10
- 2.2.1 Potential development methods	10
- 2.2.2 Pros and cons of development methods	12
- 2.2.3 Conclusion	13
Chapter 3: Requirements Analysis and Specification	14
3.1 Requirements Process	14
3.2 Requirements Analysis	14
3.3 Function Definitions	16

Chapter 4: Design	21
4.1 User Interface Design	21
- 4.1.1 UI Design principles and issues	21
- 4.1.2 Concept layouts	22
- 4.1.3 Colour scheme concepts	23
- 4.1.2 Colour palate:	24
- 4.1.3 Application theme	24
- 4.1.2 Layout templates	25
- 4.1.3 Optimisation/ dimensions	26
- 4.1.4 UI justification	27
4.2 Software System Design	33
- 4.2.1 System design	33
- 4.2.2 General android lifecycle diagram	33
- 4.2.3 Interface Diagram	34
Chapter 5: Implementation	35
- 5.1.1 General layout	35
- 5.1.2 Chord library	35
- 5.1.3 Tutorial sections	38
- 5.1.4 Android Manifest	44
- 5.1.5 Navigation	45
- 5.1.6 Finger placement section	45
- 5.1.7 Database	46
- 5.1.8 PHP code	48
- 5.1.9 Register function	49
- 5.1.10 Login function	51
- 5.1.11 Logout function	54
- 5.1.12 Achievements feature	55
- 5.1.13 Chromatic tuner feature	56
5.2 Testing specification and justification	57
- 5.2.1 Requirements testing	57
- 5.2.2 Testing process	57
- 5.2.2 Java Unit testing	60

Chapter 6: Evaluation and Conclusion	61
- 6.1 Introduction	61
- 6.2 Evaluation of objectives	61
- 6.3 Evaluation of project management	61
- 6.4 Areas that could be improved	62
- 6.5 Closing comments	
Chapter 7: References/ Bibliography	63
Chapter 8: Appendices	64
-8.1 Login class code	64
- 8.2 Chord transition class code	67
- 8.3 Definitions, acronyms, abbreviations	75

Introduction

This document aims to explain the design process of the Guitar tutor application. It contains information specific to the purpose of the application, the design process and the various functions and interfaces used to achieve the app's intended functionality.

The opening chapter acts as the problem specification, setting out the various problems that aim to be addressed in the developed software. Identified are problems on general with learning guitar and the limitations and issues with current systems.

Chapter 2 explains the proposed solution to the problems outlined in the problem specification, with particular attention given to features that would prove beneficial to beginner guitarists. The features identified here acted as the starting point for the application and provide the basis for the sections which are present in the finished product. This part of the document also justifies the proposed solution to the problem as well as highlighting and explaining the chosen design strategy used for developing the project software.

Chapter 3 of the document deals with the user requirements which were identified as essential for the successful implementation of the proposed solution from chapter 2. This chapter also contains a list of the various methods and functions used in the project.

Chapter 4 highlights the User Interface Design of the application with specifics features of the app and their instructional benefits explained. Also included are the various design details and prototype user interface designs and colour schemes as well as the finalised versions used in the finished product. The features the use can interact with are also explained in some detail here. Diagrams explaining the system design are also present in this section.

Chapter 5 details the implementation of the code used to provide the main features of the application. Here specific attention is given to how the main features of the app were implemented in the java code as well as explanation of important methods and java classes. Screenshots of important pieces of code are used to explain particular features and functions of the app. Significant attention is given to the login and register features and the app as well as the feature which allows users to record and send audio.

The final chapter is a concluding evaluation of the work on the project including an analysis of the success of the completed software in relation to the requirements specification. This chapter also highlights the areas in which the project could have been improved or further developed. Following the conclusion is a bibliography of the reading materials utilised during and in preparation for the project. Finally, an appendix is included of certain sections of code that relate to passages describing the implementation of features in chapter 5 as well as a table of terms, abbreviations and definitions.

Chapter 1: Problem Specification

1.1 Problem specification

In relation to learning to play the guitar there exist several problems which could be potentially off-putting to someone interested in picking up the instrument. Perhaps one of the most off-putting aspects of learning guitar is the number of hours the learner potentially needs to set aside to allow them to develop their skills. This can be a particularly daunting prospect to someone who may already have a limited amount of time due to their job, university or other commitments. Two of the most common methods to learning the guitar amongst beginners would be to practice in one's own spare time, perhaps by using a guide book or instructional videos or to attend lessons with a guitar teacher. Both of these have advantages and disadvantages for someone learning to play.

1.1.1 General problems with learning guitar

The main problem with learning from a tutor is that this can add to the problem of time-consumption, as the prospective player will first have to search to find a suitable tutor, set aside a specific time to attend the lessons, and part with money to pay the tutor. This could be particularly off-putting if the prospective learner already has limited free time as they may be unwilling to sacrifice anymore or change their schedule to suit attending lessons. However, this method provides the significant benefits of giving the learner a structured program for learning to play all while being guided by an experienced and qualified individual, as well as learning in a controlled environment. Learning with a tutor present also has the significant advantage of the tutor being able provide advice and help to the learner as well as the learner being able to ask the tutor directly about something they may be having difficulty with.

1.1.2 Identification of problems and analysis of current solutions

To someone with limited free time or a busy schedule learning via apps and instructional videos may be preferable option for learning to play guitar but is also not without its shortcomings. The main disadvantage of learning through apps and instructional videos being the lack of structure and guidance available to help learn. Without preforming a consistent routine of practice and exercises to help develop a player's learning, their progress may stagnate. Trying to learn on one's own also can be problematic as, if a learner has nothing to focus their learning they may lose motivation and interest. Another significant disadvantage compared to learning from a tutor is that the player can't ask for help when experiencing difficulty and doesn't have someone to guide them through the learning process or prevent them from falling into any bad habits. Also using online resources and mobile applications can also be negative once the user reaches a certain point as when a player is familiar with the basics of the guitar these resources often have little left to teach them and their progress with the can instrument plateau. However, a strong advantage of learning on one's own is that numerous

resources are available to someone learning to play guitar which can make the process easier, such as instructional videos and various mobile applications, though these resources can vary greatly in quality with the more useful teaching services for example Yousician usually requiring payment. Another significant advantage to being self-taught is that the player can learn at a pace that suits themselves which will make the process of learning more enjoyable to them as it is more like a hobby rather than an obligation.

Based on the above analysis, an effective solution to the problem of learning guitar would encompass the best aspects of both taught and individual schools of learning with the user being able to learn at their own pace but also having access to someone who can provide advice and help when the user is experiencing difficulty. Preferably the solution should also expand on the basics of learning the guitar so that intermediate level players are also catered for.

- Learning chords

In terms of actually learning to play the instrument there are a number of further difficulties for beginner players. One of the fundamentals of playing guitar is learning the various chords which are used to play songs. This is in part due to the sheer number of chords and their variations present. This can be complicated for beginner players as they aren't likely to know which chords are going to be more beneficial for them to focus on. In many current guitar learning apps and resources like GuitarTuna this is further complicated by the apps seemingly including as many chords and chord variations as possible, many of which are too complex and obscure for a novice to learn. It would be more beneficial to a learner to focus on learning the more commonly used chords rather than over-exposing them to chords that they aren't likely to use until they are more experienced and are too technically difficult for a beginner to play.

- Finger placement

Another significant point of difficulty when learning the chords as a beginner is knowing where to place their fingers to make the correct chord shape. Often in current resources for learning guitar, this leaves a lot to be desired as many instructional websites etc. tend to display the chord shape to the user but not describe which fingers need to be used and the preferred order in which the player's fingers should be placed to make the chord. This is an important feature to have as it will aid the player's development further down the line when they progress to learning chord transitions and move on to basic songs, as having an idea of where to place their fingers and in which order will enable them to make chord progressions effectively. A useful and fairly basic feature that tends to be overlooked in many guitar tutorial resources is that though they will visually display a chord to the user, they won't provide the user with the option to hear how it sounds. Including the ability to hear how the chord should sound when played correctly is a crucial piece of knowledge for someone learning for the first time as they can then compare the correct chord sound with their own attempt.

- Tuning

Another common problem for a beginner guitarist is keeping their instrument in tune. This can be particularly difficult if the guitarist doesn't have access to a tuner or isn't around someone who is experienced enough to tune the instrument for them. This is another feature that many current instruction applications overlook but is a crucial part of learning to play correctly as, before attempting to learn the notes or chords the player must first be familiar with how the guitar is supposed to sound when it's in tune and how to tune it themselves. This is a feature which is also overlooked in some current guitar tutor applications which tend to focus more on the learning aspect and don't include a tuner, so users are forced to download both an app that teaches guitar and an app specifically for tuning their guitar. The alternate tends to be an app like GuitarTuna which acts foremost as a guitar tuner but also has limited tutorial based features.

- Learning songs

Another difficulty when learning guitar for the first time is that when the user is familiar enough with the instrument to play basic chords, is using these skills to progress to learn chord changes and then basic songs. Most current apps such as GuitarTuna and iJam will go as far as teaching the user the various chords but fall short of teaching a sequence of chords or basic songs. The few that do go as far as teaching basic songs or chord progressions do so in way that the beginner is likely to find difficult to grasp. For example, many resources such as UltimateGuitar use static images of the chords that neglect to specify the necessary finger positions. Other apps also lack a count in feature or a metronome to help the user keep in time. A helpful alternate should be that in teaching a chord progression the user is displayed the visual interpretation of the chord including finger placements etc. along with an audio backing track to attempt to play along with. This audio track should also include a drum beat or a metronome click to help the user keep time.

- General problems with current applications

In general terms a failing of many guitar learning apps like GuitarTuna, Yousician and iJam is that many employ a dark interface which can be off-putting to the user as well as distracting. This is particularly evident when these apps are trying to convey multiple pieces of information at once, as the number of different components used on screens against the dark backdrop combine to make the screen appear cluttered and therefore more visually unappealing and difficult for the user to learn. This problem is worsened in apps like iJam which also include advertising that pops up at intervals and other free apps which ask for users to rate their experience. Another failing of many apps is that outside of chord memorisation games, they offer little opportunity for users to learn when they don't have their guitar, for example on the train to work etc. A helpful solution to the problem of learning guitar would capitalise more on the fact that a learner guitarist can use that app anywhere and that they may not always have their guitar with them, for example if they are commuting.

1.1.3 Problems identified with current systems

- Little to no opportunity for users to ask questions/ receive feedback
- Tutorials lacking visual components
- Generally using static images to teach
- Lack of audio utilisation
- Dark, often cluttered interfaces, with too many components and advertising
- Overabundance of chords which are too complicated for beginners
- Little opportunity for use when user without their instrument
- Few tackle all the identified “main” problems associated with learning guitar
- Lack incremental teaching

1.1.4 Conclusion

In conclusion the identified problem that will provide the basis of the project is creating a software system that will be helpful for someone learning guitar. The software should cater to all the fundamental needs for someone learning guitar and address the common difficulties mentioned above as well as the shortcomings identified in current applications available. These include having an effective way of teaching the basic chords to the user in a concise and clear manner, having a feature that helps the user to tune their guitar and teaches the user basic chord progressions and songs. The solution should also have features that a learner can make use of even when they don't have their guitar with them.

Chapter 2: Proposed solution and justification of the development model

2.1 Proposed solution

To reiterate the above, the proposed solution for the problem of learning guitar for beginners is an android app that can be utilised both when the user has access to their guitar and when they don't. The app will contain a number of features designed to help the user learn the basics of playing guitar.

2.1.1 Sources of information

Information for the proposed solution has been derived from the following domains:

- User domain- information based on my own experiences as a guitar player as well as feedback taken from users of current applications
- Application domain- information derived from existing applications in the relevant field.

The following illustrates common difficulties when learning guitar as well as common established features that are present in existing systems that can be improved.

2.1.2 Chord library

One of the essential aspects of learning that the application should cater to, is a library of guitar chords showing the user the correct finger placements and how the chord sounds when played correctly. Due to the vast number of chords that can be played on the guitar and their variations this feature of the app should feature only the "basic" major chords and their minor and seventh variations. This should be more than enough content for a beginner and provide them with a good starting point for their learning and provide the building blocks for most basic songs which they can then attempt to learn. As learning chords can be one of the more difficult parts of learning guitar, it is essential that this feature is as clear as possible in its depiction of the chords to the user, with the highlighting of finger placement being particularly important. Colour coding the chords by their finger placements could be used here to help the user. The desired order the players fingers should be placed on the strings could also be added here. The playback of chord audio must also be of sufficient quality, so the user can easily hear what the chord being played sounds like as they may attempt to play along with it themselves or compare the audio with their own attempts at the chord.

2.1.3 Guitar Tuner

Another inclusion that would be desirable for a beginner is a feature that acts as a guitar tuner. A common problem for beginners is having difficulty in tuning their instrument properly. Ideally the tuner aspect of the application would detect the user's plucking of the strings and decipher the note being played and determine if the audio frequency is too high or too low in a manner the user can easily interpret such as a sliding scale that veers between out of tune and in tune. However, from the difficulty in trying to implement DSP (digital signal processing) in the initial incarnation of the project, this could prove difficult. If creating a tuner is unachievable a possible alternative is having a visual output of a guitar neck which will play the in tune note the user selects, allowing them to tune their own guitar by adjusting the pitch of their strings to the audio being outputted from the app. This may not be an ideal solution as a beginner guitarist may have some difficulty in carrying out this process. The tuner feature should also include the option for the intermediate guitar player to tune their instrument to some common "alternate" tunings such as half step below standard, a full step below standard and perhaps common open tunings.

2.2.4 Chord changes

A further feature of the application should be a section more overtly aimed towards the instructional side of learning guitar and include exercises to develop the user's skills. This section should make up the bulk of the application's content. Foremost this section of the app should include tutorials for basic chord changes as this is a highly important part of learning to play to the guitar. These chord changes should be illustrated to the user by animations of some sort, so they can clearly see the chord transition being made and attempt to play along with the app. As with the chord library section of the app, inclusion of audio would be imperative here. For the beginner, even the most basic chord changes can prove difficult so the tutorials should be as user friendly as possible, the chord changes should be displayed clearly and the audio, as well as being of high quality should include a metronome count or drum beat which will help users play in time with the track which will in turn teach them the basics of keeping time with music. The display of the chords here should also be consistent with how they are depicted in the aforementioned chord library section. This consistency will help to reinforce the user's learning and familiarity. Within the application there should also be the option for the user to hear and practice the chord transition at a slower pace until they are comfortable enough to play the chord change at the full tempo. Another feature that could be incorporated here is adding a way for the user to record their own performance of the chord change and then comparing their attempt with the correct way to play the sequence. Hearing the tracks side by side would allow the user to hear clearly any mistakes they may have made and subsequently make the necessary corrections in their next attempt.

The chord change exercises available to the user should also be incremental in difficulty so that when the user has perfected one, they can move on to a more complex and longer sequence with more complicated chords at a quicker tempo. Similar tutorials should also be implemented for guitar riffs,

which unlike chords usually comprise of a sequence of individual notes. This feature should be broadly similar to the chord transitions feature and display the correct finger placement of notes in sync with an audio track. The songs included should be standard ones, commonly learned by beginner guitarists. These songs should be somewhat familiar to the user which will assist their learning as they will be more likely to recognise the song they are trying to learn. This will make the process more enjoyable to users as they should achieve a certain level of satisfaction in learning to play something they are already familiar with. Again, like the chord change tutor, the song tutor will provide the user with the option to try and learn the song at a slower speed until they are confident enough to attempt to play it at the correct pace and should again allow them to record themselves playing the note sequence and have it compared with the correct way the sequence should sound. As guitar riffs are comprised many of individual notes the visual depiction will likely have to be changed from the way chords are depicted.

2.2.5 Finger placements

Another feature that could be included in the tutor section of the app is a feature that more directly teaches the user the finger placement of the various chords. This could take the form of an image of a guitar neck with buttons or text displaying to the user where they should place their fingers and in which order to play various chords. Here it would help if the user was prompted to hold their phone in a similar fashion to a guitar neck allowing them to make the chord shapes on the screen of their phone as they would on an actual guitar. Like the chord library this feature should have some visual element that displays to the user where their fingers should be placed and in which in order to make the correct chord shape. This visual aspect of the chord instruction should be consistent with its use in the chord library and anywhere else it is featured in the app i.e. that method of explaining which fingers go where should be the same. This would be beneficial to the user as it would allow them to develop their skills while they may not have access to their instrument, for example while they are commuting to work or are waiting with some free time to spend.

2.2.6 Feedback

The tutorial aspects of the application should also feature some way the user can receive help from an actual guitar tutor for their own playing. This would be highly beneficial as it would provide the user with access to qualified tutor without having to attend lessons and would improve their learning by providing a way for learners to receive help and criticism about their playing. Possibly this feature could be used in tandem with the record audio feature mentioned above. Here the user could record themselves playing a sequence of chords or notes and send the mp3 of the recording to the guitar tutor. The tutor would then receive the mp3 and could offer advice or criticism based on what they hear. It would be helpful here if the user had the opportunity to specify to the tutor any difficulties they are having and have this included with the sent audio.

2.2.7 User account

Finally, the app should allow users to register an account with the app, with their details being saved to a database on a server. Once the user has registered they should be able login to their account. This could be expanded upon to track a user's learning progress, for example by monitoring the number of chord pages they have viewed or the features they have used.

2.1.8 Features of proposed solution

In conclusion the following features should be included in the produced software:

- A chord library of basic chords to intermediate chords that display finger placements, and play audio of the chords
- A section that allows the user to tune their guitar
- A section that teaches basic chord transitions which are illustrated to the user and incremental in difficulty
- A section that teaches basic riffs, which are again illustrated to the user and incremental in difficulty
- A section that teaches chord finger placements to the user
- A way for the user to receive feedback on their playing or ask for help if they are experiencing difficulty
- A way for the user to create an account with their details being stored to a database

2.2 Development strategy

Based on the analysis above, the most preferable format for the software being produced is an android application. This format is desirable for a number of reasons, firstly it is a format that is wildly available and widely accessible. This provides the advantage of the finished product being able to reach a large number of people via the Google Play Store. Android's prevalence also provides an advantage during the development stage as there are numerous recourses aimed at android development that can be utilised during the product lifecycle. This also means that development costs are relatively low. Using the android format also makes the software testing process easier as the software produced can be tested extensively through unit tests as well as being tested on a device or emulator. Android also allows features to be modified and integrated easily. However, there are some disadvantages with the chosen system, the most prominent of which being the issues of optimisation over different devices and the issues over the different versions of android that are available. To a lesser extent, availability of memory can also be a problem in android development.

