KaraoKey Design Proposal

Anna Gerchanovsky, Anita Ma, and Kelly Woicik

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**Vocal coaches are hard to find and are expensive, so there have been many apps created to mimic the feedback one would get at a singing lesson. However, we have noticed that current state of the art vocal coach apps are overly technical and do not provide real time feedback to the user.

KaraoKey is a casual karaoke-inspired web application targeted towards reducing anxiety behind recreational singing. We aim to reduce concerns about users' pitch by providing real-time visual feedback informed by the YIN algorithm. This feedback will be shown to the user within 250 milliseconds of their input vocals.

***Index Terms*—Pitch detection algorithm, real-time feedback, YIN algorithm, Vocal Coach**

I. Introduction

Music is an everyday part of life. People bond over the shared taste of songs and artists, and casual singing is commonplace at many social events. However, for some, singing is an activity that is anxiety inducing. Whether it is due to concerns about pitch, or confidence issues, some might be embarrassed to actually partake in casual singing.

When it comes to reaffirming or growing one's skills as a casual singer, the options are limited. Vocal coaches are expensive and hard to find. They are also often more than what the average singer would need, as vocal coaches target upper echelon singers. The content that vocal coaches teach can be overly technical and focus more on concepts that are irrelevant to the casual singer, such as breathing technique, vocal enunciation, and correct posture.

After doing a competitive analysis of existing vocal coach apps, we also realized that many apps do not have effective feedback mechanisms. Many apps did not give real-time feedback, which we think is essential in any educational setting. This is the main focus in our feedback mechanism loop that helps us stand out from other existing methods of vocal improvement. In addition to real-time feedback, we will also provide quantitative scoring so that users can gauge their performance and track growth.

With KaraoKey, we aim to create a web-app that provides real-time feedback for casual singers to reaffirm their singing abilities. We are taking a gamified approach inspired by one of the most popular and successful educational apps, Duolingo.

Users will sing to hardcoded melodies and songs, and the app will give them real-time visual feedback on a five-line staff as to whether they are on pitch or not. After they are done singing, a more comprehensive analysis will be done, and a scoring will be provided.

Our app's feedback mechanism is the main focus– we will not be overly technical and provide only baseline, rudimentary feedback so as to not overwhelm the user. We will be encouraging but honest, as reaffirmation and confidence in one's singing abilities is our main goal. We will gamify this approach, as to build on the user's intrinsic motivation to improve their singing. With these core goals in mind, we believe that this is an effective approach to tackle our use case.

II. Use-Case Requirements

We have set quantitative and qualitative benchmarks for our app to ensure the accuracy and effectiveness of our app. Most of them regard the ability to provide accurate, real-time feedback. We also have benchmarks on our user interface. The following are our requirements.

A. Latency in real-time feedback

According to the Web Audio API specification [1], a popular API used for "processing and synthesizing audio in web applications," a latency of 30 milliseconds or less is recommended for real-time audio processing. The Audio Engineering Society's Technical Council recommends a latency of 20 milliseconds or less, based on research on human perception of audio delays [2].

The reason why 30 milliseconds appears to be the standard in terms of real time feedback is due to a phenomenon called the Haas Effect. The Haas Effect states that if a sound follows another sound within 40 milliseconds, the two sounds are perceived as a single sound. These standards will be different for us, as we are providing real-time *visual* feedback for audio input. The Haas Effect provides guidelines on human perception of real-time *audio* feedback.

According to the Nielsen Norman Group [3], 100 ms is the limit for having the user feel like the app is reacting instantaneously to the user input. We will aim to have our page be as responsive as possible, but this metric is not feasible considering the amount of feedback processing needed. We've decided to relax this requirement to match the average human reaction speed, the time it would take for a user to react to visual stimuli.

For this reason, we are aiming for a latency of 250 milliseconds or less for visual feedback. If the latency is too high, the user may perceive a major delay between their singing and the resulting visual feedback, which may be distracting and make our application difficult to use. It is pertinent that we minimize the duration between when the user sings a note into the microphone and when the user gets the feedback for that note.

B. Pitch Detection Accuracy

The Journal of the Audio Engineering Society suggests that a level of accuracy between 90% and 95% is achievable for most pitch detection algorithms [4].

Ideally, we aim for such a level of accuracy. This would provide a more useful, effective, and engaging experience for the user. However, this comes with the risks of requiring more processing and complexity, thus potentially increasing the latency of the pitch detection algorithm.

As our visual feedback consists of simply plotting their input note on a five line staff, we have decided to aim for a 87% accuracy rate as a starting point. This means that our pitch detection algorithm correctly identifies the input note in most cases, but occasionally misidentifies a note. We believe that this visual representation of feedback gives a little more leeway, and is thus more forgiving to minor errors of pitch detection. We must strike a balance between accuracy and latency, and in this case, we decided to sacrifice and lower our required accuracy rate.

C. Range of Notes

The range of notes that the average human voice can produce is 85 to 1100 Hz. For male singers, the range of notes in songs usually spans between F2 (87.31 Hz) to G4 (392.00 Hz). For female singers, the range of notes in popular songs typically spans between A3 (220.00 Hz) to C6 (1046.50 Hz) range.

Because our app is designed to provide visual feedback for all types of beginner singers, our web application should be able to support a wide range of notes. We are aiming to recognize frequencies and detect notes from 85 to 1100 Hz.

D. Latency in post-song analysis

According to the same article by the Nielsen Norman Group [5], a 10 second delay is the upper limit to how long a user will stay engaged with an app. We believe that 10 seconds is more than enough time to process any post-song statistics, so we have lowered the latency requirement to be 5 seconds.

E. User Interface Interaction

Our user must be able to seamlessly use our app with little to no learning curve. Thus, our app should be easily navigable and easy to understand. There are many UX Metrics that Adobe recommends, including task time, completion rate, and task satisfaction [6]. We have decided on the following quantitative requirements.

