CSCI 370 Final Report

SoundByte

Delaney Lim
Jack Durfee
Trevor Sheehy
Bennett Gehr

Revised June 13th, 2024

CSCI 370 Summer 2024

Mr. Tree Lindemann-Michael

Table 1: Revision history

Revision	Date	Comments
New	5-16-2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done
Rev – 2	6-7-24	Modifications to sections I. Introduction and IV. Risks Addition of sections VII. Software Test and Quality, IX. Project Completion Status, X. Future Work, XI. Lessons Learned, XII. Acknowledgments, and Team Profile
Rev – 3	6-10-24	Modifications to sections I. Introduction, X. Future Work, XI. Lessons Learned, and Team Profile. Addition of VIII. Project Ethical Considerations
Rev – 4	6-12-24	Modifications to VI. System Architecture
Rev – 5	6-13-24	Minor revisions made based off review swap

Table of Contents

I. Introduction

This student-proposed project explores the intersection of music and technology, focusing on algorithms and computing for processing audio pitches. The goal of this project is to develop an intuitive and user-friendly software application that allows vocalists to visualize their pitch in real time. The software will enable users to monitor their vocal performance, identify pitch deviations, and make necessary adjustments to achieve greater pitch accuracy. This innovative tool aims to revolutionize pitch training and empower vocalists of all skill levels.

This is a student-proposed project and thus the client for this project is Mines graduate student, Delaney Lim. Delaney has been a musician and vocalist since a young age, and since studying her bachelor's in computer science progressively has been finding ways to intersect music with coding. During her musical studies and teaching free vocals lessons, it came to light how many people enjoy singing, want to get better, but do not have the resources to improve and grow their vocals. The purpose of this project was a software that aims to provide vocalists with a real-time pitch visualization tool akin to a guitar tuner. This software will enable vocalists to enhance their pitch accuracy, improve their overall performance, and develop their vocal skills.

This project will not be based on an existing code base and will be created by the group from scratch. The final product will be intended to be used by vocalists looking to monitor their vocal performance and improve their pitch accuracy. Once the project is delivered to the client, future maintenance or modifications will be performed by the client.

II. Functional Requirements

There are two components to this project required to meet the client's needs. There is the software side that will process input audio and compare it to a baseline, and there is the GUI side that will display the results in a clean format.

Software

- Reads in live vocal audio directly from microphone
    - iPhone 15 Pro microphone, running iOS 17.0 or greater
    - Intended for one singer at a time
- Processes audio and compares it according to pitch theory
    - Extracts frequency from audio sample
    - Analyzes extracted frequency from audio sample to produce what pitch it is closest to
    - Uses the pitch to determine the musical note it’s on
    - Compares the actual pitch sung to the pitch of the note it is nearest to
    - Displays the pitch of the input on a scale of standard music notes

GUI

- Displays a visual “graph” which shows live pitch compared to notes, see *Figure 1* for initial concept and *Figure 3* for the final implementation
    - Displays the current pitch user is singing with a dot on the right-hand side of the screen
    - Displays recent history of the pitches a user has sung with a line
        - Exact duration of recent history will be determined by the programmers after testing
    - Displays horizontal lines each indicating a note on the melodic scale
        - Exact number of horizontal lines/ notes will be determined by the programmers after testing
        - Represents the frequency for the perfect pitch for each note
    - Displays the duration of recording in minutes, seconds and milliseconds at the top of the screen
        - Formatted as such: 00:00.0
    - Displays a play/ pause and stop button for the user to control recording
- Is designed for mobile devices running iOS
    - Specifically runs on iPhone 15 Pro, running iOS 17.0 or greater

