The Musical Notepad

Technical Manual

Name: Darragh Connaughton
Student no: 13469978

Date: 22-May-2017
Supervisor: Dr Donal Fitzpatrick.

Table of contents

Introduction

Overview

The musical notepad is an application designed to document musical ideas, wherever and whenever they may arise. It converts recorded audio into sheet music at the touch of a button. Musicians now have a musical documenting tool that fits into their pocket.

Recorded audio is processed using the Yin algorithm, which allows for real-time fundamental frequency estimation. This data is collected as a set of ClusterNodes, each containing the position the song was played in, its length and its note. The note length is quantified using k-means clustering, classifying similar notes under the same note value. The clusters are then reordered, chronologically representing the notes and sent to an ABC notation format, whose responsibility it is to generate valid ABC notation for printing and playing from the aggregated cluster set. Music is displayed in both ABC notation and stave notation in a WebView.

These recorded ideas may be stored in a database for later use. Printing of ideas is available, along with storing them onto the phone's internal memory. The musical notepads friend system allows users to connect and share songs, facilitating the rapid sharing of ideas. The musical notepad also comes with a tuner, allowing the user to remain in tune through each recording.

Motivation

Musical ideas occur in a fleeting moment. These musical phrases are often lost to memory, or mutated in the time and effort it takes to capture them with a pen. The musical notepad is designed to quickly capture these ideas, allowing our musician to return to making music while also capturing and documenting his musical progression. As time moves steadily by, the musicians archive of ideas grows. These song fragments can be mixed and matched, allowing their full creation to be realised.

Research

Android Studio

Research was carried out to become proficient at android studio. The primary source of information came from https://developer.android.com/studio/intro/index.html. Android has well documented how to use their IDE, providing easy to follow tutorials for a myriad of components and also a comprehensive getting started section.

Firebase

Firebase provides the database storage for my application. Google have followed a similar approach to their Firebase docs as they did for their Android studio docs. Fully comprehensive tutorials and docs were accessed at https://firebase.google.com/docs/.

Fast Fourier Transformation

My initial plan for the pitch detection was to FFT with a sliding into to isolate the frequencies. I used princeton FFT.java in an attempt to capture frequencies. The results received were inconsistent and wildly inaccurate, most likely due to my inability to interpret the array of complex numbers I received. Material online suggested this approach was not the most appropriate for the task at hand, as the loudest frequency is not always the fundamental frequency.

Yin Algorithm

I got the idea for this algorithm from Tunepal, a query-by-playing search engine for traditional Irish tunes. More accurately, I got idea from Tunepals creator Dr Bryan Duggan, a computer science lecture in the Dublin Institute of Technology.

Glossary

ABC Notation
ABC notation is a shorthand form of musical notation. In basic form it uses the letters A through G to represent the given notes

Stave Notation
is a set of five horizontal lines and four spaces that each represent a different musical pitch

Key Signature
Any of several combinations of sharps or flats after the clef at the beginning of each stave, indicating the key of a composition.

Time signature
An indication of rhythm following a clef.

Accidental
an accidental is a note of a pitch that is not a member of the scale
Pager