2.2.1 Potential development methods

- **Waterfall model**

Example of a defined production process. Often viewed as the traditional method for software development, the waterfall method provides a liner approach to software development, a result of which being that there is little room for deviation from the original development plan. This model has distinct goals/ milestones for each stage of the development process with the finished product being delivered at the end. Because of this there is little room for reviewing/ amending an area of the project that has already been completed.

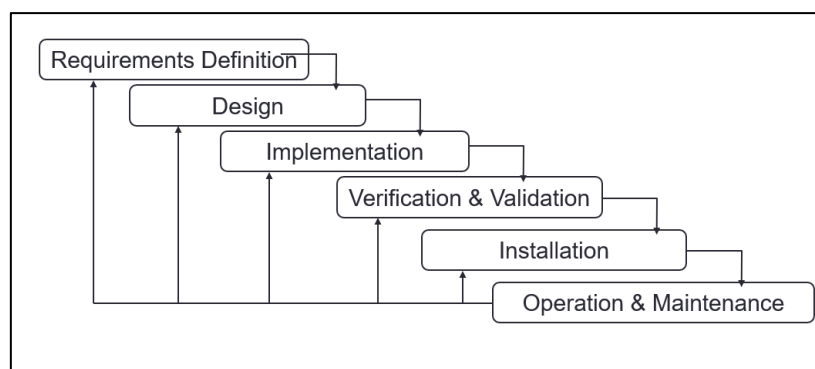


Fig. - 1 Waterfall model

- Incremental model

The software design is separated into individual sections which makes the process easier to manage. Each section passes through the necessary requirements, design, implementation and testing phases. Software is produced in the first module so working software is produced early on in the development lifecycle. The software is continuously added to and tested until it is complete.

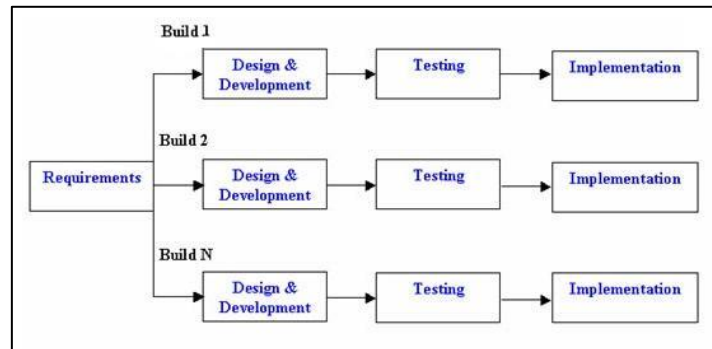


Fig. 2 - Incremental model

- Spiral model

Similar to the incremental approach though with more emphasis on risk management. Process begins by determining objectives, evaluating alternatives and then identifying and resolving risks in the project. Generally split into four stages:

- 1- Planning phase where requirements are gathered
- 2- Risk analysis where process is undertaken to identify risk and find alternate solutions. A prototype is produced at the end of this phase.
- 3- Engineering phase where software is developed and tested
- 4- Evaluation phase where the output is evaluated

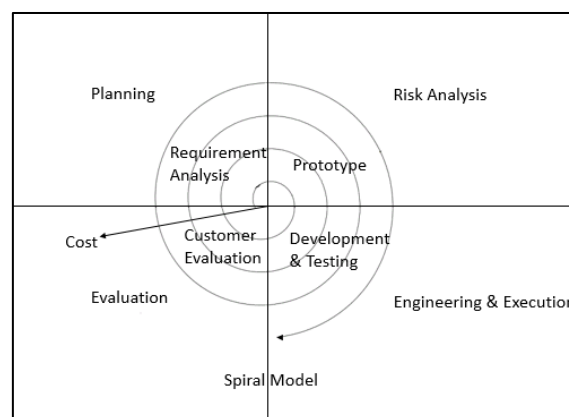


Fig. 3 - Spiral model

- **Agile model**

Another example of an incremental approach model, with software being developed in rapid incremental cycles. Each iteration includes requirements analysis, design, implementation and testing and there is constant integration with software developed in previous iterations, with the final system evolving over the series of cycles.

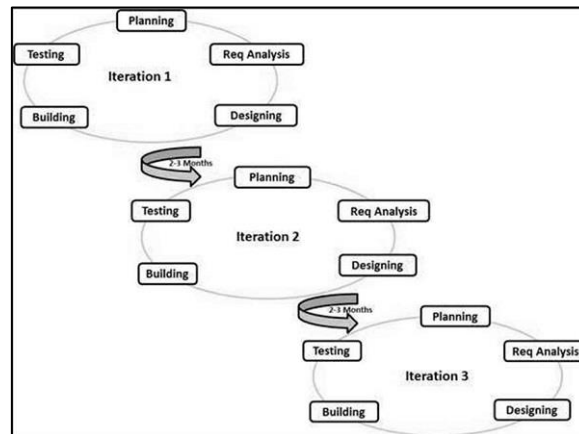


Fig. 4 – Agile model

2.2.2 Pros and cons of development methods

- **Waterfall model**

Advantages	Disadvantages
No overlap between phases	Difficult to make changes
Clear project milestones	High amount of risk
Easy to track progress	Unsuitable for complex projects
Documentation produced for each phase	Software not produced until late in the development cycle

- **Incremental model**

Advantages	Disadvantages
Software produced early in the development cycle	Requires precise planning and design
Easier to manage risk	System needs to be defined completely before it can be broken down and built incrementally
Allows for change of scope and requirements	Total cost higher than waterfall model

- **Spiral model**

Advantages	Disadvantages
Effective at managing risk	Risk management is a complex process
Effective for large projects	Project success highly dependent on the risk management stage
Software produced early in the lifecycle	Not very flexible

- **Agile model**

Advantages	Disadvantages
Working software delivered frequently	Lack of emphasis on documentation
Adaptive to changing circumstances	Difficult to access effort necessary for larger projects
User focused	More difficult to predict what project will deliver
Allows for requirement changes	Harder to assess risk
Constant integration between software produced	Can be time intensive

2.2.3 Conclusion

Based on the analysis above, the most logical strategy for development would be the Agile approach. This strategy is preferable as it allows for changes to be made to the software after the initial planning stage, so improvements and extra features can be added to the software during the development process. Features which are present in the app already can also be refined throughout the development lifecycle to help make the finished product as functional as possible. The Agile method also provides the advantage of the produced software being tested constantly during development, meaning bugs are discovered and fixed during the development cycle rather than at the end of the process when the software is finished. This approach also suits the type of software being developed as an android application will be run constantly during design to check it works as expected. This will be useful as minor changes to one area of an app developed in android can have knock on effects in others. This can be attributed to the variation of code present, as both java and xml are used to create applications and need to be utilised in tandem with each other for an app to run successfully, if not the app may fail to run. The application running successfully is also reliant on many other variables like availability of memory, manifest declarations and build files needing to be synced constantly.

Chapter 3: Requirements analyses and specification

3.1 Requirements Process

The requirements that were gathered for this project were primarily derived from my own experience as a guitarist and drawn from the what I would have found helpful when I began playing as well practices I used to help develop my skills as a player. This information was compounded with analysis of existing systems used to solve the same problem. Requirements were also gathered by asking other guitarists with various levels of experience what features they would like from an instructional app as well as people who have considered learning the instrument.

3.2 Requirements

3.2.1 Purpose

The purpose of this application is to help people who are trying to learn to play the guitar.

3.2.2 Intended Audience

The intended audience is people who are starting to learn the guitar but may not have enough free time or the opportunity to take lessons or hire a tutor.

3.2.3 Project Background

Initially the project was designed to be an application that would act as an amplifier and an effects processor for electric guitars. However, it became apparent that this was too ambitious a task to undertake and the project was scaled back to the guitar tutor idea.

3.2.4 Objectives

The main objectives of this project were to create an application that would provide a number of services and instructions that would be useful to someone learning how to play guitar.

3.2.5 Dependencies on existing systems

The app is dependent on Android version 7.1.1 or later

3.2.6 Assumptions

Users are already somewhat familiar with using mobile applications

3.2.7 Functional requirements

From the user perspective the app should be easy to navigate, with user being able to find the features they need access to with relative ease.

The app should provide information that will be beneficial to someone learning guitar for the first time including the following;

- 1- The app should include a library of chords
- 2- The app should include a tuning feature
- 3- The app should contain a tutorial section for chord changes
- 4- The app should contain a tutorial section for guitar riffs
- 5- The app should contain a tutorial section for teaching finger placements
- 6- The user should be able to register to a database by providing their details
- 7- The user should be able to login to the app
- 8- The user once logged in should be able to log out

3.3 Function Definitions

3.3.1 Functions in MainActivity class

- onCreate()- returns the tabbed layout xml of the application
- onCreateOptions()- adds menu to the action bar
- onOptionsItemSelected()- handles action bar items clicks
- getItem()- returns instances of tabs
- getCount()- returns the number of tabs
- getPageTitle()- displays the tab of each page in the tab layout

3.3.2 Functions in MainActivity_logged_in class

- onCreate()- returns the tabbed layout xml of the application modified for logged in users
- onCreateOptions()- adds menu to the action bar
- onOptionsItemSelected()- handles action bar items clicks
- getItem()- returns instances of tabs
- getCount()- returns the number of tabs
- getPageTitle()- displays the tab of each page in the tab layout

3.3.3 Functions in Configuration class

- Defines details that refer to the server and database of the application as a series of Strings including variables and server URL

3.3.4 Functions in Login2 class

- onCreate()- returns the layout of the Login class xml
- onResume()- checks information from shared preferences and will log the user in if conditions are true
- login()- logs the user in or produces an error message depending of the accuracy of credentials passed by the user
- Map()- handles information to the server
- onClick()- begins the login method when the user presses the login button

3.3.5 Functions in Register class

- onCreate()- returns view of the Register xml
- onClick()- posts the user information to the server
- onFinish()- displays a success or error message depending on if the user has been registered successfully

3.3.6 Functions in tab1 class

- onCreateView()- returns layout of the "Chord Library" tab
- onClickListeners()- several are used to open the individual chord sections of the chord library and to access the login and register features

3.3.7 Functions in tab1_logged_in class

- onCreateView()- returns layout of the "Chord Library" tab modified for users who are logged in
- onClickListeners()- several are used to open the individual chord sections of the "Chord library"

3.3.8 Functions in tab2 class

- onCreateView()- returns layout of the "Tuning" tab
- onClickListeners()- several are used to open the individual tuning sections and to access the login and register features

3.3.9 Functions in tab2_logged_in class

- onCreateView()- returns layout of the "Tuning" tab modified for users who are logged in
- onClickListeners()- several are used to open the individual sections in the "Tuning" tab

3.3.10 Functions in tab3 class

- onCreateView()- returns layout of the "Tutor" tab
- onClickListeners()- several are used to open the individual tutor sections and to access the login and register features

3.3.11 Functions in tab3_logged_in class

- onCreateView()- returns layout of the "Tutor" tab modified for users who are logged in
- onClickListeners()- several are used to open the individual sections in the "Tutor" tab

3.3.12 Functions in Chord Handler class

- onCreate()- returns layout of chord handler xml which consist of a viewPager that the chord pages are inflated to
- onBackPressed()- finishes the activity with finish() when the back button is pressed
- getItem()- returns new instances of chord classes
- getCount()- returns number of pages in page viewer

3.3.13 Functions in Major Chord class

- onCreateView()- returns xml layout of the major chord xml layout
- onClick()- plays chord sound
- chordName()- create a new instance of the major chord

3.3.14 Functions in Minor Chord class

- onCreateView()- returns xml layout of the minor chord xml layout
- onClick()- plays chord sound

3.3.15 Function in Seventh Chord class

- onCreateView()- returns xml layout of the seventh chord xml layout
- onClick()- plays chord sound

3.3.16 Functions in Tuning class

- onCreate()- returns xml layout of the tuning page
- onClick()- numerous onClickListeners are used to play the various individual notes that make up the tuning

3.3.17 Functions in chord_transitions class

- onCreate()- returns the xml layout of the chord transition page
- onClick()- used to play the audio of the chord transitions and the corresponding animation, onClickListeners are also used to detect user presses of buttons related to recording/ playing back of audio
- recorderSetup()- creates a new MediaRecorder, sets the audio sources, file save path and the sound output of the recorder

- `createFileName()`- creates random file name for recorded audio file using the `StringBuilder` class
- `requestPermission()`- requests permission to write external storage and record audio
- `onRequestPermission()`- handles request for writing external storage and recording audio permissions and displays appropriate message based on if permission is granted or not
- `checkForPermission()`- checks for write storage and record audio permissions
- `unlockSendAudioAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated with the send audio achievement for the logged in user.
- `unlockRecordAudioAchievement()`- executes .php file which updates the database containing the user's information with the send audio achievement. The achievements table is updated with the record audio achievement for the logged in user.
- `updateLearnBasicChordsAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated to increment the figure in the unlocked column by 1.
- `updateLearnIntermediateChordsAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated to increment the figure in the unlocked column by 1.

3.3.18 Functions in `basic_riffs` class

- `onCreate()`- returns the xml layout of the guitar tab page
- `onClick()`- used to play the audio of the guitar tab and the corresponding animation, `onClickListeners` are also used to detect user presses of buttons related to recording/ playing back of audio
- `recorderSetup()`- creates a new `MediaRecorder`, sets the audio sources, file save path and the sound output of the recorder
- `createFileName()`- creates random file name for recorded audio file using the `StringBuilder` class
- `requestPermission()`- requests permission to write external storage and record audio
- `onRequestPermission()`- handles requests for writing external storage and recording audio permissions and displays appropriate message based on if permission is granted or not
- `checkForPermission()`- checks for write storage and record audio permissions
- `unlockSendAudioAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated with the send audio achievement for the logged in user.
- `unlockRecordAudioAchievement()`- executes .php file which updates the database containing the user's information with the send audio achievement. The achievements table is updated with the record audio achievement for the logged in user.

- `updateLearnBasicRiffsAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated to increment the figure in the unlocked column by 1.
- `updateLearnIntermediateRiffsAchievement()`- executes .php file which updates the database containing the user's information. The achievements table is updated to increment the figure in the unlocked column by 1.

3.3.19 Functions in `finger_placement_handler` class

- `onCreate()`- returns xml of handler page which consist of `viewPager` which the chord tutorials will be inflated to
- `onBackPressed()`- finishes the activity
- `getItem()`- returns new instances of chord tutorial classes
- `getCount()` returns the number of fragments in the `ViewPager`

3.3.20 Functions in `finger_placement_chord` class

- `onCreateView()`- returns layout of the chord tutorial pages
- `onClick()`- used to handle the correct sequence of button presses

3.3.21 Functions in `User_Activity` class

- `viewAchievements()`- allows the user to view the achievements they have unlocked by using the app.

Chapter 4: Design

4.1 User interface design

4.1.1 UI Design principles and issues

Below are some of the common design issues and principles which should be taken into account when designing user-orientated software in Android.

- **Screen size**

Unlike desktop computer system android screens are much smaller with different display densities and aspect ratios. This can cause issues due to the number of different models of Android mobile devices as each may represent the visual content of the application slightly differently.

- **Text size and icons**

Because the Android devices generally contain smaller screens, any text and icons being displayed to the user must be of a suitable size, allowing the user to view easily. The application shouldn't be too reliant on text information but should aim to present information through sound and images.

- **Buttons**

The size of any buttons used in the application should be designed bearing in mind the fact that they are to be used by touch so must be a suitable size to be pressed by the user.

- **Size of application windows**

Fixed xml window sizes in the application should be avoided as the different types of android devices and their various screen sizes may mean only part of the picture is displayed.

