KaraoKey Final Project Report

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 not easily accessible.

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 visual feedback and playback informed by the YIN algorithm.

***Index Terms*—Pitch detection algorithm, real-time visualization, YIN algorithm**

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.

We created an app that is targeted toward the casual singer. Our app's feedback mechanism is the main focus– it is not overly technical and only provides baseline, rudimentary feedback so as to not overwhelm the user. The feedback is encouraging but honest, as reaffirmation and confidence in one's singing abilities is our main goal. With these core goals in mind, we believe that this is an effective approach to tackle our use case.

In our app, users sing hardcoded melodies and songs, and the app displays their sung pitches on a chart and displays whether they are on pitch or not. After they are done singing, a more comprehensive analysis is shown and scoring data is provided.

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. As an addendum, we have removed "latency in real-time feedback" as a use case requirement. In user testing, we confirmed that this did not impact the success of the app with regards to the use case requirements. The following are our requirements.

A. 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 [1].

Having high pitch detection accuracy provides 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 chart, we aimed for a 87% accuracy rate as a starting point. This meant that our pitch detection algorithm correctly identifies the input note in most cases, but occasionally misidentifies a note. This visual representation of feedback gives a little more leeway and is thus more forgiving to minor errors of pitch detection.

B. 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 currently supports a wide range of notes. We recognize frequencies and detect notes from 85 to 1100 Hz.

C. Latency in post-song analysis

According an article by the Nielsen Norman Group [2], 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 5 seconds.

D. 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.

In addition to this, the user interface should not be overwhelming, in both the language presented and the formatting of the data. Note that this specific use case requirement has been adjusted from the design proposal version to be more relevant to this specific app.

III. Architecture and/or Principle of Operation

A. *WebApp Design*

Upon opening KaraoKey, users are greeted with a home screen prompting them with 2 buttons: login and sign up. Based on their choice, they are directed to separate pages in order to enter the necessary information. Once verified as a user, they are directed to a song selection screen with buttons associated with different songs. These buttons both bring up different recording screens based on the song selected, respectively. The recording screens display the target pitch in blue with a red line following the timing of the backing track with lyrics being displayed at the top. After completion of the song, users are prompted to start their visualization in which the target pitch is displayed in blue and the users pitch is displayed in red with the lyrics up above and the backing track playing. From this screen, the user is given statistics on their singing. A detailed diagram of this process can be found in Appendix - Diagram 1.

B. *System Diagram*

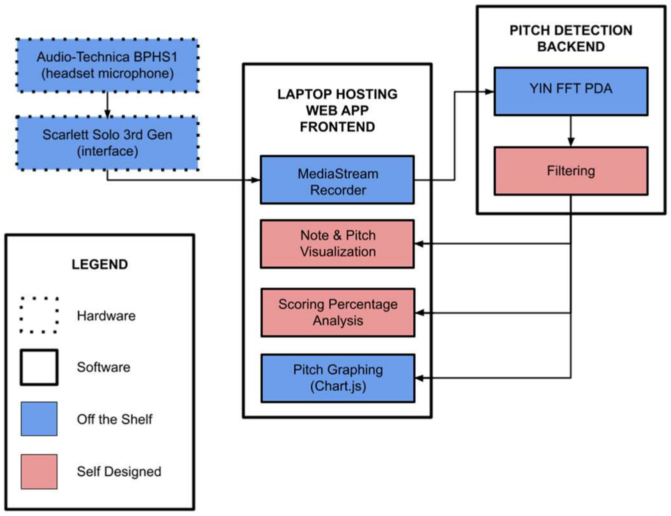

Audio-Technica BPHS1 (headset microphone)
Scarlett Solo 3rd Gen (interface)
LAPTOP HOSTING WEB APP FRONTEND
MediaStream Recorder
Note & Pitch Visualization
Scoring Percentage Analysis
Pitch Graphing (Chart.js)
PITCH DETECTION BACKEND
YIN FFT PDA
Filtering
LEGEND
Hardware
Software
Off the Shelf
Self Designed

Fig. 1. A system diagram of KaraoKey.

C. *Django Framework*

KaraoKey runs on the Django framework as it has an intuitive MVC layout and also supports dynamically updated web pages. Additionally, this framework efficiently stores the users along with their song/melody results within a User model. Views handles all of the website navigation.

D. *Pitch Detection*

The MediaRecorder module records the user's voice. The Aubio module takes in the user's voice recording as a .wav file and determines a pitch. The pitch is then processed to ensure validity and is scored for accuracy upon comparing the user's pitch to the target pitch.

E. *Feedback Generation and Scoring*

We provide feedback to the user via real-time visualization. The backing track plays while the user sees a chart displaying the target pitch as well as their pitch. The target pitch is displayed via a blue line while the user's pitch is displayed in red. Shown below is an example image of this process.