Task time:

- Time to complete registration: 2 minutes. We believe that the registration page should be short and concise. If the registration page is too verbose, we believe that we will lose the user's attention.
- Time from login to singing page: 30 seconds. The user should be able to easily navigate through these pages and choose a song of their liking.
- Time from song analysis to singing page: 10 seconds. This navigation should be very trivial, as to make practicing over and over again an easy experience.

Completion Rate: We hope to have a 95% completion rate. We hope to provide an experience that allows users to stay completely engaged with our app. Task Satisfaction: We hope to have a 95% user satisfaction rate. We hope to provide feedback that is helpful to the user. We hope to have them feel like they have gained something from this experience.

III. Architecture and/or Principle of Operation

A. WebApp Design

Upon opening KaraoKey, users will be greeted with a home screen prompting them with 2 buttons: login and sign up. Based on their choice, they will be directed to separate pages in order to enter the necessary information. Once verified as a user, they will be directed to a mode selection screen where they will have the option of “Song Mode" or “Practice Mode”. These buttons will both bring up different screens filled with hardcoded songs or melodies, respectively. Upon clicking on one of the songs or melodies, the user will be directed to a song mode page that displays song lyrics, notes, song progress, and the users pitch. After completion of the song/melody, a graph of user pitch vs. desired pitch will be displayed as well as percentages of hit and missed notes. From this screen, they will have the option to return to the mode selection screen or try the song again. A more detailed diagram can be found in *Appendix - Diagram 1*.

B. System Diagram

A detailed system diagram can be found in *Appendix - Diagram 2*.

C. Django Framework

Karaoke will run on the Django framework as Anna and Kelly have experience working with this framework and it also has an intuitive MVC layout. Additionally, this framework will allow us to efficiently store the users along with their song/melody results within a user model. Views will handle all of the website navigation.

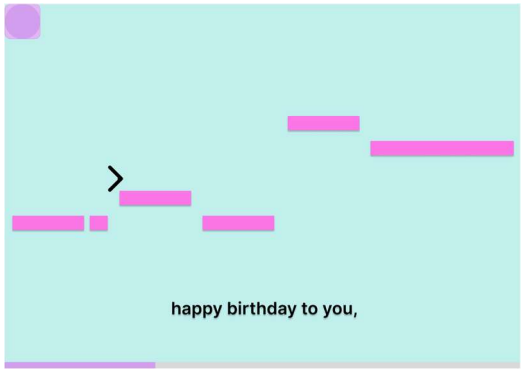
D. Pitch Detection

The pyaudio module will allow us to access the users’ microphone for recording while the aubio module will handle the pitch detection of the users’ voice. The recorded notes will then be compared with hardcoded note values for each respective song/melody to detect accuracy.

E. Feedback Generation and Scoring

We will be providing feedback during the song/melody in real time as well as after the song/melody. The real-time feedback will display where the user's pitch is relative to the hardcoded recorded pitch. The hardcoded pitch for the song will be displayed as blocks on the screen while the users’ pitch will be displayed via an arrow that traverses the screen as the user progresses through the song. Shown on the next page are example images of this process.

(a)



(b)

Fig. 1. (a) and (b) show the progression of gameplay during singing feature of Karaokey.

There will also be after song/melody feedback that will show a graph of the hardcoded desired pitch vs the users' recorded pitch. This graph will be a line graph with time on the x-axis and pitch in Hz on the y-axis. The 2 lines graphed will separate the users' pitch from the desired pitch and data will be gathered via a .txt file that will be written to once the user has finished the song/melody. This .txt file will display timestamps as well as recorded pitch. Additionally, we will generate metrics in the form of percentages displaying hit notes and missed notes.

IV. Design Requirements

A. *AJAX Update Requirements*

The AJAX update requirement, namely that it be called every 100 milliseconds, is satisfied trivially. AJAX is asynchronous, so it does not wait for a server response about the last request before sending another one, meaning that we can assign this 100 milliseconds period for AJAX calls. Even if a call takes above these 100 milliseconds, which would fall into our 250 milliseconds latency, an issue will not occur.

We will likely use MediaStream Recording API to record audio, which provides an ondataavailable event handler, which can collect audio information at a given frequency, or whenever it is called. If this is called by the AJAX update, we do not need to worry about problems sampling information at this frequency.

B. *Latency Requirements*

As we require our latency in real-time feedback to be less than 250 milliseconds from the time the user sings a note to when visual feedback for that note is displayed on the screen, we will have to make sure our pitch detection algorithm takes only a fraction of this time. As the best pitch detection algorithms have a latency of up to 60 seconds, we will also require that our algorithm's latency be under 80 milliseconds.

C. *Accuracy Requirements*

We will require that our pitch detection algorithm be at least 85% accurate for input frequencies within the range of 85-1100 Hz, to meet our accuracy and range of notes use-case requirements. This means that we must be effective at filtering out everything above 1100 Hz and below 85 Hz. We will be using the aubio module for this.

V. Design Trade Studies

A. *Pitch Detection Approach*

We have chosen to use an existing module for pitch detection. We researched the risks of developing a homegrown algorithm. It is very complex and difficult to create a fast algorithm to detect pitch. Some of the best pitch detection algorithms have a latency of 10-60 milliseconds [7]. There are many factors and optimizations that the best audio modules use. Especially as a group with limited experience with signal processing and debugging signal processing algorithms, these features will be extremely difficult to implement and optimize if we were to implement them by hand. Using a 3rd party module for pitch detection will help us achieve our latency use-case requirement faster.

B. *Pitch Detection Module*

After deciding to use a pitch detection module instead of creating it ourselves, we are using aubio for pitch detection. While there are many other python audio modules that exist, such as librosa, crepe, and pyAudioAnalysis, aubio is the most commonly used module for real-time pitch detection. This way, if we were to run into any bugs or issues, there would be many resources in the form of documentation, stackoverflow posts, etc. that could help us out. We have also used this module in the past, so we are most comfortable iterating on this project with this module. By relying on the standard pitch detection module, we will be able to reach our latency use-case requirement more confidently.

