Pitch Perfect

Authors: Oluwafunmbi Jaiyeola, Sai Chandana Korivi, Carlos Taveras
Carnegie Mellon University Department of Electrical and Computer Engineering

***Abstract*—**A web based application that provides novice singers with intuitive lessons and actionable feedback to improve their pitch, timing, and foundation in music theory. Feedback is generated by features derived from pitch detection and clap detection algorithms. Additionally, Pitch Perfect will provide ample resources and exercises needed to improve users' musical theory expertise, as well as access to resources on more advanced topics in vocal performance.

***Index Terms* — Yin Fundamental Frequency Estimation, Clap Detection, Note Strain Analysis, Vocal Coach, Rhythm, Note Detection, Cents**

INTRODUCTION

Music is ubiquitous and has been a central part of what it means to be human for several millennia. In fact, the Smithsonian [1] claims that making music is a universal human trait that goes back at least 35,000 years. Since then, music has evolved and manifested itself in many shapes and forms, but, despite this, the most prevalent instrument continues to be the human voice.

When learning how to perform a task, it is common to seek instructors who can use their expertise to guide your development. This is no different in learning how to sing. However, as a novice singer, it can be discouraging and difficult to seek the help of a professional to develop your singing ability, especially considering the steep cost of personal singing lessons which tend to range between $50-$100 per hour [2], and the dependency of available instructors nearby. These concerns are now exacerbated by the COVID-19 pandemic, which has almost certainly increased the prices and decreased the availability of in-person vocal instruction. In response, many instructors have started hosting their lessons online, but it can be difficult to justify paying full price for a singing lesson whose effectiveness is affected by the degradation of voice quality on platforms like Zoom.

To address this issue, our group, The Ensemble Methods, is creating Pitch Perfect: a web application that helps users perfect aspects of their singing from the comfort of their own home. Pitch Perfect is not an automatic voice quality assessment tool, which is still an open research problem, but instead a platform to help novice singers perfect their pitch control and identification, understand and develop rhythm, and polish their music theory knowledge through tailored lessons and exercises with feedback on how to improve. To accomplish this, we will employ the use of pitch and clap detection algorithms to measure pitch and rhythm, respectively, as well as, a note strain detector based on pitch contour statistics over a note.

DESIGN REQUIREMENTS

Our requirements are broken up into several parts, highlighting the main areas of user interaction with our product, namely: pitch to note accuracy, clap detection accuracy, note strain analysis, user interface interaction, feedback, and latency. Since much of our application's functionality depends on the effectiveness of our note and clap detection algorithms, we must ensure they are robust and accurate. The user interaction focuses primarily on making our application intuitive and enjoyable for the user. Finally the feedback is the means by which our application will communicate to the user how well they are performing, as a function of note accuracy, timing, and intonation.

A. Pitch To Note Accuracy

Note detection is a critical component in Pitch Perfect's functionality, therefore our system must be able to reliably detect notes. Notes are extracted from the pitch detection algorithm, so our pitch detection algorithm must accurately estimate pitch. We require our note detection algorithm to achieve a test accuracy of no less than 95%. We will thoroughly test the note detection accuracy against pure tones with added white Gaussian noise. After achieving this baseline accuracy, we will test our algorithm against a labeled dataset of sung tones [3].

B. Clap Detection Accuracy

Clap detection is an integral part of our rhythm exercises, thus we require no less than 95% clap detection accuracy on a preliminary collection of self-annotated claps that simulate the behavior we expect from our users. Ideally, we will like to test this against an existing annotated clapping dataset to test the generalizability of our algorithm.

C. Note Strain Analysis

Note or intonation strain will provide us with a metric to potentially discriminate between good and bad singing. We extract this feature by taking statistics of pitch data over a note, particularly the pitch variance. We will collect samples of good and bad singing intonation and determine how well this parameter correlates with the classification. We expect low pitch variance to be highly correlated with good singing quality, and vice versa.

D. User Interface Interaction

The main requirements for the web application's user interface are usability, and desirability. We want the users to be able to easily and intuitively navigate to different pages and singing exercises. Furthermore, it is important that users clearly and easily understand how the exercises will be performed and the purpose of each of them. Ease of use is essential in users wanting to come back and use the exercises we provide to keep on learning and improving their singing and music theory understanding. Since this web application will be used for learning purposes, with the majority of our target users most likely being children, we want our web application to be engaging with appealing visuals to grab their attention as well. In order to test the usability and desirability of our web application, we will be conducting focus group studies. We will begin conducting these studies remotely through zoom once our minimum viable product is ready to be used. Before these sessions, we will prepare a list of tasks that involve using the web application. These tasks will fall into 4 main categories:

1. Navigation to an exercise
2. Performing an exercise
3. Interpretation of feedback
4. Navigation to past feedback

During our user study session, we will provide all the participants with the same list of tasks. While they perform each task, they will be sharing their screen so that we can monitor their performance on the tasks. While we monitor their interaction and completion of tasks, we will measure how long it takes them to complete each task. After they complete all the tasks, we will provide them with a survey to rate their experience with the interface of the web application with a scale ranging from 1 to 5. These survey question responses will provide us with qualitative feedback. Overall, these focus group studies will provide us with quantitative and qualitative metrics to measure the usability and desirability of the web application, which we will utilize to improve the user interface.

E. Feedback

We require that we provide our users with useful and concise feedback. As we provide our users with both visual and textual feedback, we expect the visual feedback to be comprehendible. In order to meet this requirement, we will send in audio recordings to be analyzed by our feedback algorithms. Ensuring that all visual feedback is readable and the textual feedback is not too verbose. Based on this, we will be able to modify our feedback algorithms and representation.