4.1.2 Concept layouts

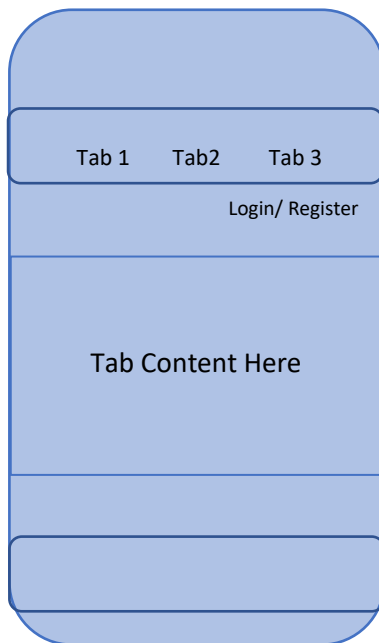


Fig.1- Main activity layout

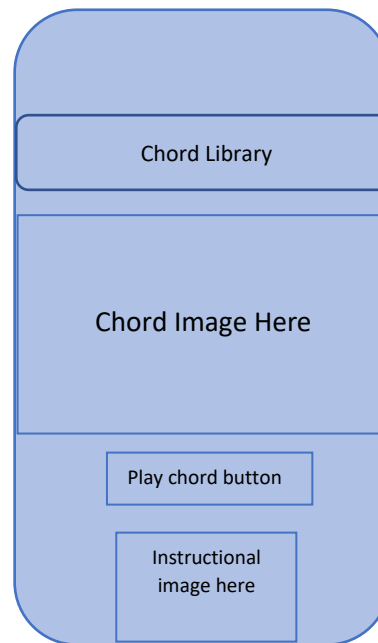


Fig.2- Chord library page layout

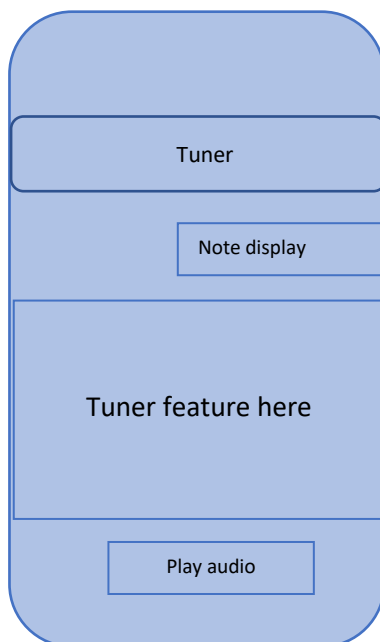


Fig.3 - Tuner page layout

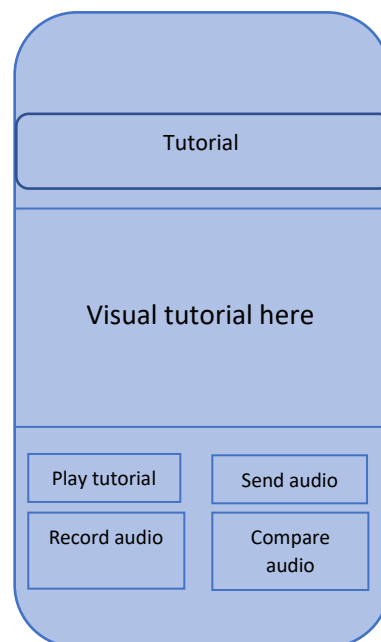


Fig.4 – Tutorial layout

4.1.3 Colour scheme concepts

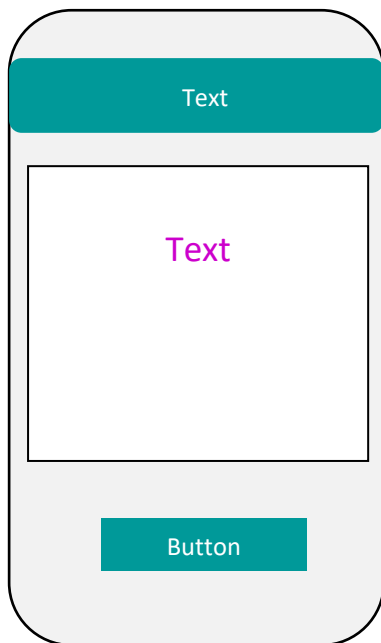


Fig. 5- Colour scheme concept 1

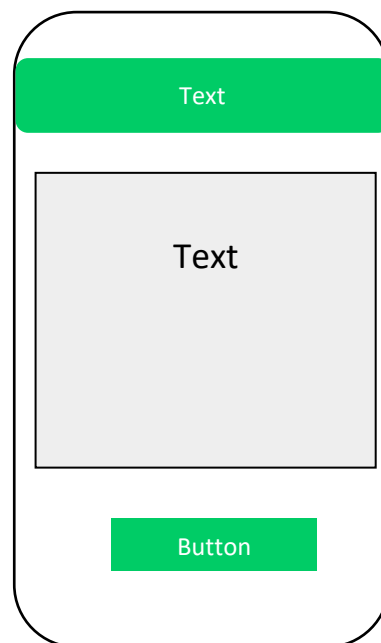


Fig. 6- Colour scheme concept 2

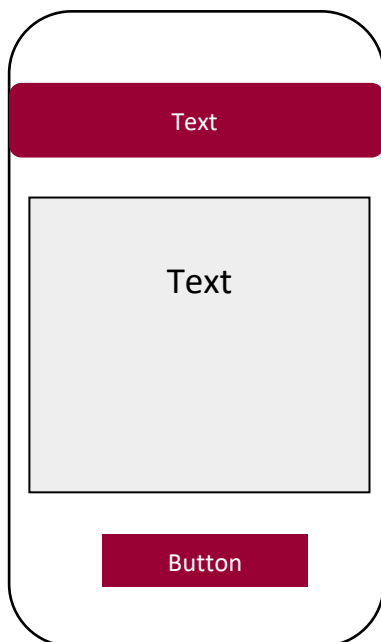


Fig. 7 - Colour scheme concept 3

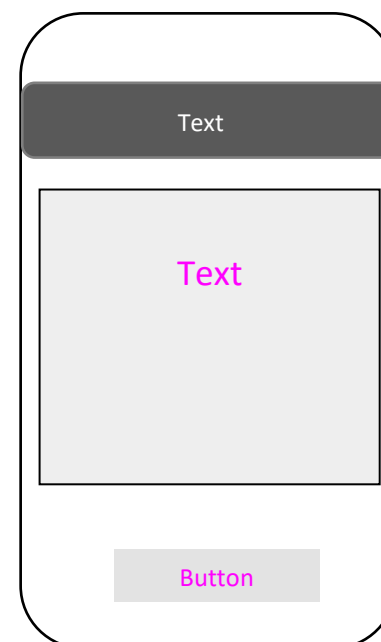


Fig. 8 - Colour scheme concept 4

4.1.2 Colour palate:

Below is the standard colour palate used throughout the application defined in the colors.xml file in the values folder of the application. Throughout the layout of the app these colours are used consistently.

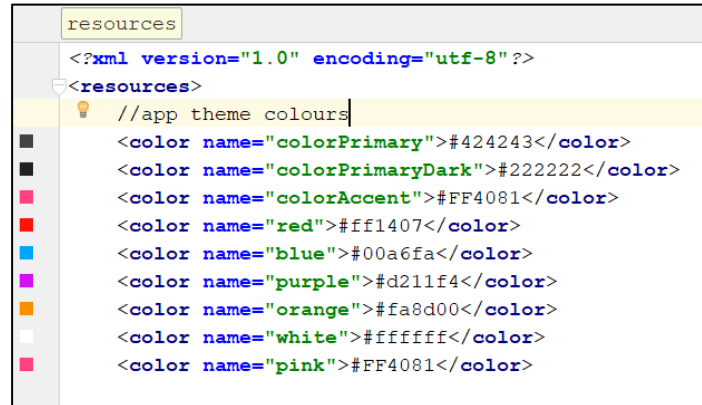


Fig. 9- The colour palate used for the user interface

4.1.3 Application theme

Below is the app theme defined in the styles.xml file in the application's values folder.



Fig 10. - App theme

4.1.2 Layout templates

- General layout

The finalised layouts for the various pages that make up the application are displayed below.

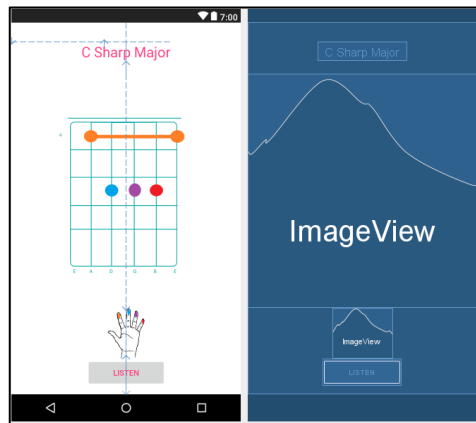


Fig. 11- Chord page finalised layout

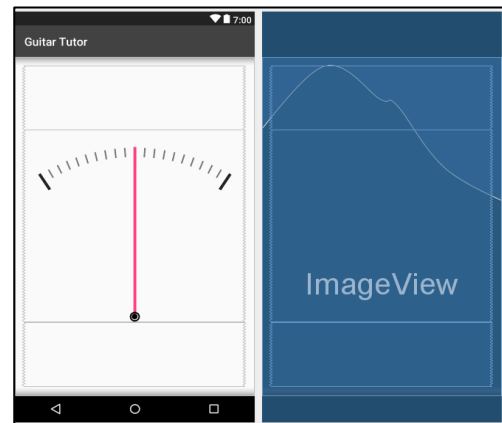


Fig. 12- Tuner page finalised layout

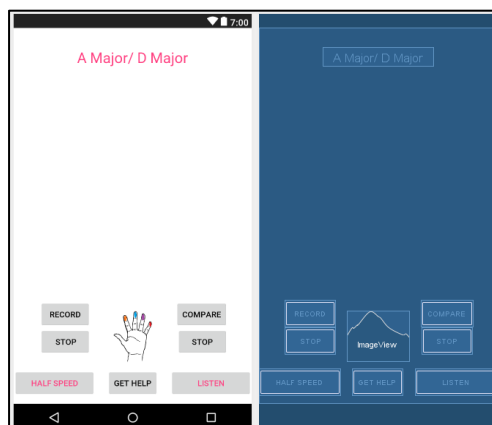


Fig. 13 - Chord tutorial page final layout

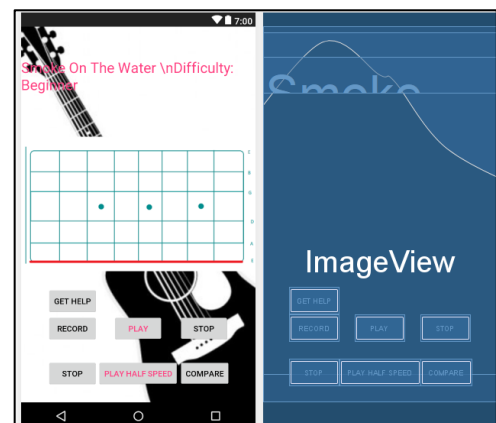


Fig. 14 - Riffs tutorial page final layout

- **Background image**

Displayed below is the background image used in several the pages in the app.

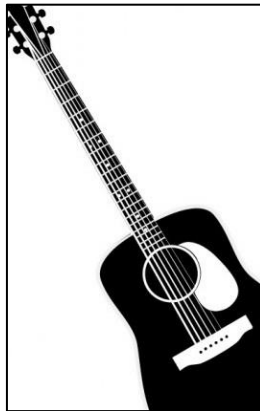


Fig 15- background image

4.1.3 Optimisation/ dimensions

The app was designed to the specifications of the Google Nexus 5x Android device as it provided a good generic platform to run the app from. Also, its screen dimensions and visual quality offer a fairly balanced representation of Android devices.

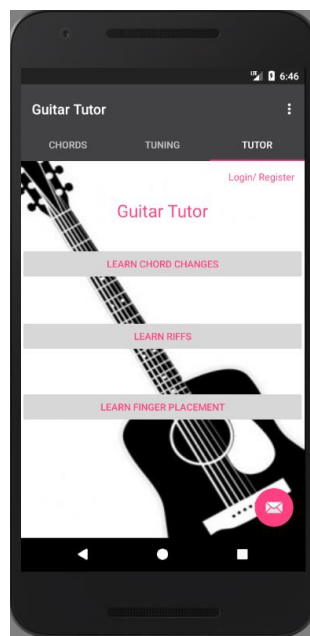


Fig. 16- the application run on the android emulator (AVD)

- **Emulator details**

Emulator details	
Emulator model	Nexus 5X
Screen dimensions	1080x1920
DPI	420
Android API	25

4.1.4 UI justification

- **Opening page**

When the user opens the application, they are met with the following layout (see Fig. 17). This design was chosen above numerous other prototype interfaces mainly because of its simplicity. The reason being that, as the objective of the application is to provide instruction and teach the user, the interface should be as accessible and as easy to navigate as possible without any unnecessary components. A plain background was chosen which includes a drawable image of a guitar to make the design more eye-catching and attractive to the user. This plain background is contrasted with bright text which will capture the user's attention.

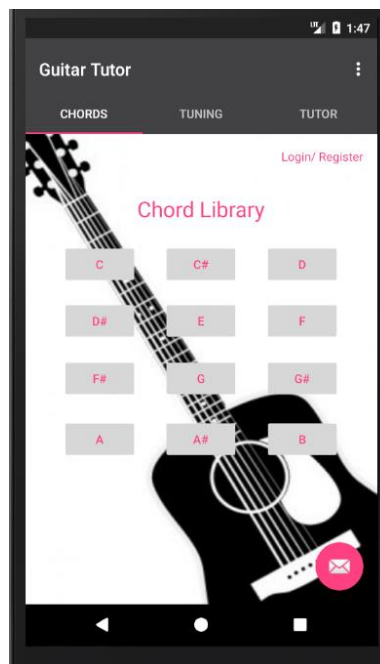


Fig. 17- Opening page of the app

For the user interface of the project it was decided that a tabbed layout would be the best approach for including all the various features of the project. The use of a tabbed layout is also beneficial to the user as they can easily swipe between their desired activity, rather than having to navigate through multiple menus, making the app more accessible. From the developmental perspective, using a tab layout helped to organise the project, as it naturally split the application into three individual sections isolated from each other which could be worked on separately. These sections include, a chord library where the user can view various chord shapes and their variations, a tuning section where the user can attempt to tune their guitar and a guitar tutor tab which will teach the user to play simple chord transitions and songs. A tutorial section allowing users to practice their finger placement was also incorporated into this section of the app.

- **Tabbed layout**

The three tabs that comprise the layout of the app are visible below (see Fig. 18). Also visible is a text view to the right below the third tab allowing the user to login or register to the app when pressed. This is changed to a message informing the user they are logged in when the user logs in providing the necessary credentials. Once logged in the user can choose to log out by clicking the options symbol at the top left of the interface and selecting “Logout”.

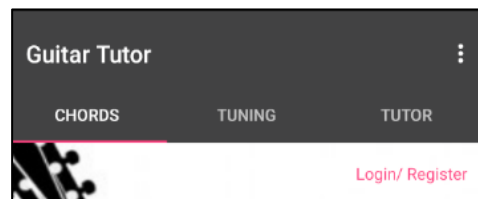


Fig. 18- Tab menu

The three tabs are derived from the proposed solution documentation. The first of these tabs acts as a chord library where the user can select a button corresponding to the selected chord. This in turn opens a separate section of the application which displays the following layout.

- **Chord library**

In keeping with the application's minimalist style, the chords in the chord library are displayed to the user in an ImageView set against a plain background. The colour of the drawable is designed to stand out to the user clearly. The notes that comprise the chord are all colour coded to the correct finger placements. This is illustrated to the user with another smaller Image View depicting a hand where the fingers are coloured to correspond to where the user's finger should be placed on the guitar neck. Again, contrasting text colour was used here (as it is in the rest of the project) to make text stand out to the user. Here it is present in a Text View that displays the name of the current chord. The final feature of the chord page user interface is a “Listen” button that plays a brief audio track of the current chord. From any of the major chord pages the user can easily swipe between the minor and seventh

variants of that chord in a similar fashion to swiping between the three main sections of the app in the main menu. An example of a chord page in the app is displayed below (see Fig. 19)

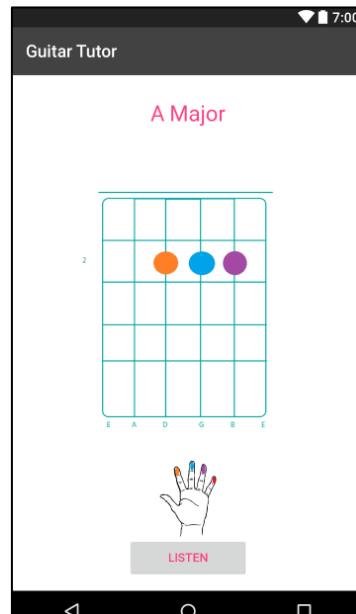


Fig. 19- Chord page example

- Tuner section

The second main section of the application is the “Tuning” section. When the user selects this option using the swipe view they are taken to a sub section displaying various guitar tunings. From here they select their desired tuning and are displayed the following layout (see Fig. 20).

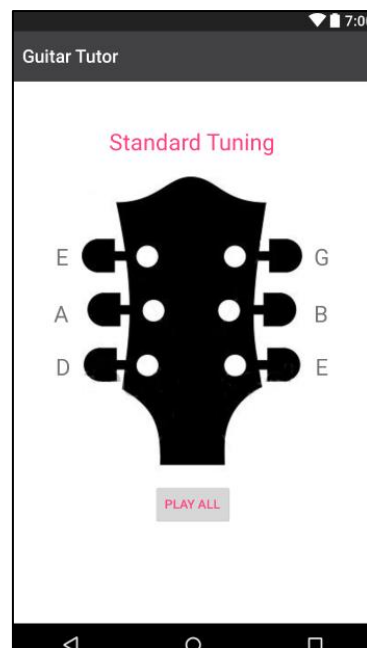


Fig. 20- Tuner page example

Here an Image View of a guitar head is displayed to the user with Text Views corresponding to the various tuning pegs. When the user selects one of the Text Views an audio track of the correct note is played, allowing the user to match the pitch of their own instrument with the audio. The page also features a button that plays all the correctly tuned notes in order from low to high, allowing the user to, again compare the played audio with the tuning of their own instrument.

- Tutor section

The third tab in the user interface of the project consists of the tutorial features of the application. Navigating to this tab opens a menu where the user can select to practice chord changes, learn basic songs and practise the various finger placements of chords. Opening the “Learn Chord Changes” opens a sub-menu where the user can select the chord sequence they wish to learn. These are categorised by difficulty, starting with basic transitions at a slow pace, and progressing to more complicated sequences which include more chord changes at a faster pace. The layout of a chord transition page is displayed below (see Fig. 21).

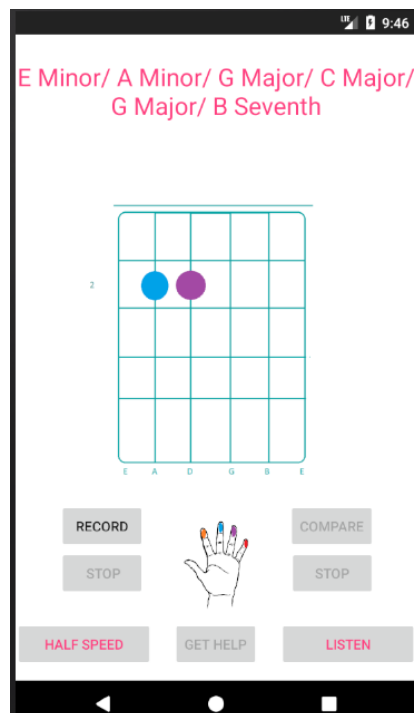


Fig. 21- Tutor page example

Being consistent with layouts used elsewhere in the app the, chord names in the sequence are displayed at the top of the page, with a graphic of the chord displayed in the centre of the screen and the hand graphic to explain finger placement present below. In addition to these, a number of buttons are also present, each carrying out a specific function. The “Listen” button plays an audio track of the specific sequence, including a metronome count in and rhythm track, should the user wish to play along. Clicking this button also starts the animation displaying the chord changes which are synced to

the audio file. The “Half Speed” button performs a similar function but plays the audio and animation at half the pace, allowing the user to practice the chord sequence at a lower speed until they are comfortable enough to try playing the sequence at the full speed. The chord transition pages also contain buttons which allow the user to record their own performance of the chord change and then have it compared with how the sequence should sound. Initially all but the “Record” button will be unavailable to the user until they begin the process of recording themselves. The user then has the option to stop their recording by pressing the “Stop” button below “Record”. Pressing the “Compare” button will then play their recorded audio followed by the pre-recorded version of the chord sequence. Pressing the “Stop” button below the “Compare” button will then end the audio playback. The “Get Help” button allows the user to send their recorded audio to a guitar tutor to receive feedback or criticism.