C. *Pitch Detection Algorithm*

In the aubio module, there are 6 pitch methods to choose from: schmitt, fcomb, mcomb, specacf, yin, and yinfft [8]. In order to decide which algorithm would work best for our use case, we tested all the pitch detection algorithms 4 times under the same conditions. The testing environment was as follows: a quiet apartment with a fan in the background and the Audio Technica BHPS1 as well as the Scarlett Solo 3rd Gen interface connected to a laptop. The four tests were as follows: Kelly singing to "C Major Scale - Slow", Kelly holding her phone playing "C Major Scale - Slow" to the microphone, Kelly

singing to “C Major Scale - Fast”, Kelly holding her phone playing “C Major Scale - Fast" to the microphone.

The results from all these tests can be found in the following linked google sheets document: https://tinyurl.com/karaokey-pda-tests. For the yin, specacf, and fcomb algorithms, the singing vs. recording tests outputted vastly different frequencies in Hz (i.e. 270 vs. 2000 Hz), which does not lend well to comparing the two audios for scoring. For the mcomb algorithm, these same discrepancies tended to happen, but to a lesser degree (i.e. 200 vs. 900 Hz). Nonetheless, this discrepancy was not ideal. For the schmitt algorithm, the outputted frequencies were quite close along the same song. However, our most accurate algorithm was by far the yinfft, reporting incredibly close pitches to Kelly’s recordings. Thus, we decided to move forward with the yinfft algorithm.

D. Backend Language

We have chosen to use a Python backend instead of C++. Despite the fact that python, a high-level interpreted language, is about 20 times slower than C++ , a lower-level compiled language, there are ways we plan on mitigating this slowness factor [9]. Most of the serious computation for the pitch detection will be handled by modules such as aubio, which is implemented in C++. We are also cognizant of the aspects of python that make it slow, and we will note this as we are coding. For example, as lists in python can hold objects of different types, each element needs to store additional metadata, which hinders runtime and memory consumption. We will avoid using lists and instead opt to use numpy arrays, which are much faster. Another factor that led us to choose python over C++ is development cost. This project will require a lot of iteration and improvement initially, and it will be much easier to do so using python. At least in the early stages of the project, we have decided to value Python’s lower development cost over its slowness factor. Eventually, if we truly cannot optimize our code further to meet our latency requirements and have reached a point where iteration has slowed, we will transition to C++.

E. Hz vs. Note Names

We have chosen to handle all of the pitch detection in Hz. Originally, we had mockups that suggested notes being displayed on a staff, which would require the mapping of Hz to note names for better placement on this staff. However, after pivoting our use-case to be targeted to beginners, we decided that this staff approach and subsequent note placement was not extremely user friendly or intuitive.

F. Feedback Delivery

To not overwhelm the amount of information going through the user’s audio channel, as they are already hearing themselves and the backing track, we’ve decided to provide feedback in the form of visuals. Such visuals will be basic and easy to interpret, so that we are not inundating the user with even more information they need to process.

G. Feedback Delivery Timing

There are advantages and disadvantages to real-time and asynchronous feedback in education. Real-time feedback is useful for users to make corrections in real-time, while asynchronous feedback allows for deeper reflection on their performance. However, singing on pitch is a skill that requires immediate adjustments if they are singing off-tune, so it is beneficial to provide a type of feedback that would allow them to make these changes in real-time. This also mirrors the type of real-time awareness that the singer will eventually develop to sing on pitch. Therefore, we have chosen to have our main feedback loop be synchronous. We will still include elements of asynchronous feedback in the form of a more detailed breakdown of the user’s performance after they have sung the melody/song. This way we can both build up the user’s pitch awareness and allow them to reflect more deeply on their performance.

H. Song Selection

Initially, our feature that has users sing to songs included a sub-feature that would allow users to input a song of choice to sing to. However, since then, we have decided to remove that sub-feature and only allow users to choose from a list of hard-coded songs. This decision allows us to meet our latency and accuracy use-case requirements, as this would limit the amount of preprocessing required by the (sometimes inaccurate) pitch detection algorithm to get the song’s pitches.

I. Frequency Range

Ideally, we will be able to accommodate all vocal ranges. The lowest note ever sung was 0.189 Hz (G-7), sung by Tim Storms, and the highest note was 1134.9 Hz (D#10), sung by Tsetsegsuren Munkhbayar. However, accommodating for such a wide range would make our pitch detection algorithm less accurate. To meet our pitch accuracy use-case requirements, we’ll be limiting our range of frequencies. This is so that the pitch detection algorithm can focus on a more specific range of fundamental frequencies for detection. This way, we can also limit the amount of noise and harmonics that can interfere with the pitch detection.

J. Microphone and Interface

When picking a microphone, it was important to consider options that minimize background noise and so our advisor, Professor Sullivan recommended a headset microphone as it ensures the microphone is always around 2 inches from the users mouth, picking up the most prominent signal. A lot of the headset microphones we found online tended to be incredible headphones with an okay microphone attached. However, our web app depends on accurate audio in order to provide correct feedback to our users’. Thus, we looked into broadcasting headsets and settled on the Audio-Technica BHPS1. This headset not only cancels out background noise via the headphones, but also works as a dynamic microphone to decrease peaking in the microphone despite it being very close to a users’ mouth.

Additionally, we purchased an interface in order to translate the signal from the high quality microphone we have into something that our computer could process in full. We decided on the Scarlett Solo 3rd Gen as it was compatible with our Audio-Technica headphones inputs/outputs and also was a

more budget friendly, yet high quality interface option.