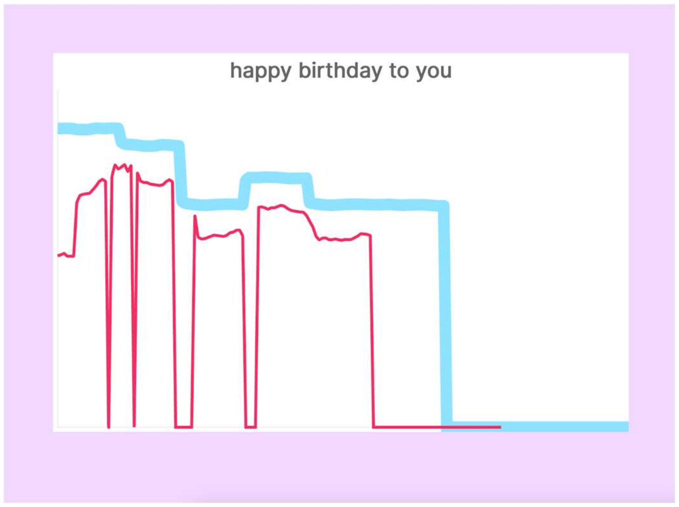
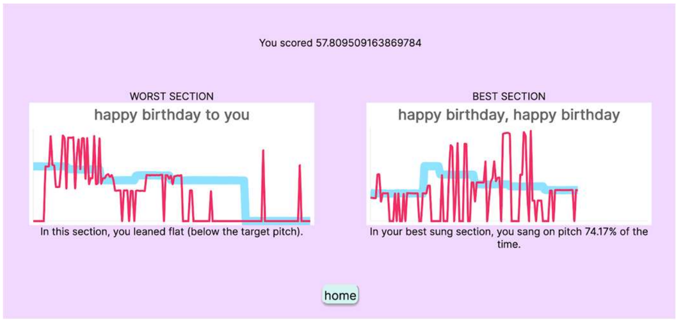


Fig. 2. A screenshot from the KaraoKey app displaying the main feedback mechanish. The user's pitch (in red) is plotted against the target pitch (in blue).

There is also statistic-based feedback given to the user at the end of this visualization. These statistics include: score, percentage of notes sharp, percentage of notes flat, percentage of notes on pitch, best section details, and roughest section details. The score is based on singing the correct note as well as the deviation from the correct note in cents. Percentage of notes sharp or flat tells the user whether they are singing too high or too low, respectively. The best section and roughest section details provide the user with a graph of the target pitch, users pitch, and lyrics for the section they scored highest in and lowest in, respectively. A sample of this statistics page can be seen below.


You scored 57.809509163869784

WORST SECTION

happy birthday to you

In this section, you leaned flat (below the target pitch).

BEST SECTION

happy birthday, happy birthday

In your best sung section, you sang on pitch 74.17% of the time.

Fig. 3. A screenshot from the KaraoKey app displaying the statistics page.

IV. Design Requirements

A. *AJAX Update Requirements*

Ultimately, our project shifted from real-time feedback, which the AJAX latency requirement was originally meant to help support. KaraoKey no longer performs pitch detection on every AJAX call, which actually allowed us to make these calls at a much higher frequency. Our focus is now on providing high quality visualization and playback of the user experience, which made us prioritize more frequent updates to the UI.

The period of AJAX updates that we landed on is 20ms, i.e. 50 updates a second. With this, we were able to achieve smooth and very granular animation of pitch, and more than meet our AJAX update requirements. However, because the nature of what we use AJAX has changed, so had the expectation of the frequency of updates.

B. *Latency Requirements*

We have removed the real-time feedback latency requirements and only have latency requirements for the post-song analysis. As stated in our use case requirements, the post-song analysis latency requirement is 5 seconds. Justification can be found there.

C. *Accuracy Requirements*

We required that our pitch detection algorithm be at least 87% 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 used the aubio module for filtering.

V. Design Trade Studies

A. *Pitch Detection Approach*

An existing module is being used for the PDA. We researched the risks of developing a homegrown algorithm, but 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. 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 would be extremely difficult to implement and optimize if we were to implement them by hand. Using a 3rd party module for PDA helped 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 used aubio for pitch detection. While there are many other python audio modules that exist, such as librosa, crepe, and pyAudioAnalysis, aubio suited our needs of processing a .wav file. We have also used this module in the past, so we were most comfortable iterating on this project with this module. By relying on the standard pitch detection module, we were able to more confidently reach our latency use-case requirement.

C. *Pitch Detection Algorithm*

In the aubio module, there are 6 pitch methods to choose from: schmitt, fcomb, mcomb, specacf, yin, and yinfft [4]. In order to decide which algorithm would work best for our use case, we tested all the pitch detection algorithms 2 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 two tests were as follows: Kelly singing to “C Major Scale - Slow” and Kelly singing to “C Major Scale - Fast”.