F. Latency

Long response times and high latency can adversely affect a user's experience with a web application, which can lead to low product use and a poor perception of the product itself. Thus, we want to provide the ideal response times for different types of user inputs. According to an article, "Response Times - The 3 Important Limits" written by Jakob Nielsen, a web usability expert and human computer interaction researcher, there are three main response time limits for a web application [4] . First, for user inputs that don't require special feedback from the application and just a display of results, 0.1 seconds is the response time limit for the user to feel that the web application is reacting instantaneously. Second, for user inputs that do require special feedback, 1.0 seconds is the response time limit for the user's flow of thought to remain uninterrupted. Third, to keep the user's attention without then wanting to perform other tasks outside of the application, the response time limit for the special feedback is 10 seconds. In the case that the special feedback does take longer than 10 seconds, some form of time indication of when the feedback will be displayed should be provided to the user. Therefore, following the response time limits of this article, we will be limiting response times in a similar fashion based on the types of user inputs and form of feedback. For feedback on users' pitch and rhythm exercises we expect the response time of the feedback to be at most, 10 seconds. Since audio processing and analysis times can be variable due to many factors, in the case that the pitch or rhythm response time is over 10 seconds, we will be providing the user with a graphic percent done progress indicator that represents the progress of the audio processing and analysis as Nielsen recommends so that the user is assured of how long they need to wait. For feedback that doesn't require special processing such as navigation from one page to another, scrolling through a page, and filling in text inputs, the response time will be at most, 0.1 seconds. In order to measure the response times reliably and accurately, we will be using the django-debug-toolbar, a configurable set of panels that display debug information and CPU times about HTTP requests and responses. It allows us to see what tasks the code is doing to handle a request and give back a response as well as the time is spent for each of those tasks. Based on the insights and times this toolbar provides us, we can determine if we need to optimize our code in the case of any long response times.

ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

A. WebApp Implementation Diagram

All users have to go through an authentication process, as an existing user, a verification would be required at the login. If not an existing user, they would be navigated to a registration page where the user would fill out the necessary personal information as well as take a quiz on their vocal range. From this stage as well as after verification of login, the user will be redirected to the dashboard page where they have access to all the lessons, their feedback as well as external resources. *Appendix-Diagram*1 below demonstrates the flow of the web application.

B. System Diagram

Appendix-Diagram 2 below represents the integration of the user inputs and outputs, our pitch and clap detection algorithms, feedback generation, and hardware.

C. Django Framework (models, views, forms, templates)

The entire web application will run on the Django Framework. We have decided to use Django as it is based on Python, and already has MVC as its core architecture. This framework allows us to have large files, such as our audio files from the user as well as their music theory lessons. In addition, the framework is employed in several major projects, and is well established with a vast community to guide us through any errors we may run into. The Django Framework will also allow us to hold many of the media files and access them with low latency. All functionality of the system will be handled in the views.

D. Music Theory

Our entire product is focused on music training. For music theory, this aspect of the product is necessary for the user to understand some of the more advanced pitch and rhythm lessons. The music theory lessons are, however, provided in parallel with these lessons and can be run independently from the other lessons. We have gathered lessons from websites such as www.musictheory.net, as well as resources from classes offered at CMU. These resources will guide the lessons we provide to our users. Every lesson will be followed by a quiz to reinforce the relevant concepts. After taking the quiz the user will have access to the correct answer, and all answers will be stored in the database, allowing users the opportunity to retake the lessons.

E. Pitch and Note Detection

The singing exercises will make use of this module to extract pitch from raw audio recordings. This extracted pitch will then be used in the note detector to segment the recording into musical notes. The resulting notes will then be compared against the expected notes in cents.

F. Clap Detection

Rhythm exercises consist of users clapping along to a beat. This module will detect those claps and annotate the original audio recording with their locations. To determine whether a user is clapping on beat, we will set a threshold on the allowed elapsed time between expected and detected claps. This metric will then be used to generate relevant feedback about the users performance.

G. Note Strain Analysis

Upon registration, our application will identify users' vocal range. In doing so, we must quantify their ability to sustain a note. We characterize the difficulty of hitting a note as strain which is a function of change in pitch over time. Throughout singing exercises we will also collect data on note strain as a metric to gauge user improvement. The output of this module will be fed to the feedback generation module to provide the user feedback on their performance.

H. Feedback Generation

For the 2 main aspects of the system, pitch and timing, we will be generating the feedback from the detection algorithms. For pitch feedback, after the user's pitch has been recorded and converted to notes, the desired note and the user's note are passed through the feedback algorithm, created to return the level of accuracy the user was able to achieve to reproduce the desired pitch, taking into consideration a margin of error of 50 cents. For timing feedback, the clap detection algorithm will process the user's claps and return the timestamps at which the claps are detected. Based on these timestamps, the music notes representing the user's rhythm will be shown in the visual feedback. The timestamps of the user's claps will be compared against expected times of the rendered rhythm that is in the exercise. The expected times will be following a set specific tempo or speed such as moderato or andante, which are moderate or walking pace speeds. Based on how the user's timestamps and expected timestamps align, we will be showing the user if they clapped all the beats correctly, if they missed clapping a beat, or if they clapped an extra beat. For the visual feedback on timing, the user's claps will be rendered as music notes to represent their rhythm. The expected rhythm will also be rendered with music music notes and be annotated with colors based on which beats were missed, hit, or added. This rendering of music notes will be done through VexflowAPI.

DESIGN TRADE STUDIES

A. Measure Intervals (Cents vs Hz)

One metric we will use to generate user feedback is the difference between expected and detected tone. To make this comparison will convert the measured pitch values to cents. The distance between one tone to the next (i.e from one note to another, e.g C - D) is exponential in Hz. Converting to cents allows us to linearize frequency measurements, which is extremely useful when switching between octaves. In cents, octaves are separated by a fixed 1200 cents, whereas the difference in octaves in Hz differs by a power of 2. A semitone is the smallest interval of music within an octave and is equal to 100 cents. At this point, the musical pitch is considered to be the most dissonant when sounded harmonically. Although converting to cents will add another step to the system, its addition will greatly impact the ability of our system to provide useful feedback.