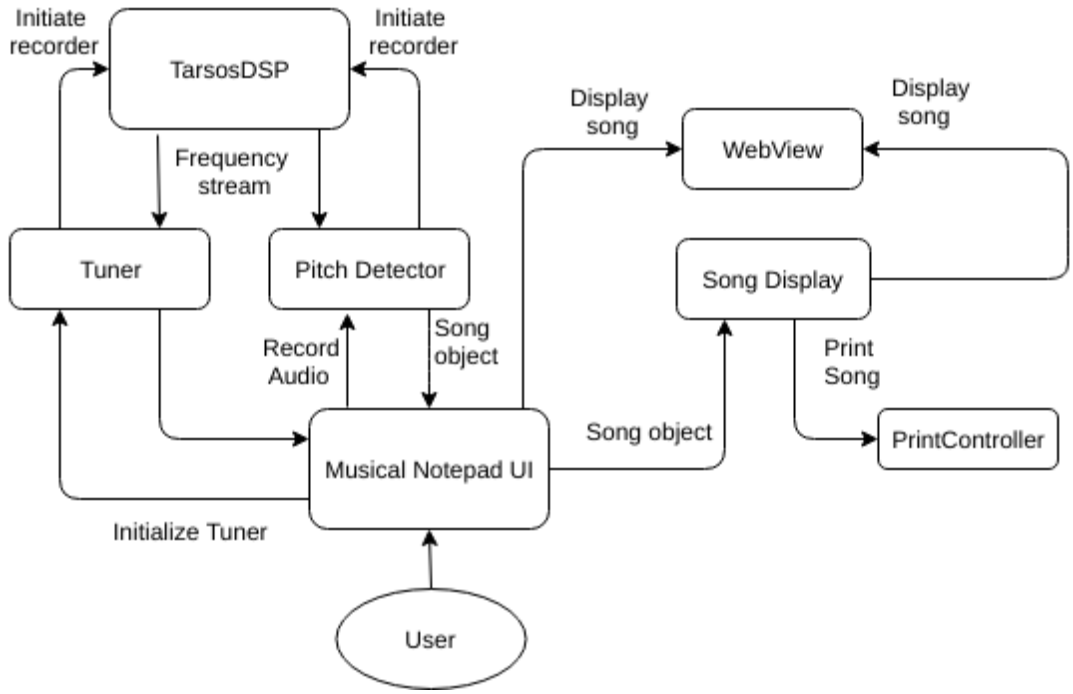
WebView

Fragment

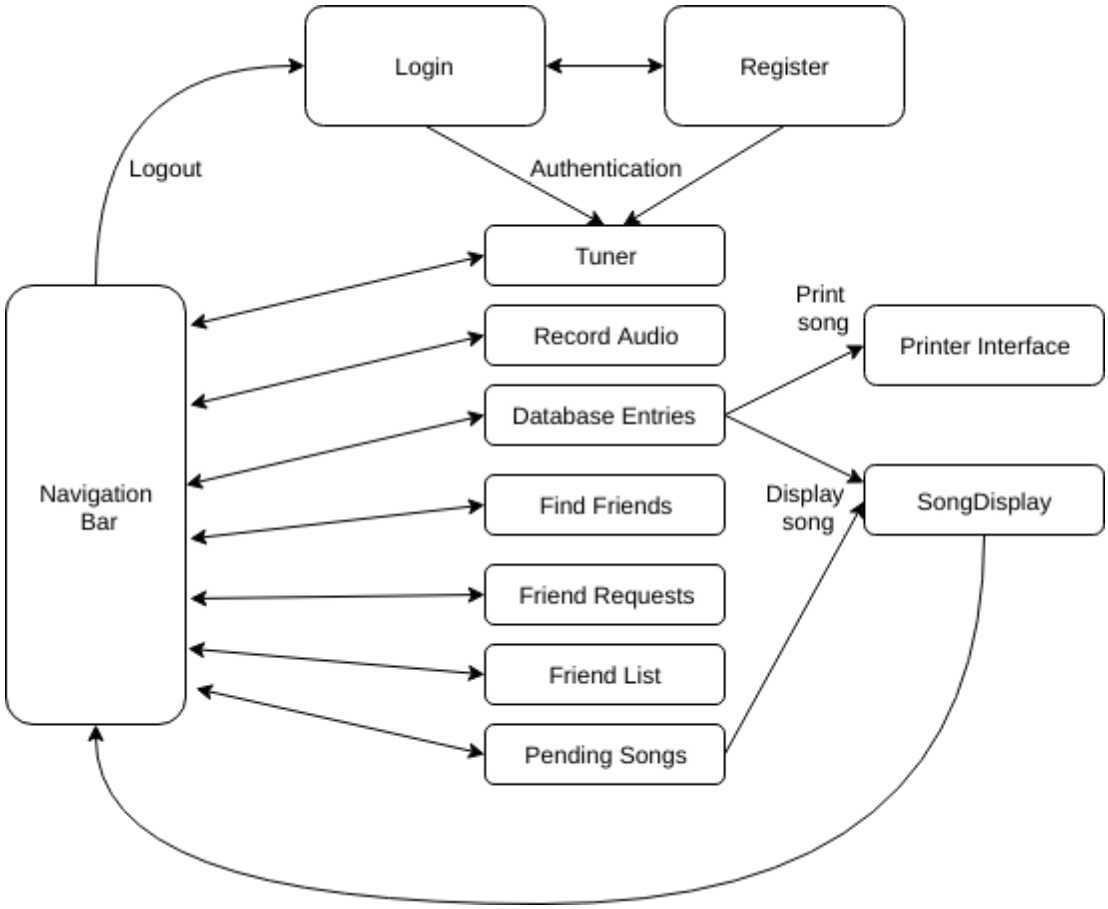
A Fragment represents a behavior or a portion of user interface in an Activity.

Design

High Level Backend Diagram



Front End Diagram



Database Structure

Users

6YG5QLbA2GPoCZffRNr2uu2NnvQ2

FriendList

BVnYi7Cp1lQffCiAty4RMvPR5hx1: "Matthew Hagan"

H2OsDePRJ4bzGc8fcd6gy9Ck2oS2: "Luke O'Regan"

JivrbMiQnqeCD71ML7yTJYofcD53: "Stephen Cassedy"

o9LNgZ3XBHhaD0EexkDZ8xqxb0P2: "Eoin Magner"

email: "darragh.connaughton5@mail.dcu.ie"

username: "Darragh Connaughton"

profilePhoto: “https://lh6.googleusercontent.com/-kYaIAR4PgCM/...”

songId

1493311608806

keySignature: “C”

l: “1/8”

name: “test1”

notes: “z4_d'3_g'|_d'2_b_g_ad2(c|\nc)z7|”

profilePhoto: “https://lh5.googleusercontent.com/-SkI...”

timeSignature: “4/4”

uid: "zIqitZrvudPfLvTsbFCVIcGepYI3"

SongRequest

1495374325422

PendingSong

1495203037802

Database Fields

username

A unique username derived from Google account or entered manually when registered by email.

profilePhoto:

This field contains url pertaining to the profile picture associated with the user's google account. This field is left blank when registering by email.

FriendList:

This field contains a list of unique ids belonging to the users you are currently friends with. This unique id gives the current user access to the users public data.

PendingSong:

This field holds all the songs that the user has sent that are awaiting acceptance.

SongRequest:

This field holds all the pending incoming songs.

email:

This field contains the email of the current user. The field is derived from the users google account, or entered manually on registration.

songId:

Songs are normalised using the timestamp of creation as a unique id. This is to ensure that all songs within the database have a unique reference, regardless of the details of the song. Within the timestamp we have the information pertaining to the song:

1. *keySignature:*

   This is the key signature of the song. To be used for displaying purposes and used in the conversion process from display ABC notation to ABC notation that is required for audio playback.

2. *l:*

   This field controls the meter, the length of one standard unit in ABC notation. I have a default value set at ⅛ or 1 quaver per note. This is done to allow for easy transition between x/4 to x/8 time.

3. *Name*

   This field represents the display name of the song in question.

4. *Notes*

   Notes represent the ABC notation note progression. These notes are in display friendly format, allowing for quick display. They will have to be processed one more time to allow for midi playback.

5. *profilePhoto*

   This field is only applicable if the song is sent to another user. On sending a song, this field is filled with the sender's profile picture url. This will be used in the Database Entries activity, displaying the sender's profile picture alongside the song. This is allow the user to identify where the song came from.

6. *timeSignature*

   This field holds the time signature of the song. Used during the initially audio processing to ensure that bars have the correct number of beats, and for display after the fact.

7. *Timestamp*

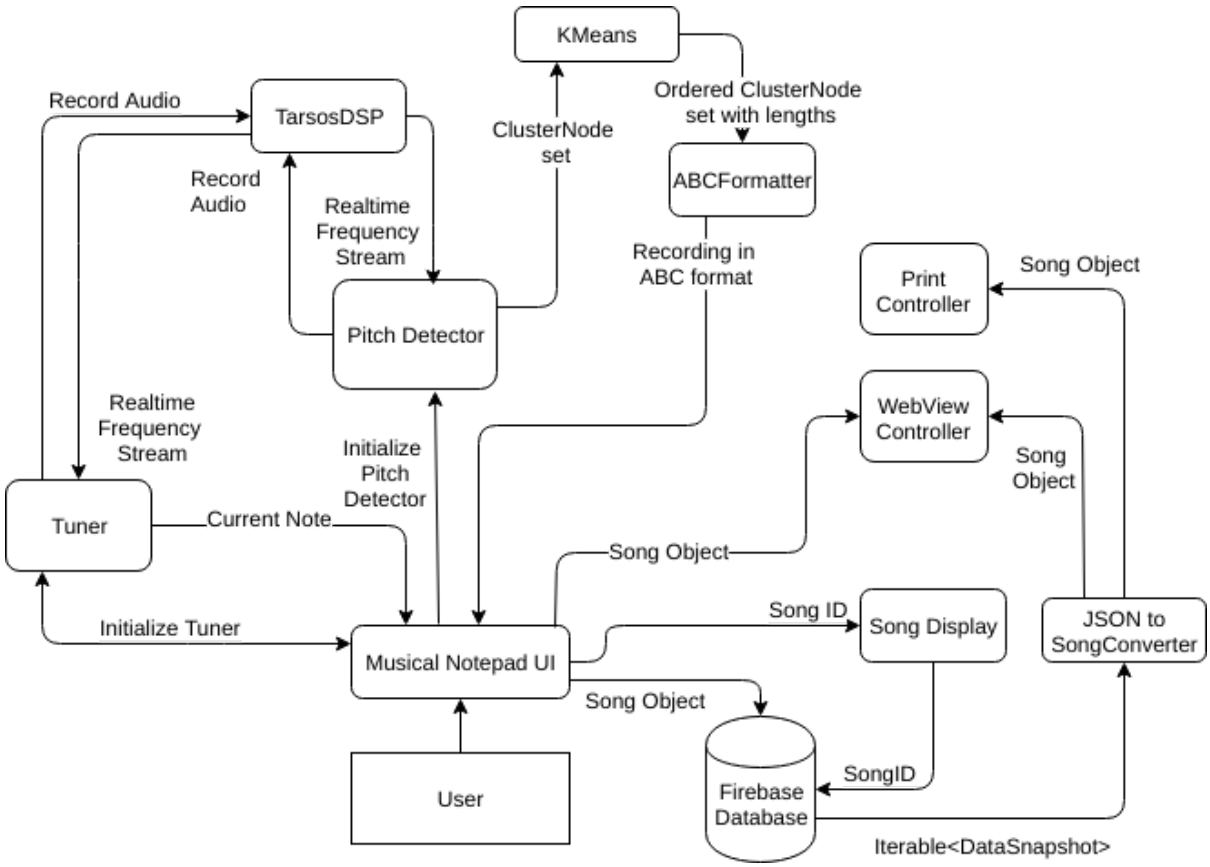
   Timestamp as mentioned above holds the unique key attached to this song.

