University of Central Florida

PracticePal: An iPhone Music Practicing App

Allison Turner

MUS 4900: Music BA Capstone Project

20 November 2023

Introduction

For my BA Capstone Project, I created an iPhone music practicing application that offers an intuitive and accessible user interface for musicians of all ages. From my experience as a music major at UCF, I learned the pivotal role that a valuable practice session plays in developing musical skills. My studies in computer science have given me the problem-solving skills necessary to develop an application. With my app, I hope to provide fellow musicians with a tool that embodies the essence of purposeful practice, helping them achieve their goals.

When designing my project, I wanted to create an app that combines all the tools musicians use for practice. My app includes a calendar with notes, a tuning drone, a tuner, a metronome, and a recorder. The calendar helps musicians organize their practice sessions, ensuring they allocate time for various aspects of their training. The tuning drone and tuner are crucial for achieving accurate pitch, helping musicians maintain the correct tuning for their instruments. A metronome enforces rhythm and timing, aiding in the development of precision and coordination. The recorder allows musicians to record their practice sessions, facilitating self-assessment and improvement tracking.

This document outlines my app's design and features. Each feature includes a comprehensive analysis of the User Experience/User Interface as well as the technical details and logic of the code. Furthermore, I will share my personal experience and user feedback to provide valuable insights into the overall performance and user satisfaction of my application.

Technical Background

I wrote the code for my app in Swift using Xcode on my Mac computer. Swift is a modern, high-level programming language designed and developed by Apple. Swift is used to create software applications for Apple devices. Xcode is Apple's integrated development environment (IDE) for building applications for Apple devices. It combines a powerful code editor, visual interface builder, debugger, and simulator for testing. Xcode also integrates with version control systems, offers extensive documentation, and includes performance analysis tools. My code is organized into various source files, each containing logic for the different features of the app. Xcode manages the project configuration, including deployment targets, build settings, and dependencies. When the project is built, the Swift compiler converts the source code into LLVM Intermediate Representation (IR) - a form of code that serves as a common language between different programming languages. This representation allows for advanced optimization techniques. The LLVM optimizer then refines the IR for efficiency, enhancing the performance of the app. Next, the backend generates machine code tailored to the target device's architecture, ensuring optimal execution. The linker combines app code with external libraries create an "executable," a file that contains machine code instructions that a computer's processor can directly execute. Finally, the app is deployed to a device where it runs.

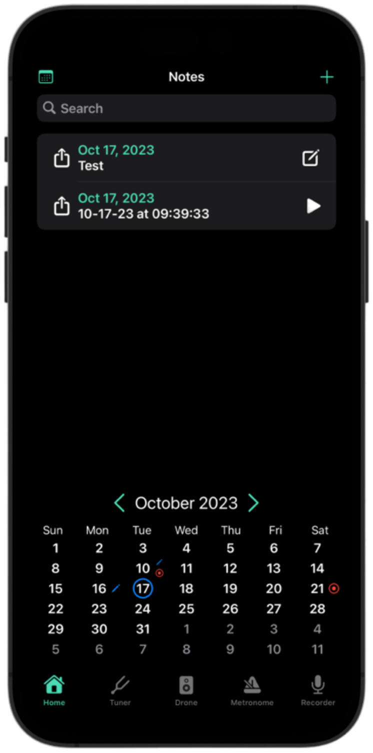
Design

I wanted my app's icon to be cute, fun, and motivate students to practice! Cute and appealing imagery tends to evoke positive emotions, which can create a more enjoyable and engaging learning environment. When students associate their practice sessions with a cheerful image, it may enhance their overall attitude towards practicing. Additionally, a memorable logo like a cute animal can create a strong brand identity for my app. It may also make the application more memorable and distinctive.

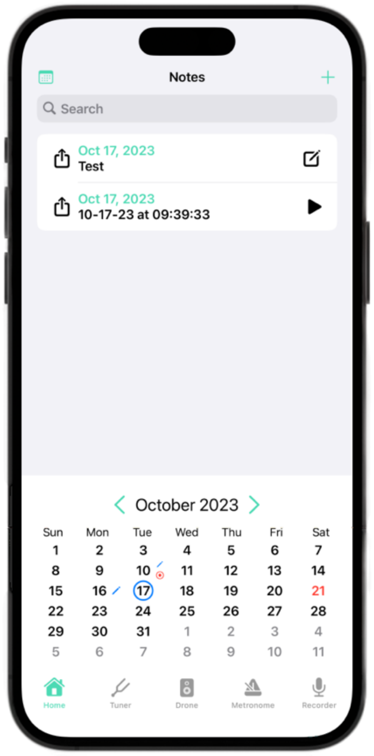
App Icon

I named my app "Practice Pal." This name is easy to remember and encapsulates the app's core mission. The term "Practice" directly addresses the app's primary function, which is to assist users in their musical practice sessions. Meanwhile, "Pal" conveys a sense of camaraderie and support, suggesting that the app is a friendly companion in the user's musical journey. Additionally, "Practice Pal" is a versatile term, applicable to a wide range of musical instruments and styles. The name's catchiness ensures that it is likely to stick in users' minds, increasing the likelihood of both retention and recommendation.