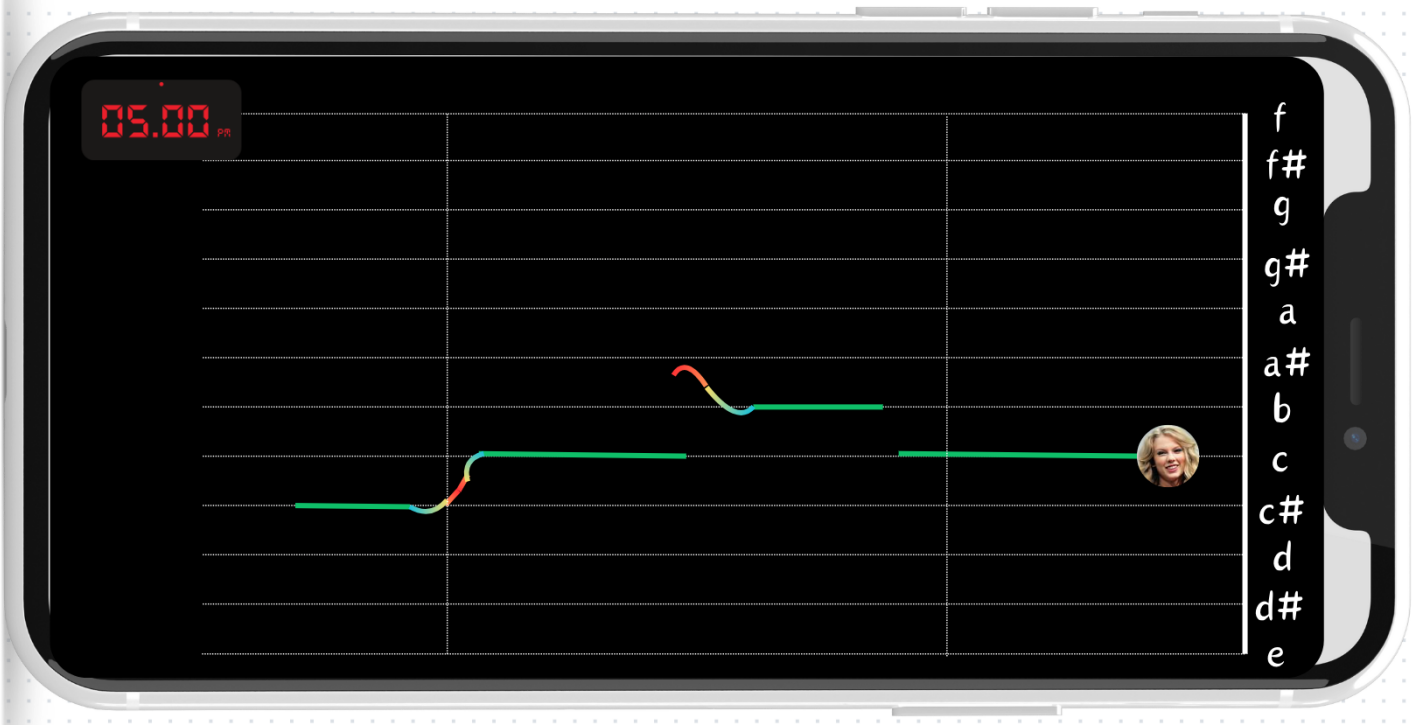

Figure 1: Initial UI Design for our App

III. Non-Functional Requirements

There are a few non-functional requirements we would like to address:

- Design an App Logo for the cover of the application. This is not required for the application to work, but it would allow our product to be more marketable.
- Download the app demo on iPhone 15 Pro, running IOS 17.0 or greater. This is not necessary for the code to work, but is necessary for furthest extent of testing, and user acceptance.
- Utilizing Apples TestFlight Service to reach a group of test users with versions of our product. This isn't necessary for completing this project but **is necessary** for a successful product.
- Building the application to have support for other devices other than the iPhone 15 Pro, so we can expand our user group. This is a helpful functionality that could allow our app to be run on many different iPhone devices and thus increasing the accessibility of our product.

IV. Risks

Team Skill Risks

- Learning Curve for Swift and SwiftUI Programming:
    - *Likelihood:* Very Likely
        - None of the group members has any experience developing code using Swift or SwiftUI, so it could be necessary to learn these skills to develop an iOS app. Our groups inexperience with the language is likely to cause a higher frequency of issues and longer time spent fixing them.
    - *Impact:* Major
        - Failure to learn to use SwiftUI, could make it impossible to launch the project as an iOS app.
    - *Risk Mitigation Plan:*
        - Allocate additional time in the schedule for learning and experimentation with SwiftUI, and for the solving of a high frequency of issues, especially early on in development. The use of pair programming can accelerate skill acquisition and reduce code errors.