8. *Uid*

   This field is only marked when the song is sent between users. It holds the UID, the primary key associated with the user who sent the song.

Implementation

Data Flow Diagram.



Objects:

Frequency

A frequency object is made up of a float frequency and an int octave. It is used in the FrequencyController.

KeySignature

KeySignature contains a getNotes() method that returns the notes associated with the current key signature (eg) D major would return “D” and “C”. KeySignature also contains three String:

- keySign - saves the current key.
- abcFormat - saves the key in ABC friendly format.
- type - indicating whether the key is sharp, flat or nothing.

Song

A song object contains the fields discussed under songId in the previous section of type String. This include: name, notes, timestamp, timeSignature, keySignature, L, UID, profilePhoto.

SongArrayObject

Contains ArrayList relating to the information of multiple songs. The same fields discussed under Song are present.

UserProfileDetails

UserProfileDetails is an object containing the details associated with a user. It contains the users name, email address, unique ID and their profile photo, if applicable.

Backend Components:

All backend components are in app/src/main/java/com/darragh/musicalnotepad

Cluster/Cluster.java

Is a cluster object containing a centroid, an ArrayList of ClusterNodes and a default constructor. Clusters are used in k-means clustering.

A single node unit within the cluster. It contains four fields:

1. Position

   The position the note occurred chronologically. This is used to ensure the output of the cluster is in the correct order.

2. Length

   Is the length of the note.

3. Cluster

   Cluster number relates to the cluster the node is currently in. This is used after the aggregation of the ClusterNodes from separate clusters and is used to indicate the length of the note.

4. Note

   This the note that has been played.

Cluster/ClusterToABCFormat.java

This method formats the aggregated cluster, writing a String containing the combination of all the notes in the ClusterNode set in the correct order and in the right format. A beat counter keeps track of the current notes within the bar, creating a new bar once the current counter grows greater than the time signature a bar line is drawn and the barCounter is incremented.

A new line is trigger on every second bar, ensuring the output is clear and readable to our user. processNode() handles the three types of note we expect: no accidental present, accidental present and space (no note). Three paths are created that process the node accordingly. The sortOctave method handles the current octave, searching the String for the integer attached to the note that identifies the octave, (eg) A1 -> A; A2 -> a; The length of the current note is derived from setNoteLength() which returns the cluster number within the node, being careful to set cluster 1 to “”. Conditions are set to handle when the current note exceeds the amount of space that is left within the bar. In ABC notation this is handled by parenthesis, generating a tie between the notes. All accidentals are stored as flats by default. To overcome this, the sortAccidentals() function controls which accidental is printed with the note, by checking the current Key Signature type against the current note. (eg) D Major is made up of F# and C#, this is marked as a sharp key, ensuring all other accidentals are printed as sharps. Finally there is a fillExcessSpace() that ensures that the remaining space within the last bar is filled with rests.

Cluster/kMeans.java

KMeans handles the k-means clustering. This is used to determine the note length from steam of notes received from the Pitch Detector. The notes are stored in an ArrayList<ClusterNode>. The kMeans is initialised using this dataset. The number of clusters is determined using the shortest and the longest length note. The number of times the smallest note fits within the greatest note is the number of clusters we will have, the centroids for these clusters are multiples of the shortest length note. After the centroids have been set, each ClusterNode is assigned the closest cluster, based on the distance from the centroid of that cluster. Once all ClusterNodes have been accounted for, the centroids are calibrated by getting the average of all the element lengths within the cluster, each cluster is then emptied and the process is continued until convergence is reached. Convergence is when the clusters do not change for 2 iterations in a row. Once this has occurred, the aggregateCluster() then places the notes from all clusters into a single cluster, in chronological order; based on the position variable associated with each node.

FriendFinder/FriendFinderController.java

This class is responsible for finding users within the Firebase based on a search query entered from the frontend. All the users are gathered using their unique id. This list is iterated through, adding the users whose name or email contains the search query as a substring. The current unique identifier is checked to ensure it is not the user unique identifier and that the person you are adding is not already a friend.

FriendList/FriendListController.java

This class is responsible for gather the FriendList of the current user. It stores this detail in an ArrayList of UserProfileDetails.

FriendRequest/FriendRequestController.java

This controller class deals with the functionality relating to adding friends. The sendFriendRequest() function writes a friend request in the desired user's /FriendRequest/ directory and writes the a marker to their /pendingFriendRequest/ directory. The /FriendRequest/ directory has public access, allowing any users to write a request to another user. The acceptFriendRequest() function is called from the desired user's profile when they accept the request. The requesting user is written to the FriendList directory, the requests are then removed using the removeFriendRequest() function. This function is also called when the request is rejected.

Modules/JavaScriptInterface

The sheet music is printed using javascript. This interface acts as the middleman between the java code and the javascript running in the WebView.

Modules/JSONToSongConverter

This code is responsible for converting the JSON received by the Firebase into a Song object that can be displayed or sent to another user. The data within the JSON is iterated through using Firebases Iterable<DataSnapshot>. A switch statement is then used to capture each component and store it within the song object.

Modules/MIDIController.java

The ABC format being used to print stave notation has been modified based on the current key signature. (eg) for D Major, all sharp symbols beside C and F would be removed and accounted for by the key signature at the start of the line of music. In order to use midi playback, the MIDIController places all the accidentals that have been omitted due to they key signature back into the music.