K. Framework

When considering how to build our web application, we thought about both Django and Flask as a framework. We selected Django, for a few reasons. The first consideration was familiarity. Both Kelly and Anna are comfortable building Django web applications and have li ttle experience working with Flask. So, out of time considerations, Django was preferred, unless Flask had specific features we needed that were not available with Django. Django also provides a stronger base and more support for front end features, both of which would allow us to develop more in a short time scale. Flask is more flexible as a framework and tends to make deployment simpler. However, the flexibility didn't outweigh the benefits of Django for us and, because we don't currently plan on deploying our web application, the ease of deployment wasn't a factor for us.

L. Graphics Display Mechanism

When considering how to display feedback, we considered using an existing graphics package, like Graph.js, or designing it on our own with HTML, css, and JS. Using an existing package has the benefit of being easier, more sophisticated and visually pleasing, and likely having more responsive design in order to scale with window size. However, an existing package could be limiting and not provide the specific functionality we want. It may be difficult to achieve the design we had in mind exactly with an existing framework. Currently, we are using Graph.js to provide feedback post-song. During the song, we are experimenting with using a stepped line chart for target pitch, and a moving element to indicate the user pitch in the y direction and time in the x direction. This sacrifices our original plan of a scrolling target pitch, but looks cleaner, and we are, in general, more confident in it working.

M. Graphics Display

As mentioned in the previous tradeoff, the original design included continuously scrolling target pitch, but we are currently considering a static graph with target pitch, which the user pitch scrolls through horizontally as time passes. This means we do not have to worry about the current target pitch remaining aligned on the screen if the window is resized. However, we liked the idea of a scrolling target pitch because it focused on a smaller portion of the song at a time. As a compromise, our current plan is to break the target song or melody into chunks, similarly to how karaoke machines show a portion of the lyrics at a time. Each target pitch chunk is displayed statically, but is switched out after it is done, allowing the user to focus on smaller portions of the song at a time.

N. Overall tone

Since our initial proposal, our project has shifted from being focused on a vocal coach to being more gamified and providing an experience more similar to traditional karaoke. This approach was heavily inspired by one of the most successful language education apps, Duolingo. A reason for this shift was to appeal to our target audience. Our target audience is less experienced and has a casual interest in singing, and does not need extremely sophisticated feedback. This allows us to focus on the user experience and making the feedback more engaging. It also limits our scope to simpler analysis and excluding advanced vocal techniques from consideration.

VI. System Implementation

A. Subsystem A- Pitch Detection

Our Pitch Detection subsystem will have the workflow of the diagram shown on the following page. First, the user will be singing into the Audio Technica BPHS1 headset. This headset will be connected to the Scarlett Solo 3rd Gen interface via a XLRM 3-pin connection and a ¼" headphone jack connection. The XLRM 3-pin connection works to upload the inputted users' voice from the headphones to the interface, while the headphone jack allows the user to hear themselves. As seen in the arrow connection, the users' voice is processed before it gets to the Scarlett Solo due to the Audio Technica's ability to filter out background noise in the inputted signal. Once at the Scarlett Solo, the users' voice will be interpreted into a readable signal for the laptop and connected via a USB-C wire to the laptop.

Once at the laptop, the processed users' voice will go through the web interface frontend and be passed to the python pitch detection backend through views.py (more mentioned in the next section). Once the backend receives the users' pitch on the PyAudio Stream, it will be passed to Aubio, which will run a YINFFT pitch detection algorithm in order to detect the pitch of the users' voice (in Hz). It will then send this pitch data back to the front end. This pitch will then be displayed on the screen via an arrow so that the user is able to see their pitch via visual feedback.

B. Subsystem B- Web Application Implementation

The organization of the web application for this project will follow the standard design of a Django web application. Navigation between pages will be handled by the Django urls.py and views.py functions, which will render generally static HTML templates. The two pages we expect to make use of JavaScript will be the gameplay page, during which the user will perform, and the feedback afterwards.

The feedback page will use JavaScript in the form of Graph.js, in which feedback will be rendered in the form of charts. We will likely use line graphs in order to represent the user pitch compared to the target pitch, over time.

The gameplay page will make use of more sophisticated JavaScript. We plan to use the MediaStream Recording API. This creates a stream in JavaScript, which will continuously record the user vocals. The stream can have a specific sampling frequency assigned to it. An advantage of this API is that it can be processed during and after recording. The ondataavailable event handler is called when new data is recorded, or on command, like we can with AJAX. We can use this event to pass information into our python views with AJAX. We will process the most recent data in python views. Here, we can update the model of the current game run in the database. The model will have a one to one relation with a user who is

currently playing, and the audio data of the vocal recording.

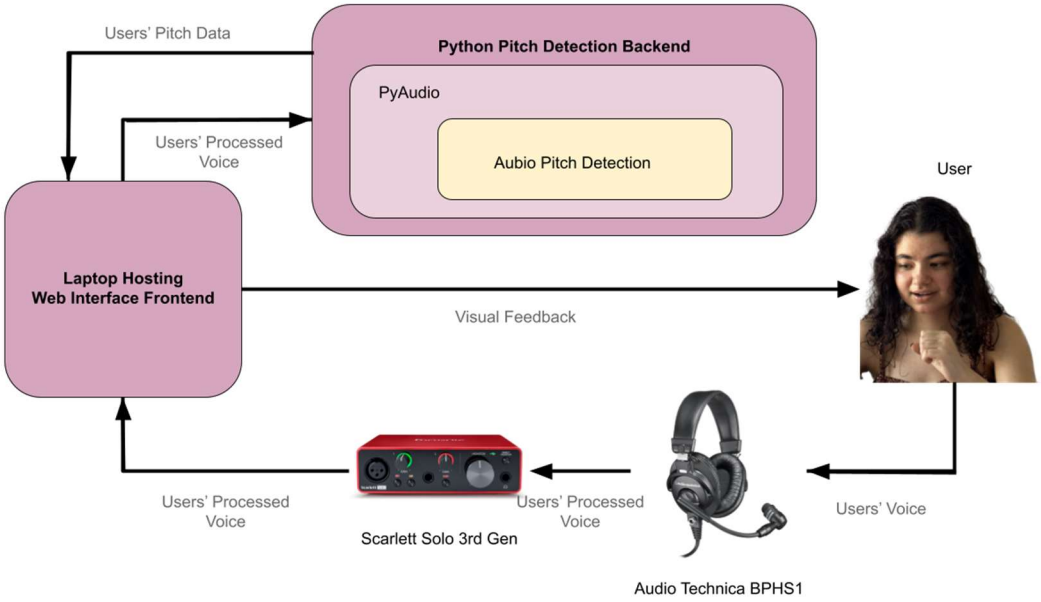


Fig. 2. Block diagram of pitch detection subsystem workflow.