- Integration of Python with iOS Development:
    - *Likelihood:* Likely
        - None of the group members has any experience with integrating parts of a project across different languages. Our group's inexperience with this part of development is likely to cause us to require extra time to implement this aspect of the project.
    - *Impact:* Moderate
        - This is expected to affect our group's development time. Failure to integrate the Python back end can be resolved by using Swift for the projects back end as well.
    - *Risk Mitigation Plan:*
        - Allocate dedicated time for researching and experimenting with Python integration.

Technical Development Risks

- Short 5-Week Development Time:
    - *Likelihood:* Very Likely
        - The short development time of this project is likely to affect the entire development process, and the project's development time may be further reduced by possible team member absences due to sickness, holidays, etc.
    - *Impact:* Major
        - The time limitations could greatly impact the project's scope.
    - *Risk Mitigation Plan:*
        - Prioritize features based on criticality and feasibility. Adopt Agile methodologies like test driven development and frequent code reviews to keep our code clean. Adopt Scrum methodologies to constantly be iteratively developing and making mini deliverables. Use rapid prototyping to validate ideas early and minimize rework.
- Difficulty of Apple iOS Development and Release:
    - *Likelihood:* Likely
        - Developing this project as an iOS app requires Apple products and services.
    - *Impact:* Major
        - Developing this project as an iOS app requires Apple products and services, which could impact our group's development process, and makes our group dependent on a third party for parts of the project development and the final release of our app.
    - *Risk Mitigation Plan:*
        - Utilize Apple's development resources, including documentation, forums, and sample code. Plan for potential delays in the App Store review process by submitting well before deadlines.

By addressing these risks proactively and implementing mitigation strategies, the project can navigate both technical challenges and skill-related obstacles effectively, increasing the likelihood of successful delivery within the constrained timeframe.

V. Definition of Done

Our project will be done once it can process live vocal audio input from the user, process and compare it to the closest pitch and display results in a way that’s intuitive for the user to understand. This product must be able to visualize a user’s pitch on a graph of standard musical notes. It also should allow a user to “record” or visualize the whole length of their singing session over the span of time, versus just comparing one note at a time. It must have a user-friendly GUI that requires less than 10 minutes to learn to use properly.

Before accepting our product, the client will test it with audio samples we provide and a live demonstration of it working.

This product will be delivered at the end of Field Session 2024, on June 14th. The code will be delivered through the GitHub repository. All documentation will also be delivered to the client throughout the project in Microsoft Teams.

VI. System Architecture

The architecture of our system is spread out over several SwiftUI views, that contain various functionalities that make up our app. To briefly explain, a SwiftUI view is simply a panel that you can place elements on, either using a coordinate space, or relative spacing using vertical and horizontal stacks. Reference Fi*gure 2* below for a visual diagram of the components below.

Main View

Our main view is composed of many individual elements, brought together to create our user interface. Its main purpose is to direct each sub view, giving it the necessary information it needs to display correctly. See *Figure 3* for the final visual of the product Main View.

Timer

The timer element allows a user to keep track of how long they've been singing for. The timer turns on when you press the play button and pauses when you press the pause button. It resets the timer back to 00:00.0 when you hit "stop" or "reset" button. The timer is also an internal clock for the code itself that fires on a set interval and updates necessary values such as the history array (which contains all the previous pitch values).

Buttons

The buttons on our UI give the user direct control over the functions of the app. The button view consists of the play, pause, reset/stop, and settings buttons. Each of these perform the corresponding action associated with their name. Thus, the play button starts recording audio and processing, the pause button pauses recording, the reset button starts the recording session over, and the settings button takes you to the settings page.

Settings View

This view allows users to change and save user preferences of the main view. You can change your key signature, the clef you're singing in, and the mode of the key (major or minor). You can also select the audio device you would like to sing from. When setting these preferences, we used the built in Apple User Defaults class that allows simple data types to be saved between uses of the application. When setting the key, it can make a key object by creating an encodable and decodable data type.

