Dublin Institute of Technology
Institiúid Teicneolaíochta Bhaile Átha Cliath

Electronic Tune Teacher for the Iphone

Dissertation

DT228
Bsc. in Computer Science
2010/2011

Sean Leonard
C05633915
Supervisor: Ronan Bradley

School of Computing
Scoil na Ríomhaireachta

Abstract

The project aims to support teachers and students in learning of guitar parts using the popular guitar tablature format. It allows the teacher or student to enter, edit and playback guitar parts on their iphone. The application is written in objective-c using XCode and Interface Builder(Iphone SDK) and it has the potential to be released for download from the App Store.

This project is based on a form of musical notation called guitar tablature. It consists of six segmented horizontal lines representing six guitar strings with numbers on the segments representing the fret the user should hold and the string that should be played. Guitar tab is usually used along with the music being tabbed as it gives users an general idea of how to play the music.

This application provides a way for guitar players to visualize musical scores in a tablature format and hear how the guitar parts should sound with triggered audio samples (.Aiff). All of this being done on a mobile device (Iphone).

The application can edit the interface using touch screen inputs and playback the correct sounds based on the users configurations and stop playback whenever the user wishes. The size of the tab can be expanded and made smaller using the add/remove staff options. There are also the playback settings: Tempo adjuster, Repeater and the Metronome which can alter the what the apps playback and potential help learners understand the music being played. The application can also read from and write to Property list files (.Plist) which are serial object file types developed by NextStep (Another company founded by Steve Jobs). These files are saved in the applications internal storage space and they can easily be converted to XML.

Declaration

I hereby declare that the work described in this Dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Sean Leonard

2010

Acknowledgements

Thanks to:

Ronan Bradley for being my Project Supervisor and pointing me in the right direction and reviewing my work throughout the process. I greatly appreciate the weekly meetings which helped me focus on the next phase of the project.

Paul Doyle (Project Co-ordinator) for the clarity and encouragement that was needed to keep me going when the work-load became overwhelming.

Bryan Duggan for giving the idea for the tune teacher and for signing my proposal and for setting up an Iphone developer account for me.

The School of Computing for allowing me have a MacBook for the year.

Table Of Contents

Table Of Figures

Chapter 1 - Introduction

1.1 BackGround

The idea for this project came about from my interest in music and the desire to learn how to play the music I like on my chosen instrument: the guitar. During my time emulating what i heard i discovered a well of information on the web that would show me how to play whatever song I wanted. This came in the form of tabs (short for tablatures). There are tabs for guitar, bass, drums, banjo, mandolin and nearly any stringed instrument. There is a tab for all the major popular songs of the last 50 years.

A tab contains a series of segmented parallel lines (six for guitar, four for bass,etc..) and the numbers on those lines which indicate which string is to be played and the number indicates which fret is to be held down. The order of the strings relate to how the guitar looks when lying flat and facing up.[1]

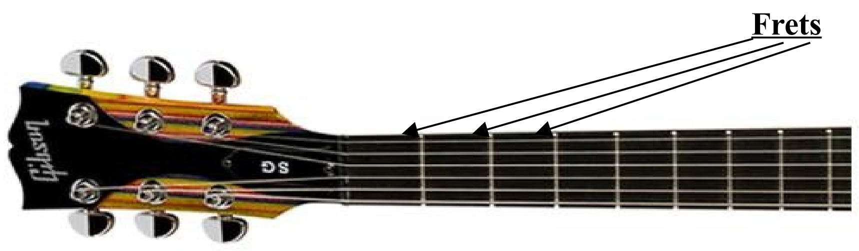


Figure 1.1 Guitar Fretboard and Head[2]

```
e|--------------------------------------------------------|  Sixth string
b|---------------------------------------5----------------|  Fifth string
g|------------------4-5------------------5----------------|  Fourth string
d|----------3-5-7------------------------5----------------|  Third string
a|---3-5-7-------------------------------3----------------|  Second string
e|--------------------------------------------------------|  First string
```

Figure 1.2 Tablature Example. (The scales of C Major and the Chord of C)

In the above example to play the C major scale the user would place a finger on the third fret on the second string and pluck the second string and follow that pattern. Chords can be identified by the numbers placed in parallel on different strings.

Tab can also specify the speed/tempo, the time signature and the style of playing of a piece of music. There may also be a legend of symbols included which indicate techniques of guitar playing like sliding, hammer-ons, tremello, vibrato, etc.. I have discarded a lot of these techniques for implementation and concentrated on the core functions as stated below in the project objectives section.

1.2 Definitions

To fully understand tablature the user must grasp the basics of reading tab and make sense of some of the buzz words used to describe parts of a tab. Also, for this dissertation I will be using certain words to describe parts of the App that could be confused for something else . So, it is important to fully understand the following phrases:

Tempo: **This refers to the speed of the Apps playback or beats per minute. The tempo can be set manually in the app. If the value is set at 120 this means that there are 120 quarter notes periods per minute and will playback the tab according to this calculation. This value will consequently change the speed of the metronome. [3]**

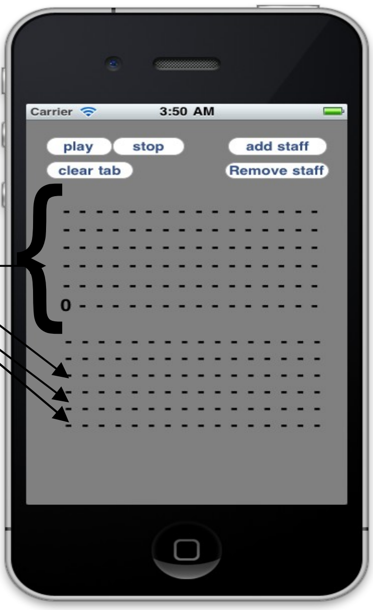
Metronome: **This is the click sound that can be set manually (on/off) that makes a sound (Cow-bell) after every quarter note length while the app is playing. It can be used to help the user keep in time with the music.**

Repeater Bars: **These are not generally part of a standard tab but i am using this idea in my App. They refer to the black bars at the left and right of each staff. When selected the bar will turn red. When there are two repeater bars selected and there is content (Notes set) between the two and the repeater amount has been set then the app will repeat the playback of that section of the tab according to the number set as the repeater number.**

Fret: **This refers to the sections on a guitar that the player uses to control what notes are being played by holding down a string on a particular fret and plucking that string then a note will be produced. Each fret for each string produces a different note (depending on the tuning).**

Note Length: **This simply refers to how long a note will play for. The longest length is a whole note or 4/4 and the shortest note length is 1/16. Each note will playback based on the tempo calculation and the given note length.**

Figure 1.3 Screenshot of First Prototype



Staff: **This refers the a section of the tab that consists of six horizontal lines and is placed one on top of another. Staffs are placed on top of each other and are to read left to right. In my App staffs can be added and removed as the user wishes but always leaving one staff on screen.**

Segments: **These are the individual parts of the tab that a fret number can be set on. For my App each staff consists of six lines horizontal parallel lines which each have 32 segments. For my App when a users touches a segment the app will display the fret number and note length view.**

1.3 Project Objectives

With my appreciation for guitar tab and the invention of the smartphone it dawned on me that it would be great to have the functionality to read and write guitar tab on a mobile device and to playback the music implemented in the tab with a virtual electronic instrument. There are applications for desktop machines that do this (E.G. PowerTab [4] and GuitarPro [5]). However, I wanted to make this application for a mobile device so it can be used in a more comfortable environment or in other words: It would be better to have this functionality without having to sit in front of a computer. Also, in my search for a project that would suit me I found the idea of the "Electronic Tune Teacher" [6] from Bryan Duggans website which spurred me on to the idea that guitar tab has the ability to teach music and is in essence a "Tune Teacher".

There are other Apps available for the Iphone that have similar functionality as my App (I.E. Guitar Tab based interface with audio playback) none of which are free to download. These Apps contain high level functionalty that are more relevant to experienced guitar players. So, I want to focus my attention on building an App that is easier to use and provides basic functionality that aids learning. When I have finished the project I would like submit it to the AppStore.

I have experience learning how to develop mobile applications on the Symbian platform from the Third Year module "Mobile Development" taught by Paul Doyle. With this Knowledge I decided to use what I've learned to develop an application on the IPhone to make use of its powerful capabilities and smooth performance.

Here is the list of functionality by the app:

Tab Editor Functions:

- **Edit tab (Choose fret number and note length)**
- **Open/Save/Delete tab**
- **Add/Remove new section (Staff)**
- **Play/Stop/tab**
- **Clear tab**

Tab settings

- **Tempo**
- **Repeat bars and repeat setter (Locate a section of tab to repeat)**
- **Metronome (Play along with music playback)**

<u>The main interface consists of:</u>

- Multiple staff sections consisting of six segmented lines representing six guitars strings.

- Segments on each line will be editable with a pop-up view of numbers indicating the frets (0-24) and a choice of note timing options that presents itself when the user touches a segment.

- A blue indicator that lies on top of each column of segments showing which note(s) is being played.

- Repeat Bars at the left and right side of each staff. If a repeat button is pressed it will turn red. If there are two repeat buttons encapsulating a section of the score and the user presses play then the app will play back that section for specified number of times.

- Toolbar consisting of Buttons and slider to indicate how the note are to be played. It consists of:

  - <u>A Tempo button</u> presents the Tempo view which consists of a slider button that increments and decrements the tempo as its slides left and right between a scale of 60 to 180 b.p.m. (Beats per Minute).

  - <u>The Add and remove staff button</u> will add or remove another section to the interface and will include additional repeater bars at the side of each staff.

  - <u>The Open button</u> will present a view to the users that displays the list of tabs that have been saved on this application. Each tab is a plist file. Each tab will have a red X next to it which will delete the tab if pressed. For a user to open a tab they can just touch the name of the tab they want to open.

  - <u>The Save button</u> save the current interface configuration to a property list in the applications storage space.

  - <u>The Play and Stop buttons</u> play and stop the audio.

  - <u>The Repeats button</u> opens a view that presents 25 numbered buttons (0-24). When pressed will set the number of repeats of playback of identifies section.

  - <u>The Metronome button</u> enables the metronome on or off and will be audible at playback.

1.4 Project Challenges

There were other functions that I wanted to implement but I found that they might take to long to develop and so I focused my attention on the fundamentals of the app which were to aid the learning of music. More specifically, The App should help people learn about tabs and their benefits and help the user progress with their musical abilities.

So, the more complex functions like the guitar techniques: string bending, sliding, vibrato and and the Tab setting options like the tuning options and time signature changes were left out. I added the repeater function so that learners can focus in on a section of the tab and have that section repeated for a set number of times and with the tempo function allowing the music to be slowed down and sped up so the learner can take their time to learn the music and speed up the playback tempo as they become faster guitar players.

I had also planned to use the open source material from the PowerTab website called PtParser which is a library of functions which converts their custom PowerTab files (.ptb) into XML which could be used to implement on the App[7]. This required for me to integrate this library into the app which would have taken to long to implement and I had learned to save the tabs as Property list files anyway.

I had initially wanted to use a Framework called AudioToolBox for the playback which allowed for audio manipulation but i found from that is was incredibly hard to use and that many Iphone developers had problem using it. So, I used the AvAudioPlayer which is just as flexible and allows for multiple instances to play simultaneously for certain periods of time.

1.5 Project overview - Roadmap

Chapter 2 - Background Research

This chapter gives on an overview of the area of study in which this project is based. It consists of discussions regarding the technologies, methods and architectures that are being used and adhered to for the development of this project. It provides an overview of the following areas:

- **Iphone (Development)**
- **Software Development Kit (Xcode and Interface Builder)**
- **IOS (Iphone Operating System) and how an application relates to its individual layers.**
- **Model-View-Controller Architecture (File structure system)**
- **Cocoa Touch FrameWork (Touch Screen Technology)**
- **Audio In Iphone (Frameworks used for Audio Playback)**
- **File Storage (How the Iphone saves to disk)**

Chapter 3 – Methodology and Design

In this chapter I give a detailed description of the Software Methodology being used and that reasons for its use. There are also UML Diagrams: Use Case, State Chart , Class and Sequence. With each diagram there are descriptions of its use and how the app architecture connects all the classes and files. As well as this there are a collection of screenshots of the interfaces from the final App prototype with explanations of how the app works. This is essentially the applications manual which discusses each individual function of the App.

Chapter 4 - Development

In this chapter I give an overview of the code and file structure and discuss each individual class and how they works and communicate with the other classes. Each function within each class will also be discussed and explain how each function the user will utilize will work and the code behind it.

Chapter 5 – Testing and Evaluation

This chapter gives an overview of the testing strategies used which are the unit tests cases and the usability tests. There are a full list of unit test that were developed as the project progressed aswell as the test results that were found after a code freeze. For the usability tests I have each test case discussed aswell as the findings from the test users feedback.

Chapter 6 – Conclusion

In this chapter I discuss what I have discovered and learnt throughout the process of developing this project and the potential future work of the App.

Chapter 2 - Background Research

2.1 Introduction

For this project I am developing an application for the Iphone on the IOS platform. I will be using a MacBook Pro with OSX 10.6.4 Snow Leopard that was kindly provided to me for the year by the school of computing. An update from 10.6.3 to 10.6.4 was needed for the installation of Xcode. I will be deploying the app onto an IPhone4 with 3G technology. I have access to the IOS Portal owned by the school of computing which allows me to deploy and test the app. This was kindly set-up for me by Bryan Duggan.

2.2 The Iphone

The Iphone has many advantages over its competitors (Android, Windows Mobile, Symbian). In particular, its ease of use and flexibility of control[8]. The Iphone is designed for universal usage so that anyone can pick it up and learn to use it quickly. As there is only one navigation button in which the user has to deal with when using an Iphone all other interaction is done through the touchscreen. This forces the applications to provide all the parameters of control through the interface and the single button is used to close windows and navigate through views[9].