B. Pitch Detection Algorithm Selection

Since we're building a singing coach application, pitch detection algorithm selection was a critical decision in our design process. Pitch detection methods leverage properties either in time, frequency, or both in some cases, of periodic or quasi-periodic signals to estimate the fundamental frequency, or pitch, of a sound. The autocorrelation method is a common time-based approach that correlates segments of a signal in time with a shifted copy of itself over a small range of values. It estimates the periodicity of a signal by locating the shift that produces the largest peak. Another approach analyzes a signal in the Cepstral domain where periodicity is easier to analyze. Using a method similar to the Short-Time Fourier Transform (STFT), we can attain a time-cepstrum representation of the original signal which can then be used to estimate the pitch. While the cepstral method can reasonably estimate the pitch, the autocorrelation method, especially when augmented with post processing steps like in the Yin fundamental frequency estimator [5], outperforms in detecting monophonic pitch. Of all autocorrelation based pitch detection algorithms, we chose the Yin estimator because of the impact of the post processing steps on detection accuracy. In the paper, the Yin algorithm was compared to several other leading pitch detection methods on 5 databases of speech and achieved the lowest gross error percentage of all the algorithms on each database. These results encouraged us to go with the tried and tested Yin algorithm.

C. Visual Feedback

Visual feedback will be provided to users for each exercise to help them understand their pitch frequency accuracy and timing accuracy from the note, scale, and rhythm exercises. To represent their pitch frequency accuracy, we chose to use a pie-donut chart from the Highcharts library to represent their accuracy as a percentage. Since our target users are beginners with little to no knowledge of the technical details of pitch, such as cent margins, we thought percentages would be the most intuitive way for them to get a good understanding of how accurately they sang a note. The alternative would be to show them the exact frequency that they sang a note in and show that against the frequency of the note they were supposed to imitate. However, due to varying acceptable and unacceptable cent margins, this isn't an accurate representation of whether the user was singing the correct pitch. A percentage is also more encouraging of their progress and motivation to keep continuing the exercises since the pie-donut charts represent a relative accuracy percentage based on many other contributing factors other than the user's frequency. A pie-donut is also a very simple and compact representation of summary of large data. To represent a user's timing accuracy from the rhythm exercise, we chose to represent the user's rhythm with music notes, such as whole notes, half notes, quarter notes, and eighth notes in 4/4 time. We also color code notes in the expected rhythm's music notes based on which beats were missed, hit, or added. This rendering of music notes will be done through VexflowAPI. We chose this form of visual representation over representing their user's rhythm symbolically with durations indicated by bars instead of notes because durations would be useful to represent when providing real-time feedback to the user on their claps. However, since we are representing their rhythm post-processing, it will be not as intuitive to understand how to clap for durations. Therefore, we will assume the user understands the number of beats each music note represents and render the rhythm for the timing exercise and the user's claps with music notes and rather than durations for the feedback.

D. Django Framework vs Flask

Django Framework was compared to Flask, as these are the two top web development frameworks in the industry both boasting of having a vast community. One of the first considerations was the database. As we would be handling the storage of a lot of files, a relational database is considered, and as Django has an inbuilt ORM we can be able to manage testing on a small scale before integrating our Amazon S3 database. With Flask, although there is a flexibility to the database, we considered issues of compatibility as well as the possible learning curve in correlation with the amount of time allotted to building the application. There is an upside of Flask when it comes to routing and views, as in this case unlike Django which requires explicit statements of response handling it's request objects are always readily available. Finally, the big decision for Django over Flask is the security. Django comes with its in-built protection against common attack vectors with injections like CSRF, however with Flask so reliant on third party extensions, there is more pressure to maintain security by monitoring these third party extensions

E. Real-time vs. Post-processing

Initially, we considered detecting and displaying users' pitch in real-time to gauge their performance at any point in time. However, this task proved to be much more difficult than we had expected. For our application to truly be considered real-time, we would need to display users' pitch within no more than 100 ms, which severely constrains the amount of processing we can perform. Under these constraints, pitch post processing would have been minimal, thus increasing the opportunity for pitch, and note detection error. Furthermore, we reasoned that the inclusion of real-time pitch feedback will distract the singer from singing leading them to perform worse than usual.

F. Singing Quality Feature Extraction

In our ideation process, we considered developing a system that would extract rich features from a user's performance based on objective metrics of good singing. In our literature review, we found a paper [6], that attempts to solve this problem. In their paper, they identify 12 generally accepted criteria for good singing, including appropriate vibrato, resonance/ring, intensity, and dynamic range, some of which the authors tried to quantify as features. In order to evaluate a user's performance, though, we would need to provide the user with a song to perform and a reference by which to compare it to. While we considered allowing users to perform a song and compare it with a reference, much like this paper does, we figured that the problem was too unconstrained to solve over the course of a few weeks, so we constrained the problem to one that was feasible to solve within the allotted time. Instead of focusing on a breadth of features, we decided to focus on two key aspects of a singer's performance: pitch and timing. Instead of providing users feedback on their rendition of a song, we will develop lessons and exercises aimed to improve their pitch control and recall, rhythm, and music theory foundation.

G. Posture Detection

In the initial design there was the introduction of a posture detection component, that would allow users the opportunity to be corrected about their singing posture, using the OpenPose library. However, due to a few considerations this was left out of the final design. Firstly, there would need to be an existing sample pose library that would allow for us to correlate the user's posture to the sample posture. There currently doesn't exist a sample pose library of singing poses and building a sample library for a subsystem would be too time consuming. For this reason and additionally the expected time to complete a subsystem of our project which based on research could be classified as the entirety of a system, we have decided to leave this feature out of our application,

H. Hardware

To have users provide an audio input when recording their voice and their claps, as well as allow them to listen to their recordings, we will be utilizing a noise isolating, wired headset, which includes headphones and a microphone. More specifically, we will be using the Shure BRH440M Broadcast Headset. We chose to use an external microphone as opposed to utilizing a standard laptop's built-in microphone because external microphones aid audio processing. There are many factors of built-in microphones that can obscure the audio input. Laptop fans can add noise to your audio recordings, built-in microphones can pick up background noise since it picks up noises as far as three feet away and the microphone is more omni-directional. Overall laptop built-in microphones are ideal for facetime or virtual meetings rather than vocal recording. Furthermore, a user's mouth needs to be very close to the microphone when recording their audio so as to ensure that the audio quality of the recording is optimized. The microphone in the SHURE headset is a dynamic cardioid microphone with a boom microphone mounting type, which is ideal for clear vocal reproduction. We also chose to use external headphones as opposed to a standard laptop's built-in speakers because a user will still be able to hear external noises from whatever environment they are in. However, the SHURE headphones, which are noise-isolating, block out any surrounding and background noise. Additionally, since it can be inconvenient to purchase and use headphones and a microphone separately, we thought it would be ideal to use a headset so as to combine the two. Lastly, we chose the SHURE headset over other headsets because this headset is ideal for media production applications and is compatible with many audio processing softwares.

