RepBuddy

Ethan James, Pushkar Seshadri, Jeeson George

Engineering Design and Development

Elkins High School

Professor Edwards

April 15, 2025

**Abstract**

The goal of RepBuddy was to be able to provide assistance to those in the gym who have a hard time remembering how many repetitions of an exercise they did. The ability to track repetitions of these exercises is due to Apple's Core Motion Library, which is able to track the acceleration

of your wrists. RepBuddy has been tested to proficiently track bench press (chest), bicep curls (arms), and squats (legs), to an extent.

**Table of Contents**



**Introduction**

Reading this report, one will learn about the key steps taken to develop RepBuddy and how the app was coded from start to finish. It walks through the planning process, the technical challenges faced, and how real-time decisions and improvisation shaped the final product. From

the initial layout to complex motion tracking, this explanation highlights the major components of the app and the thought process behind each one.

The purpose of this project was to create a smart fitness companion that could automatically track repetitions and provide real-time workout feedback on the Apple Watch. The activity focused on building practical skills in SwiftUI, HealthKit integration, and motion sensing using CoreMotion. Throughout the project, there were opportunities to learn how to structure apps, manage state across views, and work with real sensor data—translating abstract concepts into a working product

**Background**

What was first done was to compare the solution with other competitive products. The likes of the FitBit, Garvin, and Nike Training club all failed in the sense that none of them could solve the problem of RepTracking. This meant that creating an app like RepBuddy would be very challenging. Research also needed to be done on what resources could be used to create a product like this. Through the process, the Xcode application was discovered. Using Xcode helped us easily code an app that was simple to put on an Apple Watch and gave us direct access to Apple's libraries. Research was done on the Core Motion library, how to track it on an Apple Watch, and how to implement it on RepBuddy. So, it was found that there are no existing solutions and that there is a resource capable of doing what RepBuddy is sought out to do.

**Materials**

- Apple Watch Series 7
- MacBook Air

- Apple USB-C Cable
- XCode
- Apple Developer Program Membership
- Gym Equipment (Dumbbells, Barbells, Bench, etc.)

**Procedure**

Present the details about how you did the lab, activity, or project. Include your specific step-by-step process including sketches, schematics, equations, and photos. Specify what was learned at each step. This section may be broken into subsections.

The development of RepBuddy began with the creation of the Start Screen, which was used as the first screen that pops up when the user goes onto the app. This screen consisted of the app's logo, a message saying “Welcome!”, and a “Get Started” button. Using SwiftUI, the layout was structured with vertical stacking elements (VStack), and navigation to the next screen was implemented using a NavigationLink. This stage was relatively simple but provided foundational experience in SwiftUI's layout system and navigation principles.

Following the start screen, the Workout Type Screen was developed. This screen allowed users to select an exercise from a horizontal carousel-style scroll view. A TabView with page style was utilized to enable users to swipe between different workout options, including bicep curls, bench press, and squats. Each option was visually represented with an icon and a label, offering a clean and interactive selection experience. During this stage, the developer gained experience in user interface design and managing dynamic content presentation.

Next, two classes were created: the WorkoutManager and the MotionManager. Do not get fooled, this is not a new view, but it is essential for starting and tracking the workout on an

Apple Watch. The WorkoutManager acted as the core of the app's workout logic. WorkoutManager handled functions like starting, pausing, resuming, and stopping workout sessions. It was designed to ensure smooth tracking of time, calories burned, heart rate, etc. The MotionManager, on the other hand, focused on utilizing Apple's CoreMotion Library to track physical motion through the accelerometer. Various data sampling rates and update intervals were tested until the ideal numbers were found for each of the 3 exercises we used.

After the logic components were established, the developer returned to the Workout Type Screen to complete the navigation functionality. Once an exercise was selected, a NavigationLink would transition the user to the corresponding Workout Screen, carrying over the selected workout type. This required effective data passing between views and enhanced understanding of SwiftUI's data flow and view lifecycle.

The Workout Screen itself was the most complex part of the app, consisting of four main subviews:

1. SessionPagingView – Displayed real-time workout metrics such as calories burned and heart rate that is retrieved from Apple HealthKit. This involved requesting permissions, setting up health data queries, and handling live updates. This is the view that first pops up when the user starts a workout, but they can swipe left and right for the other 3 views.
2. RepView – Responsible for showing the number of completed repetitions. It was directly connected to the MotionManager and instantly updated as reps were detected.
3. MusicScreen – Allowed users to listen to music during workouts. Although simple in its early iteration, it enriched the user experience and introduced basic media player integration.

