CS 4624: Multimedia, Hypertext, and Information Access
Blacksburg, VA 24061
Virginia Tech

Practice 10k Music App

Clients: Dr. John Irrera & Dr. Annie Stevens
Professor: Dr. Edward A. Fox
May 3, 2024

Rylen Bass, Minwu Kim, Johan Lee, Jillian Ylagan
# Table of Contents

List of Figures................................................................................................................................. 2
List of Tables................................................................................................................................. 4
Abstract........................................................................................................................................... 5
1. Introduction................................................................................................................................. 6
2. Motivation and Clients............................................................................................................... 7
3. Deliverables................................................................................................................................. 7
   3.1 Metronome Implementation................................................................................................. 8
   3.2 Supabase........................................................................................................................... 10
4. Problems and Solutions............................................................................................................ 15
   4.1 Timeline............................................................................................................................. 15
   4.2 Metronome Challenges.................................................................................................... 15
   4.3 Supabase Challenges....................................................................................................... 17
      4.3.1 Redux Challenges.................................................................................................... 17
5. Learning Objectives.................................................................................................................. 19
6. Future Work............................................................................................................................... 20
7. Testing and Evaluation............................................................................................................... 22
   7.1 Testing questions for app................................................................................................. 22
   7.2 Testing questions for metronome.................................................................................. 23
   7.3 App Testing Results...................................................................................................... 23
   7.4 Metronome Testing Results.......................................................................................... 24
8. User Manual.............................................................................................................................. 26
   8.1 App Manual..................................................................................................................... 26
   8.2 Metronome Manual....................................................................................................... 30
9. Developer Manual..................................................................................................................... 31
   9.1 Project Setup.................................................................................................................. 31
   9.2 Metronome Setup........................................................................................................... 34
10. Acknowledgements................................................................................................................ 36
11. References............................................................................................................................... 37
List of Figures

Figure 1: Implementation of JUCE Metronome................................................................. 8
Figure 2: Supabase table structure in the database.......................................................... 11
Figure 3: User Table with Information............................................................................ 11
Figure 4: Music Pieces Table with Information.............................................................. 11
Figure 5: Practice Data Table with Information............................................................. 12
Figure 6: Practice Data Table with Information (continued)........................................ 12
Figure 7: Changing the Authentication API to use Supabase methods and authentication... 13
Figure 8: Changing the Data Management API to use Supabase and interact with the database.. 14
Figure 9: Updating application state using Redux reducers when adding Practice Data........... 18
Figure 10: Example QR Code.......................................................................................... 27
Figure 11: Home Screen, Register, and Login Pages....................................................... 28
Figure 12: Profile Page with the User’s Information....................................................... 29
Figure 13: Add Piece and Plan Details Page................................................................... 29
Figure 14: Metronome in use......................................................................................... 30
List of Tables

Table 1: Metronome Demo Events ......................................................................................................................... 10
Table 2: Project Timeline ...................................................................................................................................... 15
Table 3: Application File Structure .................................................................................................................... 32
Table 4: Metronome File Structure .................................................................................................................... 34
Abstract

Most musicians strive to practice their instrument every day, which warrants a comprehensive companion app. Such an application should allow users to log their practice sessions and keep them motivated, among other useful features. The Practice 10k App continues in its development to bring these features to aspiring musicians. As of the prior development team, users can currently use the app to create customizable profiles, log their practice sessions, and plan future practice sessions. Our team has continued development by adding a metronome feature, switching the backend service, and fixing database issues. Practice 10k will teach beginner, intermediate, and professional musicians new ways to practice and hold themselves accountable in their musical studies.
1. Introduction

Learning to play an instrument is a journey that countless people of all ages embark upon or continue every year. There are many benefits to practicing an instrument, such as greater musical literacy, cognitive improvement, or simply honing one’s musicianship. Whatever the reason, there is one common aspect of everyone’s musical journey: practice. Most serious musicians make an effort to practice their instrument every day, but adhering to a strict practice regiment is easier said than done. Finding the motivation and building the discipline to practice daily is a challenge that most musicians face on a regular basis.

Through the Practice 10k Music App, we hope to create a comprehensive tool that will assist musicians with every aspect of their practice session and help keep them motivated. Features such as logging practice sessions and tracking the speed of their songs will all be within reach inside one application instead of separated throughout multiple apps. Users will also have greater incentive to just open the app as a result of gamified elements, which will promote more frequent practice. By facilitating friendly competition against others in online social environments, we hope to help musicians find the motivation to begin their practice session each day.