I. APIs

WebAudioAPI: In order to generate sounds of notes for users to listen to in order to identify and imitate pitches, we chose to use WebAudioAPI. More specifically, WebAudioAPI will generate piano notes to represent pitches and scales. WebAudioAPI is a very commonly used API for any developers looking to record, add effects to, or generate audio in different forms. Furthermore, many examples of different implementations of the API exist online. WebAudioAPI also allows us to create visualizations to represent our audio input. More specifically, it allows us to easily generate a piano visualization, which we plan on implementing to display to users when they're being evaluated for their voice range. This voice range can be represented on the piano visualization on which they are able to listen to the range of notes as well.

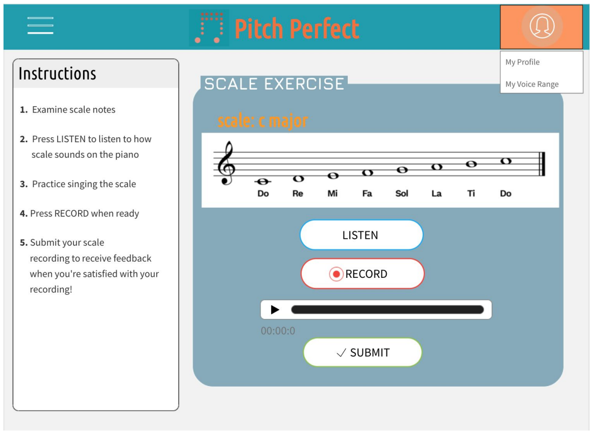
b. *MediaStream Recording API:* For the recording and playback feature of our pitch and rhythm exercises, we chose to use MediaStream Recording API. This API makes it possible to capture audio data for analysis, processing, and saving to disk. Its major interface is MediaRecorder. Mediastream represents audio tracks, and the MediaRecorder object takes the data from MediaStream and delivers it to us. We could've used WebAudioAPI to record the user's audio as well, however, there are more recording functions that MediaStream Recording provides compared to WebAudioAPI, such as pause() or resume() to pause recording your audio and resume whenever in addition to start and stop. Furthermore, it's easier and more straightforward to work with as the whole API's focus is on recording and playing back audio.

c. *Highcharts:* To provide visual feedback to the user on their exercises, we plan on using pie-donut charts with percentages representing the user's pitch accuracy and voice strain amount. Many APIs for rendering visual charts from data exist, however we chose to use Highcharts.js. Highcharts.js has an unlimited number of different chart representations compared to other APIs. Furthermore, the Highcharts.js charting library works with any back-end database or server-stack, therefore, it will work with the Django framework, Amazon Web Services, and the Amazon S3 database. We have the freedom to provide data in any form to Highcharts.js for it to render it in a visual chart. Data could be in CSV, JSON, Python, or R forms. More importantly, it mitigates one of our risks, compatibility of our web application with certain browsers or devices. The Highcharts library is compatible with any browser and device.

d. *Vexflow API:* To provide visual feedback on a user's rhythm exercise, we will be rendering the rhythm of the user's claps with music notes. We will match up the music note duration with their clap duration. Vexflow API is an open source online music rendering API. Since this is the only API we could find that has a library to assist in rendering music notes based on input data, there wasn't much deliberation on choosing this API amongst other libraries.

SYSTEM DESCRIPTION

Mockup Wireframe: Pitch - Scale Exercise



1. Examine scale notes
2. Press LISTEN to listen to how scale sounds on the piano
3. Practice singing the scale
4. Press RECORD when ready
5. Submit your scale recording to receive feedback when you're satisfied with your recording!

A. User Interface

The UI can be divided into four main categories. Each of these categories will have main features that distinguish them.

1. *Pitch and Rhythm Exercises*: Each exercise will have a rendering of the exercise, an instructions panel, a listen button, a record button, playback widget, and a submit button. The instructions panel will always be visible on the left side of each exercise until the user submits their recording. The listen button, when clicked, allows the user to listen to either the note, scale, or rhythm they are supposed to imitate, as many times as they need to. When the user is ready, they can click the record button to record their audio input. The record button will be shaded red while the user is recording and the label will change to stop, so that the user can click it when they are done recording. The playback widget allows the user to listen to their recording. Clicking the submit button submits the audio recording for processing and analysis.

2. *Feedback:* The feedback page will be provided to the user after they submit their recording for each exercise. This feedback page will either show their pitch

frequency accuracy percentage or their rhythm representation and accuracy.

3. *Voice Range Evaluation:* When the user is registering for an account with Pitch Perfect, they will be required to complete a voice range evaluation test. This voice range evaluation will consist of an instructions panel, a note scroller, a listen button, and a record button. The note scroller will allow the user to pick the note to imitate and record their vocal imitation of that note with the record button. Until the user has recorded every note in the note scroller, they will not be able to submit their test, with an "End Test" button. The vocal range test will be available for the user to take whenever they want to be evaluated again in their profile dashboard. When the user is finished taking this test, their vocal range will be displayed on a piano widget with a verbal description. This vocal range evaluation feedback will be available for the user to access at any time in their profile dashboard as well.

4. *Dashboards:* Each exercise will be listed in the user dashboard with its description and purpose. They will be categorized and put into tabbed lists based on if it's a pitch, rhythm, or music theory exercise. The is will be the same layout for accessing past feedback.