The section of the app that teaches basic songs was designed in a similar fashion to the chord transition sections, with the main difference being the guitar graphic of the notes being displayed horizontally (see Fig. 22). This style was chosen as it is a more natural view for displaying a progression of notes.

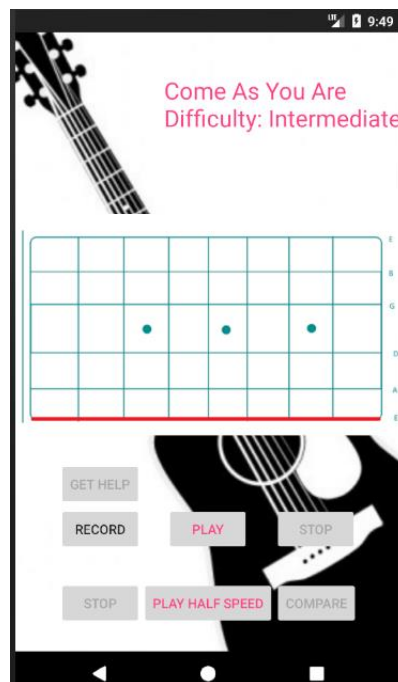


Fig. 22- Tutor song page example

As with the chord progression sections of the app, here the user can choose to hear the audio of the song and view the animation at normal or half speed. In this section, the user can again record their own attempts at playing the song and have it compared to the correct note progression as well as send their recorded audio.

- **Finger placement section**

The final section of the tutor aspect of the app is the finger placement section (see Fig. 23). Here the layout is given a horizontal orientation to better emulate the neck of a guitar. The idea being that the user can hold their phone as they would a guitar and become familiarised with the various finger placements and chord shapes.

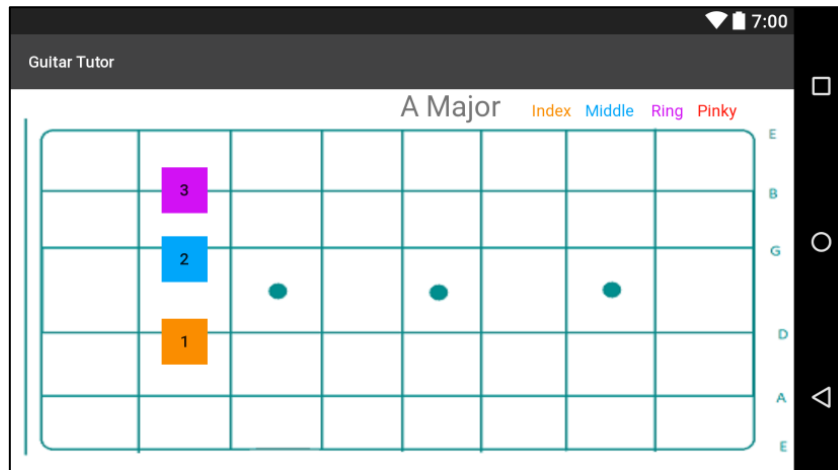
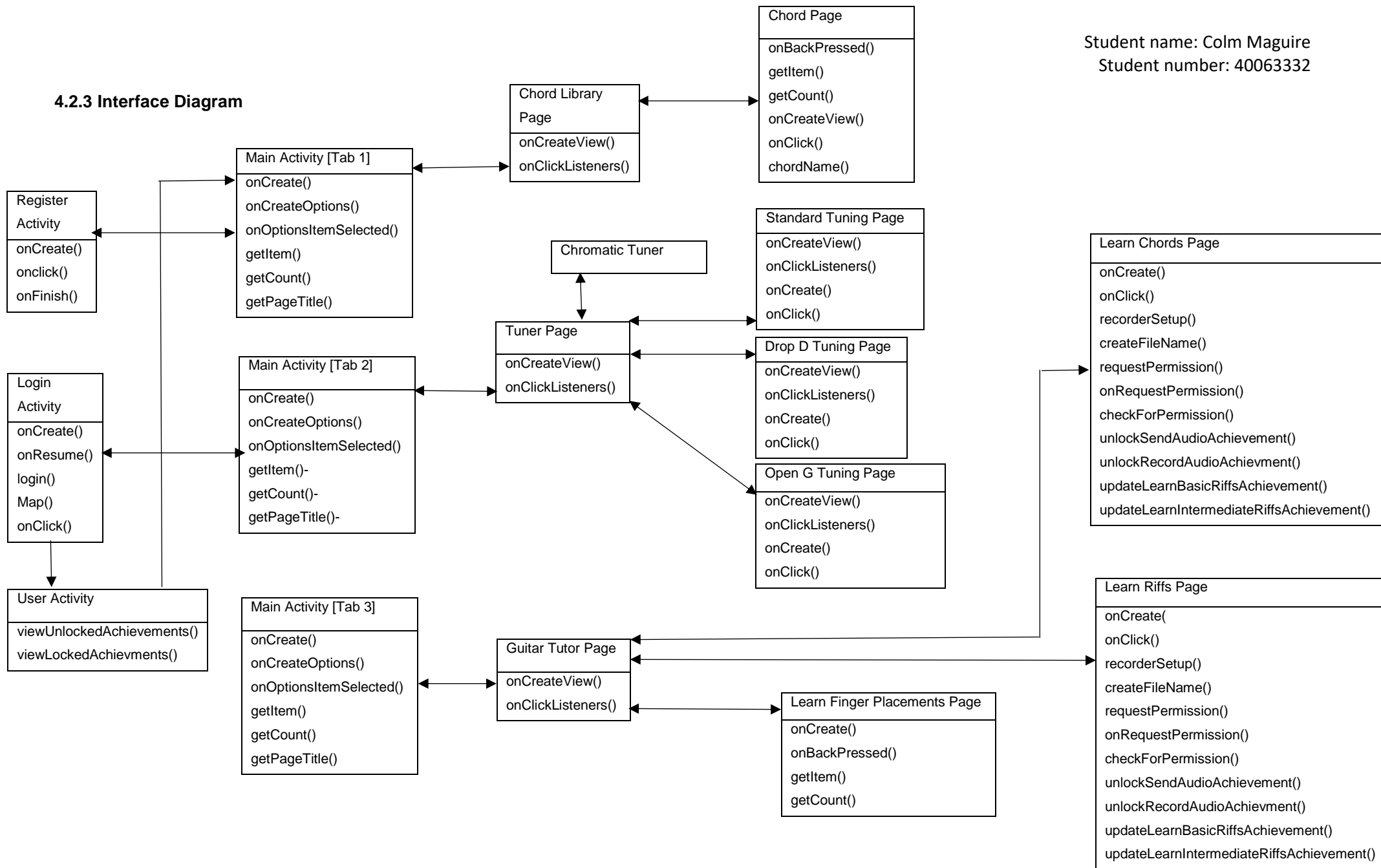


Fig. 23- Finger placement

In this layout, the user is presented an image of a chord with the name of the chord being displayed at the top of the page. The layout also includes buttons organised in the shape of the chord being taught. The buttons are numbered to communicate to the user what order they should place their fingers on the screen in order to make the chord. Similar to other sections the buttons are also colour coded in relation to which finger should be placed where. A reference is provided for the user in the top right of the layout explaining which colour relates to which finger. When the user presses the buttons in the correct sequence, the correct shape of the chord is made, and an audio track is played to confirm the user has performed the chord correctly.

4.2.3 Interface Diagram



Chapter 5: Implementation and testing

The following explains how significant aspects of the implementation of the project were achieved through code, with the functions and programs explained in detail.

5.1.1 General layout

The general layout of the application is comprised of three different tabs with the various features of the app branching off from each tab. To create the slide feature which allows the user to navigate between the three main sections of the application a ViewPager and SectionsPagerAdapter were implemented, enabling the creation of the tab layout. This enables the user to swipe through the three main tabs. The SectionPagerAdapter class extends from the FragmentPagerAdapter and includes the method “getItem()”. This method uses a switch statement to return new instances of each tab as the user swipes through them with the default set to return null. A getCount() method is also present here to return the correct number of pages in the tabbed layout (in this case there are three pages so the getCount() method returns 3). The switch statement used to return the instances of the three main tabs is displayed below (see Fig 1).

```
* Returns a fragment corresponding to  
* one of the sections/tabs/pages.  
*/  
public class SectionsPagerAdapter extends FragmentPagerAdapter {  
    public SectionsPagerAdapter(FragmentManager fm) { super(fm); }  
  
    @Override  
    public Fragment getItem(int position) {  
        // returns current tab  
  
        switch (position) {  
            case 0:  
                tab1 tab1 = new tab1();  
  
                return new tab1 ();  
                //return tab1;  
  
            case 1:  
                tab2 tab2 = new tab2();  
                return tab2;  
  
            case 2:  
                tab3 tab3 = new tab3();  
                return tab3;  
  
            default: return null;  
        }  
    }  
}
```

Fig. 1- SectionPagerAdapter class

5.1.2 Chord library

The first tab in the main layout of the app is a library of chords. This class extends from Fragment and is displayed by implementing the onCreateView() method, which includes LayoutInflater as one of its arguments. The View class is used along with a LayoutInflater in the onCreateView() to inflate the view of xml layout of the tab. This is displayed by returning the View variable at the bottom of the class. The layout of the page consists of a series of buttons which correspond to the chord content, as well as textViews which act as links to the Login and Register sections of the app. These buttons and

textViews contain onClickListeners to detect when these objects are pressed by the user. An example of the code is displayed below (see Fig. 2).

```
// button which opens c handler class
Button cButton = (Button) rootView.findViewById(R.id.C_button);
cButton.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getActivity(), chord_C_handler.class);
        startActivity(intent);
    }
});
```

Fig. 2- onClick listener to open chord handler class

Here the button which brings the user to the “C” chord section of the app is set as an OnClickListener. The listener detects when the button is pressed by the user and opens the designated activity via an Intent. Within the onClick method, a new instance of an Intent is created with, the name of the class which is being opened passed to the getActivity() method as an argument. The activity is then activated with the startActivity() method which takes the name of the Intent as its argument.

The content in the “Chord Library” section is displayed using a similar method to the tabs in the main layout of the app. However, as the content is held within a fragment, a handler class was created which extended from FragmentActivity. This class returns a blank xml layout comprising of only a ViewPager component which is used to inflate the specific xml layouts to the device screen (see Fig 3).

```
LinearLayout
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:orientation="vertical" android:layout_width="match_parent"
4   android:layout_height="match_parent">
5
6   <android.support.v4.view.ViewPager
7     xmlns:android="http://schemas.android.com/apk/res/android"
8     android:id="@+id/A_Pager"
9     android:layout_width="fill_parent"
10    android:layout_height="fill_parent"
11  />
12
13 </LinearLayout>
```

Fig. 3- ViewPager defined in xml

The onCreate() method in the handler includes a new instance of the ViewPager class, with the component being cast to the ViewPager in the xml class using the findViewById() method (see Fig 4.)

```
/*
 * Class to handle the swipe view between c chord variants
 */
public class chord_C_handler extends FragmentActivity {

    // sets layout of activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.c_handler);

        //initialize view pager and page adaptor
        ViewPager mViewpager = (ViewPager) findViewById(R.id.C_HandlerPager);
        mViewpager.setAdapter(new ScreenSlidePagerAdapter(getSupportFragmentManager()));
    }
}
```

Fig. 4- chord handler class

This ViewPager was then initialised with a new ScreenSlidePagerAdapter which provides access to the data items (in this case the c major, minor and seventh classes) and provides a new view for each item in the data set. The ScreenSlidePagerAdapter extends from a FragmentStatePagerAdapter with a switch statement returning new instances of the classes that display the content of the chord pages (see Fig.5).

```
private class ScreenSlidePagerAdapter extends FragmentStatePagerAdapter {  
    public ScreenSlidePagerAdapter(FragmentManager fm) { super(fm); }  
  
    /**  
     * Method switch implements switch to display content in the page viewer  
     */  
    @Override  
    public android.support.v4.app.Fragment getItem(int position) {  
        switch (position) {  
            case 0:  
                return chord_C_Major.newInstance("C Major");  
            case 1:  
                return new chord_C_Minor();  
            case 2:  
                return new chord_C_Seventh();  
            default:  
                return null;  
        }  
    }  
}
```

Fig.5- Switch statement returning instances of chord classes

The chord handler classes also contains a getCount() method to count the number of items in the page adapter, (in the case of the chord pages this was three, as the items returned were the chord's major, minor and seventh variants). A method which finishes the activity when the user presses the back button is also included. This onBackPressed method calls the finish() method and brings the user back to the chord library page.

Within each individual class in the chord section a LayoutInflater is used to return a view of the required xml. The chord classes also include instances of the Button class, which play a sound file of the chord. The necessary sound files are stored in the applications resources folder. The Button classes are cast to the button widget in the xml code via use of the findViewById() method. This process is shown below (see Fig 6).

```
//initialises button  
Button play_Cminor = (Button) rootView.findViewById(R.id.play_C_Minor);
```

Fig. 6- Initialising xml button with findViewById method

An `onClick` listener was then added to the button so that it would play the corresponding sound file when pressed by the user. This was achieved by creating a new instance of the `MediaPlayer` class in the `onClick` method, a class designed primarily for playing sound and video. The new instance of the class includes creates a `getActivity()` method to fetch the correct sound file from the resources folder. The sound file is then played with the `.start()` method. This process is displayed below (see Fig. 7)

```
// onclick listener to play sound file
play_Cmajor.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        final MediaPlayer Cmajor = MediaPlayer.create(getActivity(), R.raw.c_major);

        Cmajor.start();
    }
});
```

Fig. 7- Initialising MediaPlayer with sound file

In the tuner section of the app, which is the second tab in the main layout a similar process was used to implement the tuning feature.

5.1.3 Tutorial sections

For the animations present in the tutor sections of the application, layout resource files were created in the app's drawable folder. Animation lists were then added to these files by creating individual items which referred to the images from the resources folder. These items were then combined to make up the animated chord sequence defining the order the images would appear in the animation and the duration of each image. The duration values and order were set to run in tandem with the corresponding audio files. Also in the animation list the xml attribute "oneshot" was set to true, this would ensure that the animation would not keep repeating and would only play when the designated button was pressed (see Fig. 8).

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">

    <item
        android:drawable="@drawable/c_major"
        android:duration="4000" />

    <item
        android:drawable="@drawable/g_major"
        android:duration="2000" />

    <item
        android:drawable="@drawable/a_minor"
        android:duration="2000" />

    <item
        android:drawable="@drawable/f_major"
        android:duration="2000" />

    <item
        android:drawable="@drawable/a_minor"
        android:duration="2000" />

    <item
        android:drawable="@drawable/f_major"
        android:duration="2000" />

    <item
        android:drawable="@drawable/f_minor"
        android:duration="4000" />

</animation-list>
```

Fig. 8- xml animation list

This process was repeated for the half speed versions of the chord transitions and songs.

In the java code, playing the animation was handled by carrying out the following. Firstly within the onClick listener used to begin the animation, the imageView in the xml code is set to show the first image in the animation list using the setImageResource() method with the necessary drawable passed as the argument. Following this a new instance of AnimationDrawable was created with the method getDrawable() used to display the required image. The animation is played using the .start() method. The mediaPlayer, of the corresponding audio file is started simultaneously also using the .start() method (see Fig. 9).

```
a_major_d_major

final MediaPlayer a_d_a_half = MediaPlayer.create(this, R.raw.transition_a_d_a_half);

// onclick listener to play audio and start animation transition
Listen.setOnClickListener((v) -> {

    ImageView imageView = (ImageView) findViewById(R.id.Chord);
    imageView.setImageResource(R.drawable.transition_a_d_a_d);

    AnimationDrawable transition = (AnimationDrawable) imageView.getDrawable();

    a_d_a_d.start();
    transition.start();

});

Listen_half.setOnClickListener((v) -> {

    ImageView imageView = (ImageView) findViewById(R.id.Chord);
    imageView.setImageResource(R.drawable.transition_a_d_a_half);

    AnimationDrawable transition = (AnimationDrawable) imageView.getDrawable();

    a_d_a_half.start();
    transition.start();

});
```

Fig. 9- Java code used to play animations and audio

The tutor features of the app also provide the user with the option to record their own attempt at playing the chord changes or songs displayed and hear their attempt compared with the correct version. As the version of Android the app is run on requires permission before audio can be recorded from a user's device, first methods were created to request user permission. Firstly the method requestPermission() was created to ask the user to allow the app to write storage and to record audio using the device's microphone. The requestPermission() method is displayed below (see Fig. 10)

```
a_major_d_major

/**
 * Method to request permission to record audio and write external storage
 */
private void requestPermission() {
    ActivityCompat.requestPermissions(a_major_d_major.this, new
        String[]{WRITE_EXTERNAL_STORAGE, RECORD_AUDIO}, RequestPermissionCode);
}
```

Fig. 10 – requestPermission() method

The `requestPermission()` method includes the overridden method `onRequestPermissionsResult()` to handle the user's response to the request. If the user grants permission the `PackageManager` is adjusted and an appropriate Toast message is displayed to the user (see Fig. 11). Conversely, if the user doesn't grant the necessary permissions another Toast message is displayed informing them of this.