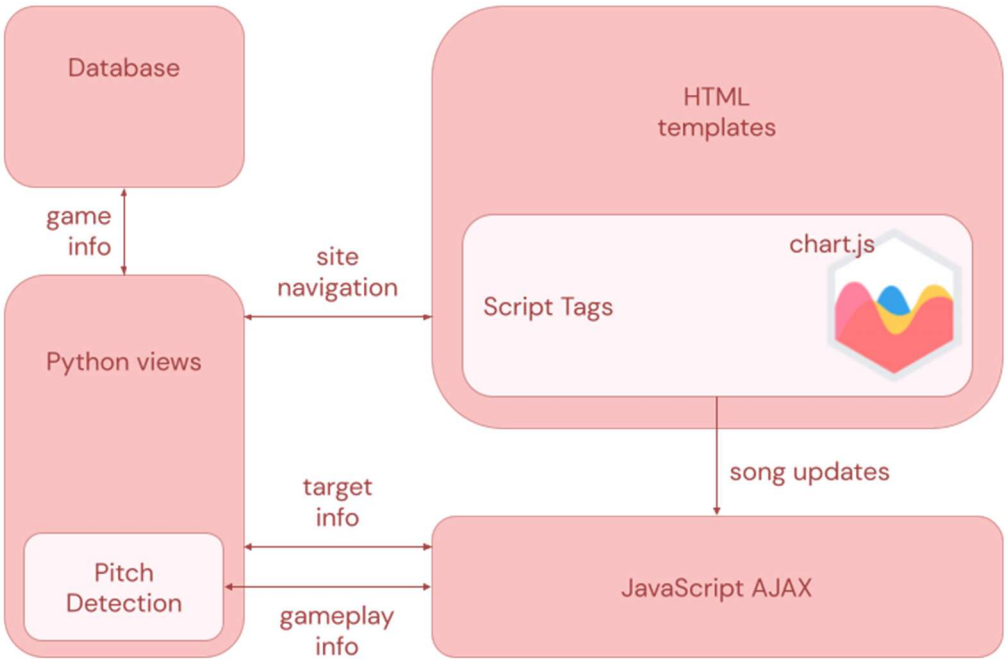


Fig. 3. Block diagram of web applicaton workflow

Pitch detection will be done at this point. Comparison with the target pitch ca n be done here or later. Then, we can pass information back to the JavaScript AJAX, which can then update the view to show where the user pitch is. Upon completing the song, the model should be updated one last time, and the stream should be stopped. At this point, the game object can be used in scoring and evaluation, providing overall feedback and comparisons to the target pitch.

VII. Test, Verification and Validation

A. *Tests for Pitch Detection*

In our use-case requirements, we are aiming for a latency of 250 milliseconds. In order to verify that we have achieved this metric, we will screen record with audio our application running the melody "Happy Birthday" and have the singer abruptly change notes. We will then review the screen recording to measure how long it took between a user changing

notes and our web application moving the users' pitch arrow. The screen recording will more accurately allow us to measure this timestamp rather than relying on our own reaction time with a timer. We will do this multiple times through the course of a song and take the average time across all trials in order to extrapolate to our true latency. A success will be achieved if our average is less than 250 milliseconds.

Another use case requirement relies on our frequency range being fixed at 85 to 1100 Hz. In order to ensure that our pitch detection algorithm does not detect frequencies outside this range, we will use our current aubio/pyaudio pitch tracking demo code, play frequencies less than 85 and greater than 1100 Hz in a continuous note and verify via the output that only frequencies between 85 to 1100 Hz were detected. A success will be achieved if our output only contains frequencies between 85 to 1100 Hz.

Our last use case requirement dealing with pitch detection relies on an accuracy of 85%. We will measure this by using our current aubio/pyaudio pitch tracking demo code and playing notes of known frequency. We will then compare the output of our code with the known frequency that we were playing through the microphone. A success will be achieved if we are able to identify the correct frequency in 85% of our trials.

B. Tests for Web Application

The first requirement we want to test, in this case, is latency in post-song analysis. Our goal is to provide users with a summary on their performance over the course of the song within 5 seconds of them completing the song. To confirm that we meet this requirement, we will perform multiple trials of all the implemented songs, and record the time it takes. We will record apparent time with an external timekeeping device that is triggered by the user on completion of the song and stopped on receiving feedback, as well as internally, by recording time stamps upon exiting the game screen to successful load of the feedback afterwards.

The second set of requirements that is handled by the web application is user interface interaction requirements. In order to test these, we want to give users who are unfamiliar with our web application a task and observe how they perform. Based on our requirements, the test will likely consist of registering, logging out, logging back in, navigating to song selection, choosing a song, performing it, viewing the results, and exiting. We will do this monitoring ourselves, to note any issues, as well as with internal logging statements. We hope to be able to perform this test on 10+ people, and it may require additional rounds if users have difficulties and the first round of user interface interaction tests don't go well. If we redo tests, we would like to find new people, in order to keep using test subjects who are unfamiliar with our application.