4. ControlsView – Contained the pause, play, and end session buttons. These controls were synchronized with the WorkoutManager to maintain consistent session behavior.

Each of these views communicated with shared data models, requiring the use of @EnvironmentObject and @StateObject to manage state and data across components. This phase deepened the developer's understanding of SwiftUI's data-driven architecture.

The next phase involved implementing the Summary View, which appeared at the end of a workout session. This screen summarized session statistics, including total reps, average heart rate, total calories burned, and duration. Data was aggregated from both the WorkoutManager and HealthKit and displayed in a user-friendly format. This step required proficiency in data formatting, including converting timestamps and numerical values into readable summaries.

The most challenging part of the project was refining the repetition tracking functionality. Within the MotionManager, custom algorithms were developed to accurately detect reps for bicep curls, squats, and bench press. This involved analyzing raw accelerometer data, experimenting with different acceleration thresholds, and designing logic to differentiate between valid reps and background movement. The developer conducted extensive testing—performing the exercises while debugging the data live—to identify patterns in motion that could reliably trigger a repetition count. This process significantly improved the developer's skills in sensor data interpretation, algorithm development, and real-time processing.

**Results**

We ran into issues with prioritizing tasks, A lesson that we learned is to prioritize designing the main feature of the app and have a clear list of priorities, ensuring that the resource heavy tasks

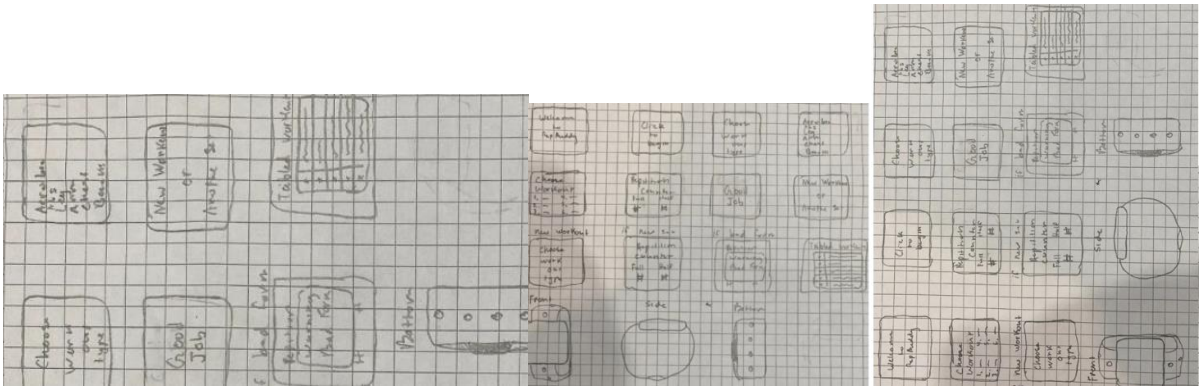
receive the resources they need. For the first few weeks we worked on the user interface and things that were not actually the main feature of the app. Though they are still very important to the function of the app, they do not have as much weight as the main selling point and does not demand as many resources such as time and are easier to build from a technical Standpoint. When we started coding the main feature, something that we did well was we took it one step at a time. We started by coding a basic exercise: bicep curls. We developed a working prototype and took it through testing and adjusted along the way. Then we moved on to another exercise that was like bicep curling movement so we could reuse the code. Since we are inexperienced in this field and our knowledge is limited, we had to start off with something simple and transfer the knowledge and experience we got from it to the building of something more complex. You can track simple and complex exercises with the same fundamental principles, the latter just has technical elements that need to be coded.