The main interface of the app consists of five navigation tabs. Each tab has an icon to represent a different feature of the app. There is a tab for the calendar, a tab for the tuner, a tab for the tuning drone, a tab for the metronome, and a tab for the recorder. The tabs work as a directory that guides you through the different parts of the app. The app functions in both light mode and dark mode.



Dark Mode



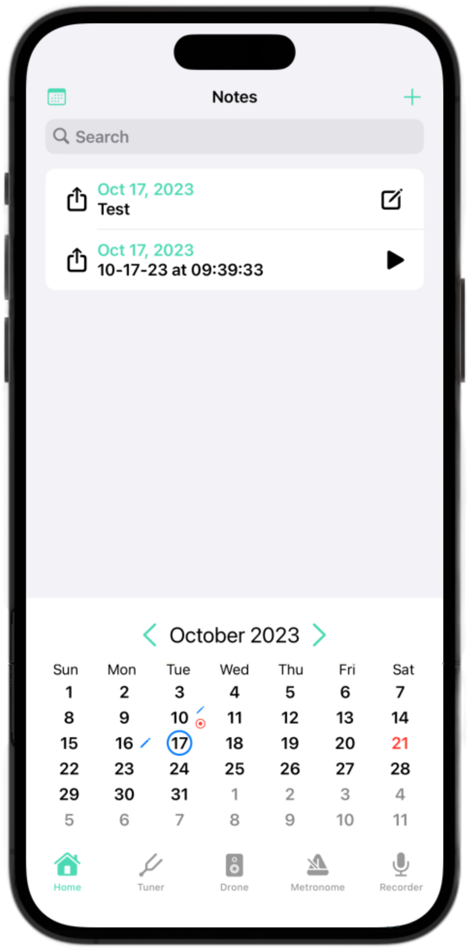
Light Mode

App Design Feedback and Personal Evaluation

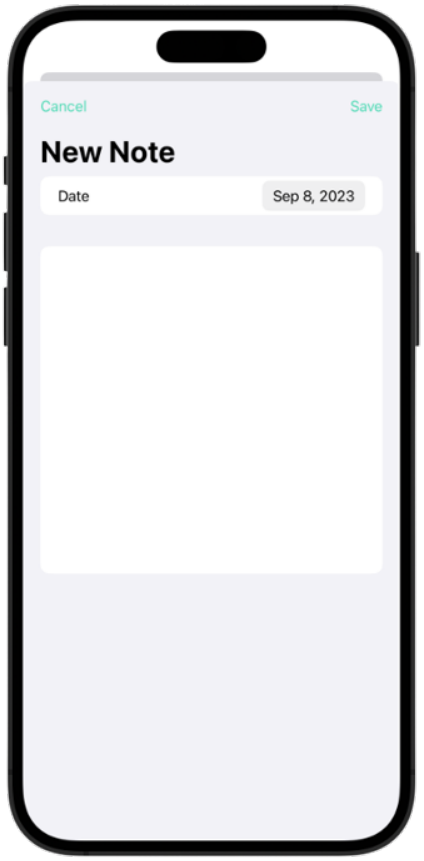
I love the easily navigable user interface, characterized by intuitive icons on each navigation tab. The app's icon makes me smile every time I see it. Users thought that the name "Practice Pal" was "adorable." Users appreciated how my application "integrates numerous features that they already use in practice every day." Many musicians utilize tools like a recorder, tuner, drone, metronome, and notebook while practicing and my app allowed users to use all those tools at once.

Calendar
The Calendar displays a list of notes and recordings and allows users to filter and search through them. The user interface includes options to add new notes, delete existing ones, and export notes as text files. Additionally, there is a calendar that can be toggled to show or hide.

The notes and recordings are stored separately, and they can be filtered based on a search and selected dates. When a note is exported as text, it creates a temporary file, populates it with the note's content, and presents a share sheet for the user to export the file. I created the calendar using a grid. The grid displays the days of the week at the top and the days of the month below. Each day is a button that can be tapped to select a date. The appearance of the days depends on whether they belong to the current month or not. There are also symbols next to each day that change color based on the presence of notes and recordings for that date. If both are present, there are a pencil and recording icon; if only recordings, there is a red recording icon; if only notes, there is a blue pencil.