```
/*
 * Method to handle user response to request permission
 */
@Override
public void onRequestPermissionsResult(int requestCode,
    String permissions[], int[] grantResults) {
    switch (requestCode) {
        case RequestPermissionCode:
            if (grantResults.length > 0) {
                boolean StoragePermission = grantResults[0] ==
                    PackageManager.PERMISSION_GRANTED;
                boolean RecordPermission = grantResults[1] ==
                    PackageManager.PERMISSION_GRANTED;

                if (StoragePermission && RecordPermission) {
                    Toast.makeText(a_major_d_major.this, "Permission Granted",
                        Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText(a_major_d_major.this, "Permission Denied", Toast.LENGTH_LONG).show();
                }
            }
    }
}
```

Fig. 11- onRequestPermissionsResult() method

The method `checkForPermission()` is also included here to return a Boolean of whether or not the user has provided the permission to record audio and write storage from the device. The constructor `ContextCompat` is used with the method `checkSelfPermission()` to check the permission status for both recording audio and writing external storage (see Fig. 12).

```
/*
 * Checks permission for record audio and storage
 */
public boolean checkForPermission() {
    int result = ContextCompat.checkSelfPermission(getApplicationContext(),
        WRITE_EXTERNAL_STORAGE);
    int result1 = ContextCompat.checkSelfPermission(getApplicationContext(),
        RECORD_AUDIO);
    return result == PackageManager.PERMISSION_GRANTED &&
        result1 == PackageManager.PERMISSION_GRANTED;
}
```

Fig. 12- checkForPermission() method

In the `onClickListener` for the button that starts recording the `checkForPermission()` method is passed as the condition in an If statement. If the result is found to be true, meaning the necessary permissions have been granted, the save path for the sound file is defined and the `RecorderSetup()` method is called (see Fig. 13). Otherwise the `requestPermission()` method is called again, until the required permissions are received.

```
a_major_d_major onCreate() new OnClickListener onClick()
buttonStart.setOnClickListener((view) -> {

    if (checkForPermission()) {

        AudioSavePathInDevice =
            Environment.getExternalStorageDirectory().getAbsolutePath() + "/" +
            CreateFileName(5) + "AudioRecording.3gp";

        RecorderSetup();

        try {
            Recorder.prepare();
            Recorder.start();
        } catch (IllegalStateException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        buttonStart.setEnabled(false);
        buttonStop.setEnabled(true);

        Toast.makeText(a_major_d_major.this, "Recording started",
            Toast.LENGTH_LONG).show();
    } else {
        requestPermission();
    }
}
```

Fig. 13- onClickListener used to start recording process

If the If statement's conditions are met the RecorderSetup() method is called. This method creates a new instance of the MediaRecorder class, as well as setting the audio source for the recorder as the device microphone. The method also defines the output format of the audio and determines the save path for the audio files (see Fig. 14).

```
/*
 * Method to setup the media recorder
 */
public void RecorderSetup() {
    Recorder = new MediaRecorder();
    Recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    Recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    Recorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
    Recorder.setOutputFile(AudioSavePathInDevice);
}
```

Fig. 14 - RecorderSetup() method

The audio save path is created by calling the CreateFileName() method which uses the StringBuidler class to help create a random file name for storing the recorded audio files (see Fig. 15).

```
/*
 * Method to create file name for user audio
 */
public String CreateFileName(int string) {
    StringBuilder stringBuilder = new StringBuilder(string);
    int i = 0;
    while (i < string) {
        stringBuilder.append(RandomAudioFileName.
            charAt(random.nextInt(RandomAudioFileName.length())));
        i++;
    }
    return stringBuilder.toString();
}
```

Fig. 15- CreateFileName() method

If the permissions have been given by the user the recorder then begins recording the user's audio by using the start() method and a Toast message is created informing the user that the device is recording. When the stop button is pressed by the user, the onClickListener for this button stops recording by calling the stop() method. A Toast message is also displayed here stating that the device is no longer recording (see Fig. 16). As audio is no longer being recorded the buttonStop button is setEnabled to false and the compareAudio and getHelp buttons are setEnabled to true as user recorded audio is required before these features can be used.

```
// onclick listener to handle stop button
buttonStop.setOnClickListener((view) -> {
    Recorder.stop();
    buttonStop.setEnabled(false);
    buttonCompareAudio.setEnabled(true);
    buttonStart.setEnabled(true);
    buttonStopPlayback.setEnabled(false);
    buttonGetHelp.setEnabled(true);

    Toast.makeText(context, a_minor_e_minor.this, text: "Recording Completed",
        Toast.LENGTH_LONG).show();
});
```

Fig. 16- code used to handle stop recording function

When the "Compare" button is pressed a new instance of the MediaPlayer class is created and the data source is set to the audio file path which was specified in the OnClickListener for the "Start" button. The user recorded audio is then played along with a Toast message stating that the user audio is being played. An onCompletionListener is then implemented to detect the end of the user recorded audio at which point the correct version of the chord sequence or song is then played allowing the user to compare the two. The code for audio playback and comparison is displayed below (see Fig. 17).

```
buttonCompareAudio.setOnClickListener((view) -> {
    buttonStop.setEnabled(false);
    buttonStart.setEnabled(false);
    buttonStopPlayback.setEnabled(true);

    mediaPlayer = new MediaPlayer();
    try {
        mediaPlayer.setDataSource(AudioSavePath+Device);
        mediaPlayer.prepare();
    } catch (IOException e) {
        e.printStackTrace();
    }

    mediaPlayer.start();
    Toast.makeText(a_major_d_major.this, "How you sound",
        Toast.LENGTH_LONG).show();

    // on completion listener to stop user audio and then play correct chord sequence
    mediaPlayer.setOnCompletionListener((mp) -> {
        mediaPlayer.stop();

        a_d_a_d.start();

        Toast.makeText(a_major_d_major.this, "How its supposed to sound",
            Toast.LENGTH_LONG).show();
    });
});
```

Fig. 17- code used to handle compare audio feature

After pressing the “Compare” button, if the user wishes to stop playback of audio then can do so by pressing the second “Stop” button. This will call the stop() and release() methods to the MediaPlayer, ending audio playback. In the java code for recording and playback of user audio, the setEnabled() method is used to disable buttons until they need to be used. For example, the “Stop” button cannot be used unless audio is currently recording and the “Compare” button cannot be used until audio has been recorded first and so on. The full code is available in the appendices (see Chord Transition class code).

After the user has recorded their own audio they can then press the “Get Help” button to send their recording to a guitar tutor to receive feedback. The functionality of the feature is handled in an onClick listener using an ACTION_SEND_MULTIPLE intent. The Intent type is set to “message/rfc822” which will display the apps the user can use to send their message. The putExtra function is used here to provide the email address of the guitar tutor as well as a subject line and some text which provides context to the person receiving the email. The feature also includes an ArrayList which locates the file name of the audio the user has just recorded and uses putParcelableArrayListExtra to include the recorded audio as an email attachment. The addFlag() method is used here to grant permission to read the URI allowing the person receiving the email to access its content. This code that handles the send audio function is displayed below (see Fig. 18).

```
// Intent to open user apps with email function
// Will create template email with the user recorded audio as an attachment
Intent i = new Intent(Intent.ACTION_SEND_MULTIPLE);
i.setType("message/rfc822");
i.putExtra(Intent.EXTRA_EMAIL, new String[]{"cmaguire82@qub.ac.uk"});
i.putExtra(Intent.EXTRA_SUBJECT, "Help with this chord transition");
i.putExtra(Intent.EXTRA_TEXT, "I'm having trouble learning this chord transition could you help? ");

ArrayList<Uri> uris = new ArrayList<Uri>();
String shareName = new String(Environment.getExternalStorageDirectory() + "/AudioRecording.mp3");
File shareFile = new File(shareName);
Uri contentUri = FileProvider.getUriForFile(getApplicationContext(), "com.example.magui.myapplication.provider", shareFile);
uris.add(contentUri);
i.putParcelableArrayListExtra(Intent.EXTRA_STREAM, uris);

// Grant temporary read permission to the content URI
i.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

String msgStr = "Share...";
startActivity(Intent.createChooser(i, msgStr));

// calls unlocked send audio method
unlockSendAudioAchievement(configuration.UNLOCK_SEND_AUDIO_ACHIEVEMENT);
```

Fig. 18 - code to handle send audio function

5.1.4 Android Manifest

To make the app run without any errors, the various activities that make up the app had to be declared in the application manifest. The manifest describes the various components, services and content that compose the application. This information must be provided to the Android system before any of the app's code can be run. Below is the declaration of some of the activities present in the app (see Fig. 19)

```
manifest application
//registering activities
<activity android:name=".Login_Register.Login"></activity>
<activity android:name=".Login_Register.Register"></activity>
<activity android:name=".MainActivity_logged_in"></activity>
<activity android:name=".Login_Register.User_Activity"></activity>
<activity android:name=".Login_Register.Login2"></activity>

//activities extended from tab 1
<activity android:name=".tab1.C_F.chord_C_handler"></activity>
<activity android:name=".tab1.C_F.chord_C_Sharp_handler"></activity>
<activity android:name=".tab1.C_F.chord_D_handler"></activity>
<activity android:name=".tab1.C_F.chord_D_Sharp_handler"></activity>
<activity android:name=".tab1.C_F.chord_E_handler"></activity>
<activity android:name=".tab1.C_F.chord_F_handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_F_Sharp_Handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_G_Handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_G_Sharp_Handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_A_Handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_A_Sharp_Handler"></activity>
<activity android:name=".tab1.Fsharp_B.chord_B_Handler"></activity>

//activities extended from tab 2
<activity android:name=".tab2.standard_tuning"></activity>
<activity android:name=".tab2.d_tuning"></activity>
<activity android:name=".tab2.open_g_tuning"></activity>
<activity android:name=".tab2.tuner.TunerActivity"></activity>
```

Fig. 19- Activities declared in the app manifest

The manifest also contains the necessary permissions the app requires. These were necessary to allow the application to have access to protected parts of the API and to interact with other applications. For example, the permission to record audio, the permission to write storage and the permission to access the internet were declared at the top of the manifest as these features are protected in Android (see Fig. 20). If the essential permissions were not declared, attempting to use the activities which required the permissions would cause the app to crash or function incorrectly.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.magui.myapplication"
    android:installLocation="preferExternal"
    >

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"></uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Fig.20- declaration of permissions in manifest

5.1.5 Navigation

User navigation through the app was predominately handled through use of OnClickListener and Intents. Buttons designed to allow the user to access different parts of the app were set with OnClickListener which contained an overridden onClick() method. The findViewById() method is used to cast the onClick() to the various buttons included in the xml code. In the onClick() method a new instance of the Intent class was created with getActivity() (or in some cases getBaseActivity()) with the class of the new activity passed as the context. The new activity is then called with the startActivity() method which includes the name of the intent passed as an argument (see Fig. 21).

```
// will open the "learn chord changes" page
Button chordButton = (Button) rootView.findViewById(R.id.learn_chord_changes);
chordButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getActivity(), chord_changes_page.class);
        startActivity(intent);
    }
});
```

Fig.21- Example of using onClick() to open new activity

5.1.6 Finger placement section

For the finger placements tutorial section of the app, the view pager technique was implemented in a similar fashion to the chord library and the main activity tabbed layout, allowing the user to easily swipe between an inflated view of the tutorial xml layouts. Again, a switch statement was used with the FragmentStatePagerAdapter to return new instances of each class that handles a specific tutorial (see Fig. 22).

```
finger_placements_handler
private class ScreenSlidePagerAdapter extends FragmentStatePagerAdapter {
    public ScreenSlidePagerAdapter(FragmentManager fm) { super(fm); }

    /**
     * Method which implements switch to display content in the page viewer
     */
    @Override
    public android.support.v4.app.Fragment getItem(int position) {
        switch (position) {
            case 0:
                //change to c major
                return finger_placements_a_major.newInstance("D Major");
            case 1:
                return new finger_placements_b_major();
            case 2:
                return new finger_placements_c_major();
            case 3:
                return new finger_placements_d_major();
            case 4:
                return new finger_placements_e_major();
            case 5:
                return new finger_placements_f_major();
            case 6:
                return new finger_placements_g_major();
            default:
                return null;
        }
    }
}
```

Fig.22- code used to return new instances of tutorial classes

The xml code of these tutorials consists of a graphic of a guitar neck set to the size of the device screen. The xml layout of the tutorial page is rotated horizontally to better mimic the layout of a guitar to the user. This would also allow the user to hold their phone as they would a guitar to support their learning. These layouts also contain a number of buttons which are colour coded in a manner consistent with the finger placement diagrams used elsewhere in the app and arranged in the shape of the chord relative to the specific tutorial. This idea is reinforced to the user with TextViews assigning colours to each of the fingers. The code for teaching the finger placements takes the form of an array which tracks button clicks from the user. Firstly, an array is declared and given three spaces. If the user presses the correct buttons in sequence the value in the array is incremented until the number of notes in the chord is met and a success message appears. If, however the buttons are pressed in the wrong order the value in the array is reset to zero and a message is displayed telling the user to try again. After the first button press the subsequent presses include for loops to change to values in the array for the success condition to be met. The code to handle the button press sequence is displayed below (see Fig. 23.)

The image shows a snippet of Java code within an IDE. The code is titled 'finger_placements_a_major'. It defines three click listeners for buttons. The first listener for 'button1' sets 'tracker[0] = 1;'. The second listener for 'button2' uses a for loop to check if 'tracker[i] == 0' for i from 0 to 1. If true, it shows a 'try again!' toast and resets 'tracker[i] = 1;'. The third listener for 'button3' uses a for loop to check if 'tracker[i] == 0' for i from 0 to 2. If true, it shows a 'try again!' toast and resets 'tracker[i] = 1;'.

```
finger_placements_a_major
1 button1.setOnClickListener((v) -> {
2     tracker[0] = 1;
3 });
4 button2.setOnClickListener((v) -> {
5     for (int i = 0; i < 1; i++) {
6         if (tracker[i] == 0) {
7             Toast.makeText(getActivity(), text "try again!",
8                 Toast.LENGTH_SHORT).show();
9         } else {
10            tracker[i] = 1;
11        }
12    }
13 });
14 button3.setOnClickListener((v) -> {
15     for (int i = 0; i < 1; i++) {
16         if (tracker[i] == 0) {
17             Toast.makeText(getActivity(), text "try again!",
18                 Toast.LENGTH_SHORT).show();
19         } else {
20            tracker[i] = 1;
21        }
22    }
23 });
```

Fig.23- code used to track user finger placement

5.1.7 Database

Another feature of the application is the ability for users to register their details to a database hosted on a server and use these details to log into the app. Once the user is registered the app tracks the user's progress through the various features of the app. This functionality was achieved through a mixture of Java, SQL and PHP code. Firstly, the database was designed to include three tables, the first of which would hold the user details which would be obtained from the app. These include the user's first and last names, a username and a password. Upon registering, each user is also given a unique user id. The second table of the database includes the achievement the user unlocks via use of the app. Lastly the final table in the database displays the achievements the user has unlocked. The

following schema was then derived by normalising the data to be included within the tables (see Fig. 24).

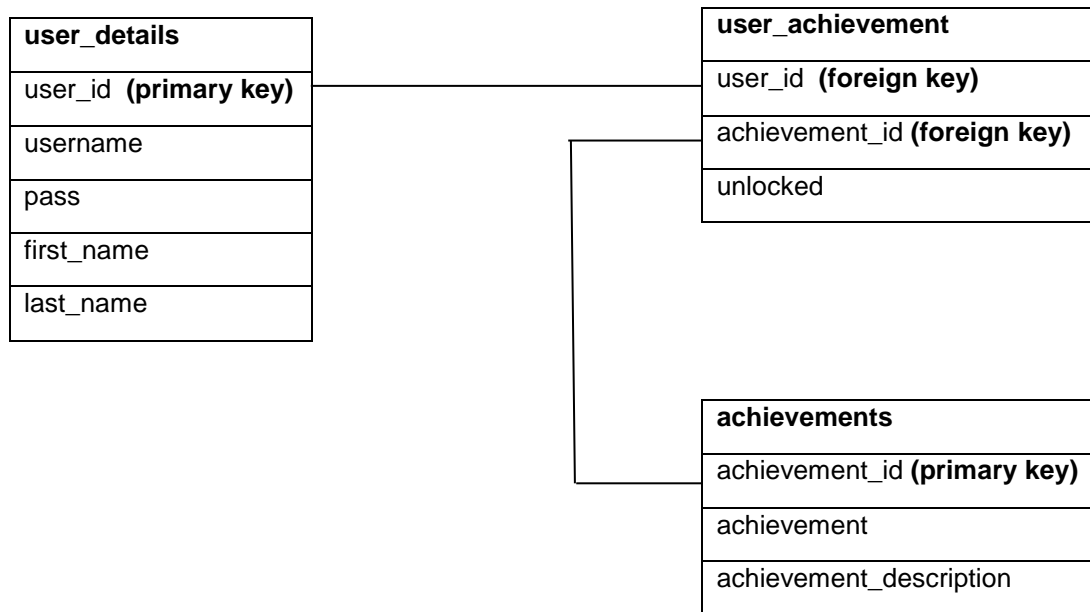


Fig.24- Database schema

In the database the `userId` attribute is set to auto-increment so the id number increases with each new user. This attribute also acts as a foreign key in the `user_achievement` table allowing the system to keep track of what achievements the user has unlocked. The foreign key `achievement_id` is also present in the `user_achievement` table to enable tracking of user achievements.

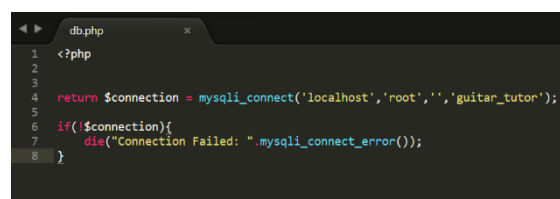
The database to hold the user information was created using the following SQL statement in phpMyAdmin (see Fig.24)

```
CREATE TABLE user_details(  
  userid INT(10) NOT NULL AUTO_INCREMENT,  
  username VARCHAR (20) NOT NULL,  
  pass VARCHAR(40)NOT NULL,  
  first_name VARCHAR(20)NOT NULL,  
  last_name VARCHAR(30)NOT NULL,  
  PRIMARY KEY (userid)  
);  
CREATE TABLE achievements(  
  achievement_id INT(10) NOT NULL AUTO_INCREMENT,  
  achievement VARCHAR(30),  
  text VARCHAR(100),  
  PRIMARY KEY (achievement_id)  
);  
CREATE TABLE user_achievement(  
  userid INT(10),  
  achievement_id INT(10),  
  unlocked boolean NOT NULL default 0,  
  FOREIGN KEY (userid) REFERENCES user_details(userid),  
  FOREIGN KEY (achievement_id) REFERENCES achievements(achievement_id)  
);
```

Fig. 24- SQL statement used to create database

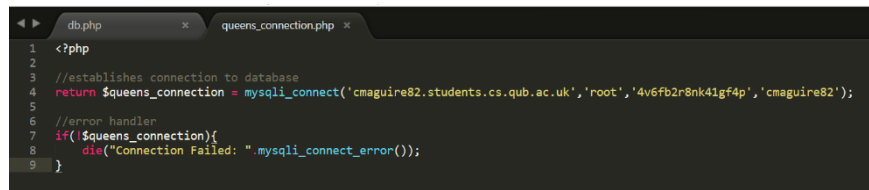
5.1.8 PHP code

Once the database to hold the user details was created successfully a series of PHP files were created to enable the database server and the application to communicate. The first of these PHP files to be created was a connection to establish a link to the database. The file returns a variable which acts as the connection to the phpMyAdmin database mentioned above. The function `mysqli_connect()` takes the database host, the username, the password and the name of the database as arguments. The PHP code used to connect to the database is displayed below (see Fig. 24 & Fig. 25).



