MUSIC OF THE WORLD

Group Members:

Eric Wysocki
Chris Schilling
Jamie Eldredge
Alex Sommer

Project Advisor:

Dr. Rick Leinecker

University of Central Florida
Senior Design I
Final Design Documentation
April 28th 2016
# Table of Contents

- Executive Summary ........................................................................................................... 1
- Project Description ............................................................................................................. 2
- Personal Motivations .......................................................................................................... 6
- Broader Impacts .................................................................................................................. 8
- Research and Investigations ............................................................................................... 10
- Goals of Project ................................................................................................................. 19
- Functional Requirements ................................................................................................... 20
- Non-Functional Requirements ............................................................................................ 29
- Front-End Design ............................................................................................................... 32
- Sequence Diagrams ............................................................................................................ 43
- User Interface Design ......................................................................................................... 48
- User Interface Mock Designs ............................................................................................. 51
- Database Design ................................................................................................................. 65
- Backend and Server Design ............................................................................................... 69
- Algorithm Design ............................................................................................................... 72
- Integration Design .............................................................................................................. 74
- Design Summary ............................................................................................................... 76
- Build Plan ........................................................................................................................... 78
- Prototype Plan ..................................................................................................................... 81
- Test Plan .............................................................................................................................. 82
- Evaluation Plan .................................................................................................................. 89
- Team Areas of Responsibility ............................................................................................ 90
- Equipment ......................................................................................................................... 92
- Budget and Financing ........................................................................................................ 93
- Project Summary and Conclusions .................................................................................... 93
- Milestones ......................................................................................................................... 94
- References .......................................................................................................................... 95
Executive Summary

This Senior Design Project, will be an Android application to be used as a music creation tool. The development codename for this application is Kwyjibo. In this tool, users from across the globe can collaborate or contribute to creating music using their own short user recorded sounds. At the heart of Kwyjibo will be a machine-learning algorithm. That algorithm will create a pattern to fit users’ sounds to render a stream of music within a chosen genre (dance, rock, latin, etc.).

There are plenty of applications that mix together sounds or songs, but none that we could find that specifically use machine learning algorithms to create music from scratch. Most of the mobile applications that exist either mix or mash-up one or more musical tracks much like a DJ would. There are many that are miniature digital audio workstations, giving the user ultimate control of the end product. While many people love absolute control, many do not have the time nor the talent to create music.

Kwyjibo is meant to appeal to a wide variety of audiences. It is intended to be a toy for novices who want to have fun and play with various sounds and let the algorithm create music for them. Musicians may use the application to create a quick song that they might be able to expand upon when inspiration strikes. Artists can create sonic soundscapes, combining sounds together that nobody has heard together before because it might take too much effort to record and compose.

This project will consist of one recording mode and two ways users can interact with the sounds. One way the users will upload their sound or sounds to a public station which will put it in a queue. The most recent sounds in the station will be analyzed by the algorithm and the station will be
given a pattern in which to play the sounds back to the user to create a stream of music. The other way is the same except users will be able to choose each sound that will be part of the music generation process.

**Project Description**

The project Kwyjibo will be a music creation tool that can create a stream music from seemingly random (or not-random) sounds that a user may record themselves using the application or download from a database of previously recorded sounds. The music creation will be rendered based on a pattern that is generated based on a machine learning algorithm. This algorithm will be fed songs from specific genres and deliver a pattern that best fits the genre given the cadre of sounds that it is using to create the music stream.

Our inspirations for the project came from a project pitch from UCF professor Dr. Rick Leinecker. He suggested several ideas and this project is a combination of two of them. The first was an app that plays back several live streams of music at once from around the world. The second was a music generation application that based on music theory can generate songs based on certain criteria. Our project will hopefully culminate in a marriage of the two ideas, since both had a few holes to fill.

With the first application, the sounds all playing together at the same time would be crowded muddy, and unlistenable without any structure. It would be akin to a mob of people yelling as opposed to a chorus of trained singers. However there is an interesting novelty of listening to what is happening at this very moment, especially in different location that could as diverse as continents.
On the second idea, a music notation generator either using music genres or specific songs as a template is very interesting, but it would not have much appeal outside of musicians. There is a great demand for music notation, but the majority of people don't play instruments well, and even fewer can successfully read sheet music or other types of music notation.

This generation has a tendency to want things easy and fast yet fun with a lot of options take something and make it their own. Kwyjibo will take the above ideas and innovatively implement them to appeal to broad amount of people, while keeping it simple for a musical novice, but being enough of an interesting tool that experienced musicians and curious artistic types will enjoy its features. This application takes inspiration from successful application such as Snapchat that has a simple interface, but disposable content. However, our content will be disposed only if the user hasn't saved the pattern-based stream of music.

Kwyjibo will consist primarily of three modes. The record mode will be how users create sounds to use within the application. The radio mode will be a public way that users can contribute sounds to a station or just listen to a music stream based on the latest sounds that other users have recorded. The studio mode will be a private part of the application where you can have complete control of the sounds used in the music stream.

The record mode is maybe where users have the most fun in the application. The possibilities are endless for what users can record and upload to tracks, stations, or to the database for other users to enjoy. A user in record mode will automatically go to the recording screen on opening it from the menu, and can start recording by pressing and holding a large red record button. There is also a timer on screen to let them know they are
getting close to the ten second time limit for a sound in Kwyjibo. After the record button is released or the timer reaches ten seconds, whichever comes first, they will be brought to another menu. This screen is the recording mode menu where they can playback the sounds they just recorded to see if they like it. If they do like the sound they can name it and store it locally on their mobile device or they add it directly to the queue of a radio station. If they don't like what they recorded then they can hit the ‘discard and re-record’ button where they will be brought back to the recording screen where they can try again.

The radio mode will be where users can choose a station named after a genre of music. They can then choose for their sound to go into the station queue where, when prompted an algorithm will generate a musical pattern based on the types of sounds that are currently in the queue. The application will then play back the sounds in a looping musical pattern until the user stops the audio stream. A user may create their own station with a name and a description, but they can also just join a station and listen to the stream without contributing any sounds. A station may be based on a genre of music or it could be themed, letting other users know what kind of sounds to contribute. For example, a station could be labeled power tools, animal sounds, nature sounds for those who want to create a public station, but still have an overarching theme to aim for. After all without some structure the sounds, even with the algorithm, could add up to just a bunch of noise.

The studio mode is almost identical to the radio mode except for a few key features that we wanted to give to users that would have conflicted with the spirit of the public radio mode. The studio mode is private, meaning only you and those you invite can contribute sounds to a track you create.
Other users will not be able to even see other tracks that have been created, you can only be invited. This is built more for the user who wants to take the time to record certain sounds, try them out and see if they can find better ones to put into their track. Invited members of the track will also have the ability to change what genre the algorithm will pattern the station’s sounds to. We plan to focus on EDM (electronic dance music) as the core genre, but there will be other pattern algorithms developed. The amount of the them is dependent on how much time we have left in the development process. Users will also have the option of altering the tempo of a generated track via a BPM (beats per minute) spinner. The BPM spinner will be set to a default setting of 120 BPM, with a lower setting giving a lower tempo and vice versa.

The last difference we would like to highlight is the fact that users may download previously uploaded sound from our database to use in studio mode. When a sound is recorded, users have the option to uploading it to Kwyjibo’s database with a name and description. Database sourced sounds will also be tagged with the author’s user name and upon them granting permission in the application settings, their general location. This could lead to a vibrant community of users sharing each other’s sounds in this ‘expert’ mode of the application.
Personal Motivations

Alex Sommer:

I wanted to be a part of a project that wasn’t boring. There was plenty of chance to be a part of a new filing system or slightly different filing system, or even just another filing system. But I wanted to be a part of something cool and unusual, and I wanted the project to bring that feeling to everybody. Lots of people like music, but most people are intimidated by trying to make music. I believe music should be accessible and fun for everybody, and that this project is a way to make it so.

Eric Wysocki:

This is a project that immediately sparked my creative juices being a lifelong music fan. Especially, innovative music that pushes the boundaries of what can be achieved by sound through an artistic lens. Whether it is conventional pop music or someone recording a freezer and putting the output through various guitar pedals. I am excited to use my knowledge and experience as an audio engineer along with my budding knowledge of computer science to the test. Also, in my education I've yet to have the opportunity to work on a project of this scope, either myself or in a team environment. Together our team will be able to make an attractive, interesting, useful, enjoyable project that will showcase our talents to prospective employers and hopefully launch our careers in computer science.