We have three time requirements: 2 minutes to complete registration, 30 seconds to navigate from login to singing, and 10 seconds to exit from post-song feedback to home screen. These tests will be monitored by internal logging keeping track of the timestamp at which the user enters and exits a given page of our web application.

The 100% completion rate will be monitored by observing the tests. If a user is unable to complete the task at hand, that will be considered a failure and we will have to reconsider our system design. Once we have made necessary changes, another round of testing will occur.

For task satisfaction, this will be measured by performing a post-task survey of the user. Although this is not finalized, the survey will likely ask users to rate statements similar to these on a scale from strongly disagree to strongly agree:

1. I felt comfortable using the web application.
2. Navigation was simple and intuitive.
3. I was satisfied with the speed of feedback provided.
4. I was satisfied with the format of the feedback provided.

VIII. Project Management

A. Schedule

Our schedule consists of two generally parallel branches - development of pitch detection and of the web application - that are progressing in parallel until they are ready to be integrated. Both the web application navigation and framework as well as pitch detection algorithms should be developed enough after spring break to be able to touch base on how information is best stored and passed between the two. Our goal is to have separate elements of both branches complete by the beginning of April, at which point the focus will be on finalizing the integration, as well as completing any remaining tasks, as, at that point, the rest of the time is slack.

B. Team Member Responsibilities

Kelly is tasked with primarily focusing on the pitch detection algorithm. Her responsibilities include learning about and selecting the best available pitch detection algorithm for our purpose, familiarizing herself with it, and setting up the basic template for using it. She is also taking the lead on purchasing materials.

Anita's main responsibility is generating feedback. This builds on Kelly's work, and also requires an understanding and analysis of the target pitch. Feedback includes both real time feedback and feedback and analysis after the song is played.

Anna's focus is building the web application. She is to focus on navigation, storage, and organization of the Django web application, which will include identifying where and how the pitch detection algorithm and pitch comparison can be integrated.

C. Bill of Materials and Budget

Item	Cost
Audio-Technica BPHS1 Headset	$234.33
Focusrite Scarlett Solo 3rd Gen	$128.39
Total:	**$362.72**

D. Risk Mitigation Plans

Our greatest concern is the quality and ability of our pitch detection algorithm. In this case, we focused on two possible

issues that could come up: issues with pitch detection accuracy and issues with pitch detection latency.

If we run into issues with the accuracy of our pitch detection algorithm, the ultimate plan for mitigating this would be to test out and ultimately move onto a different pitch detection module. However, our first course of action will be to see if any processing could be done to the input audio in order to mitigate this. For example, we are considering using a low pass in order to eliminate non-singing noises such as breathing, gasping, or sighing.

If we run into issues with latency, there are a few options to consider. First, we will reexamine how information is passed through the system and evaluate if we could make this more efficient. For example, we may find that using AJAX has a larger lack than expected, in which case we would look for alternative ways to update the view. However, if it seems that the latency issues are caused by the algorithm itself, we may have to move to C++, which is faster than Python, when we choose a pitch detection algorithm.

IX. Related Work

When working on this project, we had a few sources of inspiration both for our goal, and for our implementation.

A. At Home Karaoke Machine

As we are focusing on the karaoke aspect of our project, just with the addition of feedback, this is one of the closest alternatives we have to consider. An at home karaoke machine comes with a speaker and microphone. Optionally, it may have the ability to duet, the ability to connect to a personal device and display lyrics. These often cost greater than $200, but cheaper versions in the $40-$50 range are available.

B. Rock Band

Rock Band is a video game where users aim to imitate the performance of a real band for a song, using specialized controllers to simulate instruments like drums and guitar, and a microphone to score vocals. The singer is scored like our user will be, on the accuracy of their pitch. Target notes and lyrics scroll in order to guide the user, again, similarly to what we plan to implement.

C. Pitch Perfect

Pitch Perfect is a Spring 2021 Capstone Project that aims to provide users with feedback on their singing. Users perform an exercise, and, afterwards, receive feedback on their pitch, their rhythm detection (as measured by claps), and their posture. They provide feedback on more metrics than we aim to. However, they do not provide real time feedback, which we aim to do, because of the difficulty they had in reducing latency.

X. Summary

Our final goal for this project is to create a karaoke web application that is able to provide users with feedback on their singing by comparing their pitch with that of the target song. We will provide real time feedback as well as a summarized report upon completion of the song.

Our aim is to provide users who are beginners or just not quite confident with their karaoke skills with an opportunity to practice in a fun but productive way, where they can enjoy the karaoke format, while still receiving constructive feedback. We do not aim to provide vocal coaching or feedback on advanced techniques, and therefore KaraoKey will not be designed to analyze scatting or ad-libbing. Doing so is outside the scope of our project and, additionally, we believe that these users are likely already more confident and comfortable with practicing in a real karaoke setting.

Our aim is to keep our system accessible to our goal user. Our materials include a headset microphone costing ~$230, but we aim to develop a web application that is able to perform with cheaper materials as well if it is used in a non-crowded, generally quiet environment. We also want to emphasize the real time feedback, so keeping latency down while passing the vocal input between elements of our system will be key.