```
db.php  
1 <?php  
2  
3  
4 return $connection = mysqli_connect('localhost','root','','guitar_tutor');  
5  
6 if(!$connection){  
7   die("Connection Failed: ".mysqli_connect_error());  
8 }
```

Fig. 24- Local host connection .php file

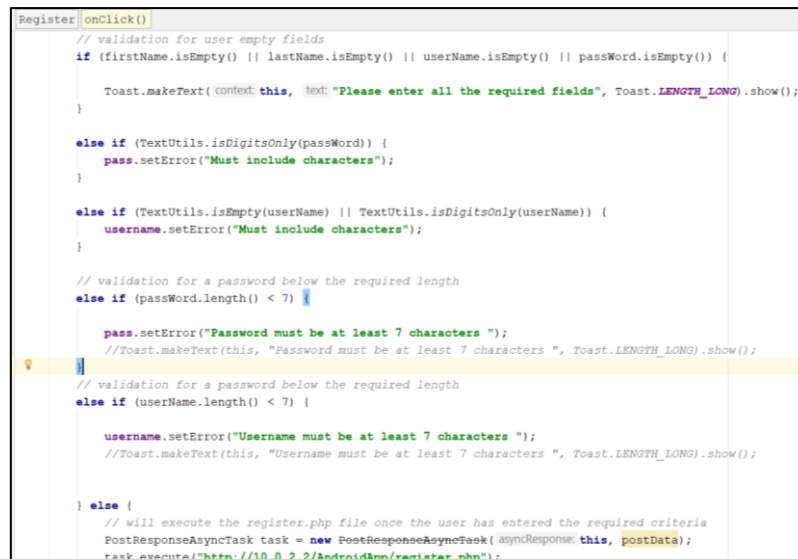


```
1 <?php
2
3 //establishes connection to database
4 return $queens_connection = mysqli_connect('cmaguire82.students.cs.qub.ac.uk','root','4v6fb2r8nk41gf4p','cmaguire82');
5
6 //error handler
7 if(!$queens_connection){
8     die("Connection Failed: ".mysqli_connect_error());
9 }
```

Fig. 25- Queens server connection .php file

5.1.9 Register function

Within the application users can register by providing a first name, last name, user name and password. The validation for these required fields was handled in the register class with If else statements to ensure that all the necessary fields were entered. Statements were also included to add validation so that a minimum number of characters have to be entered for the username and password, and that the user's given password can't be numbers only (see Fig. 26).



```
Register:onClick()
// validation for user empty fields
if (firstName.isEmpty() || lastName.isEmpty() || userName.isEmpty() || passWord.isEmpty()) {
    Toast.makeText(this, "Please enter all the required fields", Toast.LENGTH_LONG).show();
}

else if (TextUtils.isDigitsOnly(passWord)) {
    pass.setError("Must include characters");
}

else if (TextUtils.isEmpty(userName) || TextUtils.isDigitsOnly(userName)) {
    username.setError("Must include characters");
}

// validation for a password below the required length
else if (passWord.length() < 7) {
    pass.setError("Password must be at least 7 characters ");
    //Toast.makeText(this, "Password must be at least 7 characters ", Toast.LENGTH_LONG).show();
}

// validation for a password below the required length
else if (userName.length() < 7) {
    username.setError("Username must be at least 7 characters ");
    //Toast.makeText(this, "Username must be at least 7 characters ", Toast.LENGTH_LONG).show();
}

else {
    // will execute the register.php file once the user has entered the required criteria
    PostResponseAsyncTask task = new PostResponseAsyncTask(asyncResponse: this, postData);
    task.execute("http://10.0.2.2/AndroidApp/register.php");
}
```

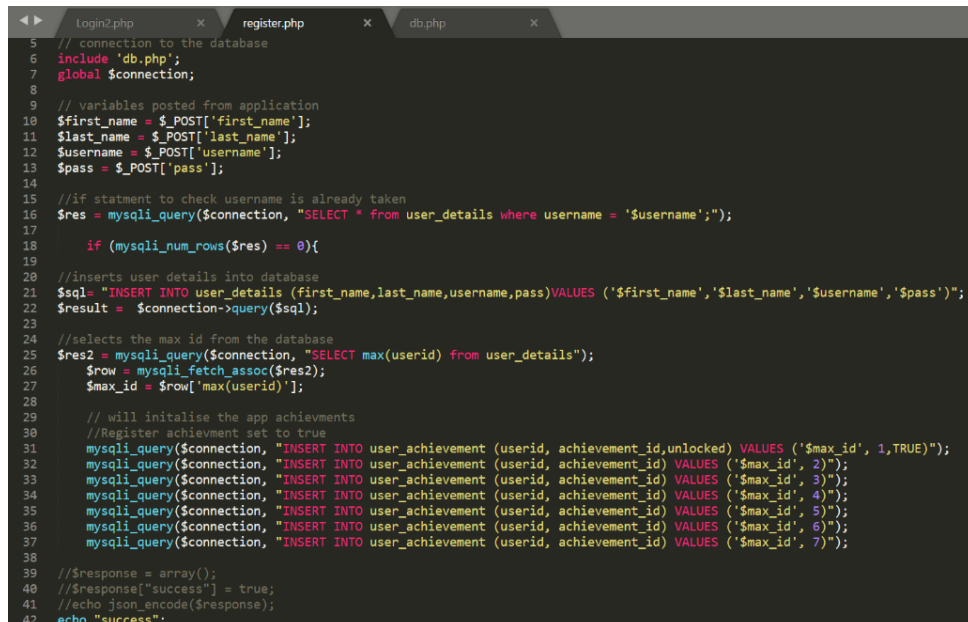
Fig. 26- Validation for the register function

When the conditions of the If statement are met, then the register.php file will be executed with the data the user has entered being sent to the server using a HashMap and the PostResponseAsyncTask class.

The register.php file which is used to register the user to the database takes the information posted by and then inserts into the user_details table of the database.

Firstly, variables are assigned to the data posted from the app. Then then a query is executed to ensure that a user hasn't already registered to the database with the same user name. If the username is not already taken an insert statement will be executed which enters the information into the user_details table of the database. The final part of the register.php file handles initialisation of the user achievements. A SQL query is used to retrieve the id of the user that has just been added to the

database, this value is then assigned the \$max_id variable. A series of insert statements are made to set up the various achievements that the user can later unlock by using the app. The first of these, the register achievement is set to "true" as the user has just registered successfully. Finally, the server echoes "success" to the app indicating the registration has been successful. The code used to insert the user details into the server database is displayed below (see Fig. 27).



```

5 // connection to the database
6 include 'db.php';
7 global $connection;
8
9 // variables posted from application
10 $first_name = $_POST['first_name'];
11 $last_name = $_POST['last_name'];
12 $username = $_POST['username'];
13 $pass = $_POST['pass'];
14
15 //if statement to check username is already taken
16 $res = mysqli_query($connection, "SELECT * from user_details where username = '$username'");
17
18 if (mysqli_num_rows($res) == 0){
19
20 //inserts user details into database
21 $sql= "INSERT INTO user_details (first_name,last_name,username,pass)VALUES ('$first_name','$last_name','$username','$pass')";
22 $result = $connection->query($sql);
23
24 //selects the max id from the database
25 $res2 = mysqli_query($connection, "SELECT max(userid) from user_details");
26 $row = mysqli_fetch_assoc($res2);
27 $max_id = $row['max(userid)'];
28
29 // will initialise the app achievements
30 //Register achievement set to true
31 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id,unlocked) VALUES ('$max_id', 1,TRUE)");
32 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 2)");
33 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 3)");
34 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 4)");
35 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 5)");
36 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 6)");
37 mysqli_query($connection, "INSERT INTO user_achievement (userid, achievement_id) VALUES ('$max_id', 7)");
38
39 //$response = array();
40 //$response["success"] = true;
41 //echo json_encode($response);
42 echo "success";

```

Fig. 27- register.php file which inserts user details to the database

To gather the user details from the app a register page was created in an xml layout with editText widgets for each of the four details necessary to register and a button that would initiate the process. The register class in the java code implements AsyncResponse which is an open source class that allows data to be posted from android. An onClick listener was specified for the register button present in the xml. Pressing the button creates a HashMap which takes user entered data from each of the Edit Texts and posts the information to the register.php page as the necessary variables. These variables are then used in the Insert statements to add the user's details to the database. The onClick which posts the user variables is displayed below (see Fig. 28).



```

@Override
public void onClick(View v) {

    HashMap postData = new HashMap();
    postData.put("first_name", first_name.getText().toString());
    postData.put("last_name", last_name.getText().toString());
    postData.put("username", username.getText().toString());
    postData.put("pass", pass.getText().toString());
    PostResponseAsyncTask task = new PostResponseAsyneTask(this, postData);
    task.execute("http://10.0.2.2/AndroidApp/register.php");
}

```

Fig.28- HashMap used to post user details

Once this process has finished the onFinish() method is called which corresponds to the echo "success" defined in the register.php file indicating the insert statement has been executed and the user has been registered successfully. If the details the user has provided have been posted and inserted to the database successfully a Toast message is displayed to the user stating their registration has been a success and they're brought to the user activity of the app. Conversely a message is display if the user registration has not been successful (see Fig. 29).

```
@Override
public void processFinish(String result) {

    if(result.equals("success")) {

        Toast.makeText(this, "Register Successful", Toast.LENGTH_LONG).show();

        Intent intent = new Intent(getBaseContext(), User_Activity.class);
        startActivity(intent);

    }else{
        Toast.makeText(this, "Register Failed", Toast.LENGTH_LONG).show();
    }
}
```

Fig. 29- If statement handling registration outcome

5.1.10 Login function

Similar to the register function, a login function was also created using PHP code which instead of requiring all the user details would need only the username and password which would be taken from text entered by the user into the app, and then sent to the server using the POST() method. The main difference regarding the login and register files is the login PHP file uses a SELECT statement to retrieve information from the database rather than an INSERT statement which the register file uses. The PHP code to retrieve data the user's data is shown below (see Fig. 29).

```
register.php x Login2.php x
1 <?php
2
3 if($_SERVER['REQUEST_METHOD']=='POST'){
4     //Getting values
5     $username = $_POST['username'];
6     $pass = $_POST['pass'];
7
8     //Creating sql query
9     $sql = "SELECT * FROM user_details WHERE username='$username' AND pass='$pass'";
10
11     //importing dbConnect.php script
12     require_once('db.php');
13
14     //executing query
15     $result = mysqli_query($connection,$sql);
16
17     //fetching result
18     $check = mysqli_fetch_array($result);
19
20     //if we got some result
21     if(isset($check)){
22         //displaying success
23         echo "success";
24     }else{
25         //displaying failure
26         echo "failure";
27     }
28     mysqli_close($connection);
29 }
```

Fig. 29- login.php file

If the SQL query is successful, then the server will echo “success” back to the app allowing the user to log in. Upon receiving the message of success from the server the system configuration sets the shared preferences Boolean for the “LOGGEDIN” variable to “true”, as well as setting the user’s specific username to the variable in the configuration class using shared preferences. The Editor class is used to commit these changes. Then an Intent is used to bring the user to the user activity of the app. The code which hands the login response from the server is displayed below (see Fig. 30, for the full login.class code refer to the appendices section 8.1)

```
//If we are getting success from server
if (response.equalsIgnoreCase(configuration.LOGIN_SUCCESS)) {

    //Creating a shared preference
    SharedPreferences sharedPreferences = Login.this.getSharedPreferences(configuration.SHARED_PREF_NAME, Context.MODE_PRIVATE);

    //Creating editor to store values to shared preferences
    SharedPreferences.Editor editor = sharedPreferences.edit();

    //Adding values to editor
    editor.putBoolean(configuration.LOGGEDIN_SHARED_PREF, true);
    editor.putString(configuration.USERNAME_SHARED_PREF, username);

    //may need to be changed!!
    //editor.putString(configuration.KEY_USER_ID, configuration.getUserID());

    //Saving values to editor
    editor.commit();

    //Starting user profile activity
    Intent intent = new Intent(packageContext: Login.this, User_Activity.class);
    startActivity(intent);
    Toast.makeText(context: Login.this, text: "Login successful!", Toast.LENGTH_LONG).show();

} else {
    //If the server response is not "success"
    //Displaying an error message on toast
    Toast.makeText(context: Login.this, text: "Invalid username or password", Toast.LENGTH_LONG).show();
}
```

Fig. 30- If statement which handles login outcome

The login functionality was achieved in the java code by firstly by creating a new instance of the StringRequest class. The .POST method is then requested with the URL of the login.php file passed as the argument, indicating this is the where the user information is being sent. Following this a new Response .listener is created to detect the response from the server. The user information is taken from editText widgets in the xml layout file and converted to String format before it is sent to the server. The code that hands the created of a StringRequest and conversion of data from editTexts to Strings is displayed below (see Fig. 31).

```
private void login() {
    //Getting values from edit texts
    final String username = editTextUsername.getText().toString().trim();
    final String pass = editTextPassword.getText().toString().trim();

    //Creating a string request
    StringRequest stringRequest = new StringRequest(Request.Method.POST, configuration.LOGIN_URL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
```

Fig. 32- section of the login class

The user data taken from the editTexts is posted to the server using the HashMap class. The HashMap is used to map the given user data to values. A new instance of the RequestQueue class is then created to put the login request in a queue by using the .add() method to add the login URL (see Fig. 33).

```
@Override
protected Map<String, String> getParams() throws AuthFailureError {
    Map<String, String> params = new HashMap<>();
    //Adding parameters to request
    params.put(configuration.KEY_USERNAME, username);
    params.put(configuration.KEY_PASS, pass);

    //returning parameters
    return params;
}

//Adding the string request to the queue
RequestQueue requestQueue = Volley.newRequestQueue(context, this);
requestQueue.add(stringRequest);
```

Fig. 33- HashMap and RequestQueue

To aid the login process a configuration class was created in the java code which includes Strings stored to the shared preferences, including the username, password and the “success” string which acts as the response from the server. An excerpt of the configuration class is displayed below (see Fig. 34)

```
//Keys for email and password as defined in login.php
public static final String KEY_USERNAME = "username";
public static final String KEY_PASS = "pass";
public static final String KEY_USER_ID = "userid";

//If server response is equal to this that means login is successful
public static final String LOGIN_SUCCESS = "success";

// Corresponds to database
public static final String SHARED_PREF_NAME = "guitar_tutor";

// Username
public static final String USERNAME_SHARED_PREF = "username";

//Used to track if user is logged in/out
public static final String LOGGEDIN_SHARED_PREF = "loggedin";

/*
```

Fig. 34- sample of configuration class

Logout function 5.1.11

Once the user has logged in successfully they are brought to the UserActivity area which displays the user achievements within the app and enables the user to then logout of the app. This function is achieved by creating a new instance of the AlertDialog.Builder class where a message is set asking the user if they want to log out. The “Yes” option is set with a DialogInterface.OnClickListener which handles the log out process. The onClick() is programmed to, amongst other features set the LOGGEDIN_SHARED_PREF to “false” and then take the user back to the app’s MainActivity by use of Intent. The code used to handle the logout function is displayed below (see Fig. 35)

```
//Logout function
private void logout() {
    //Creating an alert dialog to confirm logout
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
    alertDialogBuilder.setMessage("Are you sure you want to logout?");
    alertDialogBuilder.setPositiveButton("Yes",
        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface arg0, int arg1) {

                //Getting out sharedpreferences
                SharedPreferences preferences = getSharedPreferences(configuration.SHARED_PREF_NAME, Context.MODE_PRIVATE);
                //Getting editor
                SharedPreferences.Editor editor = preferences.edit();

                //Putting the value false for loggedin
                editor.putBoolean(configuration.LOGGEDIN_SHARED_PREF, false);

                //Putting blank value to email
                editor.putString(configuration.USERNAME_SHARED_PREF, "");

                //Saving the sharedpreferences
                editor.commit();