Chris Schilling:

Like the other members of this group, this project caught my attention immediately. Before gaining interest in software development and computer
science I spent most, if not all, of my free time furthering my musical related skills and knowledge. I've performed in a few different bands over the years, and devoted countless hours to learning music theory. Nowadays, as most of my free time goes towards my future career as a software engineer, I could not think of a better first serious project for me. Procedural music generation is a challenging and interesting idea, and I won’t have to go through as much of a problem domain learning curve because of the time I’ve spent increasing my musical knowledge. In addition, this project serves as an introduction to machine learning, something that I've been gaining an increasing passion for the more I learn about the subject. I will also be able to easily speak passionately about this project, which will be important when showing off my work to future employers.

**Jamie Eldredge:**

This was the only project proposed that captured my imagination. This is a program that I can actually see myself using, and as someone with limited musical ability I think this app has the potential to assist thousands of people with creating music. I want to be able to put this on my resume and be proud of it. I want this to be a project that will make potential employees go, “Wow, this guy knows what he’s doing!”
Broader Impacts

This app will be highly inclusive on both a local and global level. It will affect people both in and out of the stem field, anyone who likes music can use this app. It enables people from all over the world to join together and share a musical experience; it can transcend any cultural, linguistic, or geographic limitations.

One of the big questions in the art world is, “What is Art?” or in this case, music. What makes the difference between music and just sound, between creative and random? About half of the people we have told about this project have just flat out said it can’t be done. What is it about music that makes people think it can only come from special people in a special way? Our project aims to convince people that algorithmically generated art (in our case music) can indeed provide either entertainment or emotional value.

Our project should be able to answer the question of whether we can create artistically viable music through random sounds and users influencing the app in a fun and easy way. The application will also be able to connect artists and musicians across the globe to collaborate in a simple way. They will be creating art without the trouble of studio logistics, although they will be giving up considerable creative control to our project's algorithm. Perhaps a seventy year-old jazz musician, can connect with a five year-old piano prodigy, and throw in a person who just likes to stomp on soda cans. With our app users can intentionally interact with each other or be randomly grouped together into our radio stations to create an endless possibility stream of music/art while continuing to evolve the algorithm using computer science to give users more choices to explore or cater to their tastes.
While our team does not have the experience of a seasoned computer scientist or even a graduate degree level student, we could have a possible impact on machine learning research. The impacts could be in audio identification, specifically music identification. There is a lot of research and programs that are using machine learning to solve this problem. During Kwyjibo’s development process we could try some things that could influence those projects. The classification and regression steps for machine learning in music have a lot room for a creative solutions that we could try to solve.
Research and Investigations

Similar Products and Solutions

So far there has not been a mobile application that accomplishes this exact goal. There exists some automatic DJ programs, but these programs are either limited in scope or focus. Most of these products are focused on cross fading two audio sources to create a party mix, or are designed as an auto-mashup creator. The closest solution to this exact goal is a band in Iceland called múm who has done a song from sounds manually as an art project.

There are many other machine learning applications, but most are just a tool of other projects to help classify music for analyzation. None are classifying and doing something with that information, such as the music generation Kwyjibo will accomplish.

Mobile Development

For development, we chose to focus on the Android problem to cut down on problems with cross platform development. Android was chosen due to having a more favorable development environment, as we can do most of our front end and app development in Android Studio.

In addition to the standard Android SDK, we are using the MediaRecorder class to process our audio input from the device microphone. We determined that this class was the most robust, and gave the easiest inputs to manipulate for processing. Further supplemental tools come from the Android SoundPool API. Aside from audio data processing, our application is kept simple and clean with little need to use more than the standard tools on the mobile platform. More complex and processor heavy
tasks are offloaded to server-side, where more powerful equipment and tools are available.

**Database and Backend Development**

For our application backend we will be utilizing a central server and an accompanying database. The server will run the machine learning algorithm, handle stripping and normalization of sound clips, and manage the stations of radio mode. The database will hold the user data, station data, sound clips, and the different genre seeds. We will be running Microsoft SQL Server 2008 to manage our server and database on a server provided by our sponsor Dr. Leinecker.

Because most of our software was provided for us we did not have to research which versions to use, although we did have an unexpected amount of trouble installing SQL server on our machines. Not only is it not supported on Windows 10, which some of our group use, it had conflicts with SQL server 2008 R2 which comes pre-installed on Windows. To circumvent this we had to run the installation as admin and enable compatibility settings for the given machine's operating system.

**Machine Learning**

We will accomplish our task of creating computer generated music by using machine learning techniques. Machine learning, at its essence, is a system that can learn by example. Given many pictures of faces, a machine learning algorithm could learn to accurately determine if a newly presented picture is a face or not. By using data about which movies each user enjoyed, a machine learning algorithm could learn to recommend movies to other users that keep them coming back for more. Our goal is to create a system
that, when trained with songs from a particular genre, learns how to generate new rhythmic patterns that are characteristic of the training genre.

Whether or not learning is feasible is dependent upon the training set being based upon some unknown probability distribution [1]. We do not need to know the exact distribution, but it must exist. Our intuition is that songs from a particular genre share many of the same rhythmic motifs. The frequency and ordering of these recurring motifs creates a distribution which we can approximate using a training set. This is one reason for attempting the use of machine learning as the meat of our algorithm. Another reason is that machine learning is on the forefront of artificial intelligence technology [1]. Machine learning algorithms have come the closest to emulating human behavior better than any other types of algorithms in the past, and the problem we are trying to solve, musical composition, is a complex human behavior that may need the latest in artificial intelligence technology to solve well.

There are two main types of machine learning algorithms, classification and regression [1]. Classification is exactly what it sounds like; the classification of input data into one or more distinct sets. Regression is about finding a curve which approximates a set of data so that we can use the curve to estimate future input data of the same type with some degree of confidence.
The Learning Diagram

Figure ML1

Figure ML1 shows a visual description of the learning process in machine learning. It starts with an unknown target function, $f: X \rightarrow Y$, which is the classification or regression function we are trying to find. In order to find this function, we first pick a search space of functions, called the hypothesis set. The hypothesis set is generally a linear combination of polynomials where each term has an associated weight. Adjusting the weights independently from one another provides us with a large search space of
functions that hopefully encompasses a function, \( g: X \to Y \), which closely approximates the target function. The navigation of the hypothesis set to find \( g \) is done using a learning algorithm which has been trained using the training examples.

The learning algorithm, and machine learning in general, can be described as the minimization of a cost function. The cost function is a function that maps functions from the hypothesis set to an error score. Therefore, minimization of the cost function is equivalent to finding the function from the hypothesis set that has the least error. In the case of neural networks, this is done using stochastic gradient descent which, in its most basic form, uses calculus along each component of the network to update our cost function in the negative direction of the greatest rate of change.

**Audio Engineering**

On the audio side of development we must take care of the quality of the input sounds at all three phases, recording, mixing, and rendering. We are considering putting a gain level control of the recording screen, but we wanted to keep the application very simple. It is true that the input might be too loud for the device or the application to handle, but adding another control could complicate the user experience and scare away casual users who don't know what it is. We feel that the short length of the sounds and the ease of discarding and re-recording the sound would be a better user experience. In short, after they listen to it, if they don't like it, they can make a new sound. There may be some fringe artistic cases where a user may want the sound to clip or distort for a certain desired effect. For example, there are many audio effects that distort or have a dissonant output. Guitar
distortion has been around since the late 1940’s and there are even effects that alter the bitrate of the digital audio signal. Most non-mp3 audio files today are either 16-bit or 24-bit, but there are effects that can make them into or sound like a lower quality 8-bit audio file.

For the input file format we have chosen mp3. While mp3’s can be difficult to work with since they are a compressed audio file format, we will not be doing any additional processing to them. The only editing that will be done is stripping off the leading and trailing silence of each sound that is uploaded to the server. This not only makes the file size smaller for storage and transmission to the server, but it will also clean up the timing of when the audio files will be played back according to the pattern the algorithm generates.

While wav files are of a higher quality, mp3 files are a fraction of the size and since mp3 is the dominant audio file format, the public prefers the convenience of having more space as opposed to quality. A lot of that is due to the fact the lower quality systems most people are using to playback and listen to the audio masks most differences. Only audiophile hobbyists and trained professionals typically have the trained ear and listening environments necessary to tell the difference.

The mixing process will technically not take place on the client side since we are only playing back several sounds in sequence. However, after the sounds are recorded, sent to the server, stripped of leading and trailing silence we still need to make sure each audio file is of a comparable volume level. Using the Sound eXchange (SoX) utility on the server side we will use SoX’s normalize function to adjust all sound files so that on playback they can all be heard clearly. The normalization process will maximize the overall
volume of the sound. In the case where a user wants to download an audio stream of either a station in radio mode or a track in studio mode, this will need to be done on the server side as well. Our original plans were to take care of this on the client side, but upon research there seems to be no APIs, that will do this.