Modules/PrintController.java

Print controller allows the user to print a song or simply store it in internal memory. It has a similar structure to the WebViewController with the addition of androids PrintManager object.

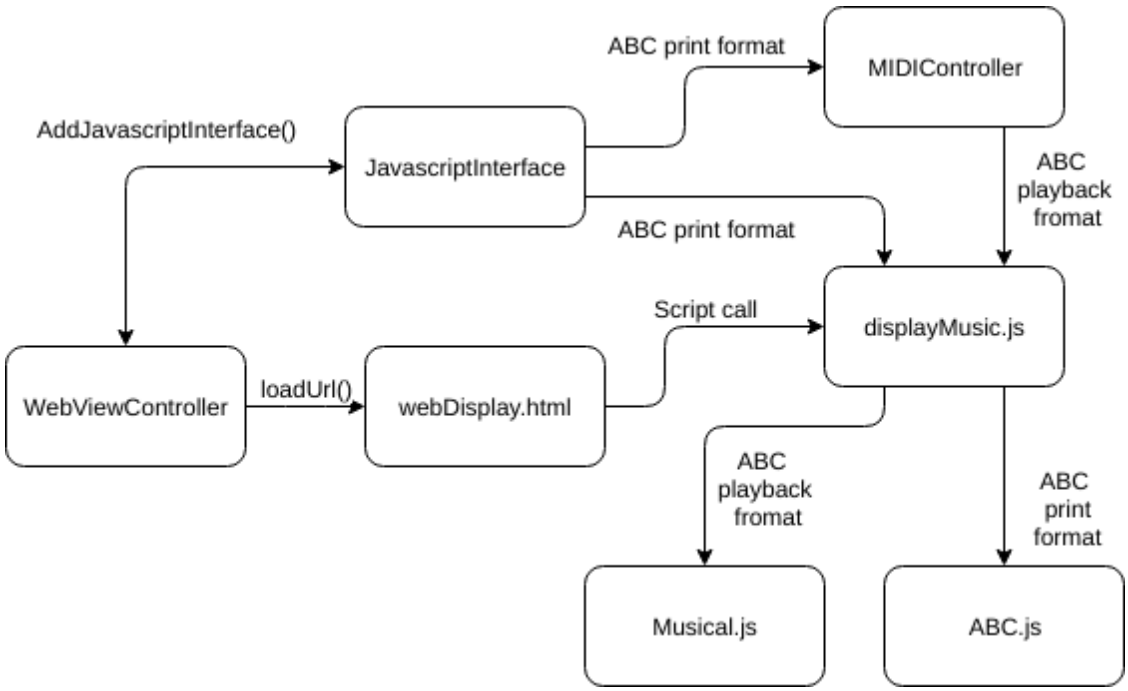
Modules/SongSharer.java

This class handles the transfer of songs between users. It takes a list of recipients and the current user's profile picture, which will be attached to the song. Similar to the friend request system above, songs are written to a provisional directory where they wait acceptance.

Modules/WebViewController.java

Songs are passed to the WebViewController and displayed on the WebView. A WebView is an Android object that is used to display web pages. A WebView is used because the two libraries used to display stave notation and for midi playback are both javascript libraries. The WebViewController utilizes the JavaScriptInterface from above to send data to the HTML, displaying the stave notation.

WebViewController Diagram



Pitch_Detector/FrequencyController.java

This class deals with scaling down the current frequency received from the TarsosDSP library into a range between 254.284 (middle C) and 538.808f (C and octave above). The frequencies are scaled down and their respective octaves are saved. This class is used by both the tuner and the pitch detector.

Pitch_Detector/PitchDetector.java

This class is used to document musical ideas. The pitch detector is started from the main activity. The recorded pitches are scaled using the FrequencyController mentioned above and the note pertaining to this frequency is extracted and stored within an ArrayList. This list is then processed, converting the stream of letters into a list of notes with their corresponding lengths (the number of contiguous occurrences of this notes). The note details are stored in a ClusterNode, ready to be passed to the kMeans cluster.

Pitch_Detector/Tuner.java

This class controls the Tuner activity. Similar to the Pitch Detector, TarsosDSP pitch detector converts the recorded audio into a stream of frequencies, which are in turn converted into a stream of notes. These notes are displayed in real time on the screen. A threshold, of 2Hz, around the perfect pitch of each note is considered in tune. If the note is less than this threshold, the difference from perfect is displayed alongside a flat symbol. If it is greater than the threshold, the difference is displayed with a sharp symbol.

SongRequest/SongRequestController.java

This class is responsible for gathering the song details from the list of song requests present in the songRequest directory.

SongRequest/SongRequestListController.java

This class has two options inside it. Either to accept the song, which will, in similar fashion to the friend request acceptor, add the song to the current users songId directory while simultaneously removing the pending request. The option is also available to simply remove the request without accepting using the RemoveSongRequest() function.

displayMusic.js

This javascript controls the display and playback of ABC notation in webDisplay.html. It interfaces with java code through the JavaScriptInterface.

Frontend Components

DatabaseEntries/DatabaseEntries

XML: databaseentries.xml

This class controls the display of the user's database contents. The entries are displayed in a ListView, controlled using a custom ArrayAdapter called SongListAdapter.java. There are two click options available for user's on this page, a long click and a normal click. A long click generates will call the generatePopupMenu() function which creates an android PopupMenu presenting the users with three options, Share with a friend, view and delete. The delete option simply removes this entry from the Firebase. Share with a friend will call DialogController.java mentioned above. A normal click will invoke the songDisplay.java activity.

Similar to all activities, a navigation bar is present and instantiated using setUpNavigationBar(), which delegates control and responsibility of the Navigation bar to NavigationView_Details.java.

DatabaseEntries/SongListAdapter.java

SongListAdapter is an ArrayAdapter that is responsible for designing each row within the ListView. Each row will contain rowlayout.xml, which is contained under the layout directory. Each row will be populated by a single song from the SongArrayObject passed as a parameter to the adapter. Fields within this layout include Song name, key signature, time signature and where applicable profile photo. An array of songs is passed.

FriendFinder/FriendFinder.java

XML: findfriends.xml

Find friends provides a EditText that allows users to search the database for a particular user. On enter, the query is sent to FriendFinderController which will return a list of users. This list of users is passed to the FindFriendListAdapter, whose responsibility it is to populate the ListView with the user's data. A navigation bar is present in this activity.