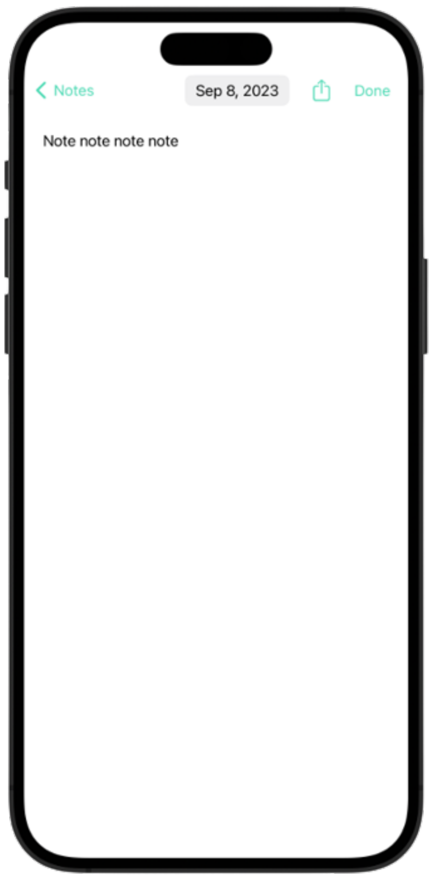
Calendar



Add Note

The user can add a new note to the calendar by tapping the plus button at the top right of the screen. The user sees a view that contains a form with sections for selecting a date and entering the text for a note. It includes a navigation bar with "Cancel" and "Save" buttons. The textbox expands as the user types. The user can scroll through their note and edit the contents before saving their note. The user can tap on a note from the main page. When the user taps on a note, they are taken to a view that includes an editable text area for the note's content and an option to select a date using a date picker. There are also buttons to export the note as text and to save any edits made. When the user edits a note, they can exit the page and the note will automatically save.

I created an algorithm to populate the dates in the calendar. The algorithm determines the number of days per month, and then determines the day of the week that corresponds with each day. The algorithm begins by initializing variables and determining the first weekday of the month. It then calculates the number of days in the previous month to account for days displayed at the beginning of the grid. The algorithm proceeds to fill in days from the previous month and continues by iterating through the days of the current month, adding them to the grid. Weeks are completed and appended to the grid as they are filled. The algorithm also displays the days from the next month that might be visible at the end of a week to complete each row. This ensures that the grid accurately represents the days of the current month, including any days from the preceding and following months that may be displayed at the beginning and end of the grid. The user can switch months by tapping an arrow or swiping left or right.



Note note note note

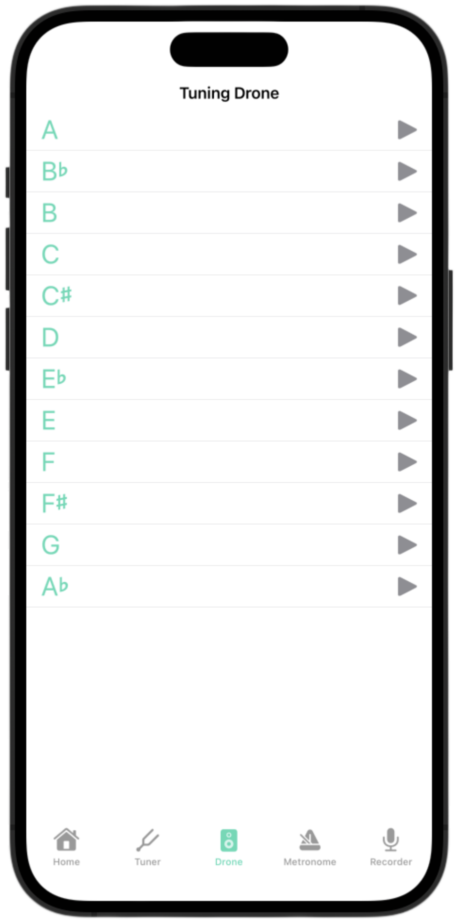
Edit Note

The app's internal storage handles storing notes. When the user taps "Save" the text is converted into a special format called JSON. The JSON version of the note is then stored in UserDefaults, a storage area that keeps data even when the app is closed. UserDefaults is part of the foundation framework in iOS a that provides a way to persistently store small amounts of data such as user preferences. When you use UserDefaults, the data is typically stored in a file on the device, managed by the operating system. When the app is opened again, the JSON data is retrieved from UserDefaults and the user can see their saved notes. When the user makes changes to a note, the code updates the note's content and date, saves it back to the notes array, and stores the updates in UserDefaults for future access. Additionally, there is a function to export the note as a text file. If the user chooses to share the note, the code creates a temporary file with the note's content and presents a share sheet for further actions.

Calendar Feedback and Personal Evaluation

Users provided positive feedback about the calendar feature, emphasizing its effectiveness in organizing notes and recordings. Users thought that the ability to share notes as text files and recordings as audio files was "quick and convenient". I think that students could use this export feature to share their notes or recordings with their music teacher. One user suggested the addition of better visual cues on the calendar (instead of my existing dots), such as "icons like a pencil or a recording symbol", to easily identify which days have associated notes or recordings. Previously, I used different colored circles to represent whether a date on the calendar had recordings, notes, or both. I added this feature to the code after receiving the user feedback.

Tuning Drone
Musicians use tuning drones in practice and performance. Primarily, tuning drones serve as a point of reference for tuning instruments accurately whether that instrument be string, wind, or even voice. Tuning drones also aid in ear training, helping musicians develop their ability to recognize and produce precise pitches. Furthermore, tuning drones facilitate intonation practice, enabling musicians to adjust their playing or singing to match the drone's pitch. In ensemble settings, such as orchestras or choir, tuning drones ensure that all members are in tune with each other. Additionally, tuning drones are used for exercises and warm-ups in a practice setting.