We ended up deciding on the YINFFT algorithm as it did the best job of introducing few peaks and also reading stable pitches. More details on this choice can be found in the testing section of the report.

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 [5]. 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.

Initially, we had our main feedback loop be a synchronous mechanism. However, due to difficulties with real-time feedback integration, we have changed it to be an asynchronous mechanism. Asynchronous feedback allows for deeper reflection on their performance.

After testing, our original use-case requirements were still met with this asynchronous mechanism. Changing the feedback mechanism from synchronous to asynchronous didn't seem to affect the efficacy of our app.

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, our app should 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 limited our range of accepted input 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. Our advisor recommended a headset microphone as it ensures the microphone is always around 2 inches from the user's 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 little 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. Recording Implementation

The recording implementation has the workflow of the following diagram:

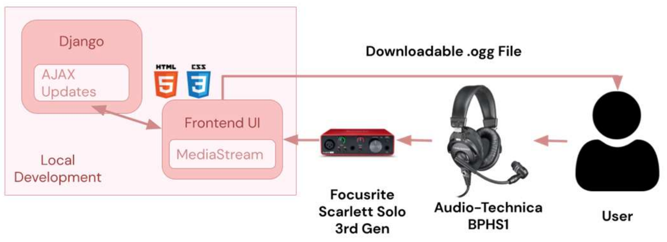


Fig. 4. Block diagram of the recording subsytem workflow.

First, the user sings into the Audio Technica BPHS1 headset. This headset is 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.

Recording of audio is done via the MediaStream Recording API. The user gives KaraoKey microphone access, from which an audio stream is created. This stream is passed into a MediaRecorder object, and incoming audio data is pushed onto the chunks of the recorder object. Upon completion of the song, the recorder is set to inactive and creates a blob object of MIME type ogg. This object is set to be available for download via url generated with the URL.createObjectURL JavaScript function.

B. Pitch Detection Algorithm and Scoring

Our PDA and scoring algorithm has the workflow of the following diagram.

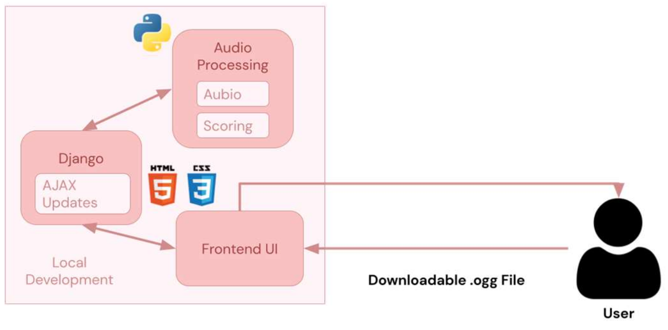


Fig. 5. Block diagram of the ptich detection and scoring algorithm workflow.

First, the frontend receives a .ogg file of the users voice which will be passed to the python backend via an AJAX request. Upon receiving this .ogg file, the python script utilizes the module PyDub in order to convert this .ogg file to a .wav file. This .wav file is then fed into Aubio to convert this .wav file into a pitch in Hz utilizing the YINFFT algorithm. These pitches are then checked for validity. If the pitch is < 200 Hz, it is replaced with 0 Hz. If the pitch is > 1000 Hz, it is replaced with the last valid pitch. This filtering is utilized to get rid of noise coming from breathing as well as the pronunciation of hard consonants such as 'p', 't', etc. Once the pitches are received, they are placed into a list and fed into the scoring algorithm. The reasoning behind this change since the design proposal can be read in the "Risk Management" section.

Accuracy scoring works in the following way: There is one main function that takes in two inputs, the user frequency and the target frequency. These frequencies are then fed into a note detection algorithm where the note, octave, and deviation in cents are outputted. Using the note, octave, and deviation in cents, an accuracy score is calculated.

The accuracy score is based on two factors: the note accuracy, and the deviation in cents accuracy. 70% of the score was determined through the note accuracy and 30% of the score was determined through the deviation in cents. This ratio was chosen after multiple rounds of tests where an average singer's frequencies were scored using the scoring algorithm. We found that the 70-30 split gave realistic, but not too low scores. It correctly penalized those who were singing completely off and vice versa.

In the note accuracy calculation, if the user correctly sang the note, they would automatically get a score greater than or equal to 70%. If the user was a half note off, they would get a score above 50%. If the user was more than a half note off, they would score a 0.

If the user sang the correct note or was only a half note off, they would get additional scoring points for their deviation in cents. The scoring for this portion was calculated using a linear scale.

The two scoring portions would then be added together. Initially, the scoring algorithm contained both a real-time and post-song feedback system, but ultimately, as the real-time feature was cut, we only included the post-song feedback in this app.

C. Web Application Implementation

The organization of the web application for this project follows the standard design of a Django web application. Navigation between pages is handled by the Django urls.py and views.py functions, which render generally static HTML templates. The gameplay page and results page both use javascript to render the pitches via line charts created with the Graph.js package.