Iphone Developers have to keep in mind all the different stimuli that will effect the apps performance like the screen rotation mechanism and the shake gesture which is a simple utility that can for example trigger an app to clear away its configurable interface when the phone is physically shuck. Also the different methods of touch commands that the phone can recognize should be noted like the pinch method which involve placing two fingers on the screen and moving them apart and closer to enable to the zooming functions on a view or the scrolling method which involves touch and dragging across the screen[9].

Another important aspect of the Iphone is its multitasking capabilities that allow apps to be loaded only once and will run in the background when the user closes the apps window. This helps the system reduce CPU usage by calling the app only when its needed and letting it sleep when its window is closed. This behavior is similar to Java threading. However, applications do close when not used for long periods and can be manually terminated. This forced the applications to be designed in such a way that allows the user to start a new instance of the app without the need to reload it every time. This is something that all Iphone developers have to keep in mind. For example, Are there any functions in the app that might timeout after a certain period? [10].

There are many other aspects of development of applications for Mobile devices that must be taken into consideration such as how the app will react when the phone receives a call or text message. If there is an app playing multimedia when it receives an interruption will it stop, pause, play until its finished or fadeout and will it still run in the background and resume when re-opened?. All of these scenarios have to taken into consideration for all Iphone Apps[10].

2.3 Xcode

Apple provides a free SDK for developers in the form of XCode which has a fully integrated IPhone and IPad simulator. This allows for code to be compiled and executed and simulated onscreen for the developer to virtually test there app. It also allow certified developers to deploy there app to a device.
The XCode bundle provides an Integrated Development Environment (IDE) with a text editor and simulator integrated with every project. It includes a GDB debugger and console which displays system and error messages and indicates any error found at the assembly level of code. I have found this to be extremely useful and is essential for developing graphical interfaces. As well as this Xcode displays the system messages from the phone when the app is deployed and running on the phone. [11]

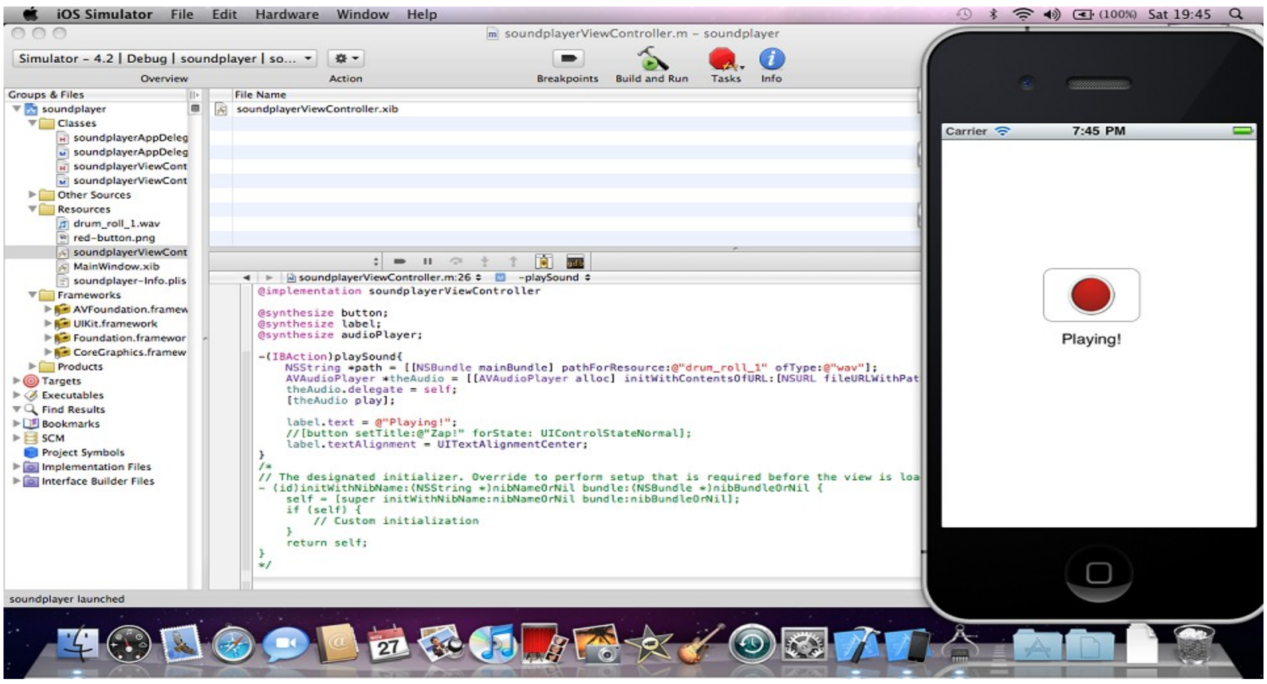
Each App developed with Xcode is made at a project level. So individual files cannot be compiled and run on there own. The file structure of a typical project can be seen in the screenshot below at the far left. The three main folder that developers are concerned with are:

- Classes: Hold the headers and main objective-c files (.h & .m).

- Resources: Holds the .XIB interface files and any other media files like image, graphics, audio and video files.

- Frameworks: Holds links to the Frameworks (API) that the App calls from its code to utilize its functionality.

The Frameworks that are linked to the project are essentially the IOS's API library that consist of individual libraries of objective-c files that are compiled along with the app at run time. They contain the interfaces used in the code to utilize the software needed to create the app. [12]



```
@implementation soundplayerViewController

@synthesize button;
@synthesize label;
@synthesize audioPlayer;

-(IBAction)playSound{
    NSString *path = [[NSBundle mainBundle] pathForResource:@"drum_roll_1" ofType:@"wav"];
    AVAudioPlayer *theAudio = [[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL fileURLWithPat
    theAudio.delegate = self;
    [theAudio play];

    label.text = @"Playing!";
    //[button setTitle:@"Zap!" forState: UIControlStateNormal];
    label.textAlignment = UITextAlignmentCenter;
}
/*
// The designated initializer. Override to perform setup that is required before the view is loa
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}
*/
```

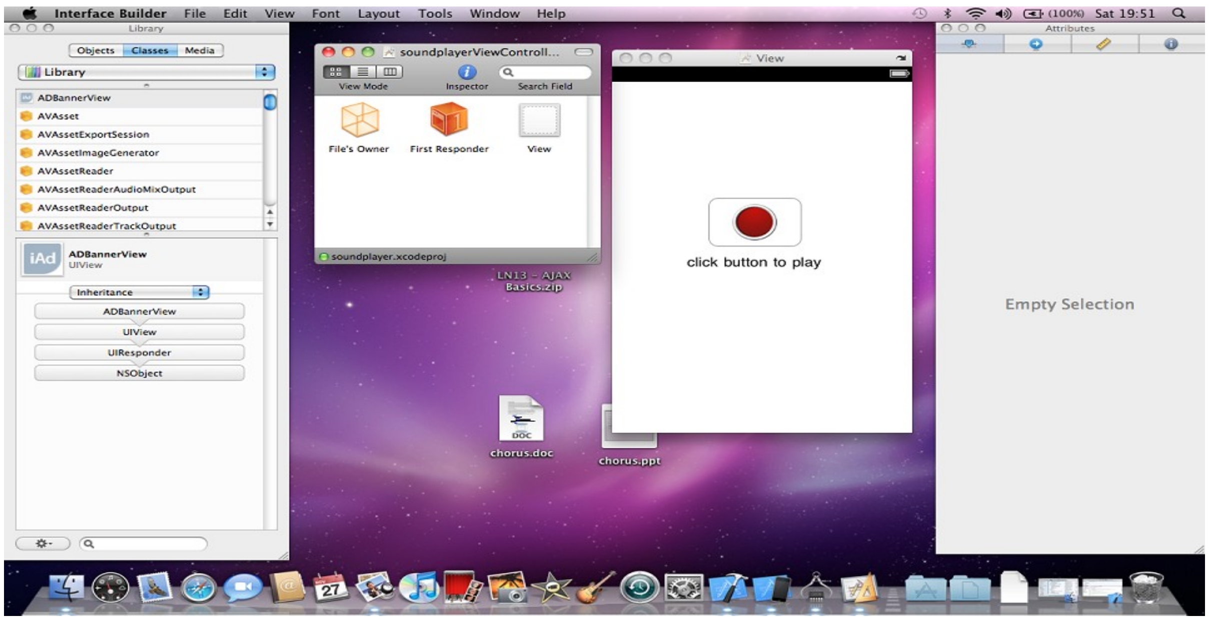
Figure 2.1: Screenshot of XCode and IPhone Simulator

2.4 Interface Builder

IPhone based projects created with XCode use .XIB file which are the actual interfaces that are seen on screen when the apps are running. The Objective-C files then control the .XIB interface by identifying the IB objects in the interface and linking them with variable instances in the code and implementing changes on those objects.

These interface files are created with The Interface Builder Suite which comes with the IOS Software Development Kit. Interface Builder lets developers build interfaces using simple drag and drop methods. It consists of four main windows: Library, Document, View and Inspector. The library window contains UI objects (windows, views, buttons, labels, text-boxes, etc..). These are the building blocks of the interface and can be customized by a detailed specification. IB Objects can be dragged from the library windows (far left) and dropped onto the view(middle right) and then manipulated with the inspector window (far right). When an object or view is selected its information is displayed in the inspector window and can be customized from here. The document window hold instances of other views and windows within the application which can be dragged from the Library window. The view is the palette on which to design the interface. There can be many views per .XIB file but there must be at least one.

The function or actions defined in the code are linked to the specific objects in the interface with the connector tools which is sub section of the inspector window. As well as functions the connector window can link instances of other classes of which an object needs to be placed in the documents window. For example, an instance of another view would involve a new view to be selected in the library and dragged to the documents view. From here it can be linked to the Files Owner and linked to an instance identified in the code. [13]



<u>Figure 2.2: ScreenShot of Interface Builder</u>

2.5 IOS Architecture

Apple IOS is the operating system that runs on the Iphone . It architectures consists of layers that communicate with its underlying layers relating to different levels of services and technology (See Diagram Below). The Cocoa Touch layer deals with the touch-screen functionality and how the device deals with multitasking. The Media provides the graphical, audio and video technology. It is at this level from which my app will get most of its functionality. The Core Services deal with such features as file storage and xml parsing and generally how data is managed. Core OS services deal with low level aspects such as networking and security[14].

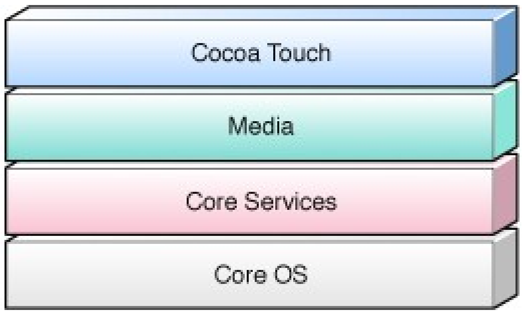


Figure 2.3: IOS Layers [14]

2.6 Cocoa Touch

The Iphone uses Cocoa Touch Technology which provides the high level services that help produce event-driven applications such as multitasking, touch-based input and push notifications. It adheres to the Model-View-Controller Software architecture. It deals with gesture recognitions and the different methods of touch-based inputs such as tapping, pinching in and out, panning or dragging and swiping rotating. It also provides the functionality of multitasking which handle apps and how they will be run and how they will be presented on the screen. For example, Weather an app will be performing in the background or the foreground.This framework (API) is integral to the apps interaction with the user and a firm understanding in this framework is essential for Iphone development[15].

2.7 Model-View Controller Architecture:

All applications that run on an IOS system abide by the MVC architecture. Each object within this architecture is assigned one-of-three roles: Model, View or Controller. The role of an object defines how it will communicate with other objects. Each collection of obects assigned to a role can be referred to as a layer of the software architecture.

MVC Roles:

Model: These classes hold data and define the functions and computation of the data relative to the application.

View: This is the interface that can be seen when the app is loaded. For this project it will be the .XIB files that use the UIKit and AppKit FrameWork within the Interface Builder.

Controller: These classes control the flow of communication between the View and the Model. [16]

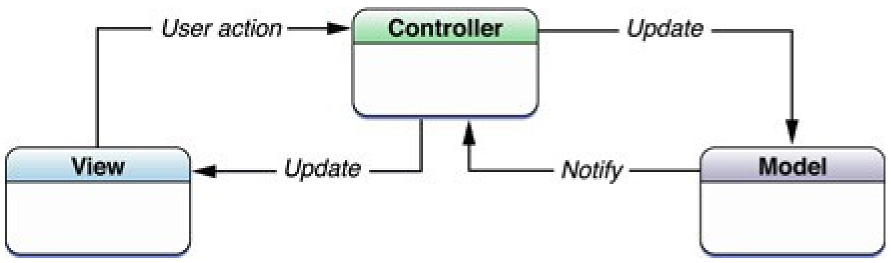


Figure 2.4: Model-View-Controller Architecture [16]

For an example of how the MVC architecture works let take a look at the scenario of a contacts list app. a contacts list app would allow users to input a new contact into the list. This would involve:

1. The user types the name and number into a form and presses a save button. This sends a *User Action* from the *View* to the *Controller*.
2. From here the data is reconfigured into an *update* message that can be passed to the *Model.*
3. Then when the user opens the contactlist the Model will send a *Notify* message to the *Controller* which then get converted into an *Update* message that the View object can interpret which is sent to the *View*. This final message updates the list with the new contact.

2.8 Audio in Iphone

For my project the audio is managed by the AVAudioPlayer Framework [17]which is closely related to the AVFoundation framework which is the standard framework for dealing with Audio and Visual services. It enables users to play simple sounds from memory but also has the functionality to play sound at a certain time and perform a certain function at a certain time after it is triggered. This allows for developers to control the length of the sound and when exactly it will be played.
I had initially wanted to use a Framework called AudioToolBox for the playback which allowed for audio manipulation at a bit level but i found from that is was incredibly hard to use and that many Iphone developers had problem using it.
I also researched the MIDI capabilities available in IOS. I found that it involved a lot of technicalities and I decided to use actual guitar sounds at an early stage. [18]

2.9 File Storage

This is managed by the NSDocumentDirectory utility which saves and reads files from the applications internal storage space using Property List files. Property list files (.plist) are standard Apple configuration files developed by NextStep. They can store arrays of arrays which are mutable which means they can increase its capacity as they are being used. Property List files can be converted to XML easily and can be stored on the phone. [19]

2.10 Programming Language: Objective-C

Objective-c is an Object-oriented Programming Language. The main difference between it and C is that it encorporates aspects of the SmallTalk OOP language adding both Procedural and OOP functionality. It is used for a vast amount of applications for the MAC OSX.