Tuning Drone

The user interface of the tuning drone is a simple list of each pitch. When the user taps one of these notes on the screen, the app will start playing that musical note over and over, like a continuous sound. These notes appear on the app's screen as buttons. When the user taps a button, it starts making a sound, and the little play button changes to a stop button. If the user taps the button again, it stops the sound.

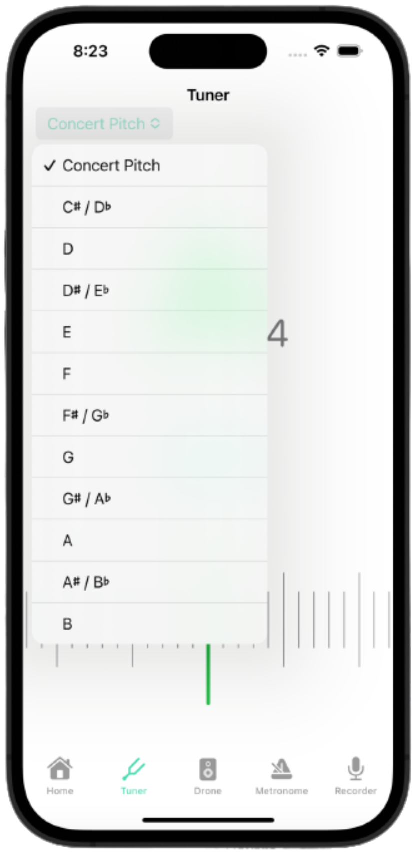
My tuning drone includes the twelve distinct pitches of Western Music. The first pitch of the list is A. As the user looks down the list, the pitches increase chromatically with the final pitch of the list as Ab. I used the free tuning drone mp3 files from https://2reed.net/2_drones.html. This is a "a collection of drones, an organ sound rich in overtones is used to help with tuning. These are not pure tones, so you will hear some pulsing as the overtones move in and out of phase. This more closely simulates a true playing condition than pure tones." I imported these sounds into Garage Band and modified them to have the volume fade-in. The modified mp3 files are saved into my app's asset folder in Xcode and are stored in your iPhone's app storage when you download the app.

Tuning Drone Feedback and Personal Evaluation
The drone is a simple feature on the app that can enhance a practice session with a little ear training. Users liked how they had the "option to tune with a drone or visual tuner." The sounds of the drone have overtones which provide musicians with a heightened level of precision, enriched tonal quality, and improved harmonic awareness in their playing. I would like to expand the functions this tuning drone in the future because some musicians may prefer single-tone drones. Single-tone drones offer a clean and focused reference pitch, allowing musicians to concentrate solely on achieving precise intonation without potential interference from harmonics or overtones. Single-tone drones could also offer different octaves to accommodate instruments with different ranges. In the future, I will consider having more options thus allowing the user to select single-tone drones or the overtone drone.

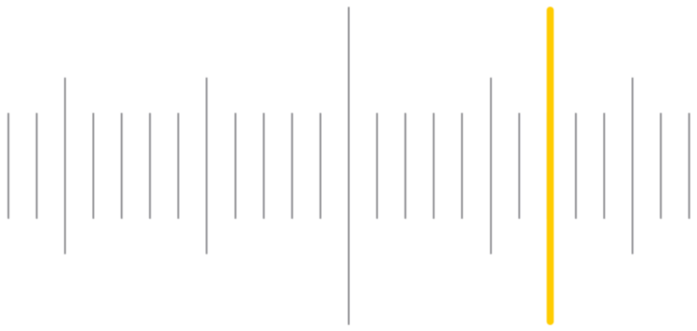
Tuner

A tuner is a device used to help people tune musical instruments accurately. It works by detecting the pitch or frequency of a musical instrument and then indicating whether that pitch is in tune or needs adjustment. To use a tuner, a musician plays a note. If the note is in tune, the indicator on the tuner will align with the desired pitch. If it's out of tune, the tuner will show whether the note is too high (sharp) or too low (flat).

The interface of the tuner includes a transposition menu, a display for the matched note based on the detected pitch and transposition, a cents difference indicator for pitch accuracy, and a visual representation of the detected note's frequency. The large note in the center of the screen is a musical note with its modifier (sharp or flat). The view presents the note and its modifier based on the closeness of the note's pitch to the correct pitch. The app displays the main note and its octave in a horizontal layout, with a modifier if the note has one. The text colors change to indicate the note's accuracy – green for a perfect match, blue for flat, and yellow for sharp. The view also allows users to tap on it to toggle their notation preference by tapping on the displayed pitch.

