M2: Agile SRS

PROFESSOR RUSSELL FOUBERT | INFO2030 – Project Design

Coachable

Firas Areibi, William Bicknell, Moises Diaz, Jacob Funes, Blake Ribble

Dec 9th, 2019

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rev** | **Date** |  | **Description of Change** | **By** |
| 1.0 | 2019-10-27 | • | Initial revision of document | W. Bicknell |
| 1.1 | 2019-10-31 | • | Added interface information and software system attributes | W. Bicknell |
| 1.2 | 2019-12-05 |  | Final revision of document | B. Ribble |
| 1.3 | 2020-01-18 |  | Changed document to apply to Bluetooth | W. Bicknell |
| 1.4 | 2020-01-20 |  | Added UI Mock-ups | J. Funes |
| 1.5 | 2020-01-30 |  | Update hardware and added sections about mobile application | W. Bicknell |
| 1.6 | 2020-02-24 |  | Adding user diagram cases for the database | M. Diaz |

Table of Contents

[1. Introduction 4](#_Toc33473249)

[1.1 Purpose 4](#_Toc33473250)

[1.2 Scope 4](#_Toc33473251)

[1.3 Definitions, Acronyms and Abbreviations 4](#_Toc33473252)

[1.4 References 5](#_Toc33473253)

[1.5 Overview 5](#_Toc33473254)

[2. Overall Description 6](#_Toc33473255)

[2.1 Product Perspective 6](#_Toc33473256)

[2.1.1 System Interfaces 7](#_Toc33473257)

[2.1.2 User Interfaces 7](#_Toc33473258)

[2.1.3 Hardware Interfaces 12](#_Toc33473259)

[2.1.4 Software Interfaces 12](#_Toc33473260)

[2.1.5 Communication Interfaces 12](#_Toc33473261)

[2.1.6 Memory Constraints 12](#_Toc33473262)

[2.1.7 Operations 12](#_Toc33473263)

[2.1.8 Site Adaptation Requirements 12](#_Toc33473264)

[2.2 Product Functions 13](#_Toc33473265)

[2.2.1 Wearable 13](#_Toc33473266)

[2.2.1.1 Tracking Skiing Metrics 13](#_Toc33473267)

[2.2.1.1 Uploading Data 15](#_Toc33473268)

[2.2.2 Website 17](#_Toc33473269)

[2.2.2.1 Viewing Data as Guardian 17](#_Toc33473270)

[2.2.2.2 Viewing Data as Athlete 19](#_Toc33473271)

[2.2.2.3 Viewing Team Data as Coach 21](#_Toc33473272)

[2.2.2.3 Viewing Team Data as Head Coach 23](#_Toc33473273)

[2.3 User Characteristics 25](#_Toc33473274)

[2.4 Constraints 25](#_Toc33473275)

[2.5 Assumptions and Dependencies 25](#_Toc33473276)

[2.6 Apportioning of Requirements 25](#_Toc33473277)

[2.7 Software System Attributes 25](#_Toc33473278)

[2.7.1 Reliability 25](#_Toc33473279)

[2.7.2 Availability 25](#_Toc33473280)

[2.7.3 Security 26](#_Toc33473281)

[2.7.4 Maintainability 26](#_Toc33473282)

[2.7.5 Portability 26](#_Toc33473283)

# 1. Introduction

## 1.1 Purpose

The purpose of this report is to document the requirements of the Coachable Wearable. The Coachable Wearable is meant for tracking a user’s skiing metrics. Coaches will use the metrics to get a better understanding of what areas their athletes need to work on. There are many devices and applications that track skiing metrics, but most are designed for self-improvement and not for situations were a coach is involved. Which is the market niche that the Coachable Wearable will fill.

Another market niche that this device will fill is competitive skiing and snowboarding. This is because many rulesets do not allow athletes to carry their mobile phones or wear accessories on their upper body. Since the Coachable Wearable will attach to the boot of the athlete, they will be able to track their metrics while competing.

## 1.2 Scope

This wearable product will track metrics such as distance travelled, velocity, altitude, run duration, and number of runs. By allowing consistent and accurate tracking of these metrics, coaches and athletes will be able to track their progress with real physical data instead of guess work. The wearable will upload the data to Coachable’s API. Our team will also create a website and a mobile application that will display the uploaded metrics.

## 1.3 Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| IEEE | The Institute of Electrical and Electronics Engineers |
| CW | Coachable Wearable |
| RAM | Random Access Memory |
| EEPROM | Electrically Erasable Programmable Read-only Memory |
| CPU | Central Processing Unit |
| API | Application Programming Interface |
| REST | Representational State Transfer |

Table 1 – Definitions, Acronyms, Abbreviations

## 1.4 References

Adafruit. (2020). *Adafruit Ultimate GPS Breakout.* Retrieved from Digi-Key: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/746\_Web.pdf

IEEE Software Engineering Standards Committee. (1998). *IEEE Recommended Practice for Software Requirements Specifications.* New York: IEEE.