This report discusses the development of the Practice 10k mobile application throughout the Spring 2024 semester. We first discuss the clients and their motivations behind this project. We also discuss the users of the app and how they will benefit from using it. We then move into the deliverables we completed this semester, as well as the problems we faced throughout this project. After that, we reflect on what we learned during the undertaking of this project and opportunities for future work that other groups could take on after we submit our deliverables. The last sections of the report include a user manual and a developer manual, which will help users and future developers or capstone teams better understand and extend the app.
2. Motivation and Clients

In his 2008 book “Outliers: The Story of Success”, Malcolm Gladwell describes the idea that in order to become an expert in anything, one must practice for at least 10,000 hours. Gladwell applied this idea to the Beatles and surgical residents, noting that the Beatles played over a thousand live shows, adding up to over 10,000 hours, and that surgical residents undergo almost 17,000 hours of work before the end of their residency [1]. Practice 10k is a music practice app designed with this principle in mind; it aims to guide users towards that 10,000 hour goal and keep them motivated throughout their musical journey.

While there are some music practice apps that already exist on the market, they are limited in the functionalities they offer, and lack user retention. One such example is Andante, which allows users to log their practice sessions and track their mood [2], but does not have any elements of gamification or social competition. The Practice 10k App will offer a variety of practice-session tracking features as well as a tuner, metronome, and audio recorder among others—all in one place. This will increase productivity within a practice session and eliminate any unnecessary navigation between apps, allowing the user to stay more focused on the tasks at hand. The practice-session tracker also allows users to plan ahead with their practice schedule, helping keep the user accountable to themselves and their own practice plan. Gamification elements will keep users more engaged with the app, while social aspects will encourage friendly competition between users and incentivize more frequent practice. For example, a group of friends could start a challenge to see who among them can log the most practice hours that week, with the winner earning a badge that can be displayed on their profile, or some other kind of reward. The friends participating in the challenge should experience greater productivity that week because of the social pressure and the shift to a different goal besides practicing for themselves.

The Practice 10k Music App project was proposed by Dr. John Irrera and Dr. Annie Stevens, who are assistant and associate professors at Virginia Tech’s School of Performing Arts. As teachers and musicians themselves, they have seen students struggle to adhere to a practice routine, inspiring them to address these challenges and work with our predecessors to develop this application [3].
3. Deliverables

3.1 Metronome Implementation

The first deliverable that our clients indicated interest in was a metronome feature for the app to help the user keep pace as they play their instrument. We developed the metronome using the JUCE framework [12], which is a cross-platform, C++ framework commonly used to create mobile and desktop applications with specialized audio functionality [4]. While it is possible to create a metronome using a React library [5], which is what the existing codebase is written with, such a metronome would lack the precision and consistency required for an accurate musical application [5]. Future development of other audio features would likely also rely upon JUCE. The solution we formulated with the lead developer was to create the metronome within JUCE and then integrate it with the rest of the React-based app. It was not feasible to create the entire app within JUCE, as JUCE is not equipped to handle backend challenges such as user authentication and database management. Currently, the metronome exists as a desktop application; we began the process of developing an iOS version, but shifted our focus elsewhere as the lead developer took over much of the integration process.

![Figure 1: Implementation of JUCE Metronome](image)

Figure 1 shows the final iteration of the metronome application, which consists of five buttons laid out horizontally, and a text display located on top. The central button is labeled ‘Play’ and begins the metronome when pressed. It subsequently changes its text to ‘Stop’, allowing the user to stop the metronome with the same button. Within the code, there is a toggle state associated
with this button which is separate from the overall “Playing” and “Stopped” states that govern other aspects of the metronome’s behavior. The metronome will make a beeping sound at a speed of 60 beats per minute (bpm) by default. The other buttons allow the user to change the speed of the metronome; the < button decreases the bpm by 1 while the > button increases the bpm by 1. Holding down either button longer than 0.5 seconds will cause the button to repeatedly click every 150 milliseconds for the user’s convenience. The << and >> buttons have the same functionality as the < and > buttons respectively, but increment or decrement the metronome’s bpm by 10 instead of 1. These features allow the user to quickly change the bpm to their desired speed through multiple options. The bpm is visible to the user at all times and is displayed through the text box above the buttons.

The logic behind the metronome’s timing comes from a four-part tutorial provided to us by the lead developer [6]. This tutorial created a metronome as a basic JUCE audio application with a play button, a stop button, and a fixed bpm. Our first change to this basic implementation was to combine the play and stop button into a single button that changes its label depending on the current state. This gives the user a visual cue of whether their metronome is playing and also saves space in the application window. The second change we made was fixing the issues with the MP3 file’s location. The tutorial stored its MP3 file in the user’s local documents folder, making it difficult to use the feature on another device. We added the sound we use into the project files where it is then converted to binary and played consistently from there.