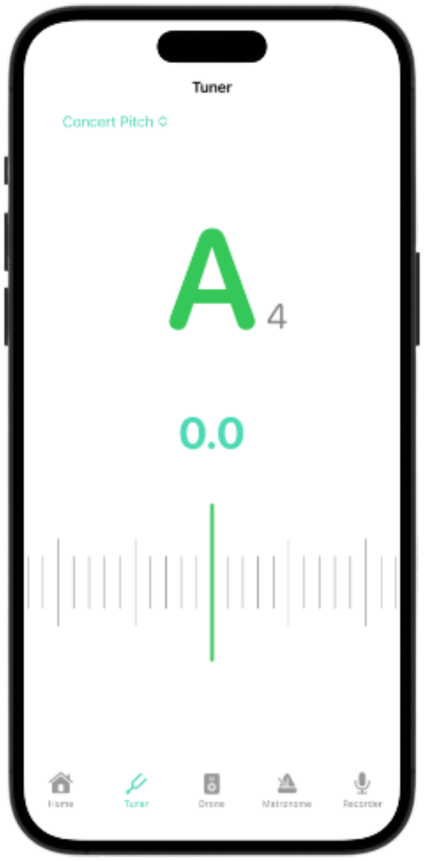


Transposition Menu



Tuning Animation

The screen includes a visual animation to show how close the user's pitch is to the correct intonation. It includes visual markers that represent different musical notes. These markers are like little lines on a screen. Some are shorter and some are taller, like different heights of bars. These bars are colored and have rounded edges. Another part of the code handles showing the current note that the tuner hears. It creates a bar that moves left and right on the screen. This bar also changes color based on how close the note is to being perfectly in tune. If in tune, the bar is green. If flat, the bar is blue, yellow if sharp. This animation provides an additional visual for the user to reference when tuning.



Tuner

In an Xcode SwiftUI project, packages refer to external libraries or frameworks that can be imported and used to extend the functionality of the application. Packages are essentially collections of pre-written code that developers can leverage to save time and effort in building their applications. They can range from small, specialized utilities to large, comprehensive libraries. I utilized a package called "Microphone Pitch Detector" in my application. This package serves as a crucial component for the tuner portion of the app, as it allows the application to listen to audio input from the device's microphone and analyze the pitch of the sound. This package utilizes code from Audio Kit. AudioKit is an open-source framework for iOS, macOS, and tvOS that simplifies audio synthesis, processing, and analysis in Swift. It provides a versatile toolkit for developers to create and manipulate audio, making it easier to build music apps, sound effects, and audio-related applications.

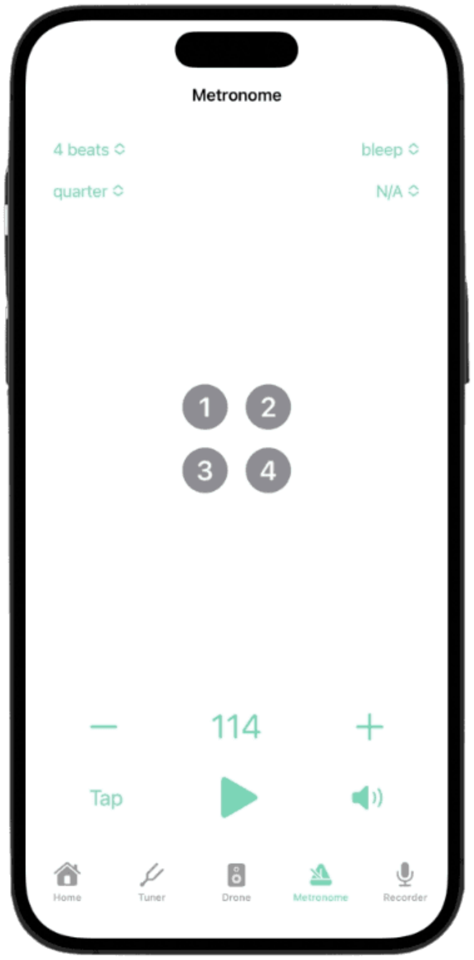
The algorithm of a tuner utilizes pitches and frequencies. It utilizes "twelve-tone equal temperament", which is a way of dividing an octave into twelve equal parts. Each note is given a unique name, like C, D, E, etc. These notes also have associated frequencies, which are the number of vibrations per second that produce the sound. Frequencies are measured in Hertz. Hertz (Hz) is a unit of measurement used to quantify the frequency or oscillations of a wave per second. In our context of sound, Hertz refers to the number of cycles or vibrations of a sound wave that occur in one second. The code provides functions to find the closest musical note to a given frequency. It adjusts the frequency to fit within the range of these defined notes. Then, it compares the adjusted frequency with the known frequencies of the notes to find the closest match. This helps musicians know which note they are playing based on the sound their instrument produces. Additionally, the code offers ways to get the names and frequencies of these notes at a standard pitch. For The code helps the computer understand and compare pitches. It calculates the difference in pitch between two notes. This difference is measured in "cents", which a unit of measurement for pitch. For example, if two notes are 100 cents apart, they sound noticeably different to us. If they're only 5 cents apart, they're very close in pitch and might sound almost the same.

Tuner Feedback and Personal Evaluation

Users found the tuner to work like a standard tuning app. One user liked “how the colors on the tuner change when you are sharp or flat.” In addition, the user thought that the tuner could display the “hertz (frequency) or decibal of the note.” Another user thought that fun tuning animations (such as an elephant to match the app’s icon) would be a fun addition to the user experience. I also would like to add a fun elephant animation but need to learn how to write the code such a complex animation.