RobotShop. (2020). *MPL3115A2 I2C Barometric Pressure/Temperature/Altitude Sensor Breakout Board*. Retrieved from RobotShop: https://www.robotshop.com/ca/en/mpl3115a2-i2c-barometric-pressure-temperaturealtitude-sensor-breakout-board.html

SET Faculty. (2015, March 1). *SET Guidelines for Documenting Requirements in an Agile Way.* Retrieved from eConestoga: https://conestoga.desire2learn.com/content/enforced/62087-D2LGroup-0059\_P00\_X\_X\_D2LGroup-0059\_1\_X/Standards/Agile/SET-Agile-SRS-Guideline.pdf

SparkFun. (2020). *SparkFun ESP32 Thing*. Retrieved from SparkFun: https://www.sparkfun.com/products/13907

SparkFun. (2020). *SparkFun ESP32 Thing Motion Shield.* Retrieved from SparkFun: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/DEV-14430\_Web.pdf

## 1.5 Overview

The rest of the SRS contains the overall description of the product, with more in-depth information with regards to how the product will function and what feature will be included along with Product Perspective, User Interfaces, System Interfaces, etc.

# 2. Overall Description

## 2.1 Product Perspective

This section details the variety of user, hardware, software, and communication interfaces that our system is using to perform the required task of collecting skiing metrics. The system comprises of a wearable device and a website that displays the recorded metrics that have been sent to the Coachable API.

Below are two of the closely-related competitors that have dominance in the market. These were found during our preliminary investigation (Stardust Crusaders, 2019):

Competitor name: **Flaik**

Type of wearable: Arm Band

URL: <https://www.flaik.com/en/>

Summary: Flaik provides a GPS tracking solution designed to allow instructors to track their students. The instructor and students each wear a GPS arm band that sends real-time GPS tracking data to a tablet device that the instructor carries. The tablet device comes with a QR code scanner that is used to add the GPS arm bands to the session. The tablet device shows a map of where the students are, which allows the instructor to know if a student has gone off course. The tablet device can also replay the lesson of a single student to review what happened.

Differences: As we have learned during research and discussion with industry partners, professional skiers aren’t allowed to wear bands around their arms. Our device would be feasible as it can clamp onto the boot and it doesn’t rely on a tablet for tracking. This would be important if you wanted to ski by yourself.

Competitor name: **Ski Tracks**

Type of wearable: Mobile Application

URL: <https://www.corecoders.com/ski-tracks-app/>

Summary: The application is compatible with Apple Watch, Android Wear and the Pebble. Its focus is to track metrics like ski distance, altitude, max speed, ski vertical, number of runs, slope, and duration.

Differences: Again, the professional cannot have their phone/wristband on when participating in events.

### 2.1.1 System Interfaces

The wearable final product will not depend on any external systems because we will be creating our own system to track the important metrics of a user’s skiing.

### 2.1.2 User Interfaces

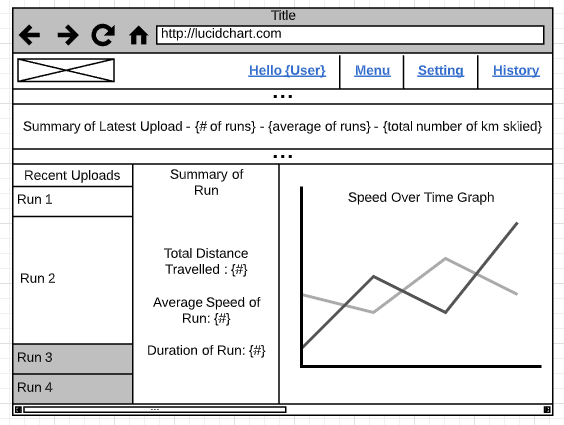
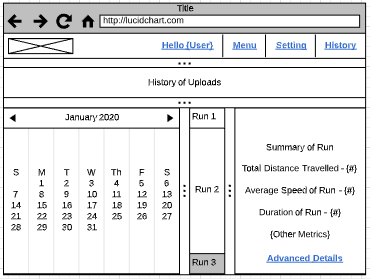
The wearable device will not directly have a user interface. However, a website and a mobile application will provide a user interface to review the collected metrics that have been sent to the Coachable API.

Figure 1 - UI Mockup for Website Home Page

Figure 2 - UI Mockup for Upload/Run History Page