B. APIs

a. WebAudio API: WebAudioAPI will be used for piano note generation and piano visualization. For the piano note generation, we will create an AudioNode and AudioContext interface. AudioNode will allow us to perform audio operations, and we will run them within the AudioContext. Inside the AudioContext, we can assign an OscillatorNode as the audio source. The OscillatorNode is a periodic waveform that acts as an audio source for which we can select the frequency, type of waveform, time, and length of tone. We can generate notes and scales by using these interfaces. For the piano visualization, we would have to make use of the AudioContext and OscillatorNode interfaces, as well as PeriodicWave and integrate these with HTML elements.

b. MediaStream Recording API: MediaStream Recording API will provide the widgets for a user to record and playback their audio inputs. To record an audio input, MediaStream and MediaRecorder objects need to be initialized. The MediaStream will capture the user's microphone stream and the MediaRecorder object will emit the recorded data as events while recording. We can use start(), pause(), and stop() commands on the MediaRecorder object. We accumulate the events into an array until a stop command is detected, and gather it all into a Blob form of data. In order to display the playback widget, we will render the Blob into an <audio> element.

c. Highcharts: Highcharts will allow us to display a user's pitch frequency accuracy percentage with a pie-donut chart. To display our calculated percentage on this chart, we would need to initialize the pie-doughnut chart by adding the library code for it in the Javascript file of our web application. This code would be attached to a <div> element in the HTML. The calculated percentage would be provided to the data field of the Javascript object. Other visual customizations can be made to the object such as size, color, and labels.

d. Vexflow API: Vexflow API, which is written entirely in Javascript, will be used to render the music notes representing a user's rhythm and color code the expected rhythm music notes for the rhythm exercise. Each note is represented as a VF.StaveNote object, for which we can specify the clef, exact note, and duration. Since the clef will always be Treble Clef and we aren't concerned about the pitch for all the rhythm exercises, we mostly care about setting the duration for each instance of the object. For a whole note, the duration will be set as "w", for a half note, as "h", for a quarter note, as "q", and for an eighth note, as "8".

For example: Rendering of a quarter note

```
var note = new VF.StaveNote({duration: “q”})
```

To render more notes, more instances of VF.StaveNote can be added to a notes array so that they can all be rendered at once. To color code our music notes based on the comparison of the user's rhythm to the expected rhythm, we will utilize the .setStyle method that's available for a StaveNote object.

For example: Coloring the notehead red

```
note.setStyle({fillSyle: “red”})
```

Overall, our visual feedback on rhythm is utilizing the StaveNote object's duration property and setStyle method in order to render the user's rhythm and represent how it compares to the expected rhythm.

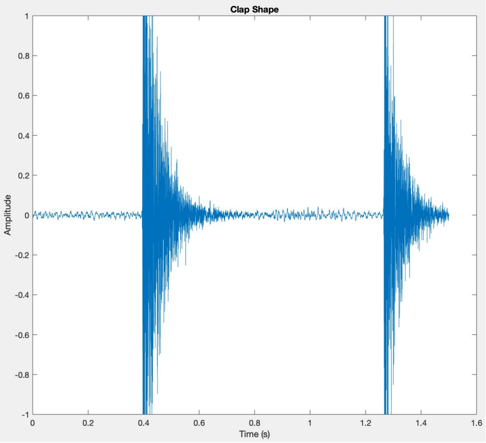
C. Pitch and Note Detection Algorithms

As aforementioned, we will be using the Yin fundamental frequency estimator as our pitch detection algorithm. We found a publicly available third-party implementation [7] of the Yin algorithm that uses modern scientific computing libraries in Python to decrease processing latency. This algorithm will be used to extract pitch from a recording of a sung performance. To extract notes from pitch, we will first convert the detected frequency to cents, take the average cent difference over an interval, and map the interval to the nearest tone within a 50 cent margin. To convert pitch measurements in hertz into cents, for a given desired frequency, we use the following equation:

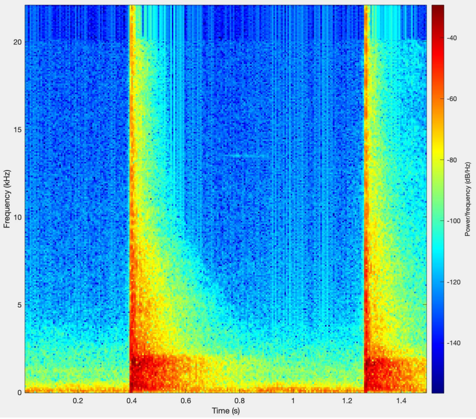
$$Cents = 1200 \times log_2\left(\frac{f}{f_{desired}}\right).$$

D. *Clap Detection Algorithm*

For the rhythm exercises of our application, we must reliably detect claps to determine whether the user is keeping the rhythm specified by the program. Claps have a distinct envelope, resembling decaying exponentials. The figures below serve examples of the shape of a clap in time and in frequency as detected by a 44.1 kHz microphone.



Plot of Two Claps



Spectrogram of Two Claps

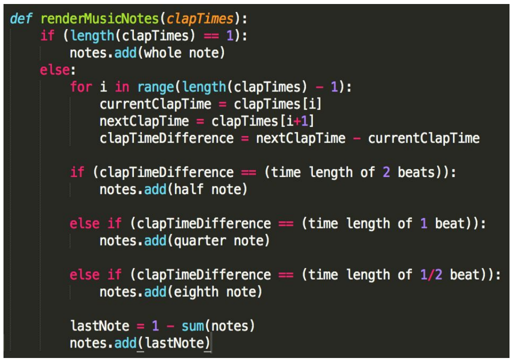
We can exploit these structures in various ways to extract timestamps of claps using a few different approaches. Some approaches to consider are convolving the claps with a matched filter, computing the root mean square and sorting on frames with the largest energy, or simply cleverly picking peaks. We will test each of these approaches, but are currently leaning towards the peak picking algorithm because of ease of implementation and testing.

E. *Note Strain Detection Algorithm*

To detect the difficulty users have maintaining a note, we will analyze the statistics of their pitch over a specified range. Vibrato which is a slight regular variation in tone, is to be expected, but large and irregular tone variations indicate poor pitch control. To capture this variability, we will collect the histogram of pitch values around a certain note and allow for a threshold in variability, which is captured in the variance of the distribution of pitch. With this metric we will be able to characterize good and poor pitch control and intonation.