Calculate Cents Off

Our calculate cent off function determines the number of musical cents off a user is singing at from the pitch of the note that they are singing closest to. This allows us to fine tune the location of pitch indicator as well as give a color to the line trailing out of it.

History View

The history view displays the user's entire performance. It draws a History Line inside of a scrollable view with all of the history data so that the user can view their pitch accuracy over the course of the whole time they've been recording.

Calculate Y Coordinates

Our calculate y coordinates function maps a given frequency to the staff that is displayed in the Main View. It calculates the physical distance between each note and the frequency associated with that note to produce a y coordinate that can then be passed to the Pitch Indicator.

Tuner Conductor (Audio Kit)

The Tuner Conductor Class is modeled after the Audio Kit tuner (open-source online library). The tuner conductor is responsible for creating an AV Foundation Audio Engine object, the device object, and creating a "pitch tap" which allows the Engine object to read in audio input for processing. The "pitch tap" specifically reads in audio input as frequency data and amplitude data. We custom set a gain on the microphone to limit background noise then, we then process this data in an asynchronous updating function that sets a data structure to contain the frequency and amplitude. We also implement a smoothing mathematical function to make the points easier to graph.

Staff

The staff view draws a musical staff in the style of a piece of sheet music. It contains the key signature as well as the clef that the user is singing in. This is the largest view in the main view, to give the user better precision when they are singing in between notes.

Pitch Indicator

The pitch indicator utilized a graphic of a musical note to represent the user's current pitch as they were singing. It also doubles as a "cursor" when a user goes to scroll back in their singing history.

History Line

The history line simply is a visual to show the past pitches the user has sung. It also colors the line to show how far off the user is singing from a note. The further away a user is singing from a note in the staff view, the redder the line becomes, but as the user sings closer to notes in the key, the greener it becomes.