When the user begins the game experience, they give KaraoKey access to their microphone and start the song. The backing track plays and the user sees the lyrics and target pitch, one chunk at a time. The target pitch is displayed via the aforementioned line chart, and is pink for parts of the chunk that have passed and blue for the remaining parts, showing the user where they currently are in this segment of the song. The portion of the line displayed in each color is updated via AJAX with a period of 20 ms.

After uploading their track to KaraoKey, users are taken to a replay of their singing experience. Again, users see the song chunk by chunk, with lyrics and target pitch displayed. This time, however, user pitch is displayed over top of the target pitch, showing the user how they were performing at this point of the song. The user pitch line is again updated via AJAX with the same period.

Afterwards, users are shown a summary of their performance, including some statistics and featuring still graphs of their best and worst chunks. These are displayed statically and not updated. Along with snapshots of these chunks, qualitative feedback is provided about where the user fell short and how well they did at best.

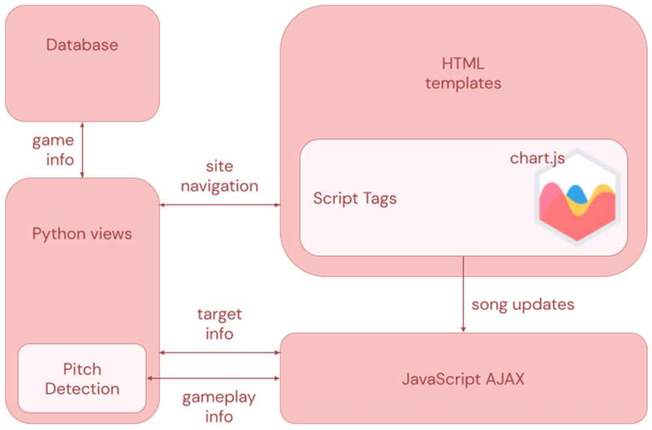


Fig. 6. Block diagram of web application workflow.

VII. Test, Verification and Validation

A. Results for Pitch Detection Subsystem

Pitch Algorithm Test Results

As mentioned above, we determined the pitch detection algorithm to use for Aubio through a series of 2 tests: Kelly singing to "C Major Scale - Slow" and Kelly singing to "C Major Scale - Fast". The results for these tests across all Aubio pitch detection modules can be found in Appendix – Diagram 2 and 3: Pitch Algorithm Test Results.

The SPECACF results produced a non-stable signal with many peaks through the C Major Scale Slow test and thus was ruled out immediately. The SCHMITT algorithm produced a non-stable signal as well with a lot of local peaks around the stable signal on the C Major Scale Slow test, lending to harder filtering of pitches and was ruled out. The MCOMB and FCOMB algorithms produced too smooth of a signal on C Major Scale Fast and thus were ruled out as they didn't seem to pick up distinct pitches as well at a fast pace. When deciding between the YIN and YIN FFT algorithms, it came down to the fact that the YIN FFT algorithm produced the most stable signals once the pitch was reached within a note on the C Major Scale Slow tests as compared to the YIN algorithm. While both had high note onset errors, the YIN FFT algorithm produced the best results overall and so we proceeded forward with it.

Wav File Input Test Results

Upon deciding to pitch track on .wav files instead of real time user audio, we tested Aubio's pitch tracking abilities once again. In these tests, we used the previously chosen pitch tracking algorithm of YIN FFT and utilized the "C Major Scale - Slow" and "C Major Scale - Fast" audio files once again. In total, we tested 6 .wav files: the raw piano note audio file of "C Major Scale - Slow", Kelly humming to "C Major Scale - Slow", Kelly singing note names to "C Major Scale - Slow" (i.e. Do Re Mi etc.), the raw piano note audio file of "C Major Scale - Fast", Kelly humming to "C Major Scale - Fast", and Kelly singing note names to "C Major Scale - Fast". The results for these tests can be seen below.

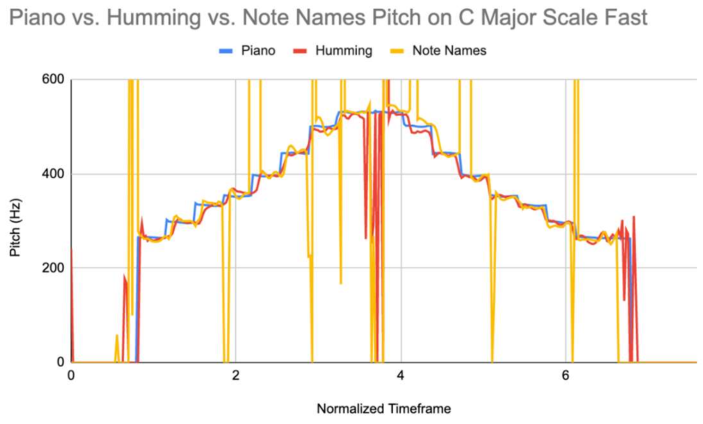


Fig. 7. Test results for Piano vs. Humming vs. Note Nmaes Pitch on a C Major Scale – Fast