FindFriendListAdapter

XML: friendrow.xml

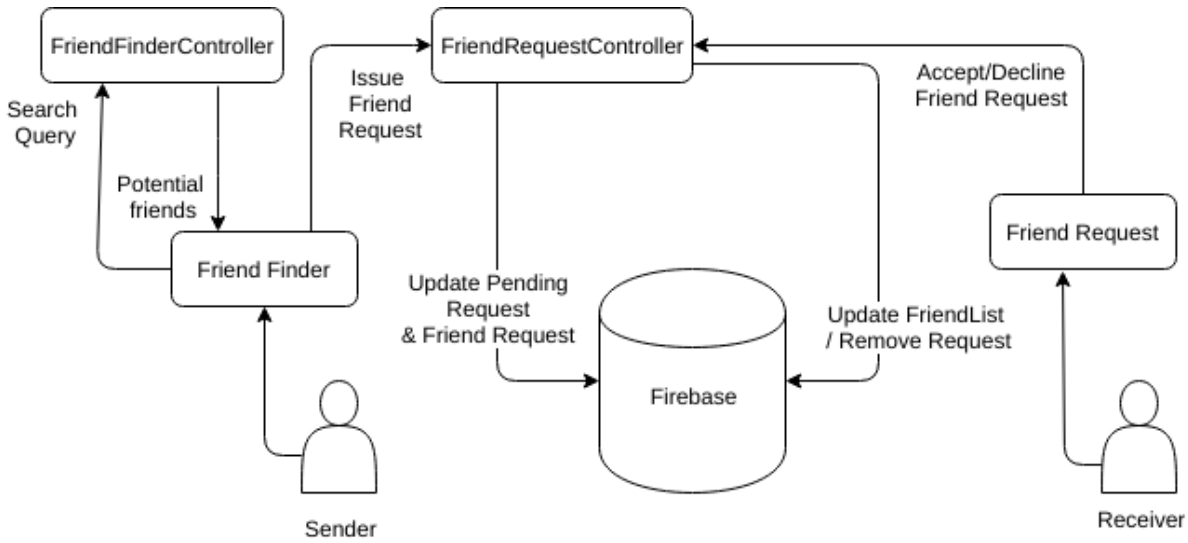
FindFriendListAdapter populates the ListView with the potential users received from the FriendFinder. Each row contains the user's name, email and potentially their picture, if applicable. A single button is present, whose sole responsible it is to send a friend request, done through the FriendRequestController mentioned above.

FriendList/FriendList.java

XML: friendlist.xml

A list of the users friends is gathered using the FriendListController mentioned above. This list is passed to the FriendListAdapter, whose responsibility it is to populate the ListView. Similar to all activities, a navigation bar is instantiated oncreate using setUpNavigationBar().

Friend Finder and Friend Request



FriendList/FriendListAdapter.java

XML: friendlistrow.xml

FriendListAdapter is a custom ArrayAdapter responsible for populating FriendList's ListView with the users passed to it as a parameter. Each row in the ListView is populated with friendlistrow.xml which contains a username, email and a profile photo, if applicable.

FriendRequest/FriendRequest.java

XML: friendrequest.xml

FriendRequest deals with the user's incoming friend requests. The gatherUsers() function gets the list of users present in the /FriendRequest/ directory. These are stored in user objects before they are sent to the FriendRequestListAdapter(), whose responsibility it is to populate the ListView. This page also has a navigation bar attached.

FriendRequest/FriendRequestListAdapter.java

XML: friendrequestrow.xml

FriendRequestListAdapter is responsible for populating each row in FriendRequests ListView with friendrequestrow.xml. The fields contained in friendrequestrow.xml include username, email and potentially a profile photo. There are two buttons per row as well, an accept and a decline button. These buttons are connected to the FriendRequestController as mentioned above.

Login_Register/Register.java

XML: register.xml

This activity deals with user registration. There are three fields to be filed, username, email and password. There is check to ensure all these fields have been filled in correctly. The user is then authenticated using Firebase.createUserWithEmail() and they are written to the Firebase. Then Tuner activity is loaded shortly after this.

This activity does not contain a navigation bar.

Login_Register/SignIn.java

XML: login.xml

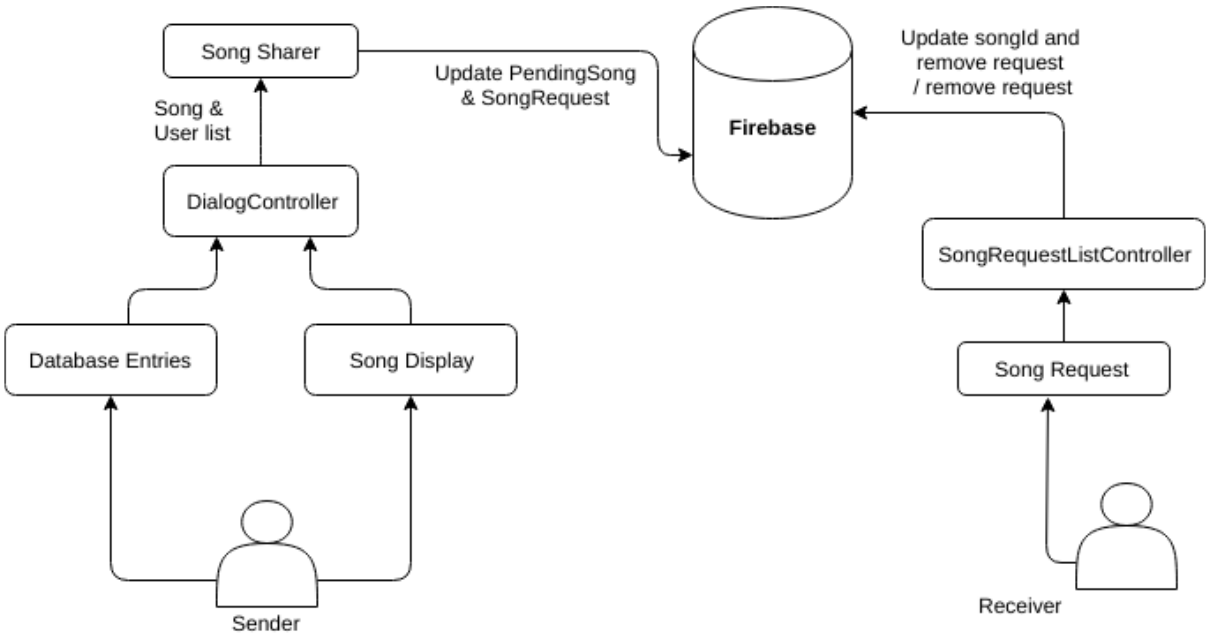
This activity handles login for current users. There are two login options available in this activity. Firstly, you can login in using the email and password that was registered under the register activity. The second method is by logging in via the user's Google account. This is done by selecting the Google button at the bottom of the screen. This is the preferred means of login and authentication in the Musical Notepad. The application is Android based and a google account is a prerequisite for an android phone, meaning all users will have a Google account at the ready. Authentication with Google is seamless and also provides a profile picture for each user. This activity does not contain a navigation bar.