Based on feedback from our client, we later added several ways to allow the user to customize the metronome’s bpm. These included small and large increment buttons as well as a repeated click functionality when a button is held down. The final change consisted of modifying the layout and adding a box to our window that displays the current bpm for the user to see.

A demonstration of the metronome’s functionality is included in the MetronomeDemoVideo.mp4 file. This video is approximately 40 seconds long. The events that occur in the demo along with their timestamps are shown in Table 1.
Table 1: Metronome Demo Events

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00</td>
<td>Clicking the Play button to hear the beep at 60 bpm.</td>
</tr>
<tr>
<td>0:06</td>
<td>Changing the bpm with the single arrow buttons.</td>
</tr>
<tr>
<td>0:12</td>
<td>Changing the bpm with the double arrow buttons.</td>
</tr>
<tr>
<td>0:18</td>
<td>Clicking and holding the increment buttons.</td>
</tr>
<tr>
<td>0:26</td>
<td>Clicking and holding the decrement buttons.</td>
</tr>
</tbody>
</table>

3.2 Supabase

The other major deliverable this semester was to convert the backend database from Firebase [7] to Supabase [8]. This involved changing Firebase-specific methods to the corresponding Supabase methods and setting up the Supabase structure. Figure 2 shows the table structure and data types that we chose to hold user profile information.

As shown in Figure 3, when a user registers with their information in the app, it will be logged in our Users table, which is connected to the authentication table. The user will set these fields accordingly in the app, which is where the information will be stored. Whenever the user updates their profile, this table will be updated as well. There were no previous registered users in this application since it has not yet been released to the App Store, so there was no need for any data migration.

In addition, when a user adds a practice piece in the app and practices it, the music piece and practice data will be added to the respective tables, as shown in Figures 4, 5, and 6. These tables have a User ID section that matches which user created the piece and the practice data associated with it.
Figure 2: Supabase table structure in the database

Figure 3: User Table with Information

Figure 4: Music Pieces Table with Information
The original code was written with an Authentication API and a Data Management API specific to Firebase functionality. As part of the conversion to Supabase, we updated the authentication and data management, as shown in Figures 7 and 8. In Figure 7, we changed methods like register and resetPassword so that users can authenticate themselves before entering the application. In Figure 8, we changed methods like updateUserProfile so that the user can change the data that is associated with their profile.
export const AuthenticationAPI = {

  // Register the user with their profile picture, name, birthday, instruments, level, email, and password
  async register(profilePicture: string, name: string, dateOfBirth: string, instruments: string[], level: string,
  email: string, password: string) {
    const { data, error } = await supabase.auth.signIn({
      email,
      password,
    });
    if (error) {
      console.error('Could not register:', error.message)
      throw error;
    }

    // After signing up, add user profile data to your database
    await DataManagementAPI.addUserProfile({
      email, profilePicture, name, dateOfBirth, instruments, level
    });
    return data;
  },

  async resetPassword(email: string) {
    const { error } = await supabase.auth.resetPasswordForEmail(email);
    if (error) throw error;
  },

  async changePassword(email: string, oldPassword: string, newPassword: string) {
    const { error } = await supabase.auth.updateUser({
      password: newPassword,
    });
    if (error) throw error;
  },

Figure 7: Changing the Authentication API to use Supabase methods and authentication
Figure 8: Changing the Data Management API to use Supabase and interact with the database

```javascript
export const DataManagementAPI =

async updateUserProfile({ email, name, dateOfBirth, instruments, level }: IUserDataProps) {
    const { data: { user } } = await supabase.auth.getUser()
    if (user) {
        const { data, error } = await supabase.from('Users').upsert({
            id: user.id, // Assuming your users table has an 'id' field that matches the
            email,
            name,
            dateOfBirth,
            instruments,
            level
        });
        if (error) {
            throw error;
        }
        return data;
    }
    else {
        throw new Error('User is not undefined. Cannot update user profile.');
    }
}
```
4. Problems and Solutions

4.1 Timeline

Table 2: Project Timeline

<table>
<thead>
<tr>
<th>Complete By</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/15</td>
<td>Meet with lead programmer and clients</td>
</tr>
<tr>
<td>2/16</td>
<td>Finish preliminary research</td>
</tr>
<tr>
<td>3/15</td>
<td>Follow metronome tutorial for basic metronome</td>
</tr>
<tr>
<td>3/19</td>
<td>Presentation 2</td>
</tr>
<tr>
<td>4/5</td>
<td>Finish metronome additional features (+/- buttons, UI)</td>
</tr>
<tr>
<td>4/5</td>
<td>Finish Supabase integration and tables</td>
</tr>
<tr>
<td>4/11</td>
<td>Meet with clients for final approval</td>
</tr>
<tr>
<td>4/12</td>
<td>Finish all manual testing</td>
</tr>
<tr>
<td>4/23</td>
<td>Final presentation</td>
</tr>
</tbody>
</table>

Throughout the project, our team adjusted the timeline according to our current deliverables and client feedback. Our final timeline is reflected in Table 2.