When capturing the pattern of audio files we also must spread the files through the stereo image. It may be beneficial, depending on the style or genre of music, to alternate or outright change where each of the files play within the stereo image at different times. For example, a sound could be balanced 100% right then 100% left for its next iteration. A rapid succession of this leads to a ping-pong effect, which is ‘ear-candy’ that mostly prevalent in electronic music, but not unheard of in other genres. It is typical to have voices or key melodies appear in center of the stereo image (50% left, 50% right), and more supportive material of a musical work is placed to the right or left as to not crowd the center where the focus needs to be.

It is also very important that the output file matches the input file. There is no use in upscaling or downscaling a lossy audio format such as mp3 [2]. Android supports recording in the mp3 format, but to render or in our case re-encode in mp3 on the server side. We will need to add an external encoder such as the open source LAME mp3 Encoder. Keeping the file at the same bit rate will also alleviate the need for applying dither while rendering the output file [3]. Dither is introducing a low volume noise to a digital audio when converting to a lower bit resolution. Dither helps reduce unpleasant distortions that could be created during the conversion.
What We Didn’t Use and Why

**Ionic:**

Ionic is an angular directive/css layer for Cordova. It allows for the concurrent development of Android, iOS, and web portal clients using a single code base. We were considering using this framework because of its cross-platform ability, but ultimately decided against it. One glaring problem with these cross-platform solutions is that they fail to feel like a fully native application to the users. There is a small 300 millisecond delay on the touch input to detect web-view scrolling vs tap input. The framework tries to get around this, but it is not perfect. Also, for complex applications, the cross-platform ability is not seamless and requires a bit of tweaking for each device’s codebase. These errors are accompanied by error messages that are of little to no use and, from experience, can be extremely frustrating to work with.

**Unity:**

Unity was recommended to us by a classmate, and initially we were excited about learning a 3D game engine. In the end, though, it was simply not the right tool for the job. Research into Unity taught us that we would have needed to build a 3D user interface in a 3D world presented from an angle that gives the illusion of a 2D interface. The entire engine would need to be loaded while the application is in use, and this would put too much strain on the device’s battery than we were willing to accept.

**Csound:**

Csound is an audio programming language similar in syntax to C. While it seems quite powerful upon further research it is more for creating sounds such as synths and other virtual instruments. Since we are recording
the sounds ourselves, we found no need for this interesting find. Once we
found SoX (Sound eXchange)'s utility, we found it a better fit for a function to
manipulate the audio editing and engineering aspects that Kwyjibo needs.

Marsyas:

Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals) is
an open source software framework for audio processing with emphasis on
Music Information Retrieval. At first glance this seemed like what we were
looking for, but this actually is a project that is the closest we have seen to
part of what Kwyjibo is trying to do. Marsyas handles more of the analysis of
a waveform for uses in applications as diverse as robotics, visualizers, and
music classification for websites such as last.fm. Marsyas is not creating
music, but we have learned of a few techniques that they employ in music
analyzation such as checking how many times and the distance a waveform
has crossed the zero boundary, going from positive to negative.
Goals of Project

The goals of Kwyjibo is primarily a learning exercise to develop our skills we have learned in Computer Science and become software developers with a quality project under our belt. Along the way, we would like to explore different methods of machine learning in regards to music theory and audio engineering. We would like to attempt a challenging project, where we come across interesting and unexpected results.

There is a lot of excitement in the field of machine learning due to the increase of human relatable systems. GPUs go hand-in-hand with the computations needed for deep learning algorithms [4]. The fact that GPU processing power is exponentially increasing gives machine learning an added boost. With that increasing power, researchers and developers can use modest machine learning algorithms on consumer level computing machines. Kwyjibo will be analyzing a large array of music, but the training the network will not be so intense, as to require such a powerful system that we may not be able to get consistent access to it.

Stretch Goals

- Develop an Apple iOS mobile client
- Implement extra music genres and features
- Develop social media integration with various websites (Facebook, Soundcloud, YouTube)
Requirements

Functional Requirements

No.1: The application will run on Android devices.

Statement: The app will be supported on android devices on API 14 through 23.

Source: The application is a mobile instead of a web portal so it requires support from a common mobile device

Dependency: None

Conflicts: None

Supporting Materials: None

Evaluation Method: App will be made using the android developer kit, and kept in compliance with the API using the developer tools. Afterwards, the app will be loaded on as many different API versions as are available for testing.

Revision History: N/A

No.2: When the application is started, title splash screen will be displayed.

Statement: When the user loads the application they are presented with a splash screen to show the app brand and have a clean slick looking package for the product

Source: To indicate that the app is loading.

Dependency: None

Conflicts: None

Supporting Materials: None
Evaluation Method: During a test run the app will be loaded and a note made on whether the expected splash screen appears, and how long of a delay there is for the splash screen to appear.

Revision History: N/A

No. 3: Users can choose to use the app in “Radio Mode” or “Studio Mode”.

Statement: The user must have a clear choice between the two primary modes of the app

Source: radio mode and studio mode have different functionalities, the user should be able to know what each mode does and how to use them.

Dependency: No 2 (Splash Screen)

Conflicts: None

Supporting Materials: None

Evaluation Method: During a test run of the app, after the splash screen the user must be taken to a menu to select between modes. A note will be made on whether this occurs.

Revision History: N/A

No. 4: The app will have a variable number of stations in “Radio Mode”.

Statement: When the user is in radio mode they have multiple “stations” available to listen and contribute to

Source: When many users are concurrently using the app there will be a need for stations to help with overflow. No one wants to wait a long time to hear their sounds played.
Dependency: menu, station functionality

Conflicts: None

Supporting Materials: None

Evaluation Method: During a test run of the app, after the menu screen displays, select Radio Mode and look for another menu screen listing radio stations that can be joined. A random selection of these stations will be joined and tested for functionality. 3 or more stations must function.

Revision History: N/A

No.5: In “Radio Mode”, users will be able to listen and/or contribute to one station at a time.

Statement: Stations must remain mutually exclusive to have a varied and interesting experience for the users

Source: You should not be listening to more than one station at a time, just like an actual radio experience. That way lies madness.

Dependency: 4, menu

Conflicts: None

Supporting Materials: None

Evaluation Method: During a test run attempt to load multiple stations by various means, including backing out of a station and loading another. At no time should two stations be playing concurrently.

Revision History: N/A
No.6: After the splash screen disappears, users will be brought to a menu to navigate the app.

Statement: As there are two modes there logically needs to be a way to navigate between them. The menu is a natural starting point for navigation.

Source:
Dependency: 2(splash screen)
Conflicts: None
Supporting Materials: None
Evaluation Method: After the splash screen on a test run, check if the next state brings the user directly to the menu.
Revision History: N/A

No.7: Users can record solo clips that can be used by them or other users to generate songs.

Statement: Users can record clips to use in composition, and upload them to be used by others from the database.

Source: Developer Requirement
Dependency: Server functionality
Conflicts: None
Supporting Materials: None
Evaluation Method: how are we going to test it
Revision History: N/A
No.8: Users can pick from recently uploaded sound clips which can be used to generate songs.

Statement: The user will have access to a database of uploaded sound clips and can use these clips in composition.

Source: Developer Requirement

Dependency: 6, Server Functionality

Conflicts: None

Supporting Materials: None

Evaluation Method: After a selection of sound clips have been uploaded to the database, test radio mode by attempting to add a sound clip from the database to the station. Listen to the output for the chosen sound clip to determine if the addition was successful.

Revision History: N/A

No.9: Recorded song clips can be up to 10 seconds long.

Statement: During recording, song clips can be a length of up to 10 seconds, and will be truncated or rejected if the length is greater than 10 seconds.

Source: Developer Requirement

Dependency: 5

Conflicts: None

Supporting Materials: None

Evaluation Method: Attempt, in record mode to record sound clips of various lengths progressing beyond the ten second limit. The recorded sound should be cut off at the limit of ten seconds.

Revision History: N/A
No.10: Users can save their privately generated songs locally to their device

**Statement:** From Studio Mode, users can save generated songs in a format playable by other music programs. (Planned: MP3)

**Source:** Developer Requirement

**Dependency:** Record mode, studio mode

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** After testing studio mode generation, attempt to save the generated song locally to the HD. If the song can be loaded in an external music player, the requirement is met.

**Revision History:** N/A

No.11: Users can name the saved songs.

**Statement:** Saving files will allow an opportunity to name the saved files. (Note: choosing save location can occur here as well)

**Source:** Developer Requirement

**Dependency:** 9 (Save Songs)

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** During the test for 9, attempt to name the saved song. In the file manager, search for the test name and check if it corresponds to the saved song file.

**Revision History:** N/A
No.12: Users can save songs up to 3 minutes long.

Statement: For saved files from studio mode, the save files should support a minimum of three minutes of audio content.

Source: Developer Requirement

Dependency: 9

Conflicts: None

Supporting Materials: None

Evaluation Method: Attempt to save songs of various lengths both shorter and longer than 3 minutes. All cases in the designated range should save, and the ones that are too long should prompt the user to truncate or attempt a different input.

Revision History: N/A

No.13: Generated songs can take up to 10 input sounds as parameters.

Statement: Song Algorithm must cap the number of used sounds in a single composition at 10.

Source: Developer Requirement

Dependency: 7 (generate from clips)

Conflicts: None

Supporting Materials: None

Evaluation Method: During studio test, attempt to add 11 sounds to a song. In radio mode do the same. In studio, the last song should be rejected, in radio mode the sound should be enqueued to be rotated in later.

Revision History: N/A
No.14: Users can playback their created song while composition in progress.

**Statement:** During Studio Mode composition, the in-progress song should play with the parameters given, continuously and in real time. As new sounds are added or old sounds removed, the composition will adapt to incorporate the changes.

**Source:** Developer Requirement

**Dependency:** 7 (generate from clips)

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** During the test of studio mode, take a note of whether the in-progress composition is playing and incorporating new additions.

**Revision History:** N/A

No.15: If recording input is above a 60 decibel threshold then recording will be stopped.

**Statement:** If during recording, the volume exceeds 60 db, that clip will be rejected.

**Source:** Developer Requirement

**Dependency:** 5

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** During recording, attempt to record a sound known to be above 60 db, and check that the sound is rejected from recording.
No.16: Recorded Sound input will not be able to be used if there is audible digital clipping.

**Statement:** During a recording, a sound byte that has digital clipping above a given threshold must be rejected from recording.

**Source:** Developer Requirement

**Dependency:** 5

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** deliberately induce clipping during a recording session, and check that the clip is rejected by the app.

**Revision History:** N/A.

No.17: UI will accommodate device orientation.

**Statement:** When the android device detects a change in orientation, the app will change its screen orientation to match, between portrait and landscape format.

**Source:** Android standards and practices

**Dependency:** None

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** During a test run of the app, at various, randomly determined points, change the device orientation and record if the screen orientation changes to match. Then check that when orientation is disabled through the android menu, that the screen does not re-orient.

**Revision History:** N/A
No.18: Users will be prompted with a short tutorial the first time they start the app.

**Statement:** Upon the first run of the program after install, the user will receive a tutorial prompt before they enter into their first menu option.

**Source:** Developer Requirement

**Dependency:** 2

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** During the first test run on a new device, check that a tutorial prompt appears before the user gets past the main menu selections. Then do the same test for a device that has had the app uninstalled and reinstalled after one or more uses.

**Revision History:** N/A

---

**Non-Functional Requirements**

No.1: UI is easy to navigate

**Statement:** UI is easy to navigate, should not be too complex as to scare away musical novices

**Source:** Per the stated Intent of the project, this app is supposed to be all-inclusive, and that necessitated a user interface that is widely accessible to as many people as possible.

**Dependency:** None

**Conflicts:** None

**Supporting Materials:** None
**Evaluation Method:** Testing can be accomplished through allowing a wide variety of users to demo the product, then getting feedback on the ease of use of the interface.

**Revision History:** N/A

**No.2: Documentation**

**Statement:** maintain documentation throughout development process. The documentation should be thorough enough that an entirely new team could pick up and complete the project.

**Source:** Developer Requirement. Also mandated by Senior Project requirements

**Dependency:** None

**Conflicts:** None

**Supporting Materials:** The Senior Project development Requirements

**Evaluation Method:** None. Documentation is either there or it is not.

**Revision History:** N/A

**No.3: The app should be available to users 95% of the time after pushed to production.**

**Statement:** The app should be available 95% of the time after pushed to production. This would include server downtime and downtime due to crashes of the mobile app.

**Source:** Developer Requirement

**Dependency:** None

**Conflicts:** None

**Supporting Materials:** None
**Evaluation Method:** None

**No.4: Source Code should be easy to maintain**

**Statement:** To prevent trouble both in production, and for any who will continue this project. The code should be written as to be easily maintained. This is to include making the code self explanatory where possible, and including copious commenting during the writing of the software.

**Source:** Developer Requirement

**Dependency:** Nonfunctional requirement 2: Documentation

**Conflicts:** None

**Supporting Materials:** None. This is a subjective Requirement.

**Evaluation Method:** None. This is a subjective requirement.

**Revision History:** N/A

**No.5: First Time Users should want to continue to use the app**

**Statement:** As we wish this app to be used, we want users to feel happy with the product. This is best expressed by wanting to use the product again.

**Source:** Developer Requirement

**Dependency:** None

**Conflicts:** None

**Supporting Materials:** None

**Evaluation Method:** Testing can be accomplished through allowing a wide variety of users to demo the product, then getting feedback on their opinion of the product.

**Revision History:** N/A
Frontend Design

Main Activity

The Main Activity (Figure UML1) is the first screen the user will see when the application is first started. The MainActivity class is responsible for providing functionality to the buttons that allow the user to enter the different modes of the application.

Figure UML1
Record Activity

The RecordActivity, RecordTab, and ReviewRecordingTab classes are responsible for providing an interface to the user for recording and reviewing sound clips. The RecordActivity (Figure UML2) class contains a ViewPager which provides a swipeable view android Fragments. The RecordTab and ReviewRecordingTab both extend android.app.Fragment.

![Figure UML2](image)
Radio Activity

The RadioActivity (Figure UML3) class provides the user with an interface to view and select radio stations that users will be able to listen and contribute to. This class depends upon StationPopulatorAsyncTask which is responsible for executing an asynchronous network call to our server to fetch the station list information. StationPopulatorAsyncTask extends android.os.AsyncTask.
Studio Activity

This class is responsible for initializing the Studio Activity (Figure UML4). The Studio Activity provides the user with an interface to build their own radio station complete with sounds that have been recently uploaded to the database. It depends upon RecentlyUploadedSoundsAsyncTask which is responsible for executing an asynchronous network call to our server to fetch the recently uploaded sounds. RecentlyUploadedSoundsAsyncTask extends android.os.AsyncTask. The SoundClipInfoAdapter class is needed to customize the ListView's list item layouts to show all the information stored in the SoundClipInfo objects.
SQLite Database Manager

SQLiteDatabaseManager (Figure UML5) is the class responsible for persistent local storage on the device using SQLite extending SQLiteOpenHelper. This local database will be used so that network calls do not have to be made every time the user uses the app, only when there are updates. Also, it will help provide a smoother experience providing existing content while network calls take their time pulling data from the server.

Figure UML5

```
<table>
<thead>
<tr>
<th>SQLiteOpenHelper</th>
</tr>
</thead>
<tbody>
<tr>
<td>- String DATABASE_NAME</td>
</tr>
<tr>
<td>- String TABLE_SOUNDCLIPS</td>
</tr>
<tr>
<td>- String COLUMN_ID</td>
</tr>
<tr>
<td>- String COLUMN_SOUNDCLIPNAME</td>
</tr>
<tr>
<td>- String COLUMN_CONTRIBUTOR</td>
</tr>
<tr>
<td>- String COLUMN_LOCATION</td>
</tr>
</tbody>
</table>

+ DatabaseManager(Context, String, SQLiteDatabase.CursorFactory, int)
+ onCreate(SQLiteDatabase)
+ onUpgrade(SQLiteDatabase, int, int)
+ addSoundClip(SoundClipInfo)
+ deleteSoundClip(String)
+ getStoredSoundClips() |
```
Station Activity

The StationActivity class (Figure UML6) is responsible for providing the user with an interface to the station they selected from the Radio Activity. The users will be able to listen to, add sounds to, and save the stations song feed. The StationSoundsAsyncTask is responsible for executing the asynchronous network call to the server that populates the ListView with the list of sounds currently being used for song generation. The SoundClipInfoAdapter class is needed to customize the ListView's list item layouts to show all the information stored in the SoundClipInfo objects.

Figure UML6
Create Station Activity

The CreateStationActivity class (Figure UML7) is responsible for providing the user with an interface to create new radio stations. It depends upon the CreateStationAsyncTask class which handles executing an asynchronous network call to the server which will insert the new station’s details into the database.