                //Starting login activity
                Intent intent = new Intent(getBaseContext(), MainActivity.class);
                startActivity(intent);
            }
        });
}
```

Fig. 35- logout() method

The logout() method also includes an onCreateOptionsMenu() method which displays the option to logout from the app’s settings menu. This is achieved by using the getMenuInflater() method. Also present is a method to handle whichever option is selected by the user. These features are displayed below (see Fig. 36)

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //Adding our menu to toolbar
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.menuLogout) {
        //calling logout method when the logout button is clicked
        logout();
    }
    return super.onOptionsItemSelected(item);
}
```

Fig.36- code which handles logout menu

5.1.12 Achievements feature

Another feature present in the app is the tracking of user use of the app. This was achieved, firstly by creating a series of .php files to update the user_achievements section of the database. These files are included as Strings in the Configuration class, so they can be easily accessed by using the POST method.

```
configuration
Configuration for hosting app locally
*/
public class configuration {

    //URL to our login.php file
    public static final String LOGIN_URL = "http://10.0.2.2/AndroidApp/Login2.php";

    //URL to .php file which returns an array response
    public static final String LOGIN_URL2 = "http://10.0.2.2/AndroidApp/loginArray.php";

    //Strings referring to .php files which will unlock user achievements
    //public final String UNLOCK_REGISTER_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Register_Achievement.php";
    public static final String UPDATE_BASIC_CHORDS_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Beginner_Chords_Achievement.php";
    public static final String UPDATE_INTERMEDIATE_CHORDS_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Intermedite_Chords_Achievement.ph
    public static final String UNLOCK_RECORD_AUDIO_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Record_Audio_Achievement.php";
    public static final String UPDATE_BASIC_RIFFS_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Basic_Riffs_Achievement.php";
    public static final String UPDATE_INTERMEDIATE_RIFFS_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/INtermediate_Riffs_Achievement.php
    public static final String UNLOCK_SEND_AUDIO_ACHIEVEMENT = "http://10.0.2.2/AndroidApp/Send_Audio_Achievement.php";

    //view achievements.php files needs to be created
    public final String VIEW_UNLOCKED_ACHIEVEMENTS = "http://10.0.2.2/AndroidApp/View_Achievements.php";
```

Fig. 32- .php files included in the Configuration class

If the user carries out a process that earns them an achievement their details in the database are updated to set the achievement as unlocked. This is achieved in code by creating a StringRequest which posts user data. This data is placed in a HashMap where the user's username and password are taken from the Configuration class, as they have already been saved during the login process via the SharedPreferences class. The request is then added to a RequestQueue (see Fig. 33).

```
public void unlockRecordAudioAchievement(String phpURL) {

    StringRequest request = new StringRequest(Request.Method.POST, phpURL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                //doesn't return anything
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                //error response
            }
        }
    ) {
        @Override
        protected Map<String, String> getParams() {
            Map<String, String> params = new HashMap<>();
            //params.put("user_id", configuration.KEY_USER_ID);
            params.put("username", configuration.KEY_USERNAME);
            params.put("pass", configuration.KEY_PASS);

            return params;
        }
    };

    requestQueue.add(request);
```

Fig. 33- HashMap and RequestQueue for unlock achievement function

The .php files for unlocking the user achievements contain an initial query that selects all the details from the row that matches the username and password posted from the application. If the query is successful a second query is then executed which selects the user id of the relevant user. Finally, another query is executed which the previously selected user id, which acts as the foreign key in the user_achievements table. This query then sets the value of in the “unlocked” row to “1” for the achievement_id which pertains to the achievement the user has just unlocked. An example of a .php file used to unlock achievements in the database is displayed below (see Fig. 34).

```
1 <?php
2 // connection to the database
3 include 'db.php';
4
5 $username = $_POST['username'];
6 $pass = $_POST['pass'];
7
8 //if statment to select row with corresponding password and username
9 $res = mysqli_query($connection, "SELECT * from user_details where username = '$username' AND pass = '$pass'");
10
11 if (mysqli_num_rows($res) == 1){
12
13     //selects the id from the database
14     $res2 = mysqli_query($connection, "SELECT userid from user_details where username = '$username' AND pass = '$pass'");
15     $row = mysqli_fetch_assoc($res2);
16     $userid = $row['userid'];
17
18     //updates user achievement
19     mysqli_query($connection, "UPDATE user_achievement SET unlocked = 1 WHERE achievement_id=7 AND userid=$userid");
20 }
```

Fig. 34- Send_Audio_Achievement.php file

5.1.13 Chromatic Tuner feature

In addition to the previously developed feature allowing users to tune their guitar by ear, an actual chromatic tuner was integrated into the project from existing open source code. The feature works by detecting the pitch of the user's instrument and displaying to the user if the note is in tune or whether the pitch is too high or too low. The user is also given the option of choosing the tuning they want to tune their guitar to.

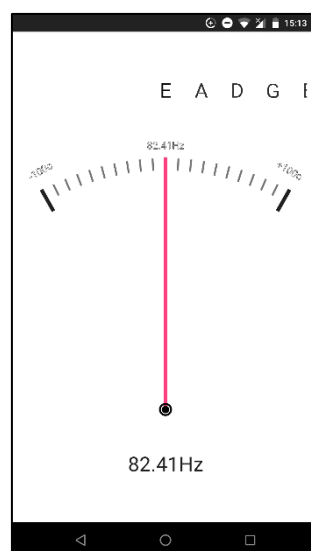


Fig. 35- Chromatic tuner

5.2 Testing specification and justification

For the testing of the project it was decided that the testing would be two-fold with a combination of user based testing to test from a user-oriented perspective and unit testing to test the java code.

5.2.1 Requirements testing

Part of the testing was based on a user perspective and derived from the following project requirements:

- The app should include a library of chords
- The app should include a tuning feature
- The app should contain a tutorial section for chord changes
- The app should contain a tutorial section for guitar riffs
- The app should contain a tutorial section for teaching finger placements
- The user should be able to register to a database by providing their details
- The user should be able to login to the app
- The user once logged in should be able to log out

In addition to the above requirements, for the purposes of test, a navigational requirement was added, to test the user would be able to access the various features of the app successfully. These requirements were tested by running the application on emulator and on device to check the various functions of the application would work when the user attempts to use them.

5.2.2 Testing process

Test conditions

- Firstly, a series of test conditions were drawn up based on the requirements specification. These conditions were assigned a test condition Id to aid test traceability, and were each given a brief description. The conditions were also assigned a level of priority based on how essential the condition is to the success of the finished system. For clarity, the source of the requirement that is being tested is also displayed. An extract from test conditions page of the test spreadsheet is displayed below (see Fig. 1).

Test Condition ID	Description	Source	Priority
Navigation_Tcon_1	To ensure a user can navigate successfully through the application	Requirement Specification	High
Register_Tcon_1	To ensure a user can register to the app by providing valid details	Requirement Specification	High
Register_Tcon_2	To ensure a user can't register to the app by providing invalid details	Requirement Specification	High
LogIn_Tcon_1	To ensure a registered user can login to the app by providing details	Requirement Specification	High
LogIn_Tcon_2	To ensure a registered user can't login to the app by providing invalid details	Requirement Specification	High
Play_Audio_Tcon_1	To ensure buttons in the application play the correct audio files	Requirement Specification	High
Play_Animation_Tcon_1	To ensure buttons in the application play the correct animation tutorials	Requirement Specification	High
Record_Audio_Tcon_1	To ensure "Record" buttons in the application record the user's audio	Requirement Specification	High

Fig. 1- Sample of test conditions

Test procedure

- After determining the test conditions, a series of test procedures were designed to test the various system requirements. Like the test conditions, each test procedure was assigned a test procedure id as well as including the relevant test case. Also, a description of the procedure was added. This would act as the instructions to be executed when carrying out the individual test cases. An extract from test procedure page of the test spreadsheet is displayed below (see Fig. 2).

Test Procedure ID	Test Case ID	Description	Comments
Register_Tproc_1	Register_Tcase_1	Attempt to register as a user by providing details that meet the specified criteria and pressing the "Register" button	N/A
Register_Tproc_2	Register_Tcase_2	Attempt to register as a user by providing incorrect details and pressing the "Register" button	N/A
Register_Tproc_3	Register_Tcase_3	Attempt to register as a user by providing no details and pressing the "Register" button	N/A
LogIn_Tproc_1	LogIn_Tcase_1	Attempt to login to the app by providing correct login details and pressing the "Login" button	N/A
LogIn_Tproc_2	LogIn_Tcase_2	Attempt to login to the app by providing incorrect login details and pressing the "Login" button	N/A
LogIn_Tproc_3	LogIn_Tcase_3	Attempt to login by providing no login details and pressing the "Login" button	N/A
LogIn_Tproc_4	LogIn_Tcase_4	While logged in, attempt to log out of the app using the "Logout" button and pressing the "Login" button	N/A
Navigation_Tproc_1	Navigation_Tcase_1	Attempt to access "Tuning" tab (tab 2) from "Chord Library" tab (tab 1) using swipe function	N/A

Fig.2 - Test procedure sample

Test cases

- Finally, the individual test cases were carried out using the test procedures as a guide line. Again, each test case was given a specific name and Id to aid traceability. Each test case also includes a specified objective for what the test case is trying to test. For example, to test if the user can login once they have registered their details. Also included in the test case details are the preconditions that must be fulfilled before the test case can be executed. For certain test cases test data was also included, this was primarily used for testing the login and register functions, with the test data in these cases taking the form of usernames and passwords. The expected result of the tests is also included, also with the pertaining test conditions and their priority. The test status is also displayed here with the options being passed, failed or not executed. An extract from test cases page of the test spreadsheet is displayed below (see Fig. 3).

Test case ID	Objective	Preconditions	Test data	Expected Result	Test condition(s)	Priority	Test completion date	Status	Tester
Navigation_Tcase_1	Check user can navigate from the "Chord Library" tab to the "Tuning" tab using the swipe function	Must be on "Chord Library" tab	N/A	User is brought to the "Tuning" tab	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_2	Check user can navigate from the "Tuning" tab to the "Tutor" tab using the swipe function	Must be on "Tuning" tab	N/A	User is brought to the "Tutor" tab	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_3	Check user can navigate from the "Tutor" tab to the "Tuning" tab using the swipe function	Must be on "Tutor" tab	N/A	User is brought to the "Tuning" tab	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_4	Check user can navigate from the "Tuning" tab to the "Chord Library" tab using the swipe function	Must be on "Tuning" tab	N/A	User is brought to the "Chord Library" tab	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_5	Check user can access "Register" page from "Chord Library" tab using TextView hyperlink	Must be on "Chord Library" tab	N/A	User is brought to the "Register" page	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_6	Check user can access "Login" page from "Chord Library" tab using TextView hyperlink	Must be on "Chord Library" tab	N/A	User is brought to the "Login" page	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire
Navigation_Tcase_7	Check user can access "Register" page from "Tuning" tab using TextView hyperlink	Must be on "Tuning" tab	N/A	User is brought to the "Register" page	Navigation_Tcon_1	High	14/01/2018	Passed	Colm Maguire

Fig. 3- Test cases sample

Defects

- The test cases section of the spreadsheet also contains an area for logging any effects found during execution of the test cases. As with all the previous test information each defect found is given an Id. The severity of the defect is also logged based on the defects impact of the system. The dates the defects were found and fixed are also displayed here. A summary of the defect is also provided, along with a description/ analysis of the defect. An extract from test conditions page of the test spreadsheet is displayed below (see Fig. 4).

Defect ID	Defect Severity	Open Date	Close Date	Summary	Description / Comment
Navigation_Defect_1	High	14/01/2018	N/A	Chromatic tuner crashes upon opening	Null pointer exception error logged in stack trace. Possible reference to null xml value in java code

Fig. 4 - Defect example

The full test spreadsheet for the system requirements is included in the electronic submission of the project.

5.2.2 Java Unit testing

Testing the project also took the form of unit testing, some of which employed the use of the Robolectric plugin. Below is an example of a unit test used to test the intents on a page in the application. The tests are designed to check the right page is opened when the corresponding buttons are clicked by the user.

```
/**
 * Created by magui on 29/01/2018.
 * Tests the intents on the chord transitions page of the app
 */
@RunWith(RobolectricTestRunner.class)
public class chord_transitions_test {

    /**
     * Test to prove clicking E_minor_C_major button opens the correct tutorial page
     */
    @Test
    public void Eminor_CMajor_Test() {
        chord_changes_page activity = Robolectric.setupActivity(chord_changes_page.class);
        activity.findViewById(R.id.E_Minor_C_Major).performClick();

        Intent expectedIntent = new Intent(activity, e_minor_c_major.class);
        Intent actual = ShadowApplication.getInstance().getNextStartedActivity();
        Assert.assertEquals(expectedIntent.getComponent(), actual.getComponent());
    }

    /**
     * Test to prove clicking A_minor_E_major button opens the correct tutorial page
     */
    @Test
    public void Aminor_EMinor_Test() {
        chord_changes_page activity = Robolectric.setupActivity(chord_changes_page.class);
        activity.findViewById(R.id.A_Minor_E_Minor).performClick();
    }
}
```

Fig. 5- Unit test example

Once the test has been run log details appear, displaying whether or not that test has passed successfully (see Fig. 6). The full unit tests are included in the electronic submission of the project.

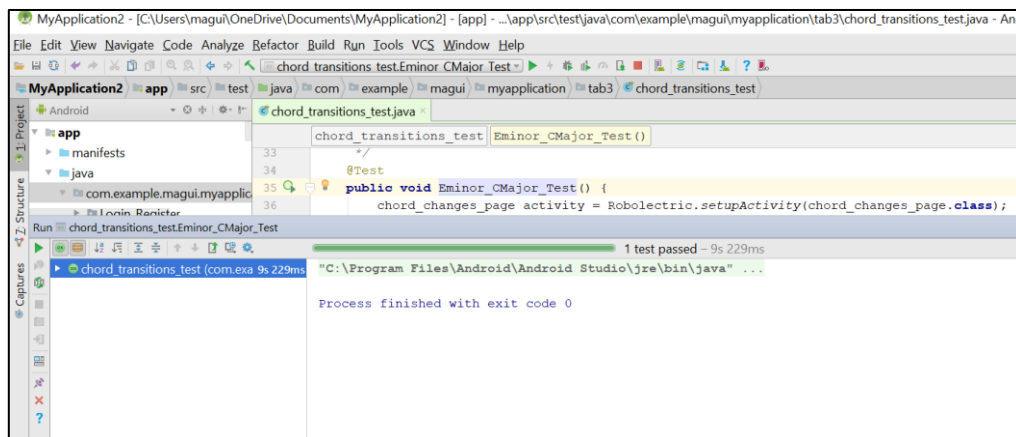


Fig. 6- Unit test which has passed

Chapter 6: Evaluation and Conclusion

6.1 Introduction

In conclusion the project was quite challenging with each section presenting its own unique difficulties and problems to overcome. However, the project was reasonably successful as it achieved much of what was set out in the proposed solution and the requirements specification, though there are a few areas of the project that could have been improved upon or developed more fully. This can be partly attributed due to the scaling back of the original project idea a few weeks into the development period, this meant time was limited for the development of the revised project. However, that being said that project included all of the features that were planned in the proposed solution with some, such as the tutor sections being developed fully to what was specified in the requirement specification as well as addressing many failings present in current systems.

6.2 Evaluation of objectives

In terms of carrying out the objectives defined in the problem specification and the project plan, the project was a success as all the key points and features are present in the finished product. The produced software is also successful in the sense that it addresses many of the problems identified in existing solutions to the problem of learning guitar. In, particular the inclusion of a feedback system is an effective solution to the problem of current applications and resources providing little opportunity for user to receive personalised guidance and instruction. The project is also successful in the sense that it runs cohesively, with the various components and interfaces working together effectively.

6.3 Evaluation of project management

Project and time management over the course of the project was reasonably successful as the required features of the project were delivered. However, the quality of the final project could have been improved with more time spent during the planning phase as a more defined idea of the features, layout and design of the app could have been developed earlier on in the project lifecycle. The use of the agile method of development during the project mean that regulating time was difficult during the project. For example, if an issue arose due to the addition of a new feature in the code, that issue had to be resolved before any further progress was made. This is an area where having a more definitive approach would have been beneficial.

6.4 Areas that could be improved

Despite meeting the objectives set out in the problem specification and the requirements specification certain areas of the project could have been developed upon and improved in future versions of the application. An example of an area which could be further developed is the user account side of the application. In future iterations of the project, this area could be improved by providing more activities

for the user to personalise their account, for example by allowing them to include a profile picture or by somehow integrating the feature to social media. The achievements feature of the application, whereby the user unlocks achievements by performing certain actions within the app could also be developed further. For example, by providing more of an incentive to complete the tasks to unlock achievements users would be encouraged to use the app more. This incentive could take the form of extra content like new chord transitions or songs to learn that the user can unlock once they have reached a certain number of achievements. Another feature of the application that could have been further developed is the finger placements feature of tutor section. This feature suffered primarily due to the sensitivity of Androids button class which was unable to recognise multiple button clicks at the same time. If this were to be rectified, possibly by using multiple onTouch listeners, the feature could be greatly improved. On a more general point the finger placements tutorials could be developed to make different chords appear at timed intervals which again would provide more of a challenge to the user and improve their learning. Another feature that could be added in future iterations of the project is a way of alerting the guitar tutor within the system that they have been sent an email when the user activates the send audio function.

If there was more time for the project, the application could also have been optimised so that the app is visually consistent over a greater number of devices and systems. In relation to this, if more time was available, xml files could have been created to represent a consistent, optimised horizontal layout for the app. In terms of the code used to implement the features present in the application, while being successful in executing the specified feature, could have been more concise in parts. Certain areas of code could also have been commented using more clarity with better descriptions of certain functions and processes.

Further improvement could also have been made to the project by including more content for the user in general. For example, more songs and chord changes could have been included for the user to learn in the tutorial section, also a greater number of chords could have been included in the chord library. A feature of other guitar tutorial systems is the inclusion of games to help the user memorise the various chords. The inclusion of this type of feature would have benefited the finished product greatly. However, inclusion of anymore content could have the adverse effect of limiting performance as memory constraints were beginning to become an issue towards the end of the project lifecycle. This could be solved in future iterations of the project by storing the necessary content (e.g. sound files, images, animations) on a server or cloud and having them accessed as they are required.