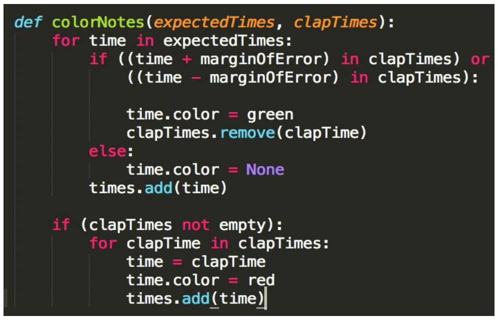
F. *Feedback Generation*

Feedback would be provided on four main metrics. The Pitch Accuracy, Timing Accuracy, Strain (both intonation and flexibility of transition) and the Music Theory Quizzes. Firstly, the pitch accuracy is measured by taking the cent difference between the user's pitch and the desired pitch. The cent difference is then used to calculate the expected percentage of pitch accuracy with a margin of 50 cents difference to be allowed. Secondly the timing accuracy. The claps of the users are sent through our clap detector algorithm which would return back the time stamps of all claps produced by the user. The timing of these claps as well as the number of claps would be compared with the expected timing and frequency of the claps and these comparisons would be reproduced as feedback for the user in a form of notes (i.e quarter note, half note, etc). The clap times would be translated to notes with the following high-level algorithm assuming that the clap times are accumulated into an array called clapTimes:



```
def renderMusicNotes(clapTimes):
    if (length(clapTimes) == 1):
        notes.add(whole note)
    else:
        for i in range(length(clapTimes) - 1):
            currentClapTime = clapTimes[i]
            nextClapTime = clapTimes[i+1]
            clapTimeDifference = nextClapTime - currentClapTime

        if (clapTimeDifference == (time length of 2 beats)):
            notes.add(half note)

        else if (clapTimeDifference == (time length of 1 beat)):
            notes.add(quarter note)

        else if (clapTimeDifference == (time length of 1/2 beat)):
            notes.add(eighth note)

        lastNote = 1 - sum(notes)
        notes.add(lastNote)
```

These notes will be rendered on a staff for the user to see. The expected rhythm will also be rendered with music notes on the staff. Additionally, based on the comparison of the expected timestamps of the claps from the rhythm exercise to the timestamps of the user's claps, we will color the notes in the rendering of the expected rhythm. For example, if there is a note in the expected rhythm that the user claps at the correct time, it will be colored green. If the user doesn't clap at that time for that note, the note will not be colored. Furthermore, if the user claps at a time which isn't expected, that time will be marked as a red note on the expected rhythm music note rendering. Here is a high level algorithm that represents this generation, assuming that expectedTimes is an array that contains the timestamps at which a clap should be detected, and clapTimes is an array that contains the timestamps at which claps were detected from the user:

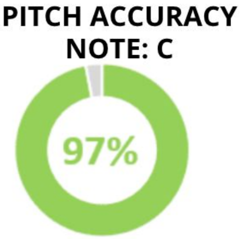


```
def colorNotes(expectedTimes, clapTimes):
    for time in expectedTimes:
        if ((time + marginOfError) in clapTimes) or
            ((time - marginOfError) in clapTimes):

            time.color = green
            clapTimes.remove(clapTime)
        else:
            time.color = None
        times.add(time)

    if (clapTimes not empty):
        for clapTime in clapTimes:
            time = clapTime
            time.color = red
            times.add(time)
```

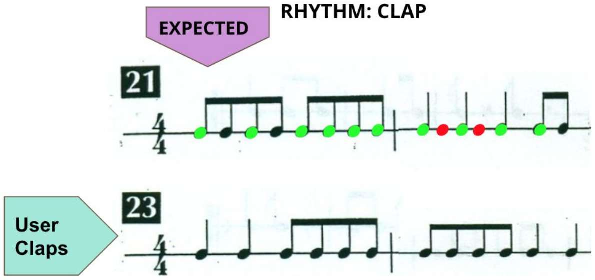
Then we have the strain both in terms of intonation and flexibility of transition.The note strain detection is going to be a function of the frequency and their ability to not deviate from the desired pitch over a certain period of time. Taking into consideration the user's natural vibrato, the cent deviation would be calculated over a period of time, and then the user's ability to stay within a certain range without too much deviation would then be collated and compared to our metric of strong, good or weak strain. For their flexibility to transition, the user is provided feedback on their ability to move from one note to the next, studying the cent difference of the user's pitch to the desired pitch as they move up the scale, and in similar fashion to the pitch accuracy they are given a percentage value on their ability to transition. Finally we have feedback on the music theory. After the user goes through the lessons on music theory they would be provided quizzes which would test their understanding of these lessons. Once completed the user would then be provided a grade based on how well they could answer the questions, as well as the solution to what they had gotten wrong.

G. *Visual Feedback*

Pitch Feedback:



Timing Feedback:



PROJECT MANAGEMENT

A. *Phase System*

Our schedule has been broken down into 3 phases: Alpha Phase, where we would be handling most of the functionality of our product. Then we have the Beta Phase, the expectation is that this phase coincides with our interim demo, so we hope to have a semi-working application with a few bugs and some functionality that may need to be improved on, and then finally our deployment phase, our product would be deployed on the cloud, free of bugs and ready for the final presentation.

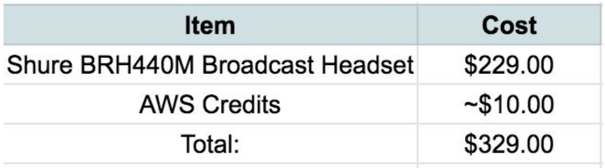
B. *Team Member's Responsibilities*

We have divided our project work such that Funmbi and Sai are developing the website's many pages and functions, as well as creating the rhythm and scales exercises, respectively. Carlos will be primarily responsible for the processing of users' clapping and singing, analyzing these detected actions, and interfacing with the website's frontend.