4.2 Metronome Challenges

One of the major technical challenges we faced this semester was the integration of the metronome JUCE application with the rest of the application in React. While we were aware of this challenge when formulating a plan, we did not anticipate how complex it would be. The most difficult aspect was simply the lack of resources; integrating JUCE and React Native is not a well-documented process. The only resource we could find was a blog post tutorial [9] and an example repository accompanying a presentation given by the blogger [10].

The core idea of this resource to achieve integration was to go from the C++ metronome code to the native API for iOS and Android, which are in Objective-C and Java respectively, and then from there to the React library via the native-to-JavaScript bridge. We chose to work on the iOS aspect because of the similarity between C++ and Objective-C. Despite our lack of experience
with iOS programming, we noted previous discussions in the JUCE forums experiencing many difficulties with building the project on Android, and decided to try iOS first instead [11]. The end goal was to be able to show the JUCE component as a custom view within the React application. This was done by defining a ReactJuceView class and a manager class that subclasses RCTViewManager. The manager class details how the view is created and updated on the native end so instructions regarding the views can be passed to the JavaScript code. This should theoretically allow us to use the metronome application within the rest of the project. However, the resources were quite outdated, and while we sought to recreate the steps taken in these resources, we encountered a number of issues along the way.

A significant challenge was our team’s unfamiliarity with the technology stack. Not only were we new to React and JavaScript, but we were also unfamiliar with the JUCE framework and C++. There was a significant learning curve regarding all of the different languages and frameworks before we even began to develop any logic. Another challenge was working around the nuances of all the software and tools involved. This project would eventually have both iOS and Android releases, so we worked with Projucer, the IDE for managing JUCE projects [12]. However, Projucer was very particular about creating or exporting files, as it was responsible for generating the build files for XCode, Visual Studio, Android, and iOS projects, so we had to be very careful throughout. Keeping in mind the different versions, build processes, package and dependency managers, and IDEs, was a new challenge for us all. We were accustomed to VSCode and Windows, having used them throughout our academic careers, so shifting to XCode and MacOS was surprisingly not easy.

The final roadblock in the integration process was an unexpected hurdle with Expo Go [17], which both we and the previous team used to quickly launch the app during testing. Expo Go displays all of the real-time changes that we make as we modify the code for the application, so it’s useful for identifying bugs and experimenting with solutions. Of the limited resources available, none commented on Expo; they only provided guidance regarding JUCE and React alone. Upon discussing with the lead developer, we learned that we needed to use Expo Go to create the host application, not JUCE. Doing so would require working with CMake, which is software that allows users to create the build files for other systems [13]. The next step would be to use CMake to generate a static library with the JUCE code to then link to the Expo app. Unfortunately, this posed a technical challenge that was beyond the scope and time constraints of
this semester, so the lead developer took over this part of the project, to continue in future iterations.

4.3 Supabase Challenges

From a technical standpoint, the shift in databases from Firebase to Supabase allows for greater scalability in the future, as Firebase has a limit to the number of users it can track in a database. Firebase is also somewhat outdated, so we hope that shifting to Supabase will help improve the app’s scalability going forward.

4.3.1 Redux Challenges

Our project uses Redux to manage application state, and reducers to update the application state in response to actions [14]. During app development, we encountered state mutation bugs with Redux, as we were manually handling state changes without dispatching them to update Redux. State mutation bugs occur when the application’s state, which includes data used to manage the app’s behavior and appearance, is changed directly in a way that violates the principle of immutability, which states that data should not be changed once it is created. Immutable data ensures predictability and prevents unintended side effects in the application.

In our original code, we directly took practice data from the Supabase table and manually set the fields in it to different values in order to update the UI for practice plans. This resulted in state mutation errors where there was a discrepancy between the application’s state and the data stored in the database as the UI was showing modified data that was taken from the database but was not saved anywhere else. Thus when the application reloaded, the data would be lost and the application would attempt to render data that could not be found.