SongRequest/SongRequestList.java

XML: songrequestlist.xml

This activity is responsible for displaying all the users current song requests. It is primarily made up of a ListView populated with song requests. These requests are gathered from the SongRequestController before they are passed to the SongRequestListAdapter to be displayed.

Song Sharer and Song Request Diagram



Pitch_Detector/Tuner.java

XML: tuner.xml

The functionality of this class has been mentioned above. The fields within tuner.xml are filled in on based on the frequency that is received. Flat and sharp indicated in Hz how flat and sharp the note is, glowing red to capture the user's attention. Once in tune, the note field located in the center will glow green and the text will be highlighted bold.

SongRequest/SongRequestListAdapter.java

XML: potentialsongrequestrow.xml

SongRequestListAdapter is a custom adapter used to populate the ListView in SongRequestList with song requests. Each row contains potentialsongrequestrow.xml which contains the songs name, key signature, time signature and the user's profile picture, if applicable. There are also three buttons per row. Accept: which accepts the song using SongRequestListController.addSongToList; Decline: which removes the song request using SongRequestListController.RemoveSongRequest; Preview: which using the songDisplay activity to preview the song.

RecordAudio/MainActivity.java

XML: activity_main.xml

This activity deals with recording and converting audio into sheet music. There are two Android Spinners, which is basically a drop down menu, located near the top of the activity. The user is able to choose the time and key signature they would like to record in from these spinners. It contains a record button. This button flashes while it records. This effect is achieved by swapping between the original record photo and a record photo with a red circle that indicates it is recording. A handler controls the frequency of transition, giving the record button its lingering effect. After recording is finished, the spinners and the button are removed. A WebView now displays the recorded audio in ABC notation and in stave notation. An EditText near the top prompts the users to enter a song name and two buttons near the bottom, save and discard, seal this recordings faith.

Modules/DialogController.java

This class controls the Dialog that is used to when the user wishes to share a song. This occurs in both DatabaseEntries activity and songDisplay activity. The DialogController generates a list of email addresses belonging to the current user's friends. There is multi-select enabled, allowing the user to send the current song a multiple friends at once. Once selected, the current songs details and the list of friends are sent to the SongSharer who manages the rest of the transaction.

Modules/songDisplay.java

XML: songdisplay.xml

This class acts as a generic song displayer and is used multiple times throughout the application. Song details are passed through the Intent when starting this activity.

Modules/NavigationView_Details

XML: userprofile, navigationitems

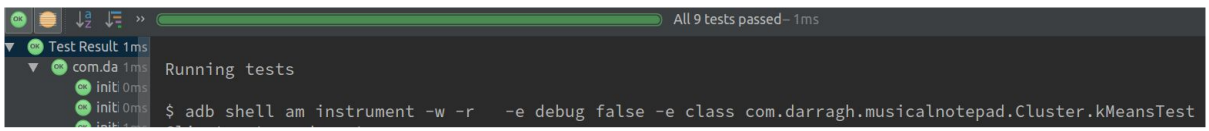
This class is attached to all activities. It controls the navigation bars display and functionality. The navigation is made up of two components, the header and the ListView containing the

menu items. The header contains the user information; username, email and the profile picture, if applicable. The bottom majority of the navigation bar is made up of menu items, derived from navigationitems.xml. A click listener is set to the navigation bar, based on which row was clicked an activity will be loaded.

Testing and Validation.

Unit Tests - androidTest/java/com/darragh/musicalnotepad

Cluster/kMeansTest.java



This testing class contains nine tests.

- testABCFormat()

  This test is designed to validate the ABC String received after k-means clustering. Using the convergence_testCluster(), a key signature and a time signature as a parameter.

- sortOctave()

  This test is designed to validate clusterToABCFormat sortOctave() function. It does so by checking the output received against the expected output.

- aggregateCluster()

  This test is designed to validate the output of aggregated cluster, this function is responsible for aggregating the clusters, in chronological order, after the k-means cluster has reached convergence. This created by simulating the k-means cluster

using kMeansMain_noABCFormat, preventing the cluster from being changed by ABCFormatter we are able to validate the output.

- convergence()

  This tests checks the cluster number and length for each cluster on convergence, before the cluster is aggregated. This ensures that the cluster is ended at the correct point with the correct values.

- AssignClosest_FourCluster()

  This simulates a single iteration, including the initiation of the k-means algorithm along with the recalibration of the centroids of each cluster present. This test ensures that the cluster is initiated correctly at the beginning, as failure to do so would ultimately lead to an incorrect output.

- initiate_OneCluster()

  This test checks the centroid initiation, checking to see if Cluster one is assigned the correct centroid.

- initiate_TwoCluster()

  This test checks the centroid initiation, checking to see if Cluster two is assigned the correct centroid.

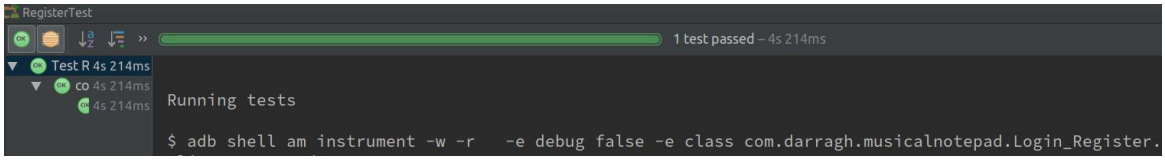
- initiate_ThreeCluster()

  This test checks the centroid initiation, checking to see if Cluster three is assigned the correct centroid.

- initiate_FourCluster()

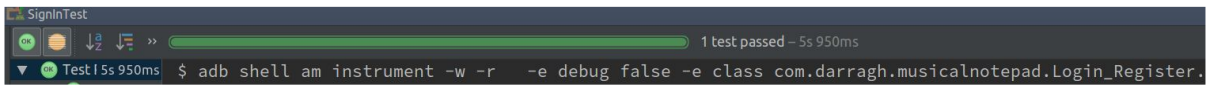
  This test checks the centroid initiation, checking to see if Cluster four is assigned the correct centroid.

Login_Register/RegisterTest.java



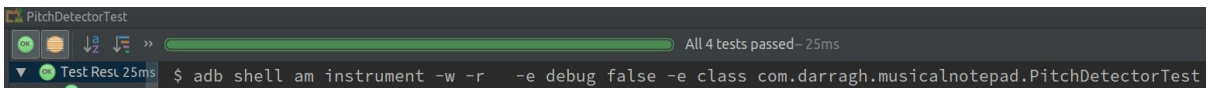
This is an Android Espresso test. It simulates a user typing in their email, username and password. It then simulates the click. If the user is successfully registered the test passes. Note, the email address must be unique for each user. The email address must be updated each time the test is ran. Running the test twice in a row will cause the test to fail, as the user is already registered.