C. *Risk Management*

There are three main risk factors that we've considered in the design of our product. One of the risks is the possibility of external noise affecting the audio input recorded from the user. In order to mitigate this risk, we have decided to use an external noise-cancelling microphone instead of a built-in microphone in a laptop. Another risk is using a third-party implementation of the Yin Pitch Detection Algorithm might be an unreliable detection algorithm that might produce inaccurate results. Therefore, we will be thoroughly testing this module with the pitch accuracy tests aforementioned in the Design Requirements and tune the parameters of the algorithm accordingly. The last risk is variation in compatibility of our web application with different browsers and devices as our web application utilizes many different APIs. Some of these APIs might have poor compatibility with a few select browsers or devices. Therefore, to mitigate this risk, we will be building user interface and Django code that is scalable to suit any browser or device.

D. Bill of Materials



Item	Cost
Shure BRH440M Broadcast Headset	$229.00
AWS Credits	~$10.00
Total:	$329.00

RELATED WORK

Based on our research, there exists some products with some existing drawbacks which are of similar functionality to our application.

A. Live Singing Coach

First we have the live singing coach, an in person tutor which according to lessons.com has a cost ranging from $50- $100 and in current pandemic situations most of these classes have gone virtual, diminishing the training quality. However, it does have the upside of being trained by a professional, as well as getting better personalized feedback and covering more singing paradigms.

B. *Yousician*

Another related work is Yousician, which costs $9.99 monthly and $119.99, which although is not as pricey as an in-person vocal coach, is still relatively costly in comparison to our free app. Also, according to some vocal instructors on this site: https://singwell.eu/singing-apps/ the scoring system seems arbitrary and some of the lessons are a bit too complicated for a novice singer. although it does provide video feedback from vocal instructors to help guide the user.

C. Voco Vocal Coach

Another application that can be found in the AppStore is Voco Vocal Coach which provides the users with lessons as well as feedback on their musical performance, this app is rated a 4.1, however in the reviews it is stated to have poor instructions that makes it difficult for the user to easily navigate the app. In addition, the app is known to be constantly crashing for some of it's users rendering it ineffective for its users.

D. Pitchy Ninja

Pitchy Ninja is an application which allows the users to perfect their pitch accuracy, grading users on their ability to accurately reproduce a specific pitch as well as their ability to hold this note over a duration of time. However, pitchy ninja does not take into consideration the user's vocal range, producing lessons that are well out of their vocal range. It also does not produce useful and understandable feedback that can help the user improve their pitch.

E. Singing Carrots

The singing carrots application allows users to train their pitch accuracy as well as be made aware of their vocal ranges so as to practice with songs within their vocal range. However, this application also comes at a cost, rangine from $1 a month to $24 a month to get the full training experience. In addition, it doesn't provide great feedback on which pitch you couldn't accurately represent or the deviation of the user's pitch from the desired pitch.

REFERENCES

[1] "What does it mean to be human?" 2020. Smithsonian National Museum of Natural History. https://humanorigins.si.edu/evidence/behavior/art-music/musical-instruments.

[2] "2021 Singing Lesson Cost." 2021. Lessons.com. https://lessons.com/costs/singing-lessons-cost#:~:text=The%20average%20cost%20for%20singing,and%20even%20by%20zip%20code.

[3] Proutskova, Polina. 2017. "Phonation Modes Dataset." OSF. https://osf.io/pa3ha/wiki/home/.

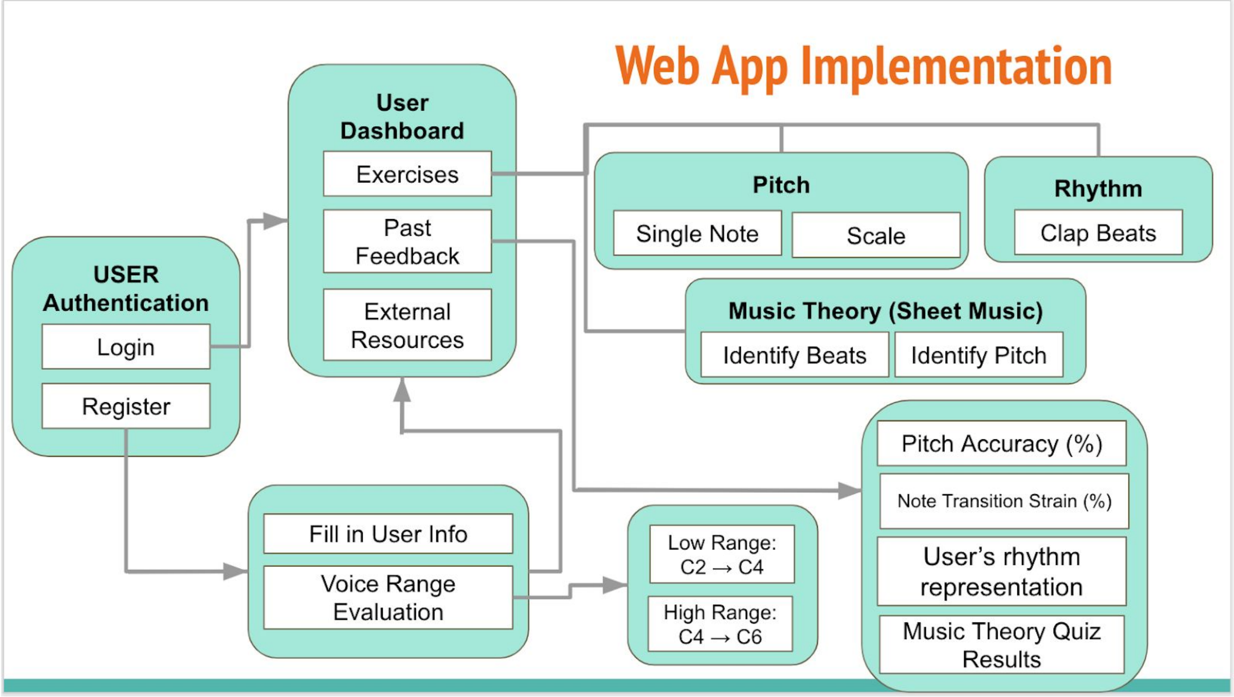
[4] World Leaders in Research-Based User Experience. (n.d.). Response time Limits: Article by Jakob Nielsen. https://www.nngroup.com/articles/response-times-3-important-limits/

[5] Cheveigne, Alain d. 2002. "YIN, a fundamental frequency estimator for speech and music." *The Journal of the Acoustical Society of America* 111 (4).