Syntax

One of the most important features in Objective-C is how it passes messages to objects as IPhone applications consist of many views the communication between these objects has to be fully understood. For example:

In Java:

```
String ButtonTitle = button.getLabel();
```

Would look like this in Objective-C:

```
NSString ButtonTitle = [button titleLabel];
```

This method is used when an action is performed as all variables declared in a class must be released to allow for better memory management. (E.G. [button release] **).**

Header and Implementation Files

The objective-c language uses two files for each class. The header or interface file (.h) declares the class to be an interface with the FrameWork it inherits from and the type of action that it has delegated itself to do. Example:

```
@interface   soundplayerViewController   UIViewController
<AVAudioPlayerDelegate>
```

The header file will then declare variable or Outlets as there called in XCode with a standard syntax:

```
UIButton button1;
```

However, It needs to be declared again for the XIB files to recognize them with the following:

```
@property (nonatomic, retain) IBOutlet UIButton *button;
```

Aswell as varibles the functions are also declared in the header. In Objective-C they are called IBActions and are declared as follows:

```
-IBAction playSound:(id)sender;
```

In the implementation class (.m) is implemented with:

```
@Implement soundplayerViewController
```

And all Outlets have to be synthesized like so:

```
@sythesize button;
```

This ensures that the Outlets have been inherited from the header file. [20] [31]

Xcode WorkFlow

The main steps involved in creating these files is to:

1. **Determine outlets and Actions (.h)**
2. **Programme Actions (.m)**
3. **Create .XIB Interface Builder files**
4. **Connect outlets and action to interface file.**

2.11 Similar Software

There are other applications that are based on the playback of guitar tab with a virtual instrument. For example, Powertab is a freeware stand-alone Windows based application that uses the local GS Microsoft MIDI WaveTable to produce synthesized guitar sounds of the editable guitar tab based interface [4]. Guitar Pro [5] is another example of this type of program. Both of these applications read and write from their own file types (E.G. .ptb for PowerTab files) and the user can save and distribute those files over the web which opens up an ever growing base of content for the end user to take advantage of. Essentially tabs of all different types of music can be shared over the internet with functionality of these types of programs.
There are also apps in the AppStore that do similar things to my App such as Tab Toolkit [21] and PocketTabs [22]. These Apps provide alot of content which is useful for experienced musicians. The aim of my App is to provide the basic functions of a tab editor and single instrument playback to aid learning.

2.12 Conclusion

I had initially wanted to re-implement some of the functionality of the PowerTab application. To implement all the functionality of PowerTab I would need a lot more time than I think I have to complete the project. So, I have focused on implementing the core functionality of the app to at least have a high level prototype working by the end of project and to give a basic demonstration using a simple piece of music.

Chapter 3: Methodology & Design

3.1 Spiral Model

For the software methodology I will be using the Spiral Model Design. It contains similar steps to the waterfall model: Initiation, Analysis, Design, Construction, Testing and Implementation. However the Spiral Model moves through these phases iteratively meaning that for each cycle there is a period where the project plan can be rewritten and allow for changes to occur as the project moves forward. Another aspect of the Spiral model is that one phase does not have to be completed before the other one begins. This means that if the for example the design document does not need to be updated then the developer can move on to the next phase with out restrictions. However some work must be done on each phase for each iteration to be completed. The flow of each each iteration involves the production of a prototype. Each prototype produced will have more deliverables than the last until the final prototype contains all the functionality and deliverable set out in the final design document. [23]

Each iteration of the methodology will include the following stages:

- **Project plan inspection**
  The first stage of each iteration involves an inspection of the project plan. What is stated in the project plan might not be the most suitable aspect of the project to be worked on. So the next stage will focus on what is the most immediate problem that should be tackled first.

- **Risk analysis and information gathering**
  This involves finding the part of the project that should be worked on for the next stage. This stage might involve interviewing people (for bigger projects) and researching areas of the project that might cause problems in the future. This phase is about eliminating risks for future development. New strategies might be introduced or the entire project might be re-organized. This phase ensures that there is a periodic stage for research within each phase before any functionality is implemented. For example, I initially wrote in the initial project plan that I wanted to save the files to XML format but after researching the alternatives I found that it was more convient to use Plist files instead as it would take let time to implement.

- **Building a new Prototype**
  When the next stage of development is defined then the new prototype needs to be built. With each iteration of the methology life cycle functionality is implemented in addition to the previous functionality implemented in the last iteration of the methodology. So, after each iteration there will be more functionality implemented than the last and eventually the final prototype will become the final product. During this phase there might be occasions where the developer realizes something about the system they are developing and find that there are more efficient ways to implement. This information will be included in the next phase.

- Updating the software design (design doc.)

  After the prototype is built the developer can update the design (Project Plan) to note any changes to the system regarding what was implemented and what the developer realized what shouldn't or couldn't be implemented. For example, for my App to set a note on the tab editor I initially tried to use a pop-over type of view in a floating window. This was difficult to implement so i reverted to a simpler method of multiple views.

- Test new functions and Update Test Plan

  This stage involves the unit testing of each piece of functionality that was implemented in the prototype. This is usually done iteratively in the prototype stage as the functions are developed. However, all the functions need to be tested separately after the prototype is built to ensure that aspect of one function does not interfere with another. If there are interferences then they need to be stated in the design document. So, for each piece of functionality that is implemented the developer will need to included multiple test cases relating to that piece of functionality within the test plan. Developers have to keep in mind that one function could have an affect on anothers performance so there may be test cases that focus on how multiple functions can relate to each other.

- Iterate (Perform Risk analysis on new Prototype)

  This is implied within the methodology that after each cycle of the methodology it will return to the Project plan inspection stage and perform the next iteration.

[30]

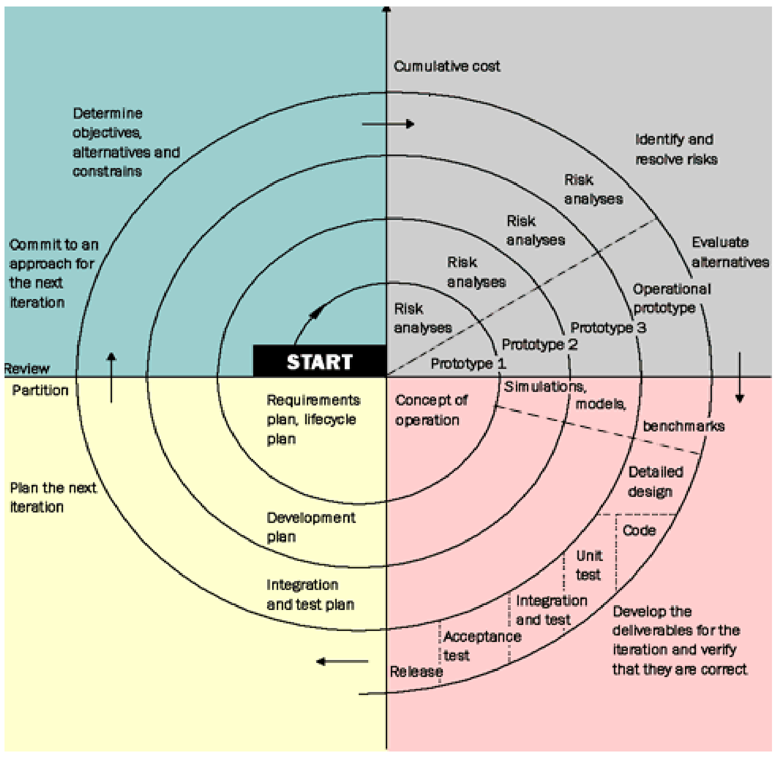


Figure 3.1: Spiral model (Boehm, 1988) [24]

For the first phase of the applications development I have:

- Gathered the main System Requirements which are:

- **Devices (MacBook and IPhone).**
- **Software (Xcode and Interface Builder).**
- **Books ("The Complete Idiot's Guide to Ipad & Iphone App Development" by Tony Brant)**
- **Registered with Apple Developer has an Iphone Developer.**
- **Developed UML diagrams**
- **Defined Project Plan**

- Performed a risk analysis and made a preliminary design:

- **Researched Guitar Tab Editors and IPhone dev.**
- **Researched Xcode**
- **Analyzed Powertab**
- **Find sound functions in xcode libraries(AVFoundations FrameWork).**

- Developed a Prototype:

- **HelloWorld from the apple developer site: "Your First IOS Application" (Free Tutorial). [25]**
- **I added to this the functionality to play a sound file (.wav) with a button("Play Sound").**

- Updated Design:

- **Defined Main Functionality(see Chapter1 Project Objectives)**
- **Wrote up Interim Report.**

- Test and Update Test Plan:

- **Tested on Simulator.**
- **Test Plan includes a simple sound test.**

From this stage I can now perform a risk analysis based on my first prototype to figure out what I can do from here and what might be a problem for future work. For example in my proposal i mentioned a guitar tuner that would receive audio signals from a guitar and analysis them to produce data to inform the user how to alter their guitar tuning. This might be too big a task as it involves a vast amount of coding in frequency analysis and developing a very detailed interface.

3.2 Project Development Summary

Here is how I used the Spiral Model for this project over the last six months:

- Gathered the main System Requirements:

First Iteration:	**Researched books and Technology, Design UML diagrams, Learnt the basics of Iphone development. [25]**
Second Iteration:	**Researched Cocoa Touch Technology. In particular the UIButton [28] and how each button can be tagged with a number which can be used to identify itself. Also learnt how classes can passes messages to each other.**
Third Iteration:	**Researched the AVAudioPlayer FrameWork [17] further and developed Audio Library of guitar sounds.**
Fourth Iteration:	**Researched the audio player further and learnt how to manipulate the sounds while playing.This allowed me to control each sound duration and exact start time [26]. Also, Learnt how add and remove sections to the main interface.**
Fifth Iteration:	**Researched the File Storage capabilities of the IOS with the NSDocumentsDirectory utility. [19]**
Sixth Iteration:	**Searched for a way to alter the tempo of the playback. Also researched methods of deploying the app to phone.**
Seventh Iteration:	**Researched for methods of looping playback for certain sections. This was research to implement the repeat function.**
Eigth Iteration:	**Researched a method to integrate a metronome to play along with the App while playing back. Used a sound sample of a cow bell.**

<u>- Performed a risk analysis and made a preliminary design:</u>

First Iteration:	**Tested the AVAudioPlayer and AudioToolBox Frameworks to test audio playback. Produced project plan and interim report.**
Second Iteration	**Decided to develop tablature-based interface That could be editable. Initially tried to use popover view but then used the simpler UIView. [29]**
Third Iteration:	**Decided to implement audio playback as it is core aspect of project. For this I had to find a way for the App to read from the interface and play sounds in the correct sequence.**
Fourth Iteration:	**Designed a way for notes to playback simultaneously to produce chords and design how the tab editor would expand and decrease.**
Fifth Iteration:	**Designed a way to save, open and load tabs from the Apps internal directory using Property List files. Initially I wanted to use XMLParser. However, .plist files work more easily on the IOS than reading XML. [19]**
Sixth Iteration:	**Designed an algorithm that would alter the timing of the playback that is used to implement the Tempo setter function.**
Seventh Iteration:	**Decided on a way for the App to repeat the playback of certain sections**
Eigth Iteration:	**Decided on a way to integrate a metronome to play at every quarter note interval during playback.**

- <u>**Develop Prototype:**</u>

First Iteration:	**Developed simple App that would playback one sound with one button on screen.**
Second Iteration:	**Developed second prototype with tab editor on the screen made of UIButtons which when presses could open the second view to set numbers onto the tab editor but does not yet play sounds.**
Third Iteration:	**Developed App that could play audio based on interface configuration of the tab editor with triggered MP3 files. This also involved developing Play and Stop functions.**
Fourth Iteration:	**Updated App to produce Fourth prototype which involved enhancing the playback functions and developing a more dynamic user interface with the functions: play/stop, add/remove Staff, Clear staff and the ability to set notes on tab.**
Fifth Iteration:	**Implemented saving, opening and delete tab file functions using Plist files and the NSDirertory Framework.**
Sixth Iteration:	**Implemented tempo function which allow the tempo value to be customized. It ranged between 60 and 180 bpm.**
Seventh Iteration:	**Implemented repeater functions with repeater bars and repeat view.**
Eigth Iteration:	**Implemented metronome to play along with playback.**

- Updated Design:

First Iteration:	**Defined main functionality.**
Second Iteration:	**Defined how the App would implement changes onto tab.**
Third Iteration:	**Defined how App would read from editor and playback audio.**
Fourth Iteration:	**Redefined how App plays back sound to include chords. Decided to on the way in which the staffs could be added and removed and a way for the app to clear the interface.**
Fifth Iteration:	**Defined how the App would Save, Read and Delete copies of tabs on the phone. Decided to use .Plist files instead of XML.**
Sixth Iteration:	**Defined Tempo function.**
Seventh Iteration:	**Defined Repeater functions.**
Eigth Iteration:	**Define Metronome function.**

- Test and Update Test Plan:

First Iteration:	**Added potential test cases relating to audio playback.**
Second Iteration:	**Added test cases relating to editing the interfaces.**
Third Iteration:	**Updated test cases relating to Audio playback.**
Fourth Iteration:	**Included test cases that involve the testing of the staff buttons (add, remove) and the clear tab function.**
Fifth Iteration:	**Added test cases for saving, opening and deleting tabs.**
Sixth Iteration:	**Updated some test cases to take into consideration the Iphone rotation capabilities.**
Seventh Iteration:	**Includes Repeater functions test cases.**
Eigth Iteration:	**Includes Metronome functions test cases.**