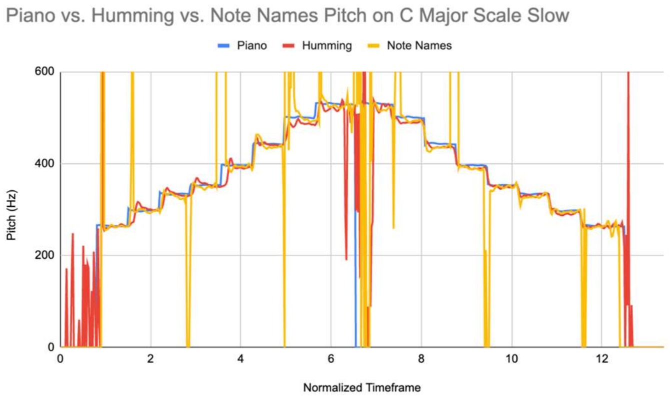


Fig. 8. Test results for Piano vs. Humming vs. Note Nmaes Pitch on a C Major Scale – Slow

From these tests, we determined that the .wav file inputting would not affect the pitch tracking performance as the lines are all quite symmetric. We also became aware of the need for filtering on the signal, especially with note names as the expending of air used to produce the consonants for these note names introduced a lot of peaks in our pitch tracking.

Accuracy Test Results

To test the pitch detection algorithm accuracy and note detection, we did two tests. First, we fed in 108 known pitches into the pitch detection algorithm and then fed the output into the note detection algorithm. The output of the note detection algorithm was compared to the expected result. There were no differences seen here– 108/108 of our input pitches matched the expected result. This was our initial rudimentary test.

In the second series of tests, we fed in a "Happy Birthday" MIDI file into the pitch detection algorithm. The same test format was used here. With this test, we had 99.4% accuracy, where most of the differences were seen in the note onset.

Note that octave errors were not included in these tests, as these errors are not penalized in our scoring algorithm.

B. *Results for Web Application and UI/UX*

The first set of tests we did for the web application was testing the latency of post song feedback. We tested songs of length 30 seconds. We recorded the time it took by using built-in time functions in Python. For song processing, our requirements were under 10 seconds, and we were able to process these songs in 0.51 seconds. For feedback generation, our requirement was under 0.25 seconds, and we were able to perform this generation in 0.05 seconds.

For user feedback, we collected user scores of the various screens of the web application on a scale of 1-5. The screens we chose consisted of the start, login, registration, home, song selection, recording, upload, playback, and feedback pages.

Our average score overall was a 4/5, with the most common score for most screens being a 4/5 as well. We were most interested in the scores of navigation pages (the start, home, and song selection pages) and visualization pages (the recording, playback, and feedback pages). Our navigation pages tended to score a bit higher, slightly over an average of 4, and the visualization pages performed slightly worse. However, we only received one unsatisfactory score - a score below 3 - on any of these pages.

VIII. Project Management

A. *Schedule*

Our schedule consisted of two generally parallel branches - development of pitch detection and of the web application - that progressed in parallel. Once a base web application and pitch detection algorithm were developed, they were integrated and were kept continuously integrated as both aspects progressed. This point was achieved after spring break and the following three weeks consisted of attempts to implement real time feedback. Around mid-April, we pivoted to an implementation that provided live visualization of the user performance during playback, rather than live feedback during the performance itself. This was achieved late April, and for the remainder of the time, we continued developing the web application and user interface, as well as in-depth analysis of user performance and formatting of the feedback. The Gantt chart can be found in the Appendix - Table 1: Schedule.

B. *Team Member Responsibilities*

Kelly's focus was the pitch detection algorithm. She focused on ramping up the Aubio module, testing the PDA in order to ensure it would work for our use case, creating .wav files to be used by our web application, and integrating the pitch detection algorithm into the web application. Additionally, Kelly took the lead on designing and implementing the User Interface as well as researching and purchasing materials.

Anita's focus was the note detection algorithm, as well as the scoring and feedback algorithms. This builds on Kelly's work with the pitch detection algorithm. She focused on designing and developing the scoring and feedback algorithm in a way such that it met the use case requirements of this project. Initially, her responsibilities included real-time feedback and post-song feedback, but as the real-time feedback feature was ultimately removed due to difficulties with integration, her responsibilities shifted to developing the post-song feedback and scoring system.

Anna's focus was building the web application. She focused 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. She also worked on passing required information between portions of the application to be used by the pitch detection.

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 Management*

As expected, the greatest risk we faced as a team was making our pitch detection algorithm work in conjunction with the rest of our application.