Metronome
A metronome produces a regular, steady beat at a specified number of beats per minute (BPM). Musicians use metronomes during practice to develop their sense of timing and rhythm. It's especially important for ensemble playing, where multiple musicians need to play in synch. My metronome had multiple iterations over the past six months.

The UI of the metronome is designed to be interactive and easy to use. It includes four menus at the top of the screen that allow the users to select different options. The top right allows the user to select a sound effect. The top left allows the user to select the number of beats, the top right picker allows the user to select a sound. The bottom left allows the user to select a subdivision and the bottom right picker allows the user to select a beat grouping. Depending on the number of beats selected, the "grouping" provides options for different beat groupings. For example, if the user selects 5 beats, they'll have options like "2+3" and "3+2" to choose from. The tempo controls include a minus button (to decrease the tempo), a numeric display showing the current tempo, and a plus button (to increase the tempo). Users can tap these buttons to adjust the tempo, and a long press on the minus or plus buttons allows for continuous adjustment. There's a button labeled "Tap" which allows users to tap to set the tempo dynamically. The play button toggles between a play and stop icon. When tapped, it starts or stops the metronome. The speaker button toggles the sound on or off. The main visual representation of the metronome is shown in the center of the screen. It's a large display that shows the current beats per measure. The visual changes depending on the number of beats and the grouping selected.



Metronome

The metronome on my app uses a timer-based algorithm. The timer mimics the metronome's "ticking". The timer is like a digital clock that counts time. When the timer "fires," it means a certain amount of time has passed. In this code, when the timer fires, it tells the metronome that a beat has occurred. The metronome can be started and stopped. It can also be set to different speeds (called "tempo") and can divide beats into smaller parts (called "subdivisions"). For example, it can play slow beats or fast beats, depending on what the musician needs. A key feature of the metronome is the ability to "tap a tempo." The algorithm is designed to calculate and manage the tempo, which represents the beats per minute (BPM) of a musical rhythm. Essentially, users tap a button in sync with a rhythm to determine its BPM. There are two main properties: "tempo," representing the calculated BPM, and "pendingTaps," indicating how many more taps are needed before the BPM calculation begins. When a user initiates a tap by tapping on the “tap tempo” button, the code measures the time interval between each tap and leverages these intervals to determine the tempo. Initially, after the very first tap, the "initialize" function sets up the initial state by clearing prior tap data, recording the time of the first tap, and starting a timer. This timer is critical for resetting the tap count after 2 seconds of inactivity. For subsequent taps, the code calculates the time gap between taps and keeps track of the number of taps and the current position. If there are fewer than three taps, the code adjusts a "pendingTaps" count, signifying how many more taps are needed for an accurate tempo calculation. It also restarts the timer to allow for further taps. Once there are at least three taps, the average tap interval is computed and then converting to BPM. However, the tempo cannot exceed a maximum value of 250 BPM. Finally, the timer is reset to clear the tap count and recommence the process.

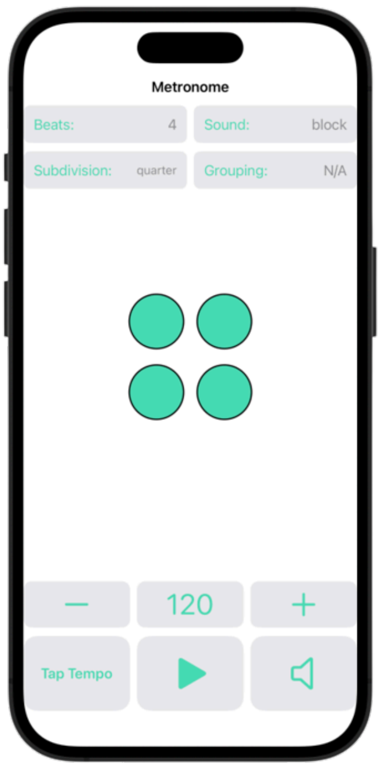
I employed MIDI for sound generation in my metronome project. MIDI facilitates communication between computers and musical devices. Initially, the program sets up timing details, determining the music's playback speed based on a specified beats-per-minute tempo and calculating a single beat's duration. Following this, it adjusts ADSR parameters - Attack, Decay, Sustain, and Release - governing sound changes over time, encompassing the start, volume increase, sustain, and fade-out. Subsequently, based on the chosen sound type, it identifies the musical note and volume level to play. Various notes and loudness levels correspond to different sound types. After these setups, the software directs the computer's sound system to initiate the note and configures details like pitch alterations if needed. Finally, it monitors musical beats and subdivisions, ensuring accurate notes at the right moments, establishing a musical rhythm.

Metronome Feedback and Personal Evaluation

Users really liked the visual representation of the beats in the user interface. They thought that this visual representation was “very customizable.” One user suggested adding “a fun metronome animation that could incorporate the elephant.” I am not sure how I could implement this suggestion, perhaps there could be an elephant that flashes for every beat of the metronome. My original metronome utilized mp3 files that played for each click of the metronome. I discovered during testing that the mp3 files could never be short enough to keep up with the fast clicking of the metronome (16h notes at the highest tempo of BPM). The next page discusses how the old metronome was built. Deciding to create an entirely new metronome one weekend in October was a huge challenge that I faced during the project. However, the new metronome has a much faster response time with fewer bugs.