3.3 UML Diagrams

Use Cases

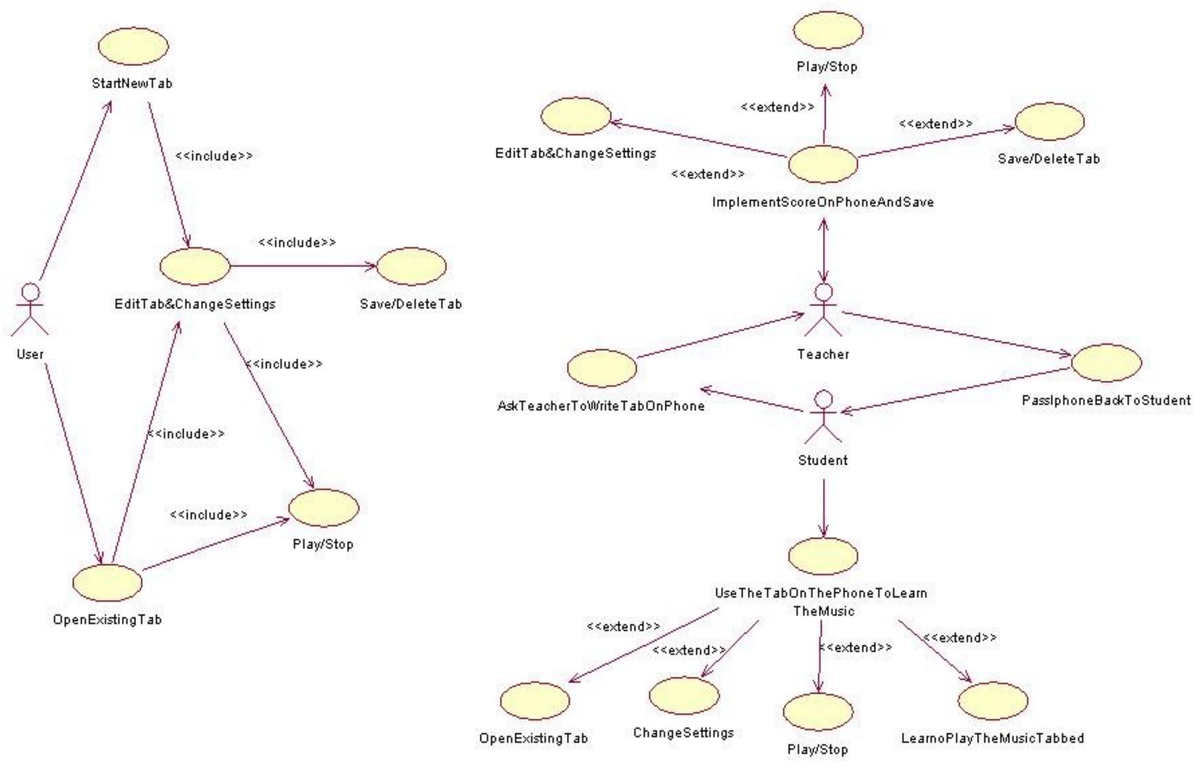


Figure 3.2: Uses Case

The typical Happy Path:

1. **When the user starts app they will have two options:**
   a. **Create new tab**
   b. **Open existing tab**

2. **Then the tab editor will load and user can:**
   a. **Edit/Clear the tab.**
   b. **Add/Remove Staff**
   c. **Open Existing tab.**
   d. **Change the playback settings: Metronome, Tempo and Repeater.**
   e. **Play/Stop the notes that are configured by the user.**

3. **Save tab to disk**

The Tune Teacher should also be able to be use in a student/teacher scenario:

1. **The student will ask the teacher to implement the tab of a particular song on the students Iphone with the Tune Teacher.**

2. **The Teacher will then implement the tab:**
   a. **Edit tab and change setting(add staff, set tempo)**
   b. **Playback tab to check tab is correct.**
   c. **Save tab to phone.**

3. **The teacher will then pass the phone back who can now learn to play the music with tab implement on the phone:**
   a. **Open the tab.**
   b. **Change Tab settings(Tempo, Repeater bars) if necessary.**

State-Chart Diagram

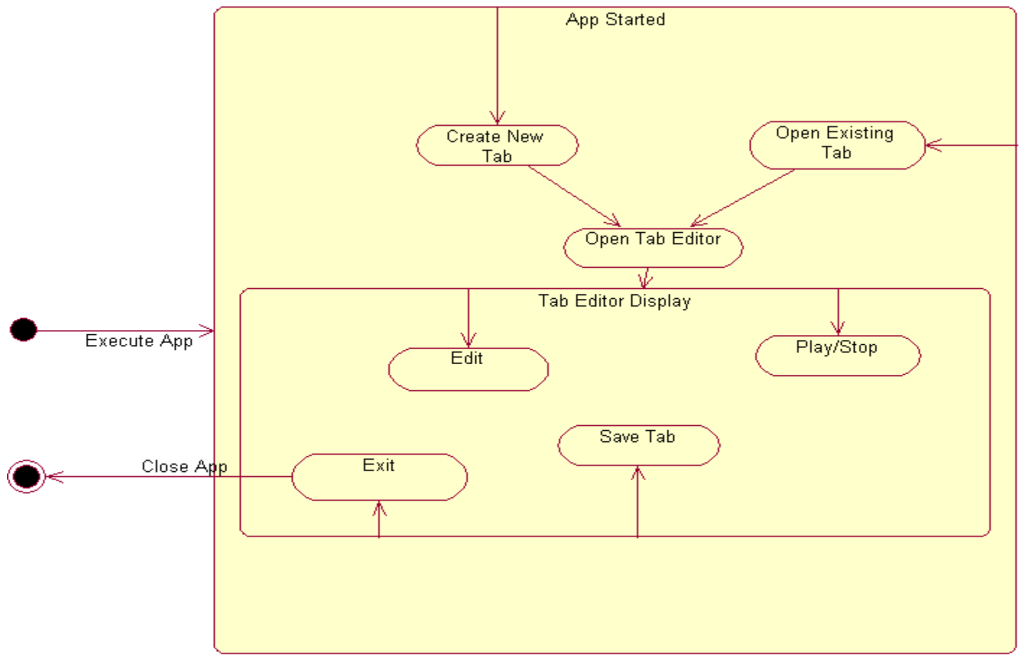


Figure 3.3: State-Chart Diagram

3.4 WorkFlow of Application

In accordance with the model-view-controller architecture each class that declares itself as an interface ("@interface") will have an associated .XIB file attributed to it. Each .XIB interface file will be reconstructed at run time when its delegate class is executed. Each new view created is placed on top of the last view created and can be discarded according to how the app functions. For an overview of how system work take a look at the diagram below. Each class in the system will consist of a header file and a main class file and have a .XIB file associated with it. [25]

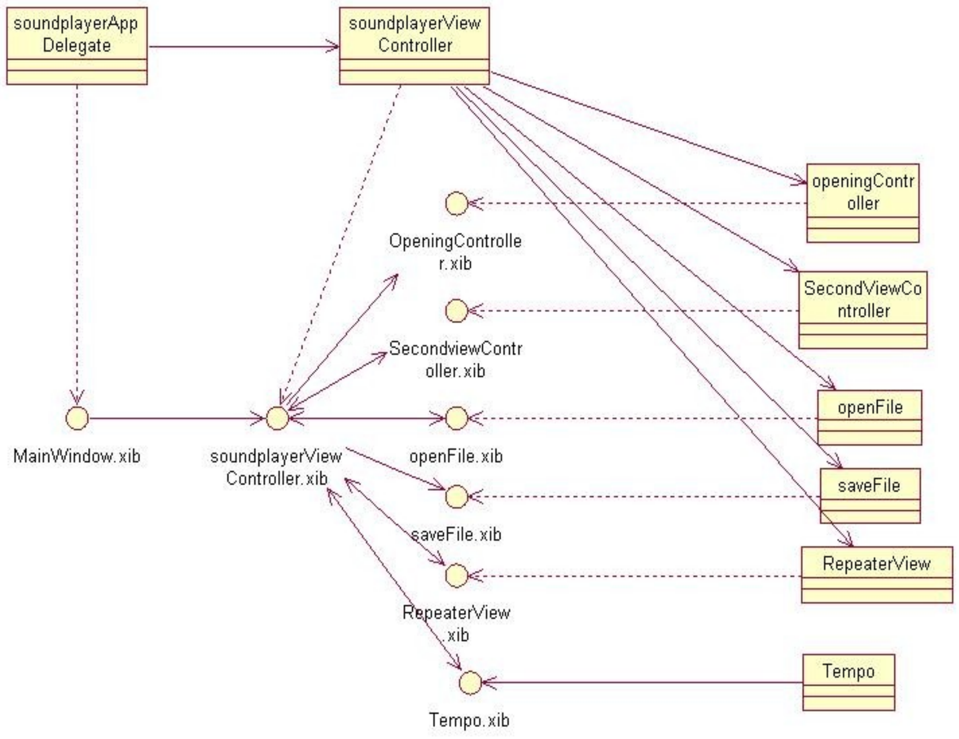


Figure 3.4: Class Diagram

When the user starts the app the *AppDelegate* class is called first. It will first construct an instance of *MainWindow.XIB* which is the interface it is associated with. This is an interface with nothing on it and will be lying underneath the other view that will be created.

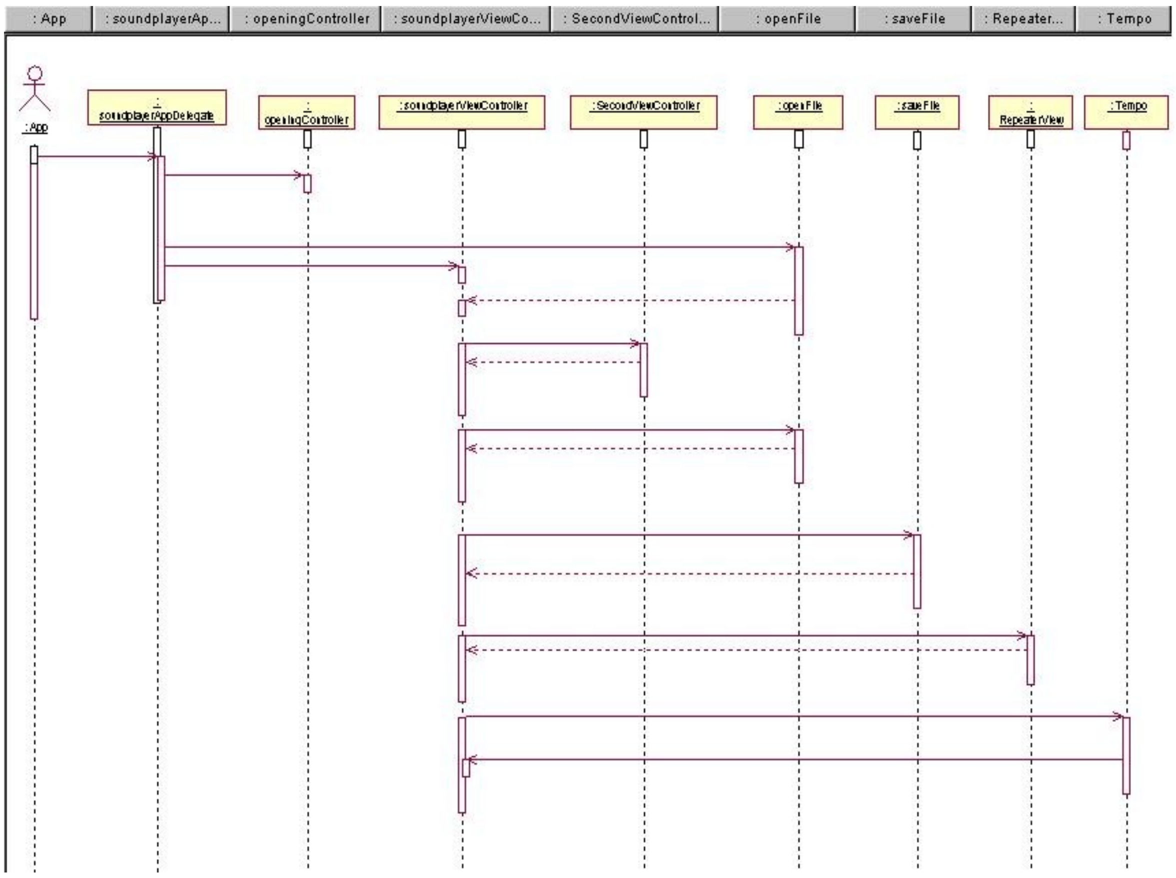
The AppDelegate will then call an instance of *soundplayerViewController* class and its .XIB file which populates the MainWindow. The *soundplayerViewController.XIB* encapsulates the main view of the app as it contains the tab editor and function buttons.

From here a tab editor will be seen on the screen and the user can edit the tab with an instance of the *SecondViewController* which creates a new view when a tab segment is pressed .This will display a list of fret numbers (0-24) and a list notes lengths in the form of buttons for the user to choose from.

When the user chooses a fret number and a note length a message is sent back to the main XIB display which will be updated with the chosen number on the segment initially chosen.

All the other views in the App communicate in a similar manner to the main view which is the *soundplayerViewController* view. All information recieved is used to editor or read from the main view.

As stated previously: For each class in the project there is an interface file associated with it. So, for the sequence diagram below I have removed the refences to the interface files for asthetic purposes.