6.5 Closing comments

In closing, the project was reasonably successful in delivering a solution to the problems identified in the problem specification, but could also have been improved with the addition of some of the features mentioned above and having spent more time on the planning stage of the project.

Chapter 7: Bibliography

“Chromatic Tuner” section derived from “guitar tuner” authored by Andry Rafaralahy. Source:

<https://github.com/andryr/guitar-tuner>

Mark L Murphy, 2011, “The Busy Coder’s Guide to Android Development”, third edition, CommonsWare, 978-0-9816780-0-9.

John Horton, 2015, “Android Programming for Beginners”, Packt Publishing, 978-1-78588-326-2.

Adam C. Champion and Dong Xuan, 2013, “Programming in Java for Android Development”, 1/8/2017, <https://web.cse.ohio-state.edu/~champion.17/4471/JavaAndroidProgramming.pdf>, pdf.

Rick Rodgers, John Lombardo, Zigurd Medneiks & Black Meike, 2009, first edition, “Android Application Development”, Safari, 978-0-596-52147-9.

Ben Phillips & Brian Hardy, 2013, “Android Programming”, first edition, Pearson Technology Group, 978-0321804334.

Tutorials point, 2014, “Android Application Development”, 10/8/2017, https://www.tutorialspoint.com/android/android_tutorial.pdf, pdf.

Keven Grant & Chris Haseman, 2014, “Beginning Android Development and Design”, Peachpit Press, 978-0-321-95656-9.

Wei-Meng Lee, 2012, “Android 4 Application Development”, John Wiley & Sons, Inc., 978-1-118-19954-1.

Reto Meir, 2012, “Professional Android 4 Application Development”

2016, “Generic AsyncTask 1.2”, 21/8/17, https://github.com/kosalgeek/generic_asynctask_v2, open source code

Chapter 8: Appendices

8.1 Login class code

```
package com.example.magui.myapplication.Login_Register;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.AppCompatButton;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.android.volley.AuthFailureError;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.example.magui.myapplication.R;

import java.util.HashMap;
import java.util.Map;

public class Login2 extends AppCompatActivity implements View.OnClickListener {

    //Defining edit texts
    private EditText editTextUsername;
    private EditText editTextPassword;
    private AppCompatButton buttonLogin;
    Button button;

    //boolean variable to check if user is logged in or not
    //initially it is false
    private boolean loggedIn = false;

    // new instance of configuration class
    configuration configuration = new configuration();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login2);

        //Initializing views
        editTextUsername = (EditText) findViewById(R.id.editTextUsername);
        editTextPassword = (EditText) findViewById(R.id.editTextPass);
        button = (Button) findViewById(R.id.Register_button);
        buttonLogin = (AppCompatButton) findViewById(R.id.buttonLogin);

        buttonLogin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getBaseContext(), User_Activity.class);
                startActivity(intent);
            }
        });

        //Adding click listener
```

```
        buttonLogin.setOnClickListener(this);

    }

    @Override
    protected void onResume() {
        super.onResume();
        //In on resume fetching value from configuration
        SharedPreferences sharedPreferences =
        getSharedPreferences(configuration.SHARED_PREF_NAME, Context.MODE_PRIVATE);

        //Fetching the boolean value form configuration
        loggedIn = sharedPreferences.getBoolean(configuration.LOGGEDIN_SHARED_PREF,
        false);

        //If loggedIn is true
        if (loggedIn) {
            //Intent to begin user activity
            Intent intent = new Intent(Login2.this, User_Activity.class);
            startActivity(intent);
        }

        private void login() {
            //Getting values from edit texts
            final String username = editTextUsername.getText().toString().trim();
            final String pass = editTextPassword.getText().toString().trim();

            //Creating a string request
            StringRequest stringRequest = new StringRequest(Request.Method.POST,
            configuration.LOGIN_URL,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {

                    //If we are getting success echo from server
                    if (response.equalsIgnoreCase(configuration.LOGIN_SUCCESS))
                    {

                        /*
                                //gets boolean response from array
                                try {
                                    JSONObject jsonResponse = new JSONObject(response);
                                    boolean success = jsonResponse.getBoolean("success");

                                    //sets user id variable to what is stored in the
                                    database
                                    configuration.setUser_id(jsonResponse.getString("userid"));

                                    if (success) {
                                        //Creating a shared preference
                                        SharedPreferences sharedPreferences =
                                        Login2.this.getSharedPreferences(configuration.SHARED_PREF_NAME,
                                        Context.MODE_PRIVATE);

                                        //Creating editor to store values to shared preferences
                                        SharedPreferences.Editor editor =
                                        sharedPreferences.edit();

                                        //Adding values to editor
                                        editor.putBoolean(configuration.LOGGEDIN_SHARED_PREF,
                                        true);
                                    }
                                }
                            */
                    }
                }
            })
        }
    }
}
```



```

        editor.putString(configuration.USERNAME_SHARED_PREF,
username);
        editor.putString(configuration.KEY_PASS, pass);

        //may need to be changed!!
        //editor.putString(configuration.KEY_USER_ID,
configuration.getUserID());

        //Saving values to editor
        editor.commit();

        //Starting user profile activity
        Intent intent = new Intent(Login2.this,
User_Activity.class);
        startActivity(intent);
        Toast.makeText(Login2.this, "Login successful!",
Toast.LENGTH_LONG).show();

    } else {
        //If the server response is not "success"
        //Displaying an error message on toast
        Toast.makeText(Login2.this, "Invalid username or
password, Try again", Toast.LENGTH_LONG).show();
    }
    /*

    } catch (JSONException e) {
        e.printStackTrace();
    }
*/

    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    }) {
        @Override
        protected Map<String, String> getParams() throws AuthFailureError {
            Map<String, String> params = new HashMap<>();
            //Adding parameters to request
            params.put(configuration.KEY_USERNAME, username);
            params.put(configuration.KEY_PASS, pass);

            //returning parameters
            return params;
        }
    };

    //Adding the string request to the queue
    RequestQueue requestQueue = Volley.newRequestQueue(this);
    requestQueue.add(stringRequest);
}

@Override
public void onClick(View v) {
    //Calling the login function
    login();
}
}

```

8.2 Chord transition class code

```
package com.example.magui.myapplication.tab3.chord_transitions;

import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.drawable.AnimationDrawable;
import android.media.MediaPlayer;
import android.media.MediaRecorder;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.support.v4.content.ContextCompat;
import android.support.v4.content.FileProvider;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.example.magui.myapplication.Login_Register.configuration;
import com.example.magui.myapplication.R;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import static android.Manifest.permission.RECORD_AUDIO;
import static android.Manifest.permission.WRITE_EXTERNAL_STORAGE;

/**
 * Created by magui on 22/08/2017.
 */

public class a_major_d_major extends FragmentActivity {

    // variables
    Button Listen, Listen_half, buttonStart, buttonStop, buttonCompareAudio,
    buttonStopPlayback, buttonGetHelp;
    String AudioSavePathInDevice = null;
    MediaRecorder Recorder;
    Random random;
    String RandomAudioFileName = "FileName";
    public static final int RequestPermissionCode = 1;
    MediaPlayer mediaPlayer;
    RequestQueue requestQueue;

    // creating configuration class
    final configuration configuration = new configuration();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tutor_chords_a_major_d_major);
    }
}
```

```
// initialising variables
buttonStart = (Button) findViewById(R.id.start_recording);
buttonStop = (Button) findViewById(R.id.stop_recording);
buttonCompareAudio = (Button) findViewById(R.id.compare);
buttonStopPlayback = (Button) findViewById(R.id.stop_playback);
buttonGetHelp = (Button) findViewById(R.id.Get_Help);
Listen = (Button) findViewById(R.id.listen);
Listen_half = (Button) findViewById(R.id.play_half_speed);

buttonStop.setEnabled(false);
buttonCompareAudio.setEnabled(false);
buttonStopPlayback.setEnabled(false);
buttonGetHelp.setEnabled(false);

// creating random class
random = new Random();

// Request queue
requestQueue = Volley.newRequestQueue(getApplicationContext());

// initialising image view
ImageView imageView = (ImageView) findViewById(R.id.Chord);
imageView.setImageResource(R.drawable.transition_a_d_a_d);

// creating media player
final MediaPlayer a_d_a_d = MediaPlayer.create(this,
R.raw.transition_a_d_a_d);
final MediaPlayer a_d_a_d_half = MediaPlayer.create(this,
R.raw.transition_a_d_a_d_half);

// onclick listener to play audio and start animation transition
Listen.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        ImageView imageView = (ImageView) findViewById(R.id.Chord);
        imageView.setImageResource(R.drawable.transition_a_d_a_d);

        AnimationDrawable transition = (AnimationDrawable)
imageView.getDrawable();

        a_d_a_d.start();
        transition.start();

        //updates user's basic chord achievement
updateLearnBasicChordsAchievement(configuration.UPDATE_BASIC_CHORDS_ACHIEVEMENT);

    }

});

Listen_half.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        ImageView imageView = (ImageView) findViewById(R.id.Chord);
        imageView.setImageResource(R.drawable.transition_a_d_a_d_half);

        AnimationDrawable transition = (AnimationDrawable)
imageView.getDrawable();

        a_d_a_d_half.start();
        transition.start();

    }
});
```

```
// onclick listener to check storage permission
buttonStart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        if (checkForPermission()) {

            AudioSavePathInDevice =
Environment.getExternalStorageDirectory().getAbsolutePath() + "/" +
"AudioRecording.mp3";
            //AudioSavePathInDevice += "AudioRecording.3gp";
            //+ "/" + CreateFileName(5) + "AudioRecording.3gp";

            // calls unlock record audio method

unlockRecordAudioAchievement(configuration.UNLOCK_RECORD_AUDIO_ACHIEVEMENT);

RecorderSetup();

try {
    Recorder.prepare();
    Recorder.start();
} catch (IllegalStateException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

buttonStart.setEnabled(false);
buttonStop.setEnabled(true);

Toast.makeText(a_major_d_major.this, "Recording started",
    Toast.LENGTH_LONG).show();
} else {
    requestPermission();
}

}
});

// onclick listener to handle stop button
buttonStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Recorder.stop();
        buttonStop.setEnabled(false);
        buttonCompareAudio.setEnabled(true);
        buttonStart.setEnabled(true);
        buttonStopPlayback.setEnabled(false);
        buttonGetHelp.setEnabled(true);

        Toast.makeText(a_major_d_major.this, "Recording Completed",
            Toast.LENGTH_LONG).show();

    }
});

// onclick to handler compare button
// will output audio user has recorded then the correct playing of the
chord sequence for comparison
buttonCompareAudio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) throws IllegalArgumentException,
        SecurityException, IllegalStateException {

        buttonStop.setEnabled(false);
```

```
buttonStart.setEnabled(false);
buttonStopPlayback.setEnabled(true);

mediaPlayer = new MediaPlayer();
try {
    mediaPlayer.setDataSource(AudioSavePathInDevice);
    mediaPlayer.prepare();
} catch (IOException e) {
    e.printStackTrace();
}

mediaPlayer.start();
Toast.makeText(a_major_d_major.this, "How you sound",
    Toast.LENGTH_LONG).show();

// on completion listener to stop user audio and then play correct
chord sequence
mediaPlayer.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        mediaPlayer.stop();

        a_d_a_d.start();

        Toast.makeText(a_major_d_major.this, "How its supposed to
sound",
            Toast.LENGTH_LONG).show();

    }
});

}

});

// onclick to handle stop playback
buttonStopPlayback.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        buttonStop.setEnabled(false);
        buttonStart.setEnabled(true);
        buttonStopPlayback.setEnabled(false);
        buttonCompareAudio.setEnabled(true);

        if (mediaPlayer != null) {
            mediaPlayer.stop();
            mediaPlayer.release();
            RecorderSetup();
        }
    }
});

// will send audio user has recorded to tutor's email address
buttonGetHelp.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        buttonStop.setEnabled(false);
        buttonStart.setEnabled(true);
        buttonStopPlayback.setEnabled(false);
        buttonCompareAudio.setEnabled(true);

        /**
         * Intent to open up email
         */
        //String fileName = "/AudioRecording.3pg";
        //File fileLocation = new
File(Environment.getExternalStorageDirectory().getAbsolutePath(), fileName);
```

```
//Uri path = Uri.fromFile(fileLocation);

// Intent to open user apps with email function
// Will create template email with the user recorded audio as an
attachment

Intent i = new Intent(Intent.ACTION_SEND_MULTIPLE);
i.setType("message/rfc822");
i.putExtra(Intent.EXTRA_EMAIL, new
String[]{"cmaguire82@qub.ac.uk"});
i.putExtra(Intent.EXTRA_SUBJECT, "Help with this chord
transition");
i.putExtra(Intent.EXTRA_TEXT, "I'm having trouble learning this
chord transition could you help? ");

ArrayList<Uri> uris = new ArrayList<Uri>();
String shareName = new
String(Environment.getExternalStorageDirectory() + "/AudioRecording.mp3");
File shareFile = new File(shareName);
Uri contentUri =
FileProvider.getUriForFile(getApplicationContext(),
"com.example.magui.myapplication.provider", shareFile);
uris.add(contentUri);
i.putParcelableArrayListExtra(Intent.EXTRA_STREAM, uris);

// Grant temporary read permission to the content URI
i.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

String msgStr = "Share...";
startActivity(Intent.createChooser(i, msgStr));

// calls unlocked send audio method

unlockSendAudioAchievement(configuration.UNLOCK_SEND_AUDIO_ACHIEVEMENT);

    }
});

}

/*
Method to setup the media recorder
*/
public void RecorderSetup() {
    Recorder = new MediaRecorder();
    Recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    Recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    Recorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
    Recorder.setOutputFile(AudioSavePathInDevice);
}

/*
Method to create file name for user audio
*/
public String CreateFileName(int string) {
    StringBuilder stringBuilder = new StringBuilder(string);
    int i = 0;
    while (i < string) {
        stringBuilder.append(RandomAudioFileName.
            charAt(random.nextInt(RandomAudioFileName.length())));

        i++;
    }
    return stringBuilder.toString();
}

/*
```

```

    Method to request permission to record audio and write external storage
    */
    private void requestPermission() {
        ActivityCompat.requestPermissions(a_major_d_major.this, new
            String[]{WRITE_EXTERNAL_STORAGE, RECORD_AUDIO},
RequestPermissionCode);
    }

    /*
        Method to handle user response to request permission
        */
    @Override
    public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[]
grantResults) {
        switch (requestCode) {
            case RequestPermissionCode:
                if (grantResults.length > 0) {
                    boolean StoragePermission = grantResults[0] ==
                        PackageManager.PERMISSION_GRANTED;
                    boolean RecordPermission = grantResults[1] ==
                        PackageManager.PERMISSION_GRANTED;

                    if (StoragePermission && RecordPermission) {
                        Toast.makeText(a_major_d_major.this, "Permission Granted",
                            Toast.LENGTH_LONG).show();
                    } else {
                        Toast.makeText(a_major_d_major.this, "Permission Denied",
Toast.LENGTH_LONG).show();
                    }
                }
                break;
        }
    }

    /*
    Checks permission for record audio and storage
    */
    public boolean checkForPermission() {
        int result = ContextCompat.checkSelfPermission(getApplicationContext(),
            WRITE_EXTERNAL_STORAGE);
        int result1 = ContextCompat.checkSelfPermission(getApplicationContext(),
            RECORD_AUDIO);
        return result == PackageManager.PERMISSION_GRANTED &&
            result1 == PackageManager.PERMISSION_GRANTED;
    }

    /*
    Program which handles the unlock Record audio achievement
    */
    public void unlockRecordAudioAchievement(String phpURL) {

        StringRequest request = new StringRequest(Request.Method.POST, phpURL,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    //doesn't return anything
                }
            },
            new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    //error response
                }
            }
        ) {
            @Override
            protected Map<String, String> getParams() {

```

```

        Map<String, String> params = new HashMap<String, String>();
        //params.put("user_id", configuration.KEY_USER_ID);
        params.put("username", configuration.KEY_USERNAME);
        params.put("pass", configuration.KEY_PASS);

        return params;
    }
};

requestQueue.add(request);

}

/*
Program which handles the unlock Record audio achievement
*/
public void unlockSendAudioAchievement(String phpURL) {

    StringRequest request = new StringRequest(Request.Method.POST, phpURL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                //doesn't return anything
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                //error response
            }
        }
    ) {
        @Override
        protected Map<String, String> getParams() {
            Map<String, String> params = new HashMap<String, String>();
            //params.put("user_id", configuration.KEY_USER_ID);
            params.put("username", configuration.KEY_USERNAME);
            params.put("pass", configuration.KEY_PASS);

            return params;
        }
    };

    requestQueue.add(request);

}

/*
Program which handles the update learn chord achievement
*/
public void updateLearnBasicChordsAchievement(String phpURL) {

    StringRequest request = new StringRequest(Request.Method.POST, phpURL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                //doesn't return anything
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                //error response
            }
        }
    ) {
        @Override

```



```
protected Map<String, String> getParams() {  
    Map<String, String> params = new HashMap<String, String>();  
    //params.put("user_id", configuration.KEY_USER_ID);  
    params.put("username", configuration.KEY_USERNAME);  
    params.put("pass", configuration.KEY_PASS);  
  
    return params;  
}  
};  
  
requestQueue.add(request);  
  
}
```

8.3 Definitions, acronyms, abbreviations

Term	Definition
System	Refers to the system developed
User	Refers to users of the system
Java	Programming language used for the majority of the project
SQL	Data managing language
DSP	Digital signal processing
MP3	Audio format file
Android	Mobile operating system
MySQL	Data management system
PHP	Server-side scripting language
Server	Provides functionality for login/ register
Fragment	Application user interface that can be placed in an activity
ViewPager	Layout manger that lets user wipe through pages of data
LayoutInflater	Object which instantiates layout xml file into View objects
Primary key	Database key which is unique to a record
Foreign key	Field in one table that identifies a row in another table
phpMyAdmin	Administration tool for handing MySQL
HashMap	Collection class used for storing keys and values
UI	User interface