Our solution to this problem was to refactor and update our reducer logic to create new functions to take the current state and action as an input and return a new state object without modifying the original. This ensured consistency between application states and eliminated potential bugs related to data discrepancies. An example of our implementation is in Figure 9, where we use reducers defined in our actions.tsx to properly update the Redux store as the user saves a practice plan in add_plan_container.tsx.
async function handleSave(plan: any) {
    try {
        const practiceId = await DataManagementAPI.addPracticeData(plan.title, plan.piece, plan.composer, plan.instrument, select
        const musicPiece = { title: plan.title, piece: plan.piece, composer: plan.composer, instrument: plan.instrument, notes:
            // Dispatch actions to update Redux store
            dispatch(addPracticeData{
                id: practiceId,
                title: plan.title,
                piece: plan.piece,
                composer: plan.composer,
                instrument: plan.instrument,
                notes: plan.notes,
                practiceDate: weekday,
                duration: 0,
                status: STATUS[0]
                })
            const existing = currentMusicPieces.musicPieces.find(item => item.title === musicPiece.title && item.piece === musicPiece.
            item.composer === musicPiece.composer && item.institute === musicPiece.
            item.notes === musicPiece.notes);
            if (!existing) {
                dispatch(addMusicPiece(musicPiece));
            }
        }
    } catch (err) {
        console.error('Error adding practice data:', err)
    }
}
5. Learning Objectives

One objective that we met was learning how to convert an application’s database from Firebase to Supabase. This required us to go into the service files of the app, import Supabase, and change the methods so that they followed the Supabase conventions instead of Firebase. For this, we also had to develop our knowledge in the TypeScript language [15]. In addition, we learned how to create tables for storing information in a database using Supabase. For the learning process, we used the Supabase website to understand how to connect the tables and backend database to the app, and watched YouTube videos on it as well [16] [17].

We learned more about C++, the JUCE framework, and build processes, through the development of the metronome. Trying to integrate it with React also gave us a more detailed understanding of how React and iOS programming works. Resources such as [9] provided insight into how different applications and technologies interface with each other to create a smooth connection, and pass data back and forth. While we were ultimately unable to finish integration of the metronome with the rest of the application this semester, we learned a great deal about the more practical side of connecting technologies together instead of just developing logic.

Finally, we gained hands-on experience with the application development process throughout the semester. We learned that it takes a lot of planning to develop and maintain an app with a team of more than just two developers. This is why we used Trello [18] to track and maintain transparency for everyone’s tasks and by when they should be completed. In addition, we used different email chains with our clients and the lead programmer to discuss different topics related to the project. For example, we had one email chain with everyone to plan bi-weekly meetings based on everyone’s schedule and availability. We also had an email chain with the lead programmer to discuss the more technical issues, and also to ask questions related to the code and development of the application.

One learning objective that we did not get to this semester was deploying the app into the App Store. The lead programmer who has worked on the Practice 10k Music App in the past will be taking the lead on this part; he will be working on it after the semester is over and when the app is ready to be released.
6. Future Work

One of the most exciting tasks planned for the future is implementing a new app design. Our clients have been working with a graphic designer to make the app more aesthetically pleasing and visually engaging. Future teams should work closely with the designer to assist in any implementations and required changes.

Another feature future teams can implement is more gamification in the app to increase motivation in newer users. Some examples include a leaderboard, badges, friending other users, and a level system. Implementing a level system where users can level up and gain badges can motivate users to practice more. Additionally, having a friend system can provide accountability between users and allow them to track each other’s progress. There could also be further implementation of these features to make the learning and practicing experience as simple and easy as possible.

For the metronome, work still needs to be done to implement it into the app. Future teams may need to work alongside our client’s lead programmer to implement CMake related changes. CMake can be used to build the metronome on non-iOS devices and is important for Android related development. In regards to the new design, the window size and buttons will need to be adjusted to fit smoothly into a mobile screen.

Other minor implementations can include:

1) Updating the Data Management API to include a method to delete from the music_pieces table in the database, and integrating that into the application, as there is currently only a method to delete practice data.

2) The tests in the auth tab would also need to be refactored for testing Supabase authentication.

3) Users should be able to see the progress of their practice plan and its updated status. The functionality for updating the practice plan status is in PracticeTimer.tsx but it could be displayed in the UI as well. Users should also be able to see the duration of time they have practiced for each plan. This way, they will be able to track the total number of hours that they have practiced for all of their pieces combined.
4) Implementing the Journal and Progress tabs to display the practice data and update the progress.

5) Removing the profile tab from the bottom, using only the top right icon to access the profile. Currently both options work but it is redundant to have two.