<u>Figure 3.5: Sequence Diagram</u>

3.5 How The App Works

Figure 3.6 Screenshot of Iphone Desktop

When App loads for the first time the user will have the options to either open an existing tab that was previously saved or start a new tab

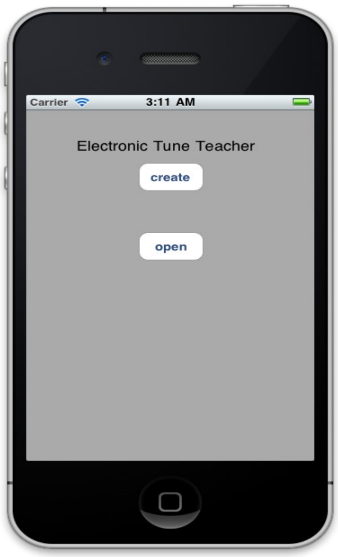
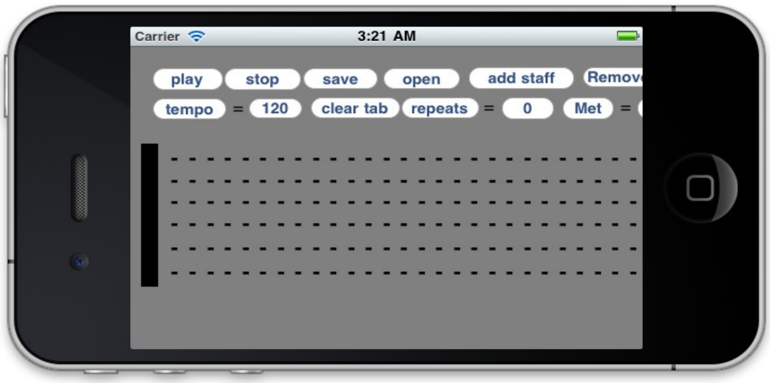


Figure 3.7 Screenshot of opening view

If the user chooses to create a new tab. Then a blank tab editor will appear.

Figure 3.8 Screenshot of main tab editor



If the user wants to open a file they can press 'open' and select a previously saved tab for it to display in the Tab Editor.

The user can then edit the tab by setting the fret numbers and adding and removing staffs.

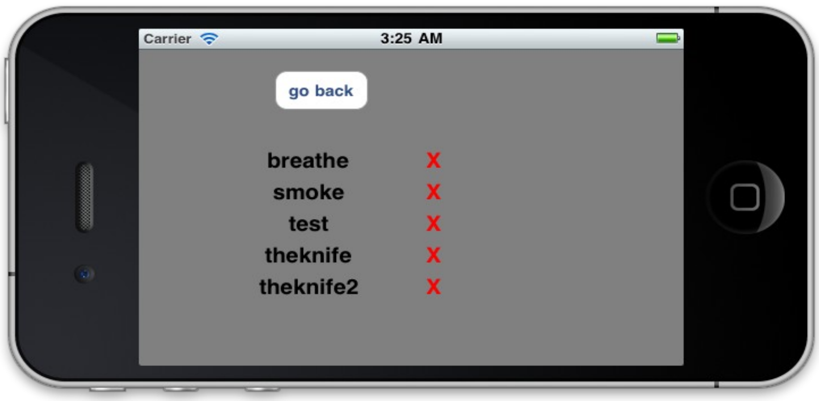


Figure 3.9 Screenshot of OpenFile view

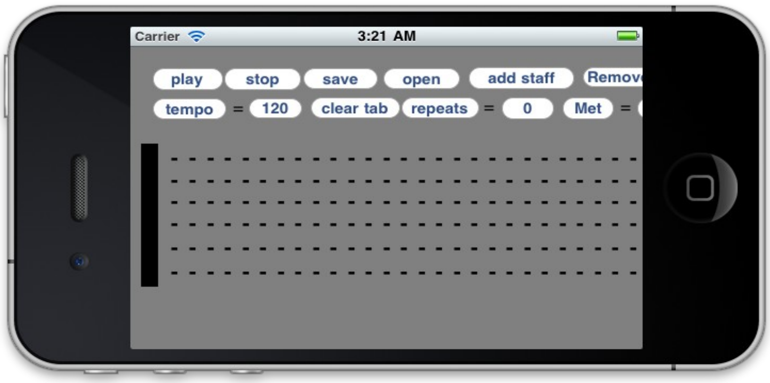


Figure 3.10 Screenshot of Main Tab Editor

When the user touches a tab segment. The fret and timing view will appear. Here the user can choose which fret number should be placed on the segment they touched and set its note length.

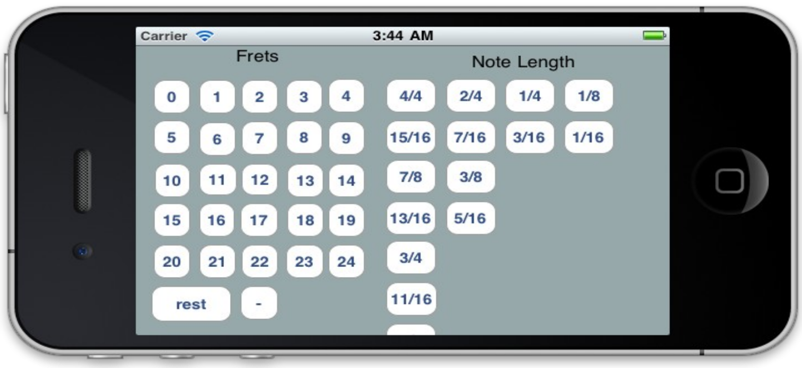


Figure 3.11 Screenshot of SecondViewController view

When the user chooses a fret and note length the app will remove that view and return to the tab.

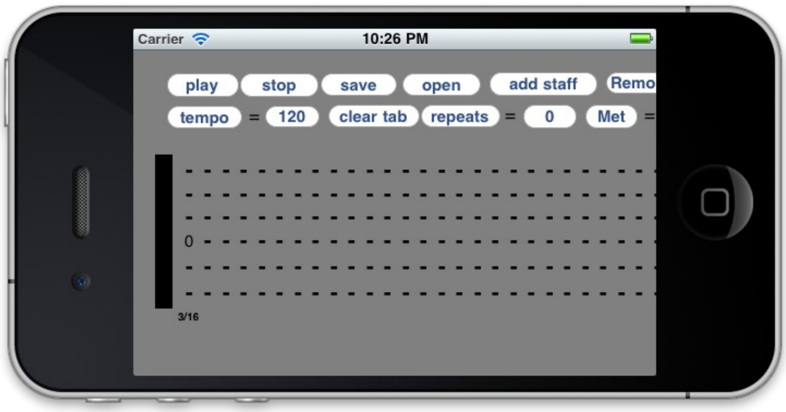


Figure 3.12 Screenshot of Tab Editor with one note implemented

When the user is finished configuring the tab they can press play which will highlight the notes being played and playback the audio in the sequence of the tab.

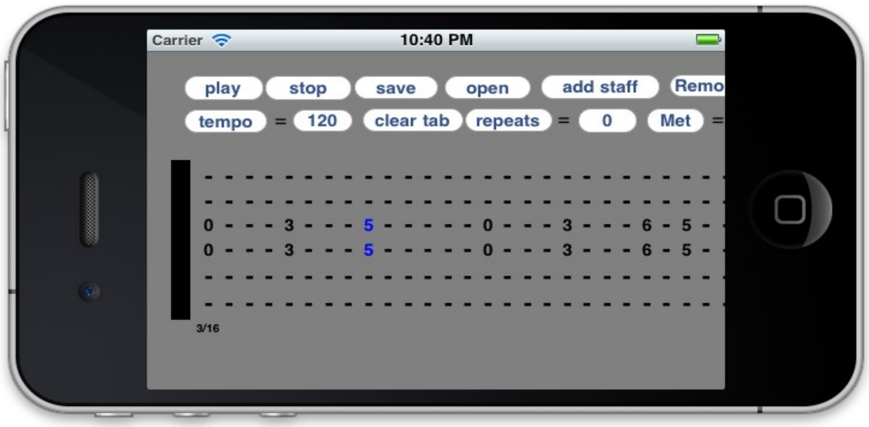
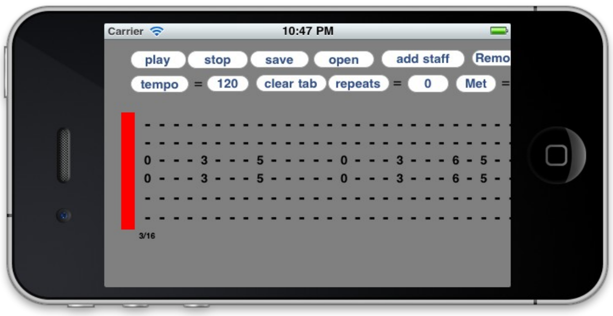


Figure 3.13 Screenshot of the tab editor with tab of "Smoke On The Water"

To aid learning the user could set the repeater bars to the left and right of each of the staffs. This sets the section of the tab to loop. Scrolling or zooming will be needed to view the right of the tab editor.



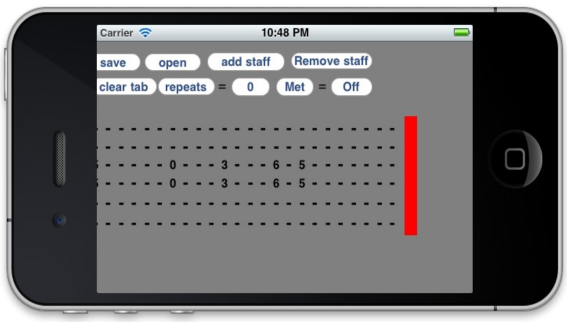


Figure 3.14 Screenshot of tab with left repeater bar set Figure 3.15 Screenshot of tab with right repeater bar set

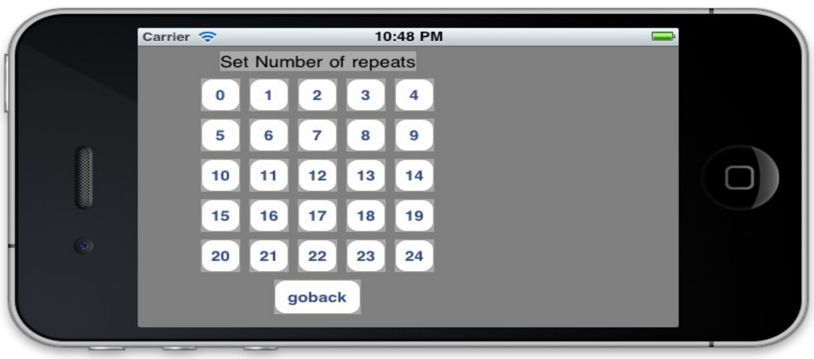


Figure 3.16 Screenshot of repeater setter view

When the repeater bars are set then the user can set the number of repeats by pressing the 'repeats' button which display the repeat view where user can select from 0 to 24. The number of repeats will set in the main tab editor view. When this is done and user presses play it will playback the set number of times.

To save the tab the user can press 'save' to display the Save view to input a file name.

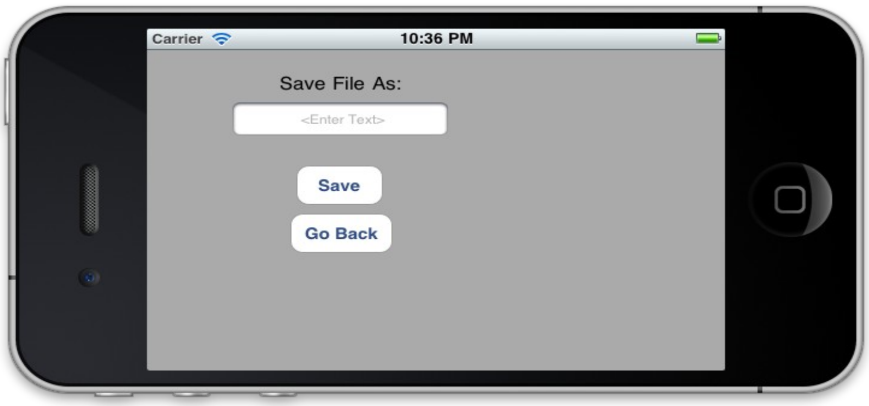


Figure 3.17 Save File View

When the user presses on the text-box the keypad will appear. The user can then type in the tab name.

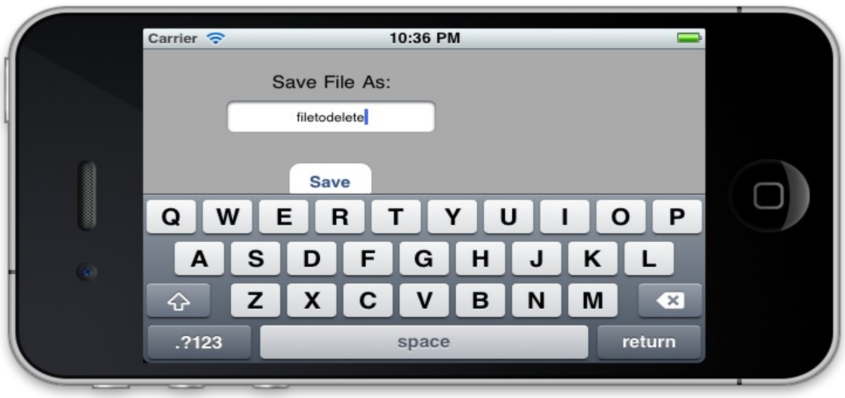


Figure 3.18 Save File View with Keypad

When the user has typed the tab name and pressed return to dismiss the text pad and presses 'save' it will save the tab and return the user to the tab editor.

When the user presses the 'open' button it will display a view with all the names of the tabs saved to the phone including the one just saved. If the user presses the red 'x' at the side of the tab name it will remove from the list.

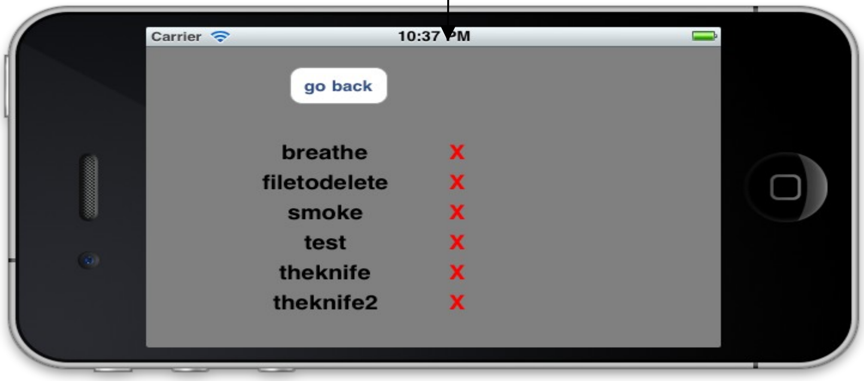


Figure 3.19 OpenFile View

When the user presses the 'tempo' button a view with a slider and a label indicating the tempo level is displayed. The tempo value will change according to the users setting of the slider.