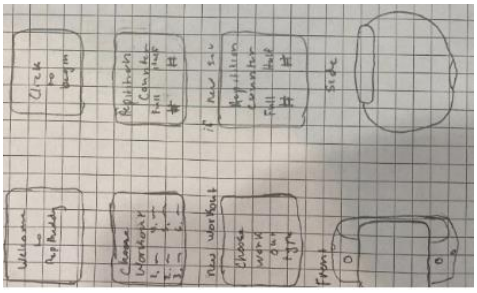
**Conclusion**

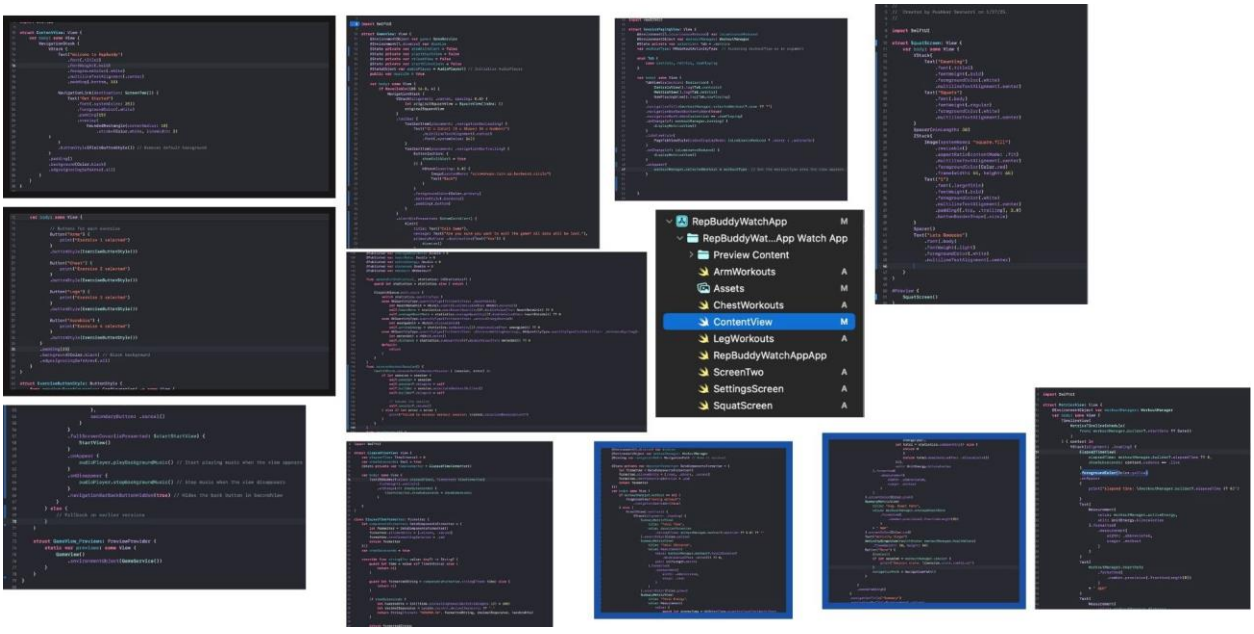
Tracking repetitions through an apple watch is very tricky, because there are many possibilities one must account for. For starters, what if people do really fast/slow reps. The truth is, everyone has their own form, with different speeds and different accelerations. RepBuddy hopes to track the ideal form someone should be doing. Developing RepBuddy helped provide a lot of insight on User Interface creation, programming in Swift, the amount of trials needed for experimenting, and most importantly, the challenge of repetition tracking.

**Appendix**

In the appendix, include the following:








## Citations

Apple. (n.d.). *Creating a watchOS app — SwiftUI Tutorials*. Apple Developer. Retrieved April 16, 2025, from https://developer.apple.com/tutorials/swiftui/creating-a-watchos-app/

Apple. (n.d.). *Core Motion*. Apple Developer Documentation. Retrieved April 16, 2025, from https://developer.apple.com/documentation/coremotion/

Gunawan, W. K. (2023, May 30). *Creating a simple watchOS app: Reminding users to stay active with Core Motion, WatchKit, and SwiftUI*. Medium. Retrieved April 16, 2025, from https://medium.com/@william.k.gunawan/creating-a-simple-watchos-app-reminding-users-to-stay-active-with-core-motion-watchkit-and-cb6b4b2ed1ed

Apple. (n.d.). *Setting up HealthKit*. Apple Developer Documentation. Retrieved April 16, 2025, from https://developer.apple.com/documentation/healthkit/setting-up-healthkit

Lijaya, H. (2023, June 15). *Tutorial using HealthKit in watchOS*. Medium. Retrieved April 16, 2025, from https://medium.com/@hendralijaya/tutorial-using-healthkit-in-watchos-93122239b43a

Kodeco. (n.d.). *watchOS with SwiftUI by Tutorials (2nd ed.)*. Retrieved April 16, 2025, from https://www.kodeco.com/books/watchos-with-swiftui-by-tutorials/v2.0