Early Metronome Version
The original UI of the metronome included shaded buttons. In the new metronome I removed these buttons to allow for a cleaner view. The basic design is very similar to the new metronome. It includes menus to change the beats, subdivision, sound, and grouping. The incremental buttons as well as the tap tempo button are the same.

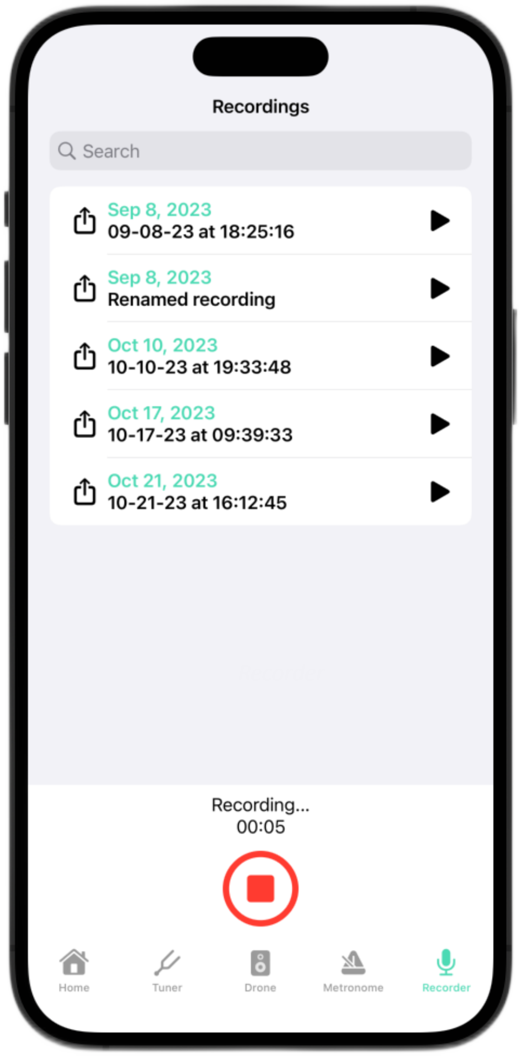
The original metronome also used a timer-based algorithm. There was still an “Audio Engine” that acted as an audio player. Unlike the new metronome which uses MIDI to generate sounds, this metronome used sound clips that were stored as mp3 files in the device folders. I recorded these original metronome sounds using the MIDI in GarageBand which is Apple's basic digital audio workstation software. Each mp3 file was a short burst of sound. When the metronome tried to play a sound, it first checked if the sound existed, and if it did, it set up the audio session to play. This proved to be an issue when trying to run the metronome at fast speeds. The mp3 files were never short enough to keep up with a fast tempo with a sixteenth note subdivision. I created the new metronome to accommodate faster tempos as previously discussed.



Original Metronome

Garage Band

Recorder
The recorder displays a list of recorded audio files provides buttons for recording and stopping. Each recording has an ID, a creation date, a file URL, and a formatted date text. The creation date is obtained from the file's attributes, and if it can't be retrieved, the current date and time are used as a backup. While recording, it shows the elapsed time. The code also handles starting and stopping the recording, updating the timer, and formatting the time in minutes and seconds. The user can search for recordings, rename them, share them, and play or stop playback with intuitive buttons. Each recording row displays its creation date, and you can tap to edit a recording's name. Swiping a row to the left reveals a delete option. The export button enables the user to share their recording or save it in their device's file manager or iCloud. In addition to being displayed on the recording list, the recordings are also displayed on the calendar.


Recordings

Search

Sep 8, 2023
09-08-23 at 18:25:16

Sep 8, 2023
Renamed recording

Oct 10, 2023
10-10-23 at 19:33:48

Oct 17, 2023
10-17-23 at 09:39:33

Oct 21, 2023
10-21-23 at 16:12:45

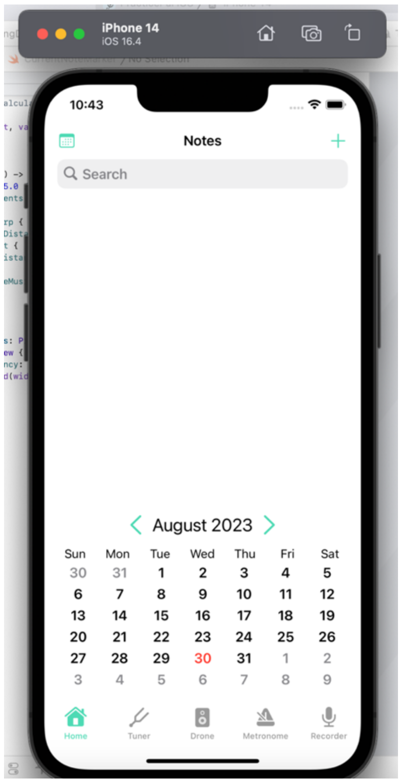
Recording...
00:05

Recorder

This code for the recorder includes two parts: one for recording audio and managing the recordings, and the other for playing back audio files. The recording part handles setting up the audio session, saving the recordings in the document directory, and managing the recording process. It also keeps track of recorded files and allows actions like start, pause, resume, and stop recording. The playback part sets up the audio session for playback, ensures the audio plays through the device's speakers, and provides functions to start and stop playback. It also handles the end of playback. Together, these parts create a system for recording and playing back audio files in a SwiftUI app.