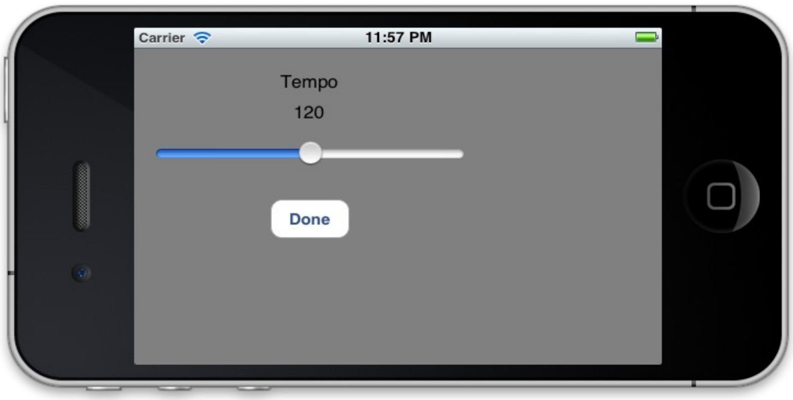
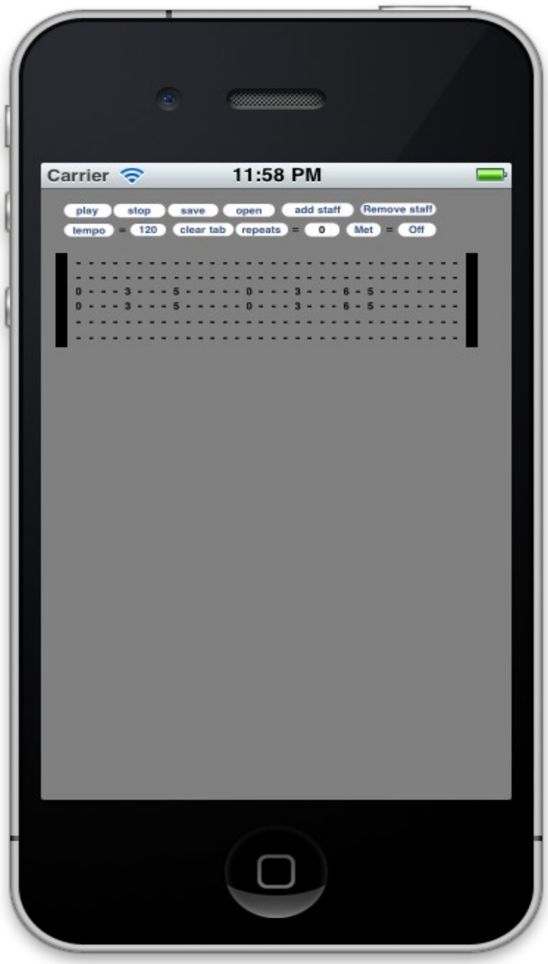


Figure 3.20 Tempo View

When the user presses the 'add staff' button an additional staff will appear and the 'remove staff' button will remove the last staff from the tab editor. The 'clear tab' button will clear away all numbers from the tab editor.



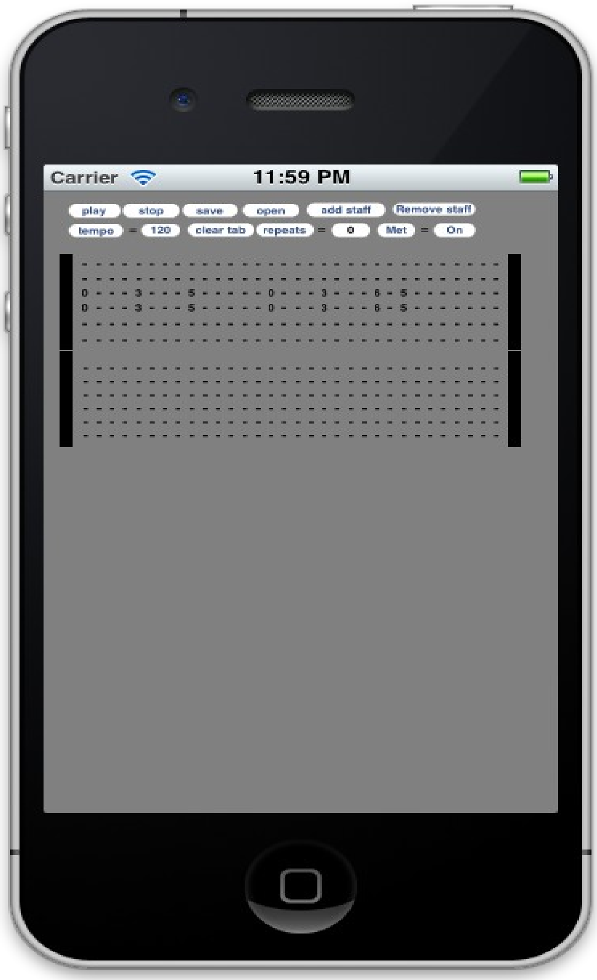


Figure 3.21 Screenshot of Tab Editor zoomed out

Figure 3.22 Screenshot of Tab Editor with multiple staffs

Chapter 4 - Development

4.1 Overview

This chapter explains how the code works and is structured. For the implementation of this project the file structure will adhere to the model-view-architecture as stated in the previous chapter. Each class consists of a header or interface file (.h) and an implementation files (.m/.mm) which contains the bulk of the code. Here are the list of classes:

- soundplayerAppDelegate

This is called when the App is first started. It is connected to the MainWindow which is not a view but a window and holds all the views of the App on top of it.

- soundplayerViewController

This class is responsible for the core functionality in the App. It is connected with the tab editor view. It contains the functions to play/stop, add/remove staffs, clear tab and connects to all other views in the App.

- SecondViewController

This class is connected to the view which sets the fret number and note length and sends this information back to the *soundplayerViewController* which implement this on the tab editor.

- openingController

This is connected to the first view that is seen when starting the App. It simply gives the users the option to start a new empty tab or open one already saved to the phone.

- openFile

This class is connected to the view which presents the list of tabs saved to the phone. It provides the functions that set the tab saved to the tab editor and delete tabs.

- saveFile

This class takes in the a tabs name and saves it to the phone.

- RepeatView

This class sets the number of repeats for the App to loop playback.

- Tempo

This class provides the function to read from the view and set the tempo value according to the reading.

For each class listed above there is an interface file (.XIB) of the same name which communicates only with that class and can only be changes by functions in that class. Each in turn can communicate to each other by instantiating other classes in its code.

4.2 soundplayerAppDelegate

.h

```
- @class soundplayerViewController;

@interface soundplayerAppDelegate : NSObject <UIApplicationDelegate,
UIScrollViewDelegate>
{
  UIWindow *window;
  soundplayerViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet soundplayerViewController *viewController;

@end
```

This header file shows hows this class instantiates the soundplayerViewController and an instance of a UIWindow which is the MainWindow.XIB interface which all views are placed upon.

.mm

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
```

Here the soundplayerViewController view is set as the first view and made visible. Also a message is sent to this view to open the openFile view right after it has loaded.

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
```

This ensures that the view can be zoomed in and out.

```
- (void)scrollViewDidEndZooming:(UIScrollView *)scrollView withView:
(UIView *)view atScale:(float)scale
```

This function enables the view to zoom and reset its co-ordinates when finshed.

```
- (BOOL) shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)toInterfaceOrientation
```

This function is in all the classes. It allows the App to rotate when the phone rotates.

4.3 soundplayerViewController

`.h`

`- @interface soundplayerViewController : UIViewController <AVAudioPlayerDelegate, UIScrollViewDelegate>`

In this header class all the UIObject in the main tab editor view are declared as well as the other view that will be called in the main part of the class. The two additional parameters indicate that this interface uses the AVAudioPlayer Framework aswell as the UIScrollView.

`.mm`

`- (IBAction)playSound{}`

This function is called when the user presses play. What it does essentially is count the tab segments that have a fret number and note length set on it and checks wheather there is a repeat selection enabled. It will then set the notes to play with the `setnotetoplay` function according to the fret numbers and notelength set on the tab editor.

When all the the notes are collected into an array the first note is found by comparing the number tag that is associated with each tab segment on the interface. The tag will indicate where the segment is on the tab editor. This information is then used to determine what note should be played and at what time.

When a note is set to play the App checks to see if there are other notes on another parallel string that should be played at the same time (Chords). If there is another note set on the tab then it will be set to play at the same time as the first. If there is no notes set to play on any of the strings directly above it then the App will read the tag of the next tag segment on the lowest string and check to see if that tag is in the array of tags that was previously saved. This process is then repeated for every tab segment until all the notes are played back.

`- (void)setnotetoplay:(UIButton *)search{}`

This function is called by the `playSound` function. It takes in the UIButton [27] which holds the fret number and checks its tab number and the number it has as its title it will then set a particular sound file to playback with a new instance of an AVAudioPlayer based on the buttons information. The name of the sound file and the AVAudioPlayer instance and the button are then sent to the `playnote` function.

`- (void)playnote:(NSString*)n :(AVAudioPlayer *)theAudiofade : (UIButton *)button{}`

Here the App will search through the array that was previously saved which contains the tab information and determine the notes length. It then checks to see if the note is set to play along with other notes if it is part of a chord or not. It will then set the exact time for the note to start playing to the nearest millisecond and prepare to send the instance of the AVAudioPlayer to the `fadeOut` function after a certain amount of time. This amount of time is based on a calculation of the note length. It also sets the time for the notes on the the tab editor to change its colour to blue while it is playing. The App then plays the sound and adds this instance of the AVAudioPlayer into a global array which is used in the `stopSound` function. [26]

`- (void)fadeOut:(AVAudioPlayer *)theAudiotofade{}`

This function will simply stop an individual sound and is executed after a period of time that is calculated based on the given notelength in the function above.

`- (IBAction)playMetronome{}`

This will set a boolean value to true that will be detected by the `playSound` function which will play the cow bell sample on every quarter note.

`- (void)firstchangeColor:(UIButton *)button{}`

This is called by the `playSound` function which sets a button to blue as the corresponding note is playing.

`- (void)changeColor:(UIButton *)button{}`

This is called by the `playSound` function which sets a button to black after the corresponding note has finished playing.

`- (IBAction)stopSound:(id)sender{}`

In this function the App will retrieve the array of AVAudioPlayer that is a global variable. It will then extract each instance of the audio-players and stop there playback using a for loop. This method ensures that the audio completely stops.

`- (IBAction)clearstaff{}`

This will clear away all tab from the editor and clear out all the notes and note settings from the global array that is filled when the user implements notes on the tab.

`- (IBAction)changeTempo{}`

This function will simply display the Tempo view which will call the `writeTempo` function when the user sets the tempo.

`- (void)writeTempo:(NSString*)tempostring{}`

Here the App will use this function from the tempo view to set the tempo value with given string variable.

`- (IBAction)setrepeater:(id)sender{}`

This function is called when the user presses one of the the repeater bars to the left and right of each staff. It sets the bar to red and if there are two repeater bars set with a section of notes between them then it sets the repeat bool value to true which is read by the `playsound` function and playbacks only the section(s) between the bars for the set number of times.

`- (IBAction)openrepeaterview:(id)sender{}`

Opens the Repeater View that sets the number of repeats in the tab editor with the `writerepeater` function.

`- (void)writerepeater:(NSString*)repeatstring{}`

This function is called by the repeater view and sets the number of repeats.

`- (IBAction) saveXML:(id) sender{}`

This will save all the relevant information of the tab in an array and send it to the SaveFile class to be saved. It will then present the SaveFile View.

`- (IBAction) openXML:(id) sender{}`

Opens the openFile view.

`- (void)FileToOpen: (NSArray *)fileArray{}`

This is called by the openFile view and passes the array of notes, note lengths and tempo relating to the tab and displays them on the tab editor.

`- (void)openfilenow{}`

Opens the a file and displays it on the App.

`- (IBAction) removestaff:(id) sender{}`

This function will remove the last staff that was added to the editor.

`- (IBAction) addstaff:(id) sender{}`

This function will add a staff to the end of the current staff along with two repeater bars at each side.

`- (void)addstaffinload{}`

This does the same as above but is contained in a void function instead of a IB action so it can be called from other functions.

`- (IBAction) displayView:(id) sender{}`

This action is called when the user presses a tab segment and displays the SecondViewController view for inputting the fret notes and note lengths.

`- (IBAction)change:(id)sender{}`

This function is also called by the App when the user presses a tab segment. It simply sets the tab segment to be the segment to be edited.

`- (void)writeToSegment:(NSString*) buf {}`

This will be called by the SecondViewController to set the fret number and time label to the tab editor.

`- (void)addnotetime:(NSString*) buf {}`

This will be called by the SecondViewController and it saves the note length value associated with note being set.

`- (void)opener{}`

This function is called when the App first starts and it called by the openController view.

`- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)Audio successfully:(BOOL)flag{}`

This function needs to implemented when using the AVAudioPlayer Framework. It is called when the audio has finished playback. The function itself will deallocate the instance to keep the memory management efiicient. [17]

`- (void)viewDidLoad{}`

This function is called atomatically when the view is loaded it will ensure that the view is an instance of a ScrollView and it detemines its co-ordinates and its zooming range.

`- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)toInterfaceOrientation{}`

This is a standard function that is called when the phone detects a movement of the screen (horizontal and vertical).