Potential issues with the accuracy of our pitch detection algorithm were managed by considering octave errors. By allowing users to perform the song at any octave they want and counting the correct note at any octave as correct, we were able to drastically reduce the amount of pitch detection errors we experienced. Other than that, we did not run into issues with accuracy. We also did not run into any issues with latency.

However, we did run into an issue we did not expect to. Integrating pitch detection to the web application proved more difficult than expected. We experienced issues with file formatting, MIME types, and RIFF headers. As a team, we were able to dedicate three weeks to working on this because of the slack we had given ourselves ahead of time. We tried a variety of methods to get around the issues we faced. For any MIME type, files sent to the server via POST request had malformatted or missing RIFF headers. We attempted to create our own, which resulted in noisy, unreadable files. We attempted to fix MediaStream Recorder settings to match our RIFF header, which did not fix the issue. After reading through documentation and reaching out to others with experience with audio processing and web application development, we ultimately pivoted.

This pivot was disappointing, as we had initially been really set on real-time feedback. However, it was a change that resulted in a functioning, engaging application and would be able to be completed in the schedule provided. We changed it to a replay of the user performance for feedback, which was created using the full recording that the user uploads to the web application themselves.

IX. Ethical issues

KaraoKey is a casual karaoke-inspired web application targeted towards reducing anxiety behind recreational singing. We don't foresee any large issues with our app in the global, environmental, and economic context, as our app mainly addresses matters in the social context. We are not at risk of exacerbating public safety issues either.

When considering the songs that would be included in KaraoKey, we kept cultural and social impact in mind. If we were to include songs that are racist, sexist, etc., we have the potential to advertise harmful content that could lead to negative social consequences. We reviewed our songs for this type of harmful content before including it into the app.

As with most apps, they have the ability to distract users. For example, if a user is using the web-app while driving, they could easily be distracted when reading the visual feedback. This can lead to increased car accidents and injuries, which is a horrible social consequence. In the future, we can potentially enable location tracking and set time limits to make sure the user is limited to using the app under controlled locations and durations.

Thinking more about the specific users of our application, the users who would be most adversely affected by this app are those who sing outside our app's accepted voice range. KaraoKey has the potential of incorrectly critiquing users who aren't in the "typical" voice range of 85-1100 Hz. These special casual singers may become demotivated with this incorrect feedback, even though they may be singing on tune. Unfortunately, we have to limit our "accepted" frequencies to a specified range to increase the accuracy of the feedback given to the "typical" population of singers.

X. 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 focused on the karaoke aspect of our project, just with the addition of feedback, this is one of the closest alternatives we considered. 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 >$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 users are, 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. However, Rock Band provides real time feedback, while we provide feedback afterwards, in the form of a replay of user performance.

XI. Summary

Our final goal for this project was 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 provide engaging visual feedback via playback of the user performance in comparison to target pitch.

Our aim was 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 did not aim to provide vocal coaching or feedback on advanced techniques, and therefore KaraoKey was not designed to analyze scatting or ad-libbing. Doing so was 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 was to keep our system accessible to our goal user. Our materials included a headset microphone costing ~$230, but we developed a web application that is able to perform with cheaper materials as well if it is used in a non-crowded, generally quiet environment.

To future student groups that may want to address this application, we recommend performing thorough research on "black box" modules before using them. While we thought that using the aubio module for pitch detection was going to greatly simplify things, working with a black box during integration proved extremely difficult. This ultimately led to us axing the "real-time" component of our app.

Glossary of Acronyms

AJAX	Asynchronous JavaScript And XML
FFT	Fast Fourier Transform

References

[1] S. Siddiq, "Data-Driven Granular Synthesis," *www.aes.org*, May 11, 2017. https://www.aes.org/e-lib/browse.cfm?elib=18619
[2] J. Nielsen, "Response Time Limits: Article by Jakob Nielsen," *Nielsen Norman Group*, 2019. https://www.nngroup.com/articles/response-times-3-important-limits/
[3] C. Harte, "Towards automatic extraction of harmony information from music signals.", 2006
[4] "Man page of AUBIOPITCH," *aubio.org*. https://aubio.org/manpages/latest/aubiopitch.1.html (accessed Mar. 03, 2023).
[5] "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).