6) Confirming the email for registering users. There is a confirm email button in Supabase that was initially turned off as it did not allow us to register while it was on.

7) Row-Level Security (RLS) disabled for some Supabase tables. We implemented Auth policies to allow create, read, update, and delete (CRUD) operations on the tables but there were permission errors when attempting to add data via the app while RLS was on.

Finally, the last goal is to make the Practice 10k App ready for the app deployment process. Our clients are ready for the app to be released in order to start gaining users and receiving feedback. A future team can work towards the app release while also working on other features.
7. Testing and Evaluation

To evaluate the Practice 10k App, our team conducted a total of seven testing sessions where each participant spent time navigating and interacting with the app or metronome. Each student completed one questionnaire after their session to record their thoughts on their experiences. Three students completed the app related questionnaire and four completed the metronome related questionnaire. We were looking for feedback regarding user satisfaction and understanding of the app. Our participants were other Virginia Tech students who had a variety of past musical experience.

7.1 Testing questions for app

1) Registration/Login Page
   a) Are the registration/login pages easy to navigate?
   b) Is there any other registration information that you think the app should collect?

2) Home Page:
   a) Is the home page easy to understand? Can you tell what each section on this page is meant to do?
   b) Is the process of adding a piece simple and fast?
   c) Is the difference between New Piece and Previous Piece obvious?
   d) How easy is it to navigate through previous pieces and find a specific one?

3) Practice Page:
   a) Is it immediately obvious that pressing the Start button will take you to a different screen with more functionality?
   b) Is the layout of the Practicing screen easy to navigate?
   c) Is it easy to switch between different pieces throughout the same session?

4) Profile Page:
   a) Is the process of updating your profile information satisfactory?

5) Overall, is the app easy to use without instruction? Do your expectations for functions correspond to the button names? (registration, adding practice plan, etc.)

6) Is the UI easy to navigate and visually pleasing?
7) Is it easy to tell if user actions have succeeded or failed? Is there sufficient feedback from the app and guidance in the event of a failure?
8) What improvements/suggestions do you have?

7.2 Testing questions for metronome

1) When holding down the button:
   a) Does the delay feel too short/long?
   b) Does the auto-repeat occur fast enough?
2) Is the beep sound pleasant to listen to? Could it be distracting while simultaneously playing an instrument?
3) Is changing the metronome’s bpm a quick and easy experience?
4) Is the metronome’s layout simple to navigate without instruction?
5) Are the buttons too big/small?
6) What improvements/suggestions do you have?

7.3 App Testing Results

We tested the application with three Virginia Tech students. One of them has played instruments in the past, but does not currently do so at the collegiate level. Another has no musical experience at all. The final student is in the orchestra at Virginia Tech.

One of the subjects thought that the layout of the application was easy to navigate overall, but one concern was that the list of previous pieces had very big boxes storing them, making it difficult to scroll through if there were more than ten of them. One suggestion to fix this was to only list the song names; the user could then click on a specific song to get relevant information such as the composer and instrument it’s written for. Another subject said that when using the Practice feature, it was inconvenient that they could only start from the first piece and had to press the next button to go through them without being able to rearrange the pieces. As for the visual aspect of the app, one of the subjects thought that adding pictures of the instruments next to the list and also more icons would make the app more visually pleasing.

A second subject also found the application to be very intuitive. They found it easy to register themselves and understood the purpose of each page. Despite most aspects of the app being easy
to use, they ran into errors when trying to add a piece. Even though all required fields were filled in correctly, an error popped up for the user. The error message did not give enough guidance to the user to allow them to fix what was wrong. They also suggested that the users should be able to adjust the daily hours of practice to fit their own needs. The last suggestion they gave was to make it so the user can see the time when in the app.

Our third subject was a music minor who plays for an orchestra at Virginia Tech. They found the UI to be easy to navigate, with each function of the app corresponding well with their subsequent names. During the process of adding a music piece, they were confused about the Title field as there was already a Piece field. They suggested removing it altogether or creating a field called ‘section’ or ‘technique’ to cover the practice parts of a piece, and be more specific. They also suggested the ability to add ‘scales’ or ‘techniques’ into a practice plan along with the music piece so they could track their practice session in conjunction with other practice elements. Another suggestion was to have the ability to edit the daily/weekly goal and to use the Journal tab to add rehearsal hours that differ from personal practice hours, having a separate record for rehearsals. This could include pressing a button when beginning the rehearsal session and then pressing another button when finishing rehearsal and recording that session inside the Journal section. Lastly, they thought it would be helpful to have an animation such as confetti whenever a user completes their goal, for motivation and dopamine release.