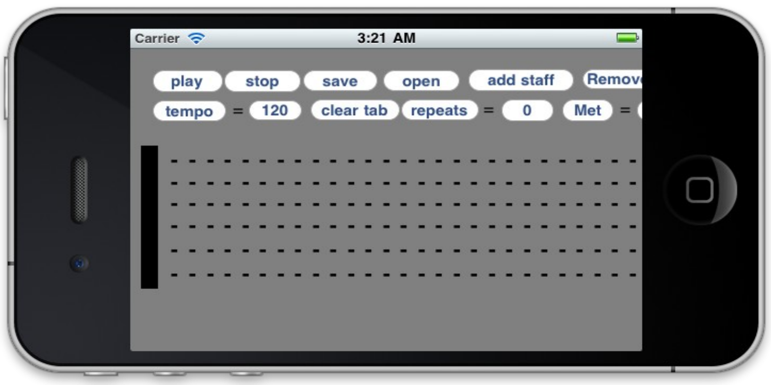


Figure 4.1 Screenshot of Main Tab Editor

4.4 SecondViewController

.h

`- @interface SecondViewController : UIViewController <UIScrollViewDelegate>{}`

Here the interface is defined as UIViewController and Scroll view delegate which means that the interface will be scrollable.

.mm

`- (IBAction) dismissView:(id) sender`

This function is called when the user presses a fret number will add a fret number by calling the functions [Obj addnotetime:notetime] and a note length with [Obj writeToSegment:fret] providing that a note length has been selected to tab editor. The Obj referenced here refers to the instance of the SoundPlayerViewController class and dismiss the view to present the underlying tab editor. The view will be dismissed when a fret and a note length have both been chosen.

`- (IBAction) changeButton:(id) sender{}`

This function is called when the user presses on a note length button and will also add a note to the tab editor providing that a fret number has been choosen. This function is essentially the same as the dismissView function. The only difference is that it is connected to the note length buttons.

`- (void)writeLabel:(NSString*)fretnumber :(NSString *)timeofnote{}`

This function simply sets the text for two of the labels in the view to display the fret number and note length.

`- (IBAction)goback{}`

This function simply lets the user go back to the tab editor without setting any notes.

`- (void)viewDidLoad{}`

Here, the actual view presented is set to be a scroll view.

`- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)toInterfaceOrientation{}`

This ensures that the apps view will rotate with movements of the phone.

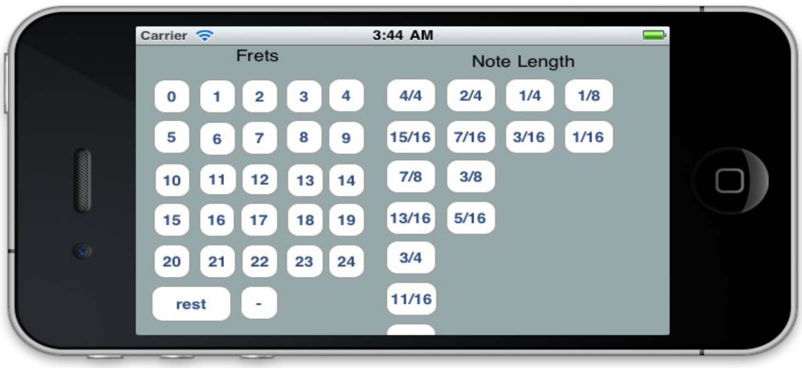


Figure 4.2 Screenshot of SecondViewController view

4.5 openingController

`.h`

`- @interface openingController : UIViewController{}`

Simply a view that does not require scrolling. This view is the first view that appears on top of the tab editor view. It gives the option to create a new tab and open an existing tab.

`.mm`

`- (IBAction)play1:(id) sender{}`

This function is called when the user chooses to create a new tab. This will simply remove this view and present the tab editor view.

`- (IBAction)openafile:(id) sender{}`

This function is called when the user chooses to open an existing tab. It will dismiss this view and open the the openFile view.

`- (BOOL) shouldAutorotateToInterfaceOrientation:`
`(UIInterfaceOrientation)toInterfaceOrientation`

This will ensure that the App will rotate the view when the user moves the phone.

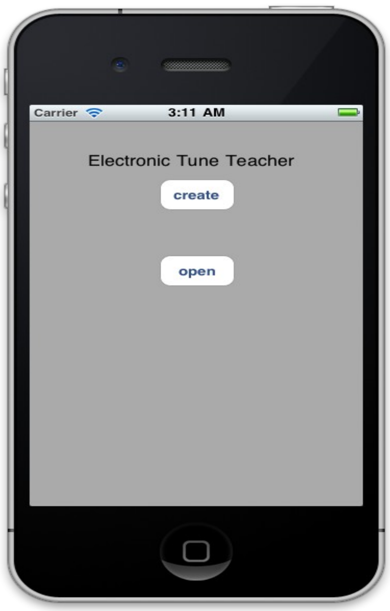


Figure 4.3 Screenshot of openingController view

4.6 openFile

.h

- @interface openFile : UIViewController{} <UIScrollViewDelegate>{}

Simple scrollview that displays the list of tabs that are saved in the apps internal storage space.

.m

- (IBAction)goback:(id) sender{}

Dismisses this view and returns to tab editor with no change to tab.

- (IBAction)selectsong:(id) sender{}

This function is called when the user selects a song. It will fetch the information from the selected tab from its co-responding property list file and save it in an array. [19] It will then send this array to the soundPlayerViewController class (tab editor class) with the code [Obj2 FileToOpen:array] which will implement the tab onto the editor. It will then close the view and return to the tab editor.

- (IBAction)deletesong:(id) sender{}

This is called when the user selects the the red x next to each song. It will locate the file choosen and remove from the apps directory. It will then refresh the view to update the list.

- (void)viewDidLoad{}

This section of code will be called when the view first loads. It will read the names from the App directory of the property list files and add them as buttons to the view. [19]

- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)toInterfaceOrientation{}

This will ensure that the App will rotate the view when the user moves phone.

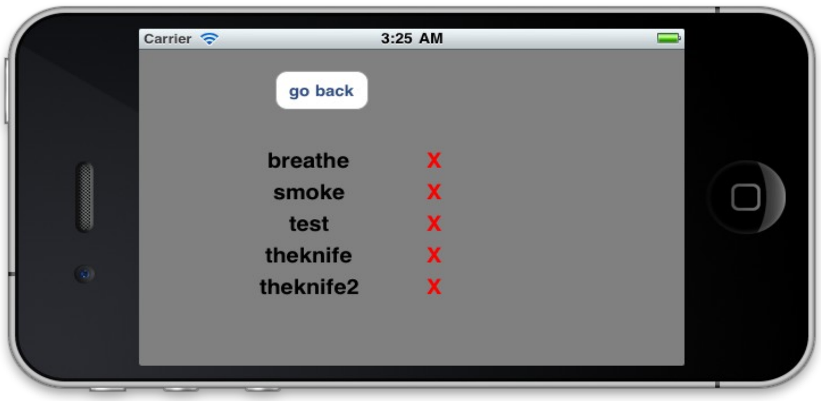


Figure 4.4 Screenshot of openFile view

4.7 saveFile

.h

- @interface saveFile : UIViewController <UITextFieldDelegate>

This is a view that uses a textfield. It takes in the name the users sets for the tab that is currently on the tab editor. The soundPlayerViewController reads the tab editor and saves the fret numbers and note lengths in an array. This class will then take the name in the textfield and save it as a property list file in the app directory.

.m

- (void)saveArray:(NSMutableArray *)theArray{}

This function is called when the user presses the 'save' button in the main tab editor. The array will be saved as a variable and held until the user inputs a name for the tab.

- (IBAction)goback:(id) sender{}

This simply dismisses the view and returns to the tab editor with no change to the tab.

- (IBAction)saveAndGo:(id)sender{}

When the user has entered a name for the tab the user should then press the 'save' button to call this function. It will take the text from the textfield use it to name the property list which holds the tab. The array that is saved will be save in that property list file. [19]

- (BOOL)textFieldShouldReturn:(UITextField *)textField{}

This ensures that the textfield returns back to the app when the user is finished inputting text.

- (void)viewDidLoad{}

This sets up the textfield to set the text to "<enter text>" when view first loads.

- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)toInterfaceOrientation{}

This will ensure that the App will rotate the view when the user moves the phone.

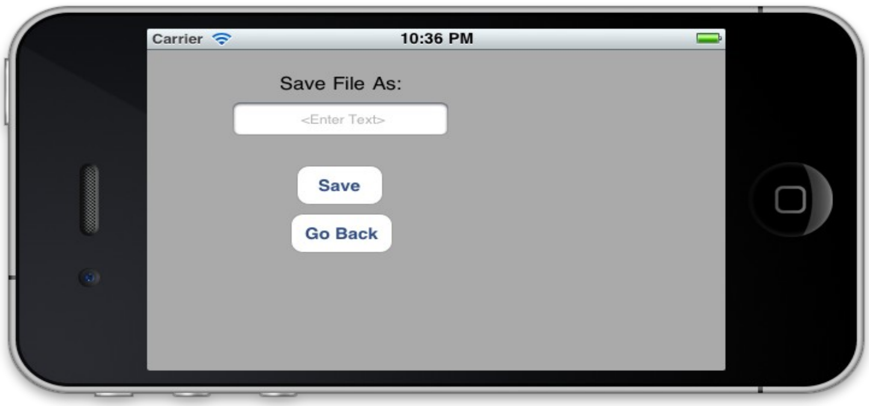


Figure 4.5 Screenshot of SaveFile view

4.8 RepeatView

`.h`
`- @interface RepeatView : UIViewController <UIScrollViewDelegate>`

This is a view that displays a selection of buttons with numbers on them. The user will select a number and that number will be set as the number of repeats that the app will playback when repeat is enabled.

`.m`
`- (IBAction)goback:(id) sender{}`

This simply dismisses the view and returns to the tab editor with no change to the tab.

`- (IBAction)setrepeaterfunc:(id) sender{}`

When a user presses a button to select a number this function will be called. It sends the data back to the soundPlayerViewController class with the code: [Obj4 writerepeater:numbertoset] . The repeater value will then be set.

`- (void)viewDidLoad{}`

This will set the view to be scrollable when view is first loaded.

`- (BOOL) shouldAutorotateToInterfaceOrientation: (UIInterfaceOrientation)toInterfaceOrientation{}`

This will ensure that the App will rotate the view when the user moves the phone.

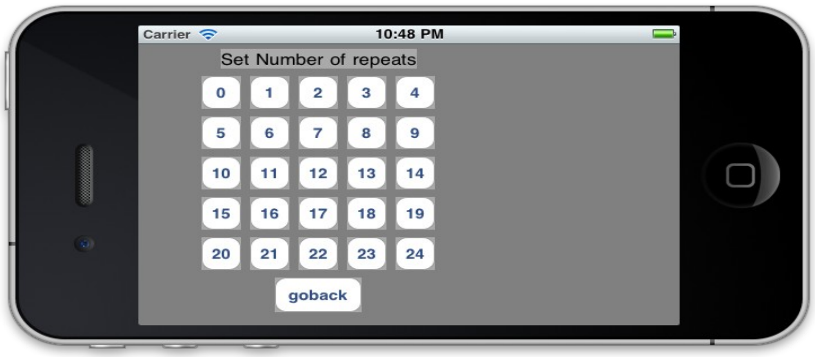


Figure 4.6 Screenshot of RepeatView

4.9 Tempo

.h

`- @interface Tempo : UIViewController`

This class controls a view that contains a slider button a label which displays the tempo value. As the user moves the tempo up and down the tempo value increases and decreases and it will set the tempo in the main tab editor view.

.m

`- (IBAction)SliderMoved:(UISlider *)sender{}`

This function is called when the user releases the slider. It will then send the data back to the main tab editor class with the code: [Obj3 writeTempo:str]. [27]

`- (IBAction)goback:(id)sender{}`

This function will dismiss the Tempo view and return to the tab editor.

`- (void)viewDidLoad {}`

Here the sliders maximum and minimum value of the slider when the view is first loaded.

```
- (BOOL) shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)toInterfaceOrientation{}
```

This will ensure that the App will rotate the view when the user moves phone.

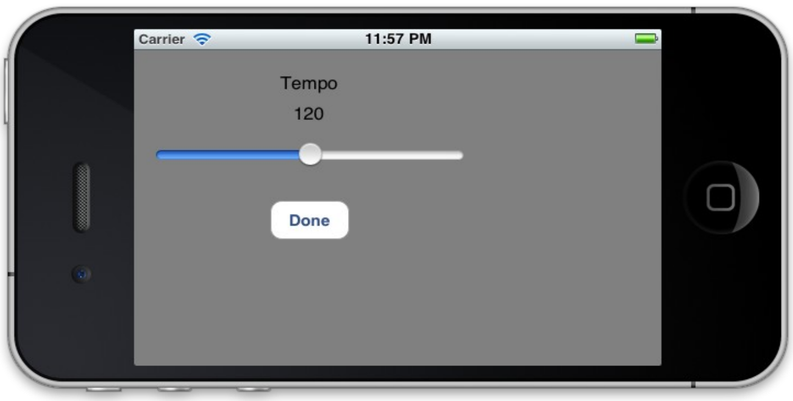


Figure 4.7 Screenshot of Tempo View

Chapter 5 - Testing and Evaluation

5.1 Testing Objectives:

For the testing of this project I will focus on the usability of the App and apply the White Box form of testing for each individual unit (functions). As each piece of functionality was implemented I tested it to ensure that it performed correctly. This was essentially to bring focusing on the performance of certain Frameworks (API) and to test the code to ensure that it would return the precise result that was specified in the design. When the code was close enough to completion the development (coding) of the Project was stopped (code freeze) and I began to use the same test cases that were collected throughout the process to do an overall test of the App to pin-point any major bugs to focus on and to decide what bug could be ignored.

The main aim of the project is to allow for this App to be easy to use and to provide accessibility to users who are learning to use tablature as they use the App. So, the main focus of the testing was based around ensuring that the audio was the correct audio and that it played back in the correct sequence. If the notes don't playback correctly then the user won't learn correctly.

Another aspect was how the potential users could learn to use the App quickly by ensuring that the interface is clear and understandable so that a user could figure out how to use the App without the need for instructions. It would also help if they have some prior knowledge of guitar tab. For this I developed a questionnaire based on those unit test cases for the people who were going to test the App.