Login_Register/SignInTest.java



Similar to the approach mentioned above. SignInTest simulates a registered user logging in with their email and password. The user is successfully logged in if the credentials are correct and the test is passed.

PitchDetectorTest.java



This test validates the Pitch Detector and in turn validates the Tuner. The tests cater for four octaves above middle C. I consider this exhaustive for the application at hand, as the highest note I can achieve would be an octave below this.

- ❏ getBeats()

  A list of all potential time signatures is passed sequentially to the getBeats() function and compared against the expected output.

- hz_to_note_octave1()

  This test validates the hz_to_note converter. It tests the first octave, using the boundaries, the lower and upper limit, of each note as a test.

- hz_to_octave2()

  This is designed to test the second octave. Checking that the note is correct and also the octave attached to the note is also correct.

- hz_to_octave3()

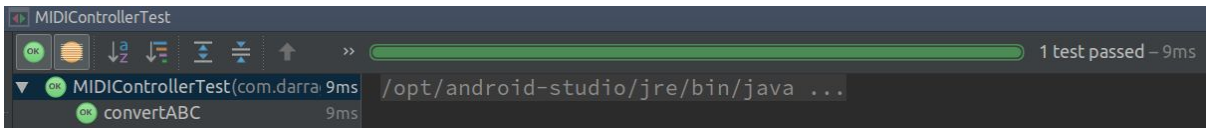
  This is designed to test the third octave in a similar fashion mentioned above.

- hz_to_octave4()

  This is design to test the third octave in a similar fashion mentioned above.

Unit Tests - test/java/com/darragh/musicalnotepad

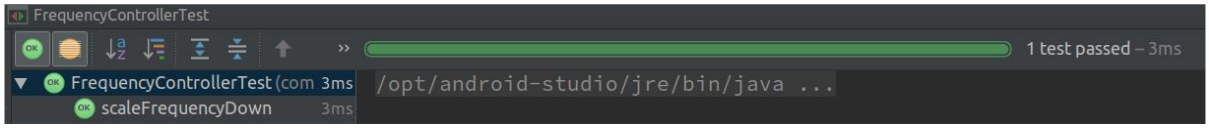
MIDIControllerTest.java



This test validates the MIDIControllers conversion of ABC notation into a playable friend format. This is done by simply adding the accidentals accounted for in the key signature back into the abc format. There are two conditions in particular that are tested for:

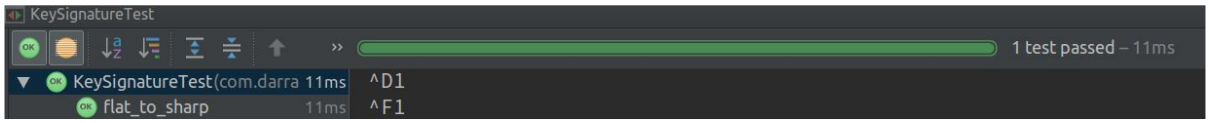
1. That natural notes are ignored.
2. Notes that are ties over the barline are accounted for correctly.

Pitch_Detector/FrequencyControllerTest.java



This test validates the FrequencyController, whose responsibility it is to scale down frequencies greater than the first octave, while also recording the octave of the frequency.

Pitch_Detector/KeySignatureTest.java



This test validates the KeySignature.flat_to_sharp() function.

User Test

Questions the users were asked are available in the Appendix under Questionnaire. Users were asked to perform a series of basic tasks on the application. These tasks use the full range of features available, apart from the printing option. The users then rated the quality, accuracy and ease of use of the project. With the users permission, I also observed them playing with the application, hoping they would uncover some bugs.

The general consensus was very good. Users found the navigation easy to use and had little difficulty finding the specified sections. The tuner was highly rated as well with most users being satisfied with the output they received. Recording accuracy was spot on with no users complaining about note omission or addition.

The length of the notes ranked rather high too, however, they very rarely scored full marks. I believe this is due to the means of deriving note length. K-means clustering provides accurate results for most cases. Since the number of clusters is derived from the shortest and the longest note, small hiccups when playing can sometimes interfere the output accuracy. The note lengths relative to each other are still fine, but on occasion the notes will be given marked as crochets or dotted crotchet where a quaver was applicable. This doesn't matter however, because the note is quantified by the tempo at which the piece is played.

Discovered Bugs:

1. Long click: display returns null:

   Options for database entries can be accessed through a long click. The display option would load songDisplay as expected, but would display null. This bug remained unnoticed as this means of display is a secondary display approach, normal click is used more frequently. The solution was a simply one. The song was not loaded due to an incorrect link to the song. The directory the song is suppose to be read from is passed through the Intent from the databaseEntries activity to the songDisplay activity. This directory was incorrect, leading to no song details being loaded, thus displaying null.

2. songDisplay sharing option: null song sent to user:

   Similar to bug number one, the songs details was loaded from the wrong directory. This lead to no details being loaded and a null object being sent to the other user.

3. Song request displays current user's profile photo, not the sending user's profile photo.

   This error went completely unnoticed during my own personal testing of the user interface of the application. Delighted to see a profile picture loaded, I failed to notice that each time it was my face. Only when multiple song requests were present did I notice the bug. I sent a String parameter, pertaining to the current user's profile picture to the function gathering the song details from the pending song list. I was then systematically adding this photo to each song instead of reading the profile photo from the song object pulled from the Firebase.

4. Friend list didn’t display the user’s profile picture or email. It also printed their unique id.

   The problem was a simple one. I had omitted the line that saved the user's email address from the code. This cause the UID to be saved as the email and for no profile picture to be recorded. Simply adding the line back in fixed the problem.

Sample Code

Cluster/clusterToABCFormat.java

processNode() is responsible for generating the abc String. There are three paths that can be taken:

1. Encounters a space, print z, the symbol for space. The length of the note here is determined by the cluster number using setNoteLength(). leftBracket and rightBracket handles tied notes.
2. An accidental is present. In this case the note is sent to sortAccidentals() which will set the sharp or flat version of the accidental depending on the key signature type.
3. No accidental. In this case, we send the note to the potentialNaturalNote() function which will check to see if this note is an element of the key signature, if so it will mark it as a natural note, or “=” in ABC notation.

```
private static String processNode(String noteProgression, ClusterNode node, KeySignature key,String leftBracket,String rightBracklet){
    if(node.note.equals("SPACE")){
        return noteProgression + leftBracket+ "z" + setNoteLength(node.cluster) + rightBracklet;
    }
    else{
        if(containsAccidental(node.note)){
            return noteProgression + leftBracket + sortOctave(sortAccidentals(key,node.note)) + setNoteLength(node.cluster) + rightBracklet;
        }
        else {
            return noteProgression + leftBracket + potentialNaturalNote(key,node.note) + sortOctave(node.note) + setNoteLength(node.cluster) + rightBracklet;
        }
    }
}
```