Glossary of Acronyms

AJAX – Asynchronous JavaScript And XML
FFT – Fast Fourier Transform

References

[1] "Web Audio API," *www.w3.org*. https://www.w3.org/TR/webaudio/#AudioContext-section (accessed Mar. 03, 2023).
[2] "Audio Guidelines for Over the Top Television and Video Streaming," 2016. Accessed: Mar. 03, 2023. [Online]. Available: https://www.aes.org/technical/documents/AESTD1005_1_16_09.pdf
[3] J. Nielsen, "Response Time Limits: Article by Jakob Nielsen," *Nielsen Norman Group*, 2019. https://www.nngroup.com/articles/response-times-3-important-limits/
[4] S. Siddiq, "Data-Driven Granular Synthesis," *www.aes.org*, May 11, 2017. https://www.aes.org/e-lib/browse.cfm?elib=18619
[5] J. Nielsen, "Response Time Limits: Article by Jakob Nielsen," *Nielsen Norman Group*, 2019. https://www.nngroup.com/articles/response-times-3-important-limits/
[6] "Usability Metrics: Measuring UX Design Success | Adobe XD Ideas," *Ideas*. https://xd.adobe.com/ideas/process/user-testing/usability-metrics-measuring-ux-design-success/
[7] C. Harte, "Towards automatic extraction of harmony information from music signals.", 2006
[8] "Man page of AUBIOPITCH," *aubio.org*. https://aubio.org/manpages/latest/aubiopitch.1.html (accessed Mar. 03, 2023).
[9] "Intel | Data Center Solutions, IoT, and PC Innovation," *Intel*. https://www.intel.com/content/www/us/en/homepage.html?ref=https://www.intel.com/content/www/us/en/develop/articles/is-python-slower-than-c-for-mathematical-computations.html (accessed Mar. 03, 2023).