7.4 Metronome Testing Results

We tested the metronome with four different Virginia Tech students; two had a recent musical background (with The Marching Virginians) while the other two did not. Each subject reported that the metronome interface was simple to navigate without instruction, although two subjects did not realize that the incrementing buttons could be held down to rapidly increment the bpm. Each subject reported that the delay between holding down the button and the bpm rapidly incrementing (500ms) was comfortable and similar enough to other familiar UIs. The speed at which the bpm incremented when the buttons were held down was agreed upon by all participants to be satisfactory, and did not trigger impatience. Overall, the participants all reported that changing the speed of the metronome was a quick and easy experience.
With regards to the beeping noise that marks every downbeat, 3 of the 4 participants responded positively, while the last participant had more neutral feelings; they stated that it was “pleasant enough” and not any more annoying than other metronome sounds. When asked for improvements, the two participants with musical experience both suggested changing the tone of the beep on the downbeat to differentiate it from the others, or adding the ability to customize the tone of any beep. Aside from that, there were no overlapping suggestions. One participant expressed interest in in-app volume control; another suggested aesthetic improvements to the UI.

8.1 App Manual

Scanning the QR code on your mobile device will launch the app in Expo Go as shown in Figure 10 [19]. The QR code shown in Figure 10 is just an example of what it will look like on the user side, and will not have any functionality. ExpoGo is a free, open-source sandbox for experimenting with React Native on Android and iOS devices. Users can easily run applications on their phone and see their mobile interactions live. An example of what the QR code will look like can be seen in Figure 11. Users will be led to a start screen with the option to ‘log in’ or ‘register’. In Figure 11, you can see the Home Screen when opening the application and also the Register and Login pages. Figure 12 shows the Profile page with the user’s information displayed. The Profile page also allows the user to edit any of their info, as well as delete or log out of their account. Figure 13 shows the Add Piece and Plan Details pages. The user can choose between adding a new piece and entering the information for it, or choosing from a previous piece.
Figure 10: Example QR Code
Figure 11: Home Screen, Register, and Login Pages
Figure 12: Profile Page with the User’s Information

Figure 13: Add Piece and Plan Details Page
8.2 Metronome Manual

![Figure 14: Metronome in use](image)

To start the metronome, as shown in Figure 14, the user will click the center button labeled “Play”. To stop the metronome, the user will click the same button, now labeled “Stop”.

To change the pace of the metronome, the “<” and “>” buttons will decrease or increase the bpm by one. The “<<” and “>>” buttons will decrease and increase the bpm by ten. These buttons can be held down to change the bpm by their respective increments rapidly. The adjusted bpm is reflected in the label at the top of the window.

9.1 Project Setup

The developer manual is intended to provide instructions for developers to set up the project on their own device in order to continue development. Expo was chosen for its toolset and powerful capabilities to build cross platform mobile applications using React Native.

1. Install ExpoGo and Expo so that you can run the mobile application from the computer.
   - Download and install Expo on your system.
     
     [https://docs.expo.dev/get-started/installation/][20]
     
     ‘Npm install -g expo-cli’
   - Install ‘Expo Go’ on your mobile device from the Play Store or App Store

2. Clone the Repository
   - To access the project you will need to navigate to a directory where you want to clone the repository.
   - Run the command ‘git clone https://github.com/blwinters/practice-10k.git’.

3. Install Dependencies for the application to run properly.
   - Install npm or yarn in your system [21] [22].
   - Run ‘npm install’ in the ‘mobile’ project directory to install all the necessary dependencies (located in your package.json).

4. Set up the environment file
   - To configure your project properly, you will need to create a .env file located in the ‘mobile’ directory. You can do so by creating a ‘.env’ file.
   - Add any necessary API keys to this file, such as your Supabase URL and Supabase Key, obtained from the project website.

5. Run the application on a local machine
   - Run the command ‘npx expo start’ to launch the app.
- For Android, scan the QR code using ExpoGo to access it on your mobile device.
- For iPhone, use the camera app to scan the QR code.

Table 3 shows the file structure for the mobile application. Once the github repository has been cloned, developers should see this file structure within the mobile folder.