Cluster/kMeans.java

Aggregated cluster puts our clusters back together again based on each node's position variable. Position keeps track of the play order of each note, ensuring the output is in chronological order. The method commences by creating an integer array, each index within this array will hold the current position in each cluster. The fragmented clusters are in ascending order. The first element is taken from the current pointer position of each cluster. The correctCluster() method compares the current position i with each of current cluster positions. The node that matches this position is added to the aggregated cluster and the cluster that node belongs to position is incremented.

```
public static ArrayList<ClusterNode> aggregateCluster(){
    int[] clusterPositions = new int[numberOfClusters];
    ArrayList<ClusterNode> aggregatedCluster = new ArrayList<>();
    for(int i=0; i<clusterNodeSet.size(); i++){
        int[] positions = new int[numberOfClusters];
        for(int j=0; j<numberOfClusters; j++){
            if(clusterList.get(j).clusterList.size()>clusterPositions[j]){
                positions[j] = clusterList.get(j).clusterList.get(clusterPositions[j]).position;
            }
            else{
                positions[j] = -1;
            }
        }
        aggregatedCluster.add(clusterList.get(correctCluster(i,positions)).clusterList.get(clusterPositions[correctCluster(i,positions)]));
        clusterPositions[correctCluster(i,positions)]++;
    }
    return aggregatedCluster;
}
```

main/assets/displayMusic.js

displayMusic() is the petite powerhouse behind the WebViewController.

- The displayMusic() function calls the JavaScriptInterface requesting the abc print friendly format. This is placed inside the ‘notation’ div tag.
- The playAudio() connects to Musical.js and is responsible for midi playback. The instrument chosen here is the piano with a tempo of 200.

```
function displayMusic(){
        var chorus = getTuneDetails();
        ABCJS.renderAbc('notation', chorus);
}
function getTuneDetails(){
    return Android.sendData();
}
function getABCTuneDetails(){
    return Android.sendABCData();
}
function playAudio(){
    var piano = new Instrument('piano');
    piano.play({tempo:200},chordProgression());
}
```

Problems and Resolutions

As stated above, my initial approach for pitch detection was to use Fast Fourier transformation with a sliding window. I was unable to make heads or tails of this output. Ultimately I decided to go down a different route and use the Yin algorithm.

My initial design was through a pager and fragments. A pager hosts multiple fragments, while fragments contain an activity. This approach clashed with the introduction of the navigation bar. Each fragment occupies the entire screen on the phone. Navigation between is done through sliding the screen left or right, depending on the amount of fragments within the pager. The navigation bar is activated by sliding from the right. The fragment and the navigation bar do not work well in tandem. The fragment in the pager takes priority. Even if you can get a hold of the navigation bar, by the time you have traversed half the distance required to reveal it, the pager will take over once more. I decided to omit the pager and fragment layout, as I felt the navigation bar as the main means of transportation is more comprehensive.

I was unable to find any library that could print stave notation. The only ones that were available are python based. In order to run python on android, I would need to code on the scripting layer, which would have caused me a lot of torment. I discovered a javascript library that does the job I am looking for perfectly. The next step was to incorporate this into my application. I discovered that android had an in built WebView, which is an embedded HTML

web page that can we controlled using javascript. This brought with it another task, how can I send data from my java backend to the javascript controlling the WebView. The solution to this problem was the javascript interface attached to the WebView. From here I was able to request data from the java code for my javascript code.

Future Work

I would like to add a delete option on my Friend List. Currently the user is unable to remove a friends. This would be a nice feature to implement. I would also like to update the songDisplay module. Currently, it doesn't differentiate between preview song and the regular song display. On preview the user is able to print and share the field. This isn't a bug as they should be able to do this anyway. However, I would like to restrict this functionality to when the song has been fully accepted first, as I feel this would be more professional.

I believe there is more work to be done on deriving the note length. The one size fits all approach, although adequate for my project, it is not sophisticated enough to make my application marketable. Although I am proud of the UI, I don't believe it is at marketable quality just yet. More work is needed to achieve a clean and crisp interface worthy of the app store.

Appendix

Consent Form

Project: The Musical Notepad
University: DCU school of computing.
Investigator: Dr Donal Fitzpatrick.

The research is undertaken as part of an undergraduate final year project, to gain an understanding of application in terms of quality and ease of use. The musical notepad is a music documentation tool, facilitating easy storage and sharing of musical ideas.

I have read the Plain Language Statement (or had it read to me)	Yes/No
I understand the information provided	Yes/No
I have had an opportunity to ask questions and discuss this study	Yes/No
I have received satisfactory answers to all my questions	Yes/No

This research project is taken on a voluntary basis, participants have the right to back out at any moment, withdrawing themselves from the research study. Data can be deleted on request at any moment and will be automatically deleted after the 25th of May. User will not be harassed or pressed for information.

Participant's Signature:

Name in Block Capitals:

Witness:

Date:

The Musical Notepad Questionnaire:

Task List:

1. Log into the application and tune your instrument.
2. Find the record audio page.
3. From this page record a track, in whatever key and time signature you wish.
4. Once saved, retrieve the track and play audio back.
5. Search the database, using the Friend Finder, for the user Darragh.connaughton5@mail.dcu.ie and issue him a friend request.
6. Once the friend request has been accepted, send your recently recorded track to this user.
7. Accept generic track sent from newly added friend.

Rating below are done on an ordinal scale from 1-10. 1 indicating low/poor and 10 indicating high/excellent.

1. Rate the navigation through the application:
    a. Explain your rating:

2. Does the output conform to your expectations?
3. Have many notes been omitted/added?
4. Rate the accuracy of the note lengths:
    a. Explain your rating:

5. Rate the accuracy of the Tuner:
6. Rate the difficulty of finding the specified user:
7. Rate the interface of the application:
8. Rate the difficulty faced sending a track to the specified user:
    a. Explain your rating:

9. Rate difficulty of accepting track from user:
    a. Explain your rating:

Any comments or other problems faced:

Plan Language Statement:

The musical notepad is a music documentation tool undertaken as a undergraduate final year project in DCU's school of computing. The principal investigator is Dr. Donal Fitzpatrick, donal.fitzpatrick@dcu.ie. The completion of a questionnaire required is undertaken by participants of this study. There is no risk involved in this project. There is no benefit for participants taking this study. User information will only be accessed by me, and shall be deleted on request after the questionnaire or after May 25th. This is a completely voluntary study, participants have the right to back out at any time. Participants interested developments of the study will be given the option to be put on a mailing list.

If participants have concerns about this study and wish to contact an independent person,

please contact:

The Secretary, Dublin City University Research Ethics Committee, c/o Research and Innovation Support, Dublin City University, Dublin 9. Tel 01-7008000, e-mail rec@dcu.ie