XI. Appendix

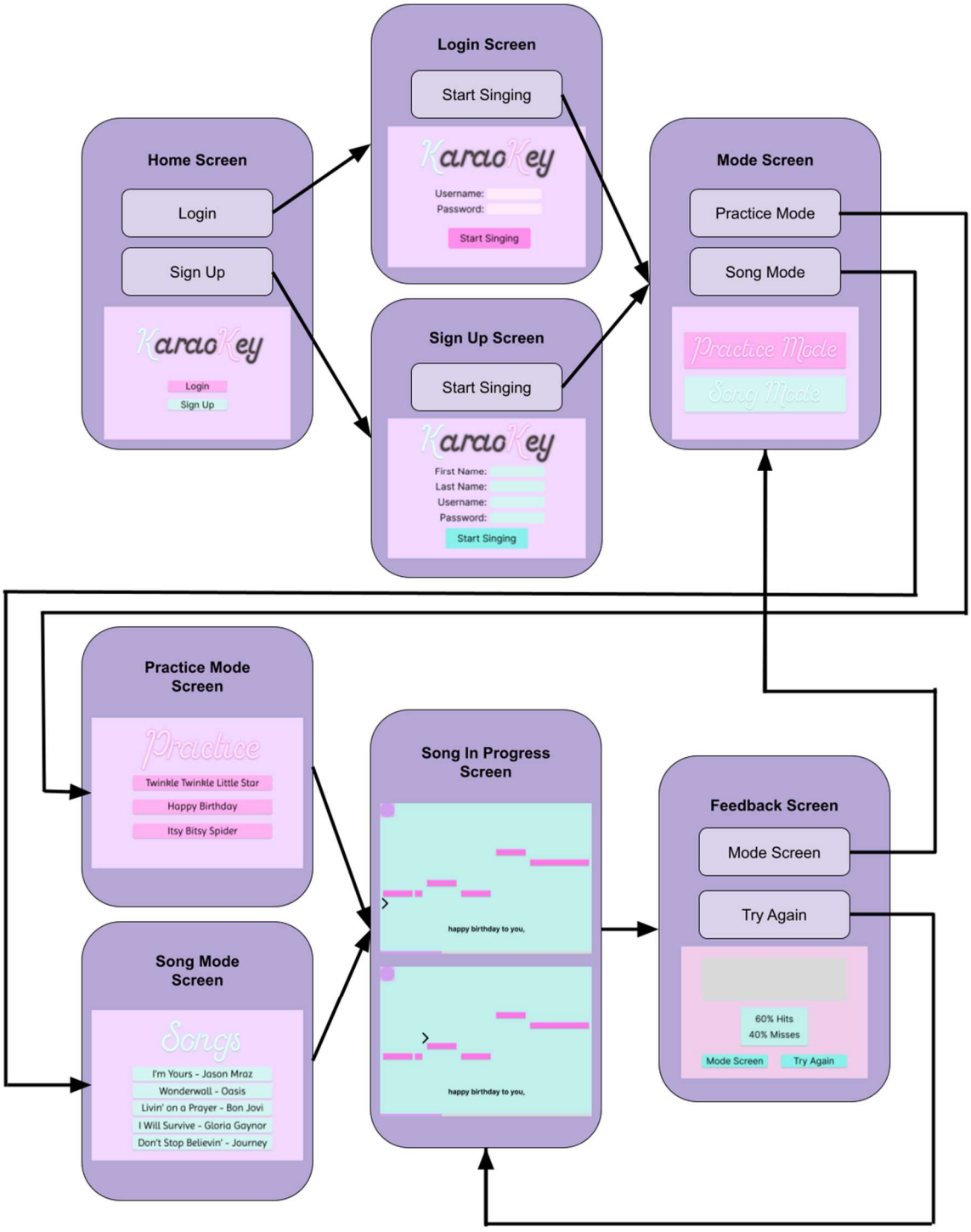


Diagram 1: Web Application User Interface Design

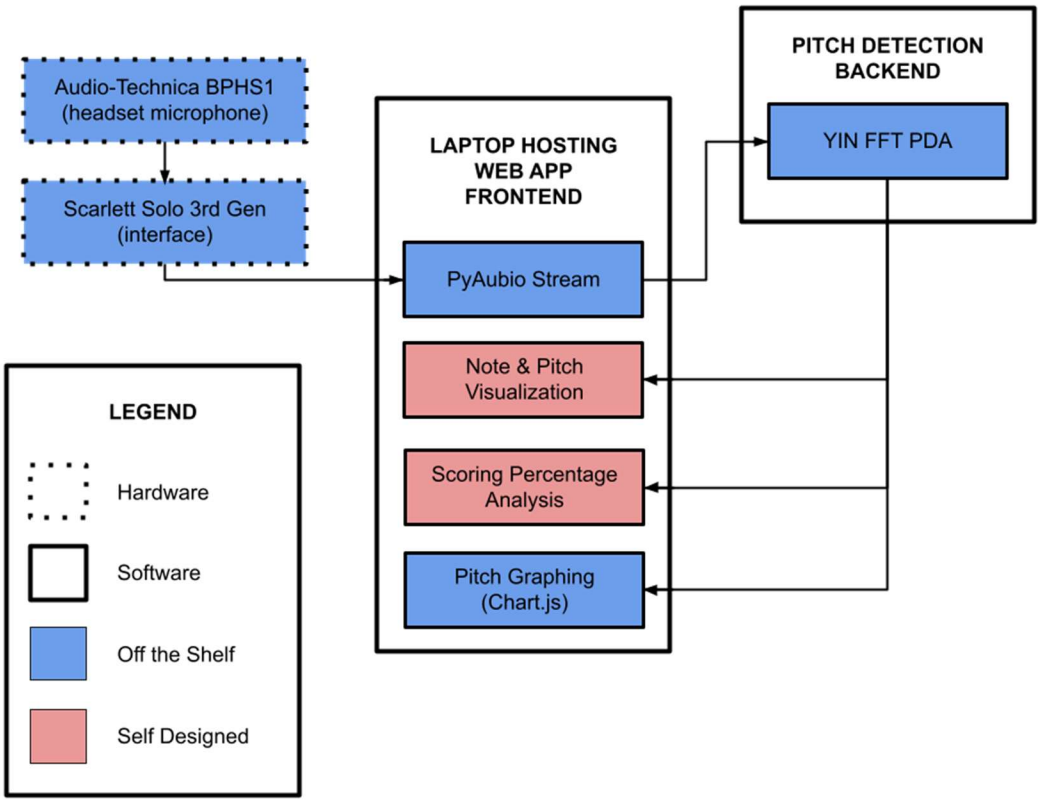


Diagram 2: System Diagram

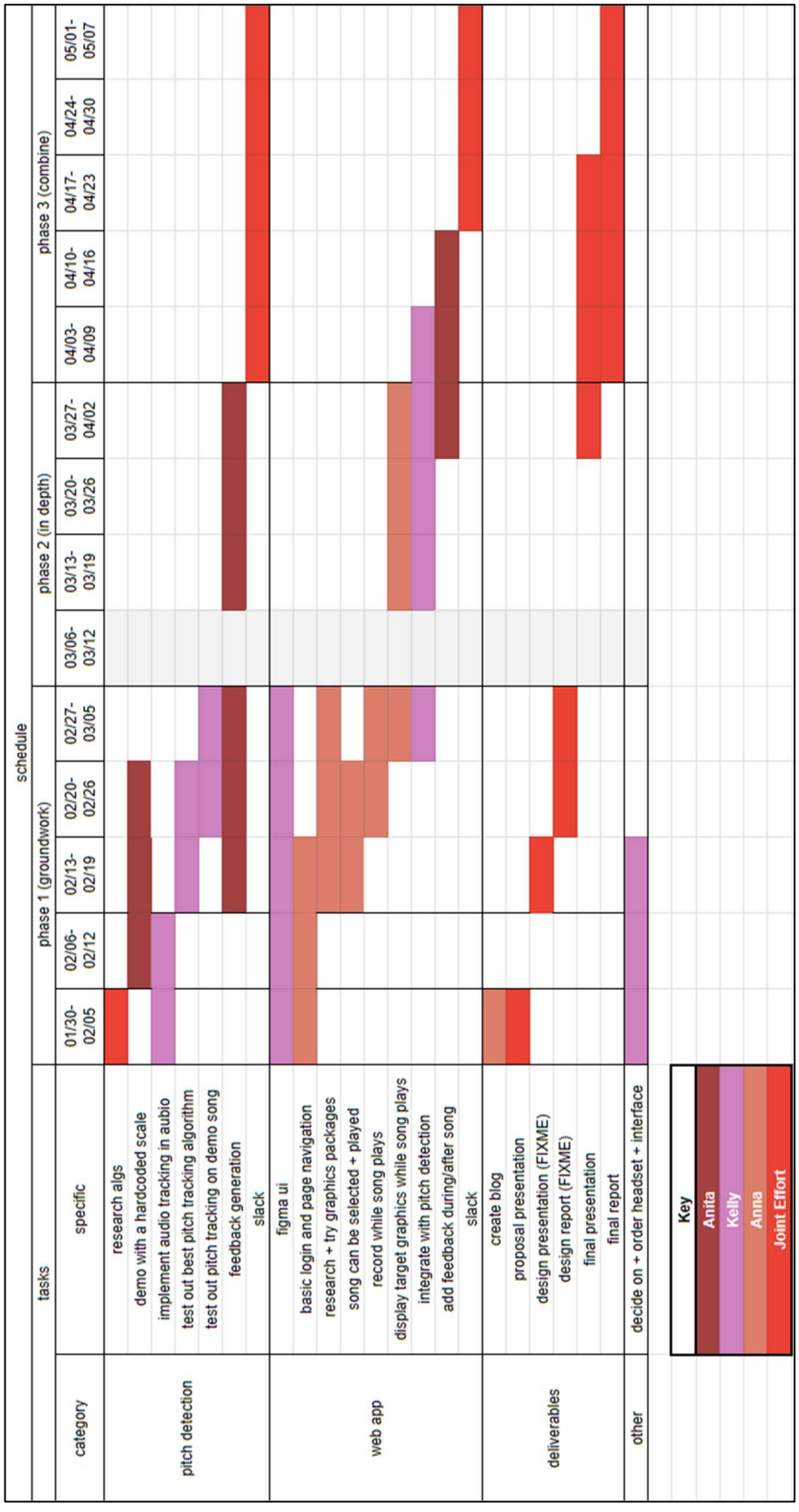


Table 1: Schedule