5.2 Unit testing

Each of the following tests are based on one of the following areas:

- **Playing back sound (notes, chords, metronome).**
- **Stopping playback.**
- **Setting notes onto the tab.**
- **Adding/removing staffs sections and clering tab.**
- **Saving, opening and deleting tab files.**
- **Use of the Repeater functions.**
- **Zooming and Scrolling capabilities.**

Each test case might not involve one test in it self but will test a certain aspect of the App. For example, The first test is to test the playback of a single note. However, This involves playing each note on the App on the tab in all possible ways one at a time. To write a full list every single test would be too much detail. So, the following list of test cases can be seen as foreseeing all the possible scenarios the App will be in.

5.3 Test Cases

Number	Module	Expected result
1	set app to play one note. This will involve playing each of the notes on each possible string	note should play
2	play multiple notes in sequence Two subtypes: on one string, over multiple strings	notes should play
3	play 1 to 6 notes simultaneously (Chords) Use complex chords with strings that are not next to each other	All notes must play
4	play one note then chord in one instance Chords should be complex and should move to single notes on different strings	both single note and chord should play one after the other
5	play note using different note length 1/16 to 4/4 + rests	note should play at different lengths
6	Change segment while playing back. This should also be done while repeat is on.	fret window should not appear All buttons should be disabled. Should not alter sequence of playback
7	press buttons while playing back.	Should not interrupt playback All buttons should be disabled while playing back.
8	press play button repeatedly	Should not effect playback This should not start new Instances of playback

9	press stop repeatedly	Should not stall app or have any effect.
10	close app while playing	Sound should stop and not fade out and when reopened should not be playing.
11	Save tab to phone and replay	file should appear in open file view. Fret numbers segment numbers and note-lengths should all be saved to property list file.
12	Open tab	Tab should appear in the correct sequence and same note lengths as when it was saved.
13	Delete tab	tab should disappear from open file view and from disk.
14	Change tempo slow and fast	playback speed should be relative to tempo set and should not effect playback.
15	Add and remove staff. Check limits of amount of staffs	There should be a limit to number of staffs. When removing staffs the first staff should not be removed and repeater bars should appear with each staff.
16	clear tab	All notes and time labels should disappear and no change to tab format should happen.
17	Repeat playback of section for up to 24 times using complex tabs.	should playback 24 times without crashing and without change to playback.

18	Set number of repeats	Should playback set number of times
19	Test zoom and scroll options while rotating. Test on all views	Should zoom and scroll with ease.
20	Check what happens when phone recieves a call while playing	Sound should stop and allow phone call to take place.
21	Create and playback very long complex tab.	App should playback sounds with out interference.
22	play with Metronome	Metronome should stay in time with the music and not interfere with the playback of the sounds.
23	Play song with metronome at different tempos	Metronome should speed up and slow down as appropriate.
24	When app starts the open tab function on the start view doesn't open the open tab view	Open File view should appear and start view should disappear.

5.4 Test Results

For each of the unit tests set down previously I only included results for the tests that a bug was found as the failed test cases did not need further examination.

Here are the results of the first test run after the code freeze:

Number	Date of completion + Description
17	25/02/11

When testing the repeat function for 24 times It repeats for 17-20 times then the Audio stops and the blue indicators still move across the notes showing that the App does not crash or give any system errors however the instance of the AvAudioPlayer returns a null.

4 25/02/11
When testing to play chords and notes when first note is set on any other string than the bottom string then the first note wont be played and the app will be stuck in a loop. This is because the app assumes that the first note will be starting at a lower tone than the second.

3 25/02/11
When testing chords the app plays chords if all notes are next to each other but if a chord is played that does not have neighboring string being played then the app will play the notes or sets of notes separately.

9 25/02/11
When pressing stop while app is playing the sound stops but the blue indicators go through the notes as if they were still playing.

22 25/02/11
When using the metronome the sound doesn't playback in time with it. It seems to stop at certain pints and stall playback.

6 25/02/11
When pressing buttons while app is playing other views appear while music is playing. I need to disable the buttons while sound is playing.

8 25/02/11
When pressing play multiple times the app will play twice over the current sounds. Need to disable play button when pressed.

12 25/02/11
When opening a tab the app adds on extra staffs. It doesn't recognize empty staffs that are there already there. Sometimes when opening a tab the music doesn't playback correctly. Note label of new tab are also not set.

15 25/02/11
For number of staffs the screen needs to be larger vertically to allow for maximum number of staffs. Need to set maximum number of staffs to less than 999.

16 25/02/11
When tab clears some segments don't have the right font.

19 25/02/11
When zooming while rotated horizontally view does not stay horizontal and becomes very small.

5 25/02/11
When setting notes of different lengths the label that is placed below the note can overlap. Need to set label size to a smaller size.

14 25/02/11
When playing back the tempo seems to be about 10bpm slower than it should be.

10 25/02/11
While playing back: if the app is closed the sound will fade. Need to make it stop completely.

5.5 Usability Testing

This type of testing involved for test users to use the App on the Iphone itself. For this I introduced the App to the test users and briefly show them how it works. I then asked them to use App to implement some notes onto the tab and playback the results and use the repeater and tempo functions while playing . I then used the following questionnaire to gain feed-back.

Questionnaire

For each question I focus on the general areas of the App and restrained from asking to many specific questions about certain functions.

Questions:

1 How did you find the playback of audio from the App?(Sound quality)

2 Did the App play the tab correctly and according to what you implemented? If not what aspect of playback is not correct.) (bad timing, wrong note, wrong sequence, metronome timing, repeater, ...)

3 Were you able to use the zooming and scrolling functions with ease? Did it help to use the App?

4 Were you able to edit the tab with ease? (Sizing of interface issues, adding/removing staffs, clearing the tab)

5 Do you think that you could use an application like this to help you learn the guitar?

6 Do you think that this App could help people learn to read guitar tab?

5.6 Test Users

The following is an overview of each of the usability tests.

Test A (Novice Level)

I showed the app to someone who would play the guitar at an novice/intermediate level. He mainly deals with chords and is not familiar with tab. I presented to him a tab of the riff from 'Smoke On The Water'. He had trouble understanding tab so I demonstrated to him how to read and play piece of music and how what I was playing was displayed on the app. Eventually, I got him to play the guitar along to the blue indicators on screen. He used the tempo function to slow down the playback and to speed it up to help him get used to the speed of the song. The main point of feedback was that there should be an explanation or demonstration of how tab works. Maybe a separate help view with diagrams explaining how tab works. He also pointed out that he found it hard to change to size of the view with the zoom function.

Test B (Intermediate Level)
I also showed the app the a guitar player who is familiar with tab. He learned how to use the app quickly. His main complaint was that he found it hard to select a tab segment and that the process of placing notes was a bit laborious. He also pointed out a few thing that I have identified in the unit testing such as the vertical size of the tab view should be larger and that the metronome doesn't work properly. He said that he saw the benifits of having something like this on the phone.

Test C (Music Teacher)
I showed the app to a music technology teacher. He was familiar with tab so he understood what he saw. As he was using the app he tried to zoom in while the phone was vertical and the tab editor disappeared. Also when he tried to play the audio back the app would highlight the strings as if it were playing but there was no audio. It then stalled and i had to reload the app. I played back a tab of a riff from a song called "Ain't Talkin 'Bout Love" (Van Halen). Luckily the teacher knew the riff and recognized it. He then used the tempo functions to increase and decrease the playback speed. He seemed satified that the app was good quality and with some work it could go commercial. He talked about other Iphone Apps and how expensive they are and how this app could integrated into an app that already exists.

5.7 Feedback Evaluation

Question	Novice	Intermediate	Music Teacher
1	good	good	good
2	yes	yes	yes
3	no	no	no
4	no	no	no
5	yes	yes	yes
6	yes	yes	yes

One of the main points of the feedback was how the tab segments were too small for the user to select and often the test user would touch a different tab segment by mistake. So, If I can make the button larger or apply a different method for inputting notes then the App could be used faster and easier. If the user can simply edit the tab and save it the App can be used for its visual aspects alone.

After considering the feedback I recieved from the usability tests and I decided to focus my work on the editing aspects of the App as opposed to the playback errors which are not as high a priority. There were other points of feedback that were relevant to discuss such how tabs could be passed between phones and how the App could read other tab file types. I will discuss these topics in the future work section in the next chapter.

Chapter 6 - Conclusion

6.1 Summary

After the implementation of the core functionality and pin-pointing the most prominent bugs and errors I can conclude that the apps performance is just about acceptable for submission. It can perform simple arrangements easily, but can struggle with complex ones. This reflects how it can be used to learn short sections of music (chords and riffs/licks).

When I started the project I intended to implement more than I had time for. So, I concentrated on the core functions of the App, but I also kept in mind that it should be used to aid learning as that is its main purpose. The main scenario in which the app can be use for is when the end user is the potential learner. So, I proritized which functionality I would implement according to what could help the user learn and discarded the more flash ideas like complex audio manipulation. This works in my favour as the latter would result a heavy workload. For example, an intermediated level guitar player would not have any need to distinguish between playing a note a certain way or just playing the note. Techniques like sliding and bending the string are used by more experienced guitar players. So, I would like to stress that this app has functionality that can benefit the novice and intermediate level guitar players specifically but might not perform exactly what would be expected from an experienced player.

The resulting experience that I can take from this project is the experience developing software for the IOS platform. I have learned Objective C which has stengthened my knowledge of the C programming language and Object-Oriented programming. Objective C is used to code a vast amount of the programmes that are developed for the IOS. I have become familiar with the integrated development environment provided with the SDK (Xcode & Interface Builder) which can be used to develop many other types of application for Mac and the Ipad. So, I feel confident that I can now develop Mac-based applications.

I will continue to work on the app after the final submission. Hopefully, I can submit the App to the AppStore for free download. I can use this a kind of portfolio as I think there are enough examples of functionality implementation that demonstrate my programming capabilities.

6.2 Future work

For the next stage of the Apps progression I would like it to have the ability to use other guitar tab formats that are availible online notably the PowerTab libraries. As described in chapter 1 Powertab files contain tabs in its own format (.ptb). There are hundreds of thousands of songs in PowerTab file format availible for free online. PowerTab have released the Ptparser [7] which is an open-source library that converts PowerTab files to XML format. XML can subsequently be converted to the Property list format and used by the App. It is a C based library so is can be intergrated easily into the App. To do this I will also need to research the Iphone downloading capabilities for it to be able to download PowerTab files.

For the App to use the powertab files it will need to have more functionalty that will manipulate the notes in complex ways such as bending the frequency of the note to simulate the sliding and bending of guitar strings. Also Power tab uses classical notation to indicate that certain sections are to be repeated to save on space on the tab editor. This will also need to be implemented. Powertab is a complex application that as many functions that are too complex to implement on the IOS.

Another area I will work on is the guitar sounds. The App uses a library of acoustic guitar samples. I want to develop an array of guitar sample libraries with different instruments (electric & aucostic guitar, banjo, mandolin). This will allow the user to select what instrument they would like the music to be played back with. Also I would like to integrate a higher level of control over the playback by introducing time signature constraints on the tab to help produce complex musical arrangements like jazz and classical guitar styles.

There are many other potential additions to the App such as tabbing for seven string guitars and other stringed instruments (I.E. Banjos, Mandalion). This is something that existing guitar tab editors like PowerTab and the Itab app don't support. Another idea I have is for the App to take in audio from the user acoustic guitar and analize it to feedback to the user wheather they are playing a song correctly. This could lead to the App being an electronic learner kit.

Appendix A:References

[1] Terry Burrows "Total Guitar Tutor" *Guitar Tablature* page 29

[2] http://www.guitarfretboard.org/

[3] Terry Burrows "Total Guitar Tutor" *Tempo and Rhythm* page 46

[4] http://www.power-tab.net/

[5] http://www.guitar-pro.com/

[6] http://www.comp.dit.ie/bduggan/projects.htm

[7] http://www.power-tab.net/developers.php

[8] http://www.iphoneuxreviews.com/?p=114

[9] http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html%23//apple_ref/doc/uid/TP40007898-CH3-SW1

[10] http://developer.apple.com/library/ios/#documentation/uikit/reference/UIApplicationDelegate_Protocol/Reference/Reference.html

[11] http://developer.apple.com/technologies/tools/xcode.html

[12] http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSFrameworks/iPhoneOSFrameworks.html

[13] http://iphoneproghelp.blogspot.com/2008/12/interface-builder-101-overview.html

[14] http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IphoneOSOverview.html%23//apple_ref/doc/uid/TP40007898-CH4-SW1

[15] http://developer.apple.com/technologies/ios/cocoa-touch.html

[16] http://developer.apple.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html%23//apple_ref/doc/uid/TP40008195-CH32

[17] http://developer.apple.com/library/ios/#documentation/AVFoundation/Reference/AVAudioPlayerClassReference/Reference/Reference.html

[18] http://developer.apple.com/videos/iphone/#video-advanced-audiodev

[19] http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html

[20] http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html

[21] http://itunes.apple.com/us/app/tabtoolkit/id325946571?mt=8

[22] http://www.appstorehq.com/pockettabs-iphone-85509/app

[23] http://www.freetutes.com/systemanalysis/sa2-spiral-model.html

[24] http://newton.cs.concordia.ca/~paquet/wiki/index.php/Spiral_model

[25] http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html

[26] Erica Sadun "The Iphone Developer's Cookbook", *Chapter 15 – Audio, Video and MediaKit* page 611 – 623

[27] Erica Sadun "The Iphone Developer's Cookbook", *Chapter 9 – Building And Using Controls* page 341

[28] http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIButton_Class/UIButton/UIButton.html

[29] Troy Brant "The complete Idiot's Guide To Ipad & Iphone Development" *Chapters 1 - 12* ALPHA 2010

[30] Roger S. Pressman "Software Engineering: A Practitioner's Approach Sixth Edition *Chapter 3* Page 86 - 88 McGraw – Hill International

[31] Erica Sadun "The Iphone Developer's Cookbook", *Chapter 3 – Objective-c Boot Camp* page 91