XII. Appendix

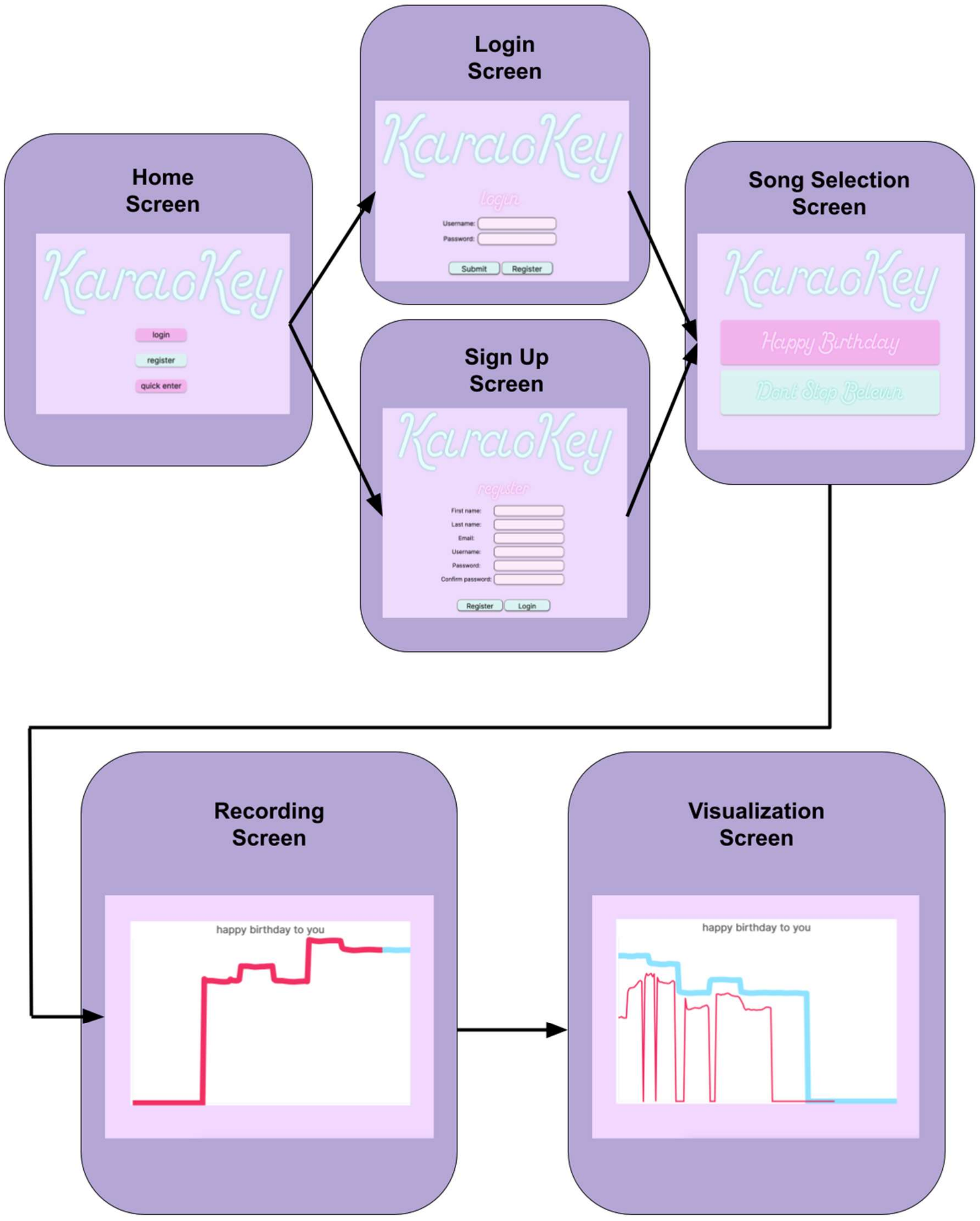


Diagram 1: Web Application User Interface Design

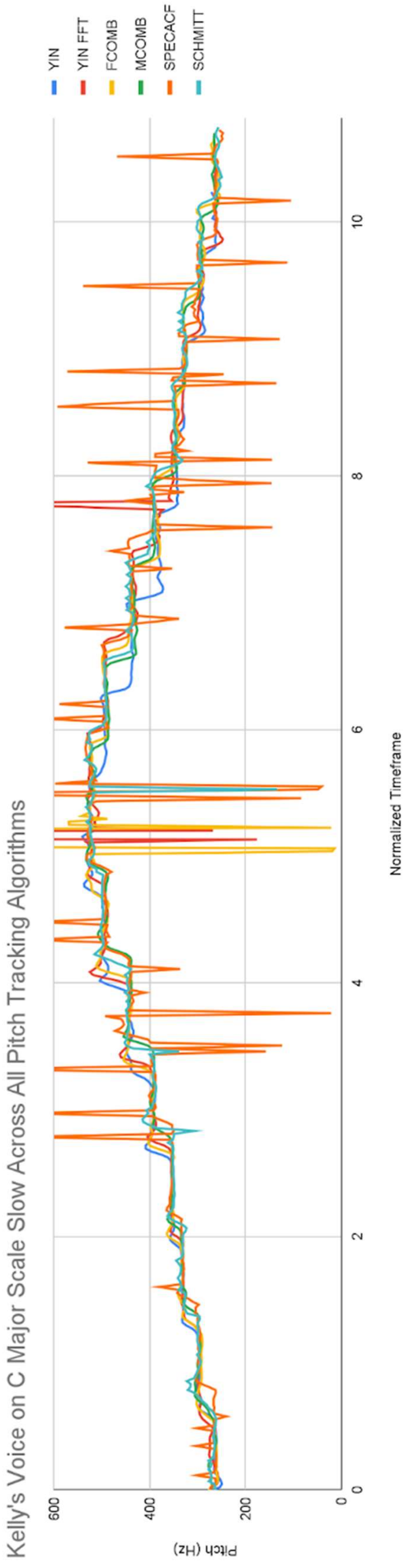


Diagram 2: *Pitch Algorithm Test Results – C Major Slow*

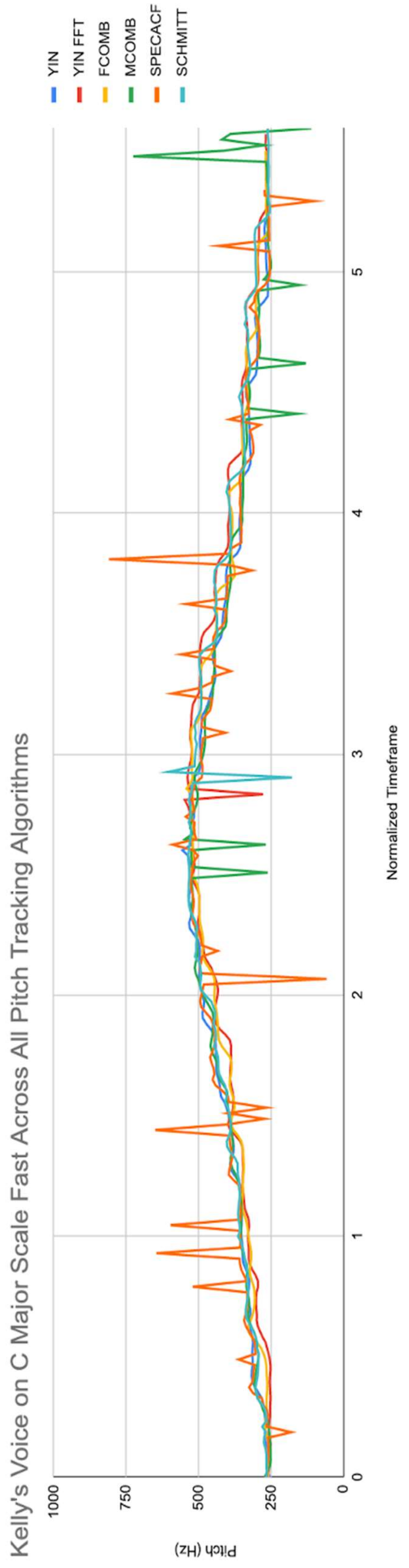


Diagram 3: *Pitch Algorithm Test Results – C Major Fast*

schedule															
	tasks	phase 1 (groundwork)					phase 2 (in depth)				phase 3 (combine)				
category	specific	01/30-02/05	02/06-02/12	02/13-02/19	02/20-02/26	02/27-03/05	03/06-03/12	03/13-03/19	03/20-03/26	03/27-04/02	04/03-04/09	04/10-04/16	04/17-04/23	04/24-04/30	05/01-05/07
pitch detection	research algs	🟥													
	demo with a hardcoded scale		🟫	🟫	🟫										
	implement audio tracking in aubio	🟪	🟪												
	find/craft midi files				🟪										
	test out best pitch tracking algorithm				🟪										
	test out pitch tracking on wav file							🟪							
	feedback generation			🟫	🟫	🟫		🟫	🟫	🟫					
	integrate with feedback algo									🟫	🟫	🟫	🟫		
	slack										🟥	🟥	🟥	🟥	🟥
web app	figma ui	🟪	🟪	🟪		🟪									
	basic login and page navigation	🟧	🟧	🟧											
	research + try graphics packages			🟧	🟧	🟧									
	song can be selected + played			🟧	🟧										
	record while song plays				🟧	🟧									
	display target graphics while song plays					🟧		🟧	🟧	🟧	🟧				
	integrate with pitch detection								🟪	🟪	🟪	🟪	🟪		
	add feedback during/after song									🟫	🟫	🟫	🟫	🟫	
	slack												🟥	🟥	🟥
deliverables	create blog	🟧 02/04													
	proposal presentation	🟥 02/05													
	design presentation (FIXME)			🟥 02/19											
	design report (FIXME)				🟥	🟥 03/03									
	Interim Demo									🟥	🟥 04/03				
	final presentation									🟥	🟥	🟥	🟥 04/23		
	final report										🟥	🟥	🟥	🟥	🟥
other	decide on + order headset + interface	🟪	🟪	🟪											

Key	
🟫	Anita
🟪	Kelly
🟧	Anna
🟥	Joint Effort

Table 1: Schedule