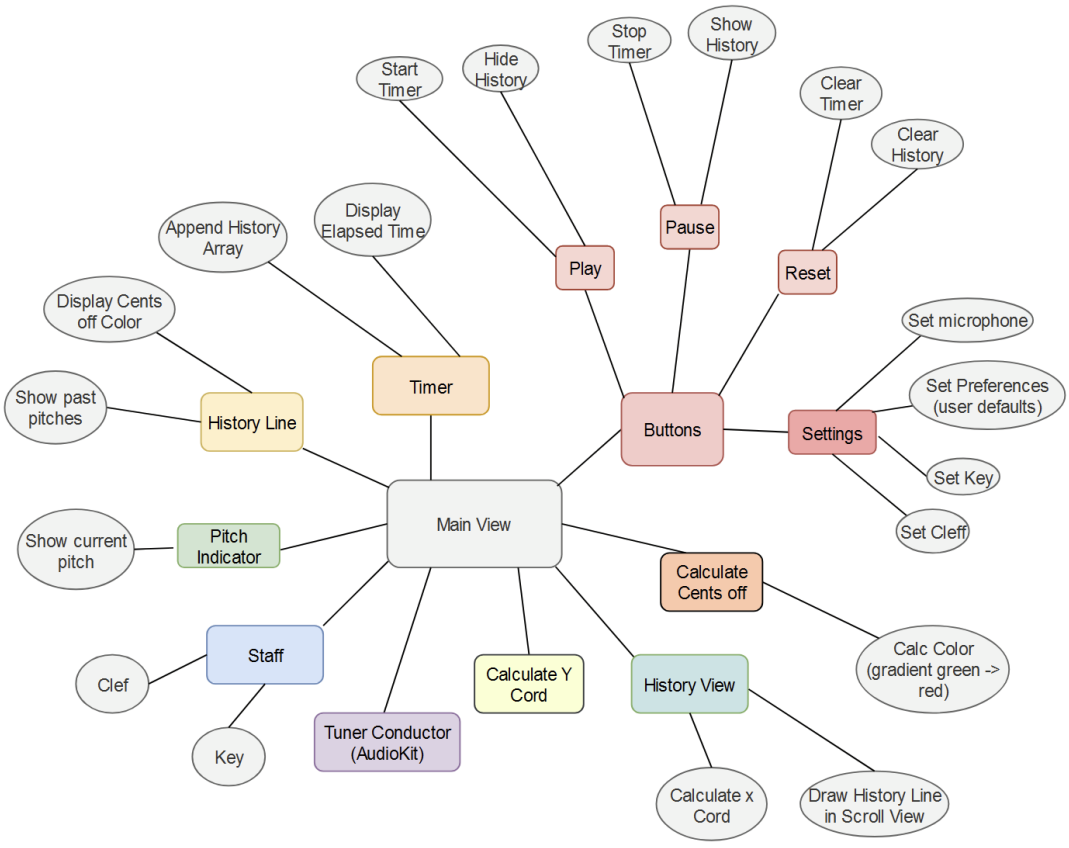

Figure 2: System Architecture Diagram

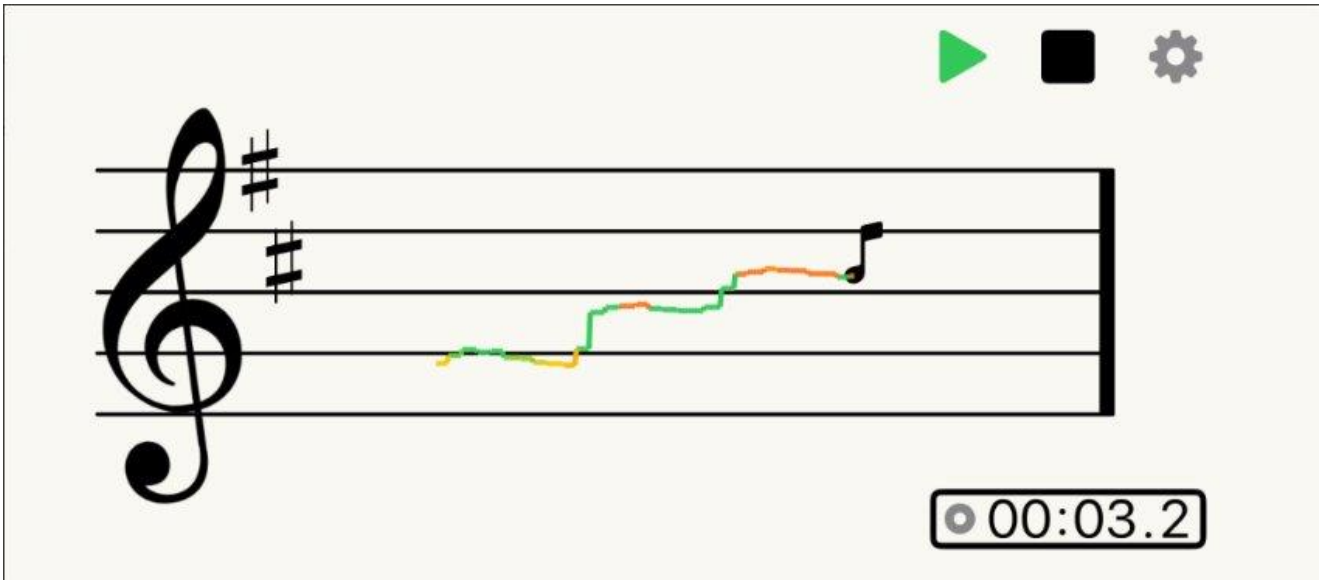

Figure 3: Final UI Design

VII. Software Test and Quality

Our group used various quality checks to ensure better quality software. These included:

- Unit testing:
  - Our group implemented automated unit testing, utilizing the built in testing frameworks for SwiftUI and Python. Unit tests contribute to identifying unexpected behavior in code, and to ensuring code stability across the project's development, ensuring that the code continues to function as expected even as changes are made.
    - We tested our Key Generator class to ensure that when it is given a specific number of sharps or flats, the correct notes in the key are generated.
    - We tested our Frequency Mapper class to ensure that when it is given a list of frequencies and physical spacing between each note on the screen, it correctly maps each frequency to the expected position.
    - We also tested several error handling exceptions thrown by the above two classes to ensure that invalid inputs to these classes were handled correctly.
- User interface testing (automated or otherwise):
  - Included among general unit testing, our group utilized the testing framework for SwiftUI to create automated unit tests for the UI elements of the project. Our group has also done extensive manual testing of the UI, by launching the application on a group member's iPhone at various intermediate stages of the development process. Both the automated and manual tests contribute to verifying that user interactions produce the expected outcomes. See *Figure 4* for the correct outcomes for each action within the UI.
- Code reviews:
  - During the project development our group has conducted regular code reviews. These review sessions included having one of the group members who worked on a code segment display the code to the other group members and provide a thorough explanation of what their code is doing and why certain decisions were made. Code reviews are used to ensure adherence to coding standards and best practices, to identify potential issues early in the development process, and to aid in knowledge sharing among team members.
- User acceptance testing:

- Partway through the project development, at the end of week two, our group made the app available as a demo beta version and had a small test group of users interact with the app and provide feedback on any issues or user preferences.

- Frequent client meetings:
  - Our group participated in daily check-ins with the client to ensure that the project direction matches with the client's desired product, and that the project development maintains the expected schedule.

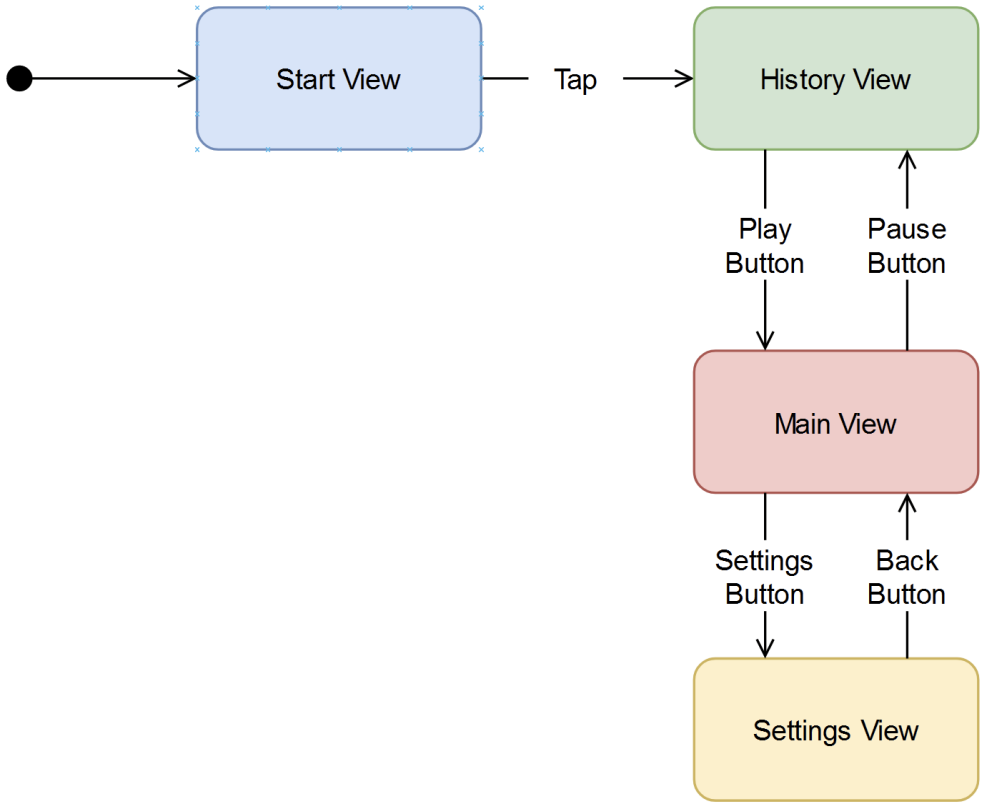

Figure 4: View Navigation Diagram

VIII. Project Ethical Considerations

Our project had several pertinent ethical considerations. These included:

- Privacy
  - Violating ethical principles ACM 1.6 and IEEE 3.13 are of particular concern to this project because we use the devices microphone to record the user. To avoid ethical complications, we obtain explicit consent to record the user upon opening the app for the first time, as well as only recording when the start button is pressed.
- Accessibility
  - The ethical principle in IEE 1.07 states that as software engineers we must consider factors that limit our software accessibility. Our project being an iOS exclusive app does limit accessibility for some potential users, however this is mitigated by the existence of alternatives and the low impact of our app.
- Quality
  - The IEE ethical principle 3.10 calls for software engineers to strive for high quality, among other things. This is particularly important to our product because of its nature as an informative tool. To avoid

misleading users by giving them false information it is paramount that we ensure the accuracy of our program's functionality.