*Table 3: Application File Structure*

<table>
<thead>
<tr>
<th>src</th>
<th>assets</th>
<th>images</th>
<th>blank-profile-picture.png</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>med-white-logo.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>small-black-logo.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>common_style.ts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>constants.ts</td>
</tr>
<tr>
<td></td>
<td>components</td>
<td>dropdowns</td>
<td>dropdown_calendar.tsx</td>
</tr>
<tr>
<td></td>
<td>components</td>
<td>dropdowns</td>
<td>dropdown_selector.tsx</td>
</tr>
<tr>
<td></td>
<td>headers_and_sections</td>
<td>app_header.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>headers_and_sections</td>
<td>edit_header.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>headers_and_sections</td>
<td>journal_details_header.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>headers_and_sections</td>
<td>practice_timer_header.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>headers_and_sections</td>
<td>profile_logo_section.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>practice_planner</td>
<td>add_practice_details</td>
<td></td>
</tr>
<tr>
<td></td>
<td>practice_planner</td>
<td>planner.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>practice_planner</td>
<td>view_plan_details.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stats_figures</td>
<td>goal_tracker.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stats_figures</td>
<td>music_distribution.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>helpers</td>
<td>index.ts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>helpers</td>
<td>index.ts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>helpers</td>
<td>utils.ts</td>
<td></td>
</tr>
<tr>
<td>Folder</td>
<td>Subfolders</td>
<td>Files</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>validate_user_data.ts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>📁 redux</td>
<td>actions.tsx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reducers.tsx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>store.tsx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>📁 screens</td>
<td>📁 auth</td>
<td>auth_navigation.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Login.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Register.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResetPassword.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start.tsx</td>
<td></td>
</tr>
<tr>
<td>📁 tabs</td>
<td></td>
<td>navigation.tsx</td>
<td></td>
</tr>
<tr>
<td>📁 services</td>
<td>📁 apis</td>
<td>authentication_api.tsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>data_management_api.ts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>📁 configs</td>
<td>supabase.ts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>📁 hooks</td>
<td>use_authentication.tsx</td>
<td></td>
</tr>
</tbody>
</table>

- The assets folder contains images and styles used within the project.
- The components folder contains components of the application that interact with the UI, and actions that the user can take such as adding practice plans, viewing plan details, using dropdown menus, and changing profile details.
- The helpers folder contains additional methods to validate user data or convert time formats.
- The redux folder contains the logic for maintaining global state management with files that define the redux actions, reducers, and store.
- The screens folder contains files that organize the assets and components in the project to render UI screens for user display.
- The services folder contains the logic for accessing the Supabase database and authentication through the use of API’s.

### 9.2 Metronome Setup

Developers should first install Projucer [4] and use it to open the Metronome.jucer file in the GitHub repository. Projucer will then prompt the user to pick an Integrated Development Environment (IDE) to continue with. Our team developed the metronome using MacOS and Xcode; we exported the project to both Xcode and Visual Studio, but have not successfully run it using Visual Studio due to our lack of familiarity with the IDE. Assuming the developer also chooses Xcode, open the metronome file seen in Table 4 in Xcode. Click the run button and wait for the metronome window to pop up.

*Table 4: Metronome File Structure*

<table>
<thead>
<tr>
<th>Source</th>
<th>assets</th>
<th>beep.mp3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🍎 IOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MainWindowIOS.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MainWindowIOS.cpp</td>
</tr>
<tr>
<td></td>
<td>Main.cpp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MainComponent.cpp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MainComponent.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MainWindow.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metronome.cpp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metronome.h</td>
<td></td>
</tr>
</tbody>
</table>

- MainComponent.cpp holds information related to the buttons, and both the sizing and layout of the window. Any changes to buttons’ behavior on click would be done here. Any additional layout changes should also be done in this file.
- Metronome.cpp contains the logic of the metronome’s timing, the minimum and maximum values of the metronome, and the audio file input. The minimum possible bpm is currently set to 10 and the maximum is set to 250. These can be changed via user input, which calls the updateSpeed function in the file.
- The audio file is currently placed in the assets folder named beep.mp3. Any additional audio files should also be added to the folder with a different name. Changes to the sound should also be reflected in Metronome.cpp for processing.
10. Acknowledgements

We would like to express our gratitude towards key individuals who have guided our team and contributed to the success of this project.

First and foremost, we are grateful to our clients, Dr. Irrera and Dr. Stevens, for the conception of the Practice 10k App. Their continuous feedback and support for the design of the application and features to implement were vital in directing our team’s progress this semester.

We would also like to thank Ben Winters, our technical lead, for providing valuable advice in organizing the overall project structure and leading us in the integration of the metronome and Supabase to the app.

Additionally, we thank Dr. Fox for mentoring our team and guiding our project timeline to ensure that we effectively achieve our milestones.

Lastly, we would like to extend our appreciation to the previous Capstone team who worked hard to develop the app from scratch and provide a starting base for us to continue improving upon.
11. References