![Figure UML7](image-url)
**Settings Activity**

The SettingsActivity class (Figure UML8) is responsible for providing an interface to the user for viewing and changing application settings. Using a PreferenceFragment inside of SettingsActivity is a common and best practice when it comes to storing user preferences. The settings preferences are persistently stored in the SharedPreferences object which is an object that is shared between applications on the android device that stores key-value pairs.

![Figure UML8](image-url)
Activities

The majority of the activities in our application will extend AppCompatActivity. AppCompatActivity (Figure UML9) is the latest Activity class Google has introduced in the Android SDK that is generic enough to support our minimum api level, API level 14.

Asynchronous Tasks

The asynchronous network calls used by our application extend android.os.AsyncTask (Figure UML10).
Record Mode Pager Adapter

In order to allow the user to swipe between views, a concrete implementation of either FragmentPagerAdapter or FragmentStatePagerAdapter is required. FragmentPagerAdapter is designed for a low number of static fragments. FragmentStatePagerAdapter is for a large variable number of fragments, working more like a list view. We only want to add two fragments to our pager adapter and do not expect that to change, so we went with the FragmentPagerAdapter. Both pager adapter types are from the android.support.v4 library.

![Figure UML11](image)

Song Generation Services

The song generation service (Figure UML12) extends android.OS.Service

![Figure UML12](image)
**LoginActivity**

The LoginActivity class (Figure UML13) is responsible for providing the user with an interface to login or create an account. It depends on LoginAsyncTask and AddUserAsyncTask which execute the asynchronous network calls to the server which will handle login authentication and updating the database. LoginAsyncTask and AddUserAsyncTask extends android.os.AsyncTask.

Figure UML13
Sequence Diagrams

These sequence diagrams show the flow of control for the methods within the android client that interact with the external server and database. They show not only the different classes and methods that are called, but the temporal ordering in which those elements are called.

Station Populator AsyncTask

Figure SQ1
Get Rhythm Pattern AsyncTask

Figure SQ6

Song Generation Service

Figure SQ7
Login AsyncTask

Add User AsyncTask
User Interface Design

For Kwyjibo, we want a simple user interface as to not provide too much of a distraction from the experience. The user interface design will start with a brief launch or splash screen with our application's logo and brand colors. While in development if we find that our application has an excessive load time then we will add a loading progress bar to the splash screen to alleviate users’ concerns on the app not freezing and loading correctly. The login screen (Figure U1) will come up on screen if it is the first time the user has opened the application or if the user logged of it. A dialog menu will pop up if the user has entered the wrong password and again prompt the user to enter the correct one.

The application screen will switch between portrait and landscape dependent on how the user is holding their device. In the application we have chosen to have a consistent look with each screen to have a label at the top below the action bar to remind the user what mode (record, radio, or studio) the application is in. The color scheme will have an appropriate contrast between background color and text.

Between active and inactive screens there will be a quick transaction using a cross-fade animation, since it is recommended that there should not any lateral movement between views [5]. The exception will be using a swipe gesture in the record mode for the user to swiftly be able to switch between recording a sound and previewing it. The top action bar will have have a menu accessible at the “three dots”. That drop down menu will contain access to application and user account settings, a help screen, and an “about” section (Figure U12).
The most unique interface feature of the user interface will be the recording screen which will come up when the user chooses to record their own sound to add to a station or track (Figure U4). The recording will start when the user presses the big red on-screen button and will stop recording when they release the button. There will also be a countdown on the screen to let the user know when they have reached the maximum of ten seconds allowed on a sound within the application. The decision was also made not to include a gain or volume meter on the recording screen. This keeps the interface clean and will not worry the user about the volume level. The feature of playing back and keeping or discarding the sound will be a suitable substitute. Once the recording is finished the app will automatically go to record menu screen (Figure U3). On this screen, the user may choose what to do with the sound they just recorded.

The on-screen raised buttons within the various screens of the app will have the recommended minimum width of 88dp and a height of 36dp [6]. The button click area will be the same with an additional 8dp of padding around the perimeter of each button, this ensures more usability for those with disabilities [7]. On screen dialog menus will pop up when a user wants to delete or remove to ensure there are no unintended accidents with their recorded sounds.

Within the track and session menus they will display the number of sounds used in the music generation currently happening (Figures U7, U11). Users will also be able to scroll and view the sounds within each track or session. Each sound will also list the username of its author and the location. Users will be able to delete unwanted sounds they’ve recorded for their
station or local storage by pressing and holding onto screen at the instance of the sound at a given list.

Each sound on all of the list based screens will also contain an option to report a sound for containing offensive material. While censorship and artistic creation don’t go hand in hand all the time, we wanted a way for the users to report those who might abuse the system. The UI will also implement a system that will prevent users from using offensive words in the saved names of sounds, tracks, stations, or anywhere else a user might be able to sneak it in throughout their use of the application.
User Interface Mock Designs

Login Menu

Figure U1
Home Menu

Figure U2
Record Mode Menu

Figure U3
Recording Screen

Figure U4
Create Radio Station Screen

Figure U5
In Station Menu

Figure U7
Save Mode Screen

Figure U8
Studio Mode Menu

Figure U9
Track Session Select

Figure U10

<table>
<thead>
<tr>
<th>TRACK SESSION LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sick Jams</td>
</tr>
<tr>
<td>Mega Super Track</td>
</tr>
<tr>
<td>Witty Track Name</td>
</tr>
<tr>
<td>Nature Soundscape</td>
</tr>
<tr>
<td>Ambient Chillax Track</td>
</tr>
<tr>
<td>...Find More Stations</td>
</tr>
</tbody>
</table>

Search Track Name
In Track Screen

Figure U11

<table>
<thead>
<tr>
<th>Currently 10 Sounds Playing In Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackhammer</td>
</tr>
<tr>
<td>Snare Hit</td>
</tr>
<tr>
<td>Cat Hiss</td>
</tr>
<tr>
<td>Female Jazz Singing</td>
</tr>
<tr>
<td>5 Gallon Drum Hit</td>
</tr>
<tr>
<td>C# Arpeggiated Guitar Chord</td>
</tr>
<tr>
<td>Kazoo Fanfare</td>
</tr>
<tr>
<td>Human Cannonball Splash</td>
</tr>
<tr>
<td>Cricket Chirp</td>
</tr>
<tr>
<td>Car Peeling Out</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Genre</th>
<th>BPM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find Sounds
Create Sounds
Save Output
Drop Down Menu

Figure U12
Delete Confirmation Screen

Figure U13
Report Content Screen

Figure U14
Database Design

SQL Tables with Descriptions

Our database will have five tables that contain the current stations playing in the app, all users registered with the app, and all sound clips currently stored on the server.

Stations

stationID(*) - int
name – varchar(50)
createdBy - int
genre - int
dateCreated - date
numListeners - int

The station table stores all the stations that at least one user is currently listening to, stations are deleted once the last user stops listening. StationID is the primary key, it is an integer that is unique to that station but will not be visible to users. Name is a string that is visible to users to help identify and differentiate stations. createdBy is an integer that represents the userID of the user that created it. dateCreated stores when the station was created so users can see how long an individual station has existed. Genre is an integer which indicates the set of inputs seeding the machine learning algorithm.
Sound Clips

soundID(*) - int
soundName – varchar (50)
userContributor - int
location – varchar (50)
uploadDate - date
Category - varchar (50)

The sound clips table stores all sounds that are currently on the server, which expire after a set period of time. SoundID is the primary key, it is an integer that is unique to that sound but will not be visible to users. SoundName is a string that is visible to users to help identify and differentiate sounds when they are searching for sounds to add to their station or song if they are in radio or studio mode respectively. UserContributor is an integer that indicates the userID of the user who uploaded the sound. Location indicates which city the sound was recorded in. UploadDate indicates when the sound was uploaded. Category is a string that identifies whether the sound clip is percussive, drone, ambient, melodic or other.

Genre

genreID(*) - int

The genre table contains a primary key integer that indicates the style of the station, it will determine how the machine learning algorithm is seeded.
**Users**

- **userID(*)** - int
- **Name** – varchar (50)
- **password** – varchar (50)
- **Location** – varchar (50)

The user table stores all users registered with the app. UserID is the primary key, it is an integer that is unique to that user but will not be visible to users. Name is a string that is visible to users to help identify and differentiate users. Password is a string that contains the user’s password for verification purposes, this will not to be visible to other users. Location is a string that indicates what city the user from, it is visible to other users to help foster a sense of global community.

**Sounds in station**

- **stationID** - int
- **soundID** - int

The sounds in station table is a go between to implement a many to many relation between sounds and table. StationID is an integer that identifies the station. SoundID is an integer that identifies the sound.
ER Diagram

The Entity Relationship diagram (Figure E1) for our database contains four entities: Sound Clips, Station, User, and Genre. The attributes of sound clips are ID (the primary key), name, location, category, and upload date. Sound clips has a many-to-many relationship with stations and a many-to-one relationship with user. Station’s attributes are ID (the primary key), name, and date created. Station has a many-to-many relationship with sound clips, a many-to-one relationship with user, and a many-to-one relationship with genre. Genre has a single attribute which is its primary key ID, it has a one-to-many relationship with station. User’s attributes are ID (primary key), name, location, and password. User has a one-to-many relationship with sound clips and a one-to-many relationship with station.

Figure E1
Sound Clip Prep

The SoundClipPrep class (Figure UML12) is responsible for taking sounds uploaded by users and transforming them into a uniform format. The stripSound method removes the empty sound from the beginning and end of the clip. The normalizeSound function normalizes the sound to an appropriate volume.

![SoundClipPrep Diagram](image)

Figure UML12
**Music Generator**

The MusicGenerator (Figure UML13) class is responsible for managing the machine learning music creation algorithm. It contains the station it is running on and a queue containing all the sounds that users have uploaded to the station. The getNextSounds method fetches sounds from the queues or database to be used in the song being generated. The createPattern method generates the musical pattern for the song, it sequences how often each sound clip is played. The outputPackage method assembles the previously created pattern and its corresponding sound clips into one neat package and outputs it so the clients listening to the station.

<table>
<thead>
<tr>
<th>MusicGenerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Station station</td>
</tr>
<tr>
<td>+SoundClipQueue queue</td>
</tr>
<tr>
<td>+getNextSounds()</td>
</tr>
<tr>
<td>+createPattern()</td>
</tr>
<tr>
<td>+outputPackage()</td>
</tr>
</tbody>
</table>

Figure UML13
**DataAccess**

The `DataAccess` class is responsible for fetching data from the database and bringing it to the server for use. It contains a list of stations and a list of sounds to pull data to. The `getStationList` method retrieves the specified stations. The `getStationSounds` method retrieves the sounds. The `addStation` method creates a new station. The `getRecentSounds` method retrieves the sounds that were last played. The `addSoundClipToStation` method inserts a sound into a station. The `getRhythmPattern` method retrieves the pattern that was generated by the machine learning algorithm. The `getSongInfo` method retrieves the information of the song. The `getSoundClips` method retrieves the specified sound clips. The `getUserInfo` method retrieves user data info. The `insert userInfo` method adds new user info to the database.

<table>
<thead>
<tr>
<th>DataAccess</th>
</tr>
</thead>
<tbody>
<tr>
<td>+StationList stations</td>
</tr>
<tr>
<td>+SoundList sounds</td>
</tr>
<tr>
<td>+getStationList()</td>
</tr>
<tr>
<td>+getStationSounds()</td>
</tr>
<tr>
<td>+addStation()</td>
</tr>
<tr>
<td>+getRecentSounds()</td>
</tr>
<tr>
<td>+addSoundClipToStation()</td>
</tr>
<tr>
<td>+getRhythmPattern()</td>
</tr>
<tr>
<td>+getSongInfo()</td>
</tr>
<tr>
<td>+getSoundClips()</td>
</tr>
<tr>
<td>+getUserInfo()</td>
</tr>
<tr>
<td>+InsertUser()</td>
</tr>
</tbody>
</table>

Figure UML14
Algorithm Design

The machine learning algorithm will be the core of our music generation algorithm. We have a number of ideas on the design of the algorithm, but the design and implementation will be flexible as this is a learning process for all of us in machine learning. The main algorithm, as of right now, is divided into two parts: Rhythmic pattern generation, and sound clip classification. I will now briefly describe our initial design thoughts and motivations behind some of the choices we have made.

The pattern generation algorithm will be a deep neural network which outputs a random pattern which is representative of a particular genre. More specifically, this pattern will encode common rhythmic motifs of the genre used to train the system. We chose rhythm as the baseline for song generation as it is something that will need to exist within every song generated by our system. Things like melody and harmony do not necessarily need to exist at all within a song. This backbone will allow us to build a song by combining sound clips the way the pattern describes.

The first order of business, then, is how we choose to encode a rhythmic pattern within our system. Our initial idea of a rhythmic pattern is an array of integers which represent a length of time. This encoding is one flexibility we will play with during design and optimization. One idea is for each integer to represent the length of time from the start of the song, to when each sound clip should start. A second idea is for each number to represent a length of time from the beginning of a sound clip to the beginning of the next sound clip. Both of these ideas have their pro's and cons. We believe the former may be the better solution because it will be easier to visualize and adjust the pattern for specific beats per minute using
scalar multiplication across each of the values. It also makes it easier to programmatically check where a single instance of a sound clip exists temporally within the full pattern by using simple division. At this point in time it is unclear how this information may be of use to the building of our system, but it may help in finding a solution to an unknown problem in the future. Also, with stretch goals in mind, it may be useful later on down the road as well.

The training set for the rhythmic pattern generation algorithm will be arrays of doubles (representing sound wave frequencies) extracted from songs files where all the songs exist within a particular musical genre. The exact number of training examples will be a function of the number of features of our network, which hasn’t been fully designed yet.

The second part of the algorithm may or may not be included in the final product depending on the effect it has on song generation listenability. This will be the classification of sound clips to different timbres. Although users will have the ability to tag their sound clip into a specific timbre, we think it may be worthwhile to play with programmatic classification. These classifications will be used to help construct the song from the rhythmic pattern. For example, a sound classified as “percussion” may be placed at each quarter beat that exists within the generated rhythmic pattern, while a sound classified as “melody” would be much more sparse and selective in its occurrence.

These two algorithms together define the heart our song generation algorithm. At this point in time we do not have enough information to properly choose a cost function to minimize for the rhythmic regression
problem. For the multi-class classification problem, we may use a neural network, or a support vector machine modified for multi-class classification.

Integration Design

Our project will run a client-server model. End users will use their android device clients to upload sound bytes to the server. The server modifies and stores sound bytes in the database to be retrieved as needed. The server computes the music structure, grabs sound bytes from the database to fill in the structure, and then sends the package to the clients. When the client receives the package it constructs the song and then plays it for the user (Figure C2).

Because we've chosen a client-server architecture for our system, seamless integration between components is important. We plan to support many concurrent users therefore it is important to keep communication between client and server to a minimum. We will not be streaming audio from the server to all of our connected users. Instead, we will simply send the generated rhythmic patterns along with the necessary sound clips to the client for construction on the client side. While this may demand more battery power and complicate the client code a bit, the scalability of our system skyrockets to allow many more concurrent users.
## Rest API

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get list of stations</td>
<td>GET</td>
<td>motw.tech/stations</td>
</tr>
<tr>
<td>Get list of all sound clips</td>
<td>GET</td>
<td>motw.tech/sound_clips</td>
</tr>
<tr>
<td>Get specific sound clip(s)</td>
<td>GET</td>
<td>motw.tech/clip</td>
</tr>
<tr>
<td>Add sound clip to station</td>
<td>POST</td>
<td>motw.tech/add_clip</td>
</tr>
<tr>
<td>Get rhythmic pattern</td>
<td>GET</td>
<td>motw.tech/pattern</td>
</tr>
<tr>
<td>Add user</td>
<td>POST</td>
<td>motw.tech/add_user</td>
</tr>
<tr>
<td>Get list of users</td>
<td>GET</td>
<td>motw.tech/users</td>
</tr>
<tr>
<td>Login Authentication</td>
<td>GET</td>
<td>motw.tech/login</td>
</tr>
<tr>
<td>Get song info</td>
<td>GET</td>
<td>motw.tech/song_info</td>
</tr>
<tr>
<td>Add new station</td>
<td>POST</td>
<td>Motw/tech/new_station</td>
</tr>
</tbody>
</table>

Figure R1
Design Summary

When the user launches Kwijybo they have three options from the main menu which are record mode, radio mode, and studio mode. If the user selects record mode they are given the option to choose a previously stored file from their device or to make a live recording of their environment. If the user is satisfied with the recording they can save it to their device hard drive or upload to the server database. If they upload the sound file then they can add descriptive tags to it.

If the user selects radio mode they are given the option to create a new station or join a preexisting one. While listening to a radio station a user can opt to add a new sound to the station. If the user wants to record a new sound they are taken to record mode and returned to the station when they are done. Otherwise the user will be sent to the database to choose a sound that was already uploaded. Users can listen to a station for as long as they want with no requirement to contribute, when all users have left a channel it is closed. Sounds are kept available on the database for a preset period of time before being removed so that other sounds have a chance to be noticed and used.

If the user selects studio mode they can create a new track or load one that was previously worked on. When the user opts to add a new sound they are directed to record mode if they want to record a new sound or to the database if they want to use an existing sound. Users can manually set the structure they want for their song and add sounds into their respective spots. After the user their desired arrangement they can generate the song and listen to it, when the user is satisfied with the song then they can save it to their device hard drive.
Use-Case Diagram

Figure C1
Build Plan

Implementation will begin with concurrent development of the android client and the server/database. Initial development of the client will be primarily focused on functionality rather than a responsive and professional looking UI. The first main objective is to develop a functioning client that can talk with the database. That way, we can then split up the work into the responsive UI/UX and the machine learning algorithm implementation. Supporting variable screen sizes will be taken into account while updating the layout files to provide a more professional and responsive UI and supporting Android versions 14 through 23 will require tweaking of the code during testing. The UML design currently makes use of the support library whenever possible, but there will definitely be unforeseen issues that arise due to the fact that none of our project developers are experts in android development.

The client functionality will be fairly straightforward to develop as we have a detailed designed ironed out in UML diagrams. The song generation algorithm development will, conversely, take much longer. This is the primary reason we want to get a, functioning system up and running as quick as possible. We need the android client as an environment to test the song generation algorithm because of possible temporal issues that may arise from things like networking load times or device processor speed.

We also will be developing the machine learning algorithm separate from the rest of the system. This will help our team to stay focused on their part of the build plan. Also while integrating the final algorithm with the rest of the system we will be able to eliminate either piece (algorithm or the rest) from being the problem. So when ruling out the different elements of the
system, we can then concentrate on debugging functions of either the algorithm, frontend or backend that are not cooperating with each other.

Another advantage is that if during the development process one path becomes a bottleneck and holds up the rest of development, the rest of the team can come together to get the project back on track.

**Client-Server Model**

![Client-Server Model Diagram](image)

Figure C2
Build Plan Diagram

UI / Frontend Prototyping

Database / Backend Prototyping

Machine Learning Research

Integrate Frontend / Backend Prototypes

Machine Learning API Testing

Machine Learning API choice

Implement Frontend Advanced Features

Build Backend (SoX, SQL) / Server

Algorithm Construction

Algorithm Testing / Feeding

Algorithm Integration with App

Testing

Deployment

Figure B1
Prototype Plan

The prototype for Kwyjibo will consist of a bare-bones skeleton of the application. It will have enough of the needed functionality to be able to move forward with the more complex stages of development, namely the machine learning algorithm. It will have the functionality of adding users and sounds to our database. Once the database is established, and our backend has installed the tools necessary to work with audio files we will start with a simple mixing of several sounds together using function within the SoX utility. Once the sounds have been merged into one audio file, we will test the client getting that merged audio file from the server and playing it back with the Android client prototype. As we get the basic functionality, up and working we will slowly expand the functions on the frontend and backend till we leave the prototype phase.

Prototype Use-Case Diagram
Test Plan

The objective of testing the software is to provide a reliable and enjoyable experience for users of the MOTW app. Tests will be conducted on all the requirements of the system, as well as subjectively evaluated for enjoyability overall.

Some of the test will be:

- Server accessible from android devices
- Database resiliency
- Algorithm functionality

Reference Documents:

- Use Case Diagram
- Class Diagram
- Requirements Document
- UI mockups

Test Environment:

Android developer kit, various android devices

Stopping Criteria:

Testing will be stopped when we are satisfied that the system of app and server is fully functional. This shall be defined as the functional requirements being met, and the three modes of the app functioning without error.

Test Procedure:

The modes of the app will be tested by a depth first traverse of the user menu, testing each action that is possible by a user. Some cases will require that additional data is added to the database.
Record Mode

1. Record a sound

   a. Objective
      i. Demonstrate the basic recording functionality
   b. Description
      i. From Record mode, use the record function to record a test case sound, preferably one that is known to be at optimal level for recording.
   c. Conditions
      i. This is defined as being less than 10 seconds and at approximately the volume of a person speaking in a clear voice.
   d. Expected Results
      i. Sound is successfully recorded and uploaded to database

2. Record a sound deliberately too long

   a. Objective
      i. Demonstrate the recording will not record clips of greater than 10 seconds in length.
   b. Description
      i. From Record mode, use the record function to record a test case sound that is longer than the set maximum length
   c. Conditions
      i. This is defined as being greater than 10 seconds and at approximately the volume of a person speaking in a clear voice.
   d. Expected Results
      i. App will prompt user to truncate the recording down to a maximum of 10 seconds in duration.

3. Try truncating the sound to fit

   a. Objective
      i. Demonstrate ability to truncate sounds to fit the desired length
   b. Description
      i. After passing test 2(record a sound that is too long) now use the displayed to select a ten second clip of the recording to be used.
   c. Conditions
i. Must have passed test 2, and then proceed directly to test 3.

d. Expected Results
   i. A menu option will allow the user to select up to 10 seconds of their recording to be uploaded. This 10 seconds will be continuous and can come from any section of the recording.

4. Record a sound that is too loud

   a. Objective
      i. Demonstrate the ability to filter out recordings that exceed the volume cap.

   b. Description
      i. From Record mode, use the record function to record a test case sound that is louder than the set volume cap.

   c. Conditions
      i. This is defined as being louder than 60 decibels.

   d. Expected Results
      i. The recording should be rejected from being uploaded and the user prompted to try recording at a lower volume.

5. Record a sound that has clipping

   a. Objective
      i. Demonstrate the ability to filter out recordings that have audible clipping.

   b. Description
      i. From Record mode, use the record function to record a test case sound that has clipping.

   c. Conditions
      i. Clipping is defined as exceeding the volume threshold of a microphone to have an audible break in the recording.

   d. Expected Results
      i. The recording should be rejected from being uploaded and the user prompted to try recording at a lower volume.

6. Repeat previous for pre-recorded clip

   a. Objective
      i. Demonstrate the ability to perform all the recording functions with a prerecorded clip being uploaded.

   b. Description
i. From Record mode, use the function of uploading from the hard drive.

c. Conditions
   i. Sound clips should be in mp3 or wav format.

d. Expected Results
   i. The recording should be tested to be compliant with upload conditions then uploaded the same as if it were being recorded directly.

Studio Mode
7. Start a new song

   a. Objective
      i. Show that new compositions can be created in studio mode.

   b. Description
      i. Enter studio mode and create a new composition.

   c. Conditions
      i. The new composition is a menu to begin accessing the algorithm package. The algorithm must be seeded ahead of time.

   d. Expected Results
      i. The user gets a blank composition with the option to add sounds.

8. Add a sound from database

   a. Objective
      i. Demonstrate the ability to add sounds to the composition.

   b. Description
      i. From studio mode, add a sound to the composition using the “add sound” menu option.

   c. Conditions
      i. Pass test 7 and have useable sounds in the database.

   d. Expected Results
      i. The program adds and incorporates the new sound into the composition.

9. Go to record from studio mode

   a. Objective
      i. Demonstrate the ability to transition from studio mode to record mode, and back again.
b. Description
   i. From Studio mode, use select record a new sound “to be taken
to record mode. Then select “back to studio” to return to the
studio composition.

c. Conditions
   i. Test 7 is passed. And a new composition is active.

d. Expected Results
   i. The user is sent to record mode, and when finished, returns to
the point in studio mode that they left.

10. Add 10+ sounds, testing the limit

   a. Objective
      i. Demonstrate the ability to limit the addition of sounds over the
capacity of the algorithm.

   b. Description
      i. From Studio mode, add 10 sounds to a blank composition,
then add an 11th sound and see if the system rejects it.

   c. Conditions
      i. The limit for sounds in a single composition is set at 10. This
test is to be performed on a composition in Studio Mode.

   d. Expected Results
      i. The recording should be rejected from being uploaded and the
user prompted to try recording at a lower volume.

11. Remove a Sound

   a. Objective
      i. Demonstrate the ability to remove a sound from a composition
in studio mode.

   b. Description
      i. From Studio mode, with an active composition as specified,
select remove sound and select a victim sound to be removed.

   c. Conditions
      i. Test must be conducted on a composition with at least 2
sounds added, preferably 4 sounds and with test 10 having
been passed.

   d. Expected Results
i. Sound is successfully removed from the composition, and the composition continues with the remaining sounds.

12. Save a Song
   a. Objective
      i. Demonstrate the basic ability to save a song once completed.
   b. Description
      i. From studio mode once a composition has been finished, save the song to the Hard Disk of the test device. Then The user can name the song, and choose a location to save to.
   c. Conditions
      i. For the purposes of this test, finished is defined as having 5 or more sound clips incorporated. The test of this function has further sub requirements.
   d. Expected Results
      i. 3 minutes of generated song are saved to the specified location with the specified name.

Radio Mode

13. Join a station
   a. Objective
      i. Show that stations exist and can be joined.
   b. Description
      i. From Radio mode, Select the Station to test and select the join function.
   c. Conditions
      i. Stations are intended to be preset at launch with the algorithm trained and waiting.
   d. Expected Results
      i. The user is taken to the selected station, where the radio mode functions are made available to them.

14. Add a Sound to the station
   a. Objective
      i. Demonstrate the ability to add sounds to the station.
   b. Description
      i. From the Radio Station, add a sound to the composition using the “add sound” menu option.
   c. Conditions
d. Expected Results
   i. The program adds and incorporates the new sound into the composition.

15. Play Music
   a. Objective
      i. Show the listener can receive music from the station server.
   b. Description
      i. From Record mode, use the record function to record a test case sound, preferably one that is known to be at optimal level for recording.
   c. Conditions
      i. This is defined as being less than 10 seconds and at approximately the volume of a person speaking in a clear voice.
   d. Expected Results
      i. Sound is successfully recorded and uploaded to database

16. Go to Record from Radio
   a. Objective
      i. Demonstrate the ability to transition from studio mode to record mode, and back again.
   b. Description
      i. From Radio mode, select “Record a new Sound” to be taken to record mode. Then select “back to Station” to return to the studio composition.
   c. Conditions
      i. Test 15 and Test 14 are passed. Record mode also must be functional.
   d. Expected Results
      i. The user is sent to record mode, and when finished, returns to the Station that they left.

17. While music is playing go back to menu and join other station
   a. Objective
      i. Show ability to switch stations.
   b. Description
i. From A radio station, with music playing, back out to the menu and enter a new station.

c. Conditions
   i. Working stations with a composition playing are required for this test.

d. Expected Results
   i. The user exits the station, and the music stops playing, and new music plays when entering the new station.

18. Add 10+ sounds to station and wait for the enqueueing to function

a. Objective
   i. Demonstrate the ability to add sounds to the composition.

b. Description
   i. From Radio mode, continue to add sounds to the station until 10 sounds are playing. Then add an additional sound and wait.

c. Conditions
   i. Working stations with a composition playing are required for this test.

d. Expected Results
   i. After the tenth sound the program adds new sounds to a queue. After a set period the program removes the first sound and adds the sound from the front of the queue.

**Evaluation Plan**

Once the application is functional, it is desirable that the application be fun and entertaining to use. One way to test general user experience and see if there are any features that need adding or adjusting, is to let test groups play with Kwyjibo on their experience. There are several groups we would like to query, to cover all of our target audience. We would ideally like to test with fellow computer scientists for possible technical insight. Testing with musicians to see how Kwyjibo shapes up as genre-based music generator. Multi-discipline artists and musical novices can give us their
feedback on what other features we might have overlooked and would like to see implemented in possible future versions of the application.

**Team Areas of Responsibility**

Each member will have areas of the project that will be their primary focus. The team will also work together as a cohesive unit in order to advise, assist and learn from one another. Eric Wysocki and Chris Schilling will be co-project leaders. Eric Wysocki will focus on the project management side, assigning tasks and making sure we stay on schedule, since he has the most experience in managing teams. Chris Schilling will focus more on the technical decisions since he has the most software development experience among members of the team.

Alex Sommer's main focus will be the frontend development and user interface design using Android studio. He also has some musical experience and his knowledge will help us in analyzing music genre selections to feed the machine learning algorithm. He will also be able to listen to see how the algorithms' patterns and user created sounds will match up to our expectations of having a 'musically listenable' output from the application.

Jamie Eldridge's primary focus will be in several areas. He will be in charge of creating and maintaining the application's database full of user information, sound information, data for the tracks and sessions, and finally the patterns themselves from which Kwyjibo will generate the music. Jamie will also spearhead all the security measures our application will put into place. We do not perceive there will be much pertinent information that could put our users at risk. However, we must be vigilant in protecting our application from having too much downside from outside threats. Finally, he will be in charge of making sure all parts of the application's infrastructure
can communicate with each other. Since there is a lot of different types of data flying back and forth integrating all parts of the application will be key to its success.

Chris Schilling will be our go-to person for machine learning. We will all be learning and assisting him to develop the algorithm at the application's heart, but the team will follow his lead on most decisions important to this aspect of the application. Since he has the most experience he will also guide all other phases of the application development. He also has over eight years of experience writing and performing music in a band. His musical experience along with Alex's will be invaluable to analyzing the music genre inputs without the users' generated outputs.

Eric Wysocki will take the lead in the project management and assisting in every other phase of development while also filling any roles and needs in development that might surface. The machine learning algorithm's theory and testing. With over a decade of audio engineering experience he will oversee that the sounds the user records will be of highest quality possible for a mobile device. His knowledge of audio signal flow and processing will be valuable in manipulating the audio files before they get to the algorithm and while the sounds are rendered to an audio stream for the user to listen to. He will be working hand-in-hand with Alex on the front end development to ensure this happens.
Equipment

Software

Audio Utility - SoX (Sound Exchange)
Android APIs - SoundPool, MediaRecorder, AudioCapture, AudioTrack
Database - Microsoft SQL Server 2008, Visual Studio (C#)
Frontend - Android Studio (Java)
Machine Learning API - SciPy (Python)
Version Control - git and github

Hardware

Server

Operating System
- Windows Server 2008 R2 Standard
- Service Pack 1

System Specifications
- Processor: Intel(R) Xeon(R) CPU E5-2630L v2 @ 2.40 GHz
- RAM: 4.00 GB
- 64-bit Architecture

Consultants

Dr. Rick Leinecker
Budget and Financing

We have self-financed the domain name of motw.tech to host our Web API, at a cost of $4.99 for one year through GoDaddy. Otherwise, at this time our project has no budget. Dr. Leinecker has graciously volunteered some space on his personal server free of charge. If that falls through or we decided later to expand the scope of our project where that will not be sufficient we will then look at renting a server and splitting the cost amongst the four group members. For software our team will use ones that are free or able to be free due to an educational discount. Microsoft Visual Studio and SQL Server are both available for free through their Dreamspark program.

Project Summary and Conclusions

In summary, computer generated music is an uncommon and noteworthy challenge. We have decided to tackle the problem using machine learning based artificial intelligence because such techniques are on the forefront of AI in computer science today. All the members of this group chose this project because we are up to the challenge. The bulk of this design document exhibits our ambition, as most of it regards the design of the talking components, rather than the solution to the problem.

The medium we have chosen to present our solution to the world is an Android application. The end goal of our product is to connect people around the world through a shared musical experience that is accessible to nearly anyone. In the process of completing this project, we will have gained necessary knowledge for our future professional lives, as well as creating an enjoyable application that uses Music of the World.
# Project Milestones

<table>
<thead>
<tr>
<th>Date</th>
<th>Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/7/16</td>
<td>Phone Recording Test (usability/limits of phone sound input)</td>
</tr>
<tr>
<td>4/1/16</td>
<td>Initial Sound Design in DAW into what the algorithm will produce</td>
</tr>
<tr>
<td>4/28/16</td>
<td>Submit Final Project Design Documentation</td>
</tr>
<tr>
<td>5/10/16</td>
<td>Finish Prototype Backend</td>
</tr>
<tr>
<td>5/20/16</td>
<td>Finish Prototype UI</td>
</tr>
<tr>
<td>6/1/16</td>
<td>Finish First Bare Bones Project Prototype (no algorithm)</td>
</tr>
<tr>
<td>7/1/16</td>
<td>Finish Testing of Prototype</td>
</tr>
<tr>
<td>9/1/16</td>
<td>Finish Music Creation Code Algorithm (alpha phase)</td>
</tr>
<tr>
<td>10/27/16</td>
<td>Finish Final Project (beta phase)</td>
</tr>
<tr>
<td>11/1/16</td>
<td>Begin Final Project Testing</td>
</tr>
<tr>
<td>12/1/16</td>
<td>Complete Project Testing / Launch</td>
</tr>
<tr>
<td>Early Dec ‘16</td>
<td>Final Project Presentation</td>
</tr>
</tbody>
</table>
References