IX. Project Completion Status

The overarching goal of this project is to create a program which takes in audio input and provides a visualization of the user's pitch. This can be broken into the steps of reading live audio input, processing the audio, and visualizing the result. Within the project's short development time, our group has created a minimal viable product that performs these tasks in addition to meeting all functional requirements. Specifically, the project features include:

- Reading in live vocal audio from the device microphone, for an iPhone 15 Pro running iOS 17.0 or greater, intended for one vocalist at a time.
- Processing audio input, including extracting input frequency and then comparing it to the frequencies of standard musical notes to determine the pitch and closet note of the input.
- Visualize the input on a musical staff in real time. Indicate the current input pitch, its' accuracy, and a trailing line providing immediate history of the pitches that have been sung.

The original project requirements also detailed some of the desired features of a potential user interface. However, with approval from the client our group has changed parts of the design from the initial concept, while still satisfying the original overarching goal. Specifically, the project UI includes:

- Visual "graph" that shows live pitch compared to notes
    - Displays the current pitch the user is singing with a note icon, rather than a dot.
    - Displays recent history of the pitches a user has sung with a line
    - Displays horizontal lines formatted to create the look of a musical staff that is standard in sheet music, including two ledger lines above and below which fade in and out as needed, rather than the original design of having a full horizontal line for every note.
    - Displays the recording's duration in minutes, seconds, and milliseconds at the top of the screen, formatted as 00:00:0, rather than 00:00:00.
    - Displays a play / pause and stop button for the user to control recording
    - Displays a gear icon used to access user settings.
- Mobile based for iOS, the project runs on an iPhone 15 Pro with iOS 17.0 or greater.

All these implemented features have been tested to ensure that they perform as expected, on the target device of an iPhone 15 Pro running iOS 17.0 or greater. The application has also been run and tested on an iPhone 10 with iOS 17.4. Any device without iOS 17.0 or greater would be expected to crash as our application uses features that are not supported in older iOS versions, as well as Apple's standards now require all Apps released to be developed in iOS 17.0 or greater.

X. Future Work

Any further additions to this project could require knowledge of coding in SwiftUI as that is the primary language the project has been developed in. Also, our project is designed as an iOS application so future developers will be required to continue development using Apple's integrated development environment Xcode, as it is required for iOS development. This also entails the requirement of having devices that run on macOS, as Xcode is only designed to work within a Mac environment.

Some ideas for other features beyond the functional requirements that could be added in the future include:

- Having the scale change dynamically, so that a wider range of pitches can be graphed, rather than the current implementation of limiting the display to a chosen musical clef. It is estimated to take one to two weeks to implement.

- Adding sing along elements to the graph visualization. Including things like a “ghost” line or another form of targets for a user to try to match the target pitch, and the addition of lyrics to sing along with. This could also be further extended to allowing users to sing along with a chosen song, either one uploaded to the app or from a library of songs already within the app, and this would require getting licensing to use some set of songs. The time to implement the full idea of being able to sing along with a full song would be estimated to take about three weeks.
- Convert audio input into a sheet music representation of the vocalization. This is estimated to take three weeks.
- Accessibility options, potentially including things like adjusting the size of text, colorblind modes, etc. These features could be added within one week.

XI. Lessons Learned

Through our project's development, our group has learned serval lessons about thorough project planning. The aspects of project development that we learned more about are:

- Timelines and Scope
    - Our group underestimated the time required for certain features, leading to tight deadlines and rushed implementations. As we progressed, we strove to uphold the importance of thorough project planning and a clear definition of the project's scope. This allowed us to prioritize tasks, allocate resources, and avoid wasting time, which allowed us to stay on pace to complete the project within the short 5-week period.
- User testing early and frequently
    - Based on advice from our advisor we started user testing as early in the stages of the development cycle as possible. We saw firsthand the benefits of feedback, as it led us to redesigning our UI to being more intuitive. Regular user testing sessions provided invaluable insights into usability issues and user preferences, enabling us to iterate rapidly and deliver a more intuitive and user-friendly experience.
- Understanding and accounting for hardware limitations
    - The development of our iOS application requires the use of Apple’s IDE Xcode which is only available on MacBooks. This was an issue because not all the members of our group have access to a MacBook, and thus were limited in their ability to contribute to the project code. Eventually we got access to more devices covering most of the group, however this issue could have been mitigated earlier had we better planned and understood the requirements of each step of the project's development.
- Learning new tools quickly
    - Given the dynamic nature of technology, we frequently encountered unfamiliar concepts and tools throughout the development process. To address this, we adopted a proactive approach to learning, leveraging online resources, documentation, and peer support. By embracing a culture of continuous learning, we were able to quickly adapt to new challenges and integrate emerging technologies into our project.
- Knowing when to ask for help
    - At times, we encountered complex technical challenges that exceeded our individual expertise. Instead of struggling alone, we learned to recognize when to seek assistance from peers, mentors, or online communities. Collaboration not only facilitated problem-solving but also fostered a sense of camaraderie within the team, enhancing our collective learning and productivity.

By reflecting on these lessons learned, we have gained valuable insights that will inform our future projects and contribute to our ongoing growth and development as software engineers.

XII. Acknowledgments

We'd like to thank our client Delaney Lim for proposing this project and being available to meet every day. We'd like to thank our instructors for putting this course together. Lastly, we'd like to thank our advisor Tree Michael for giving us his guidance and time.

XIII. Team Profile

Name - Delaney Lim

Year - Graduate Student

Software/Engineering Discipline - Computer Science

Hometown - Littleton, Colorado

Work Experience - Adaptive Innovations Intern, Law Intern

Sports/Activities/Interests - Music, Sailing, Dance

Name - Bennett Gehr

Year - Junior

Software/Engineering Discipline - Computer Science

Hometown - Louisville, Colorado

Work Experience - Math tutor at Red Rocks Community College

Sports/Activities/Interests - Video games, Anime.

Name - Trevor Sheehy

Year - Junior

Software/Engineering Discipline - Computer Engineering

Hometown - Austin

Work Experience - Whataburger

Sports/Activities/Interests - Music, Board Games

Name – Jack Durfee

Year - Junior

Software/Engineering Discipline – Computer Science

Hometown – Golden, CO

Work Experience – Senior Manager Chick-fil-A

Sports/Activities/Interests - Piano, Reading

References

[1] "General Principles The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems." Available: https://standards.ieee.org/wp-content/uploads/import/documents/other/ead1e_general_principles.pdf

[2] ACM, "ACM Code of Ethics and Professional Conduct," *www.acm.org*. https://www.acm.org/code-of-ethics#:~:text=The%20values%20of%20equality%2C%20tolerance

[3] "Home - AudioKit," *www.audiokit.io*. https://www.audiokit.io/

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Pitch	*The quality of a sound governed by the rate of vibrations producing it; the degree of highness or lowness of a tone.*
Note	*A notation representing the pitch and duration of a musical sound*
Scale of Standard Notes	*Any set of musical notes ordered by fundamental frequency or pitch*
GUI	*Graphical User Interface*
iOS	*iPhone Operating System*
Melodic Scale	*The combined scale that proceeds as natural major ascending and as Aeolian dominant descending*
Fourier Transform	*The Fourier transform (FT) is an integral transform that takes a function as input and outputs another function that describes the extent to which various frequencies are present in the original function.*
Staff	*A set of five parallel lines and the spaces between them, on which notes are placed to indicate their pitch.*
Clef	*Any of several symbols placed at the left-hand end of a staff, indicating the pitch of the notes written on it.*
Key	*A group of notes based on a particular note and comprising a scale, regarded as forming the tonal basis of a piece or passage of music.*
Key Signature	*A set of sharp, flat, or rarely, natural symbols placed on the staff at the beginning of a section of music.*