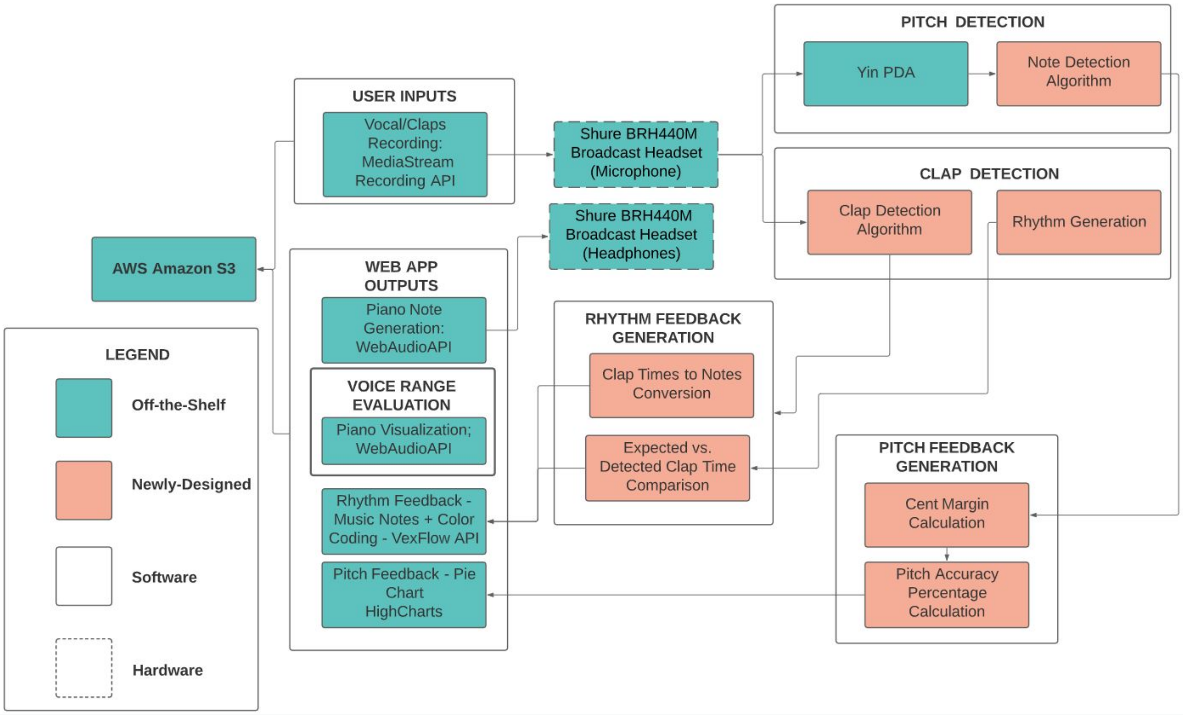
[6] Gupta, Chitralekha & Li, Haizhou & Wang, Ye. (2017). Perceptual Evaluation of Singing Quality. 10.1109/APSIPA.2017.8282110.

[7] Guyot, Patrice. 2018. "Yin." GitHub. https://github.com/patriceguyot/Yin.

APPENDIX - Diagram1



APPENDIX - Diagram2


PITCH DETECTION
Yin PDA
Note Detection Algorithm
USER INPUTS
Vocal/Claps Recording: MediaStream Recording API
Shure BRH440M Broadcast Headset (Microphone)
CLAP DETECTION
Clap Detection Algorithm
Rhythm Generation
Shure BRH440M Broadcast Headset (Headphones)
AWS Amazon S3
WEB APP OUTPUTS
Piano Note Generation: WebAudioAPI
VOICE RANGE EVALUATION
Piano Visualization; WebAudioAPI
Rhythm Feedback - Music Notes + Color Coding - VexFlow API
Pitch Feedback - Pie Chart HighCharts
RHYTHM FEEDBACK GENERATION
Clap Times to Notes Conversion
Expected vs. Detected Clap Time Comparison
PITCH FEEDBACK GENERATION
Cent Margin Calculation
Pitch Accuracy Percentage Calculation
LEGEND
Off-the-Shelf
Newly-Designed
Software
Hardware

SCHEDULE

Tasks	Phase 1			Phase 2			Phase 3	
	3/8 - 3/14	3/15 - 3/21	3/22 - 3/28	3/29 - 4/3	4/4 - 4/10	4/11 - 4/17	4/18 - 4/24	4/25 - 5/1
Pitch Detection and Testing	🟥							
Clap Detection	🟥	🟥						
Pitch to key mapping		🟥	🟥					
Strain Detection			🟥					
Feedback Generation			🟥	🟥	🟥			
Pitch Matching (single notes)	🟩							
Scale lesson (includes flexibility) piano	🟩							
Login and registration		🟩						
Home page (user account)		🟩						
Lessons Page - Category 1			🟩					
Feedback Page - UI			🟩					
Recording Functionality				🟩				
Piano Note Generation				🟩				
Voice Range Evaluation					🟩			
Database relationship	🟦							
Training your ear lesson	🟦	🟦						
Interval Lesson		🟦	🟦					
Metronome clap along rhythm generation		🟦	🟦					
Resources Page (Information)			🟦	🟦				
Resources Page (UI)			🟦	🟦				
Lessons Page - Category 2				🟦	🟦			
Feedback Page - Information				🟦	🟦			
Cloud Deployment							🟦	🟦
Design Report	⬜							
Phase 1 Integration			⬜					
Phase 2 Interim Demo Integration				⬜	⬜	⬜		
Final System Integration						⬜	⬜	⬜
Testing						⬜	⬜	⬜

Key	
Funmbi	🟦
Carlos	🟥
Sai	🟩
General	⬜