Recorder Feedback and Personal Evaluation
Users appreciated having a built-in recorder in their practicing app. One user thought that having an ability to create collections of recordings could enhance user experience. This could potentially be implemented through the introduction of a structured folder system on the recordings page. For instance, users might have the ability to establish distinct folders dedicated to scales recordings, repertoire recordings, or those earmarked for exportation. Moreover, the inclusion of a search bar facilitates swift retrieval of recordings, even those created weeks or months earlier. One user especially loved "how the recordings and notes integrate on the home screen. It makes it easier to track progress."

App Deployment
Throughout the development of each "feature" of my app, I did extensive testing. During development, I used the simulators that are built in to Xcode. These simulators mimic a phone and provide easy access to the console for testing and debugging.

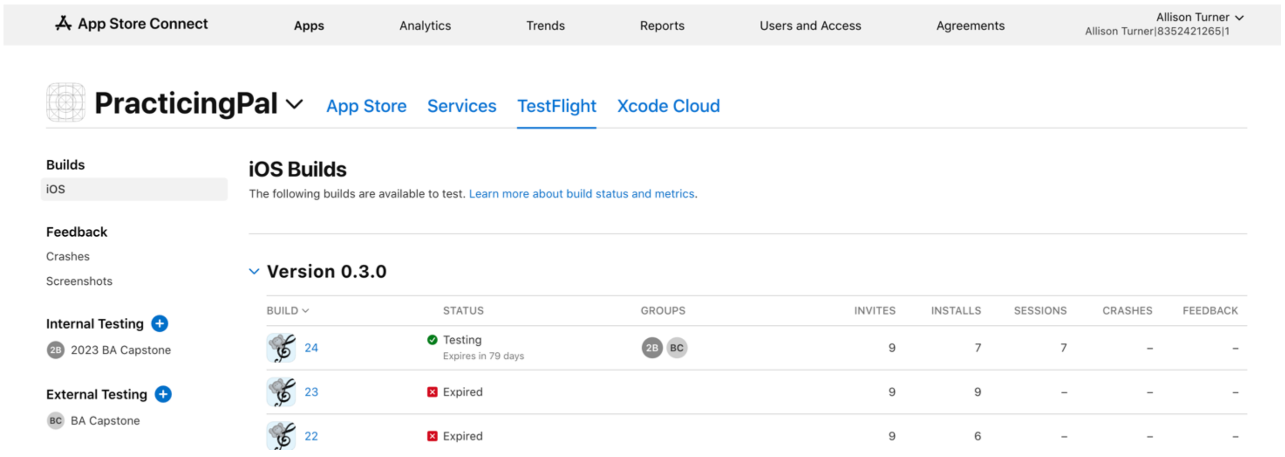


Xcode iPhone Simulator

When starting the process of testing on a real iPhone, I learned about "developer mode." Developer mode allows iOS developers use access advanced settings and debugging tools. With my phone plugged into my computer, I can access the console and logs in Xcode while running the app on my phone.

After testing in developer mode, I began beta testing my app. A beta version of an app refers to a pre-release version of the app that is made available to a limited group of users for testing purposes before its official launch. This phase of development is called the "beta testing" phase. Beta versions allow developers to identify and address bugs, glitches, and other issues that might not have been discovered during the app's internal testing. Beta testing can provide valuable insights and suggestions that help developers improve the app's overall quality and user experience.

To create the beta test version of the app I used App Store Connect. App Store Connect is a platform provided by Apple that allows developers to manage and distribute their apps on the App Store, including tasks like submitting updates, managing metadata, and monitoring user reviews and analytics. It also allows user to add their app to TestFlight.
TestFlight is a tool that allows you to distribute pre-release versions of your app. With TestFlight, users were able to download my app with a link through the TestFlight app.


App Store Connect   Apps   Analytics   Trends   Reports   Users and Access   Agreements   Allison Turner
Allison Turner|8352421265|1

PracticingPal   App Store   Services   TestFlight   Xcode Cloud

Builds
iOS

Feedback
Crashes
Screenshots

Internal Testing
2B 2023 BA Capstone

External Testing
BC BA Capstone

iOS Builds
The following builds are available to test. Learn more about build status and metrics.

Version 0.3.0

BUILD	STATUS	GROUPS	INVITES	INSTALLS	SESSIONS	CRASHES	FEEDBACK
24	Testing Expires in 79 days	2B BC	9	7	7	–	–
23	Expired		9	9	–	–	–
22	Expired		9	6	–	–	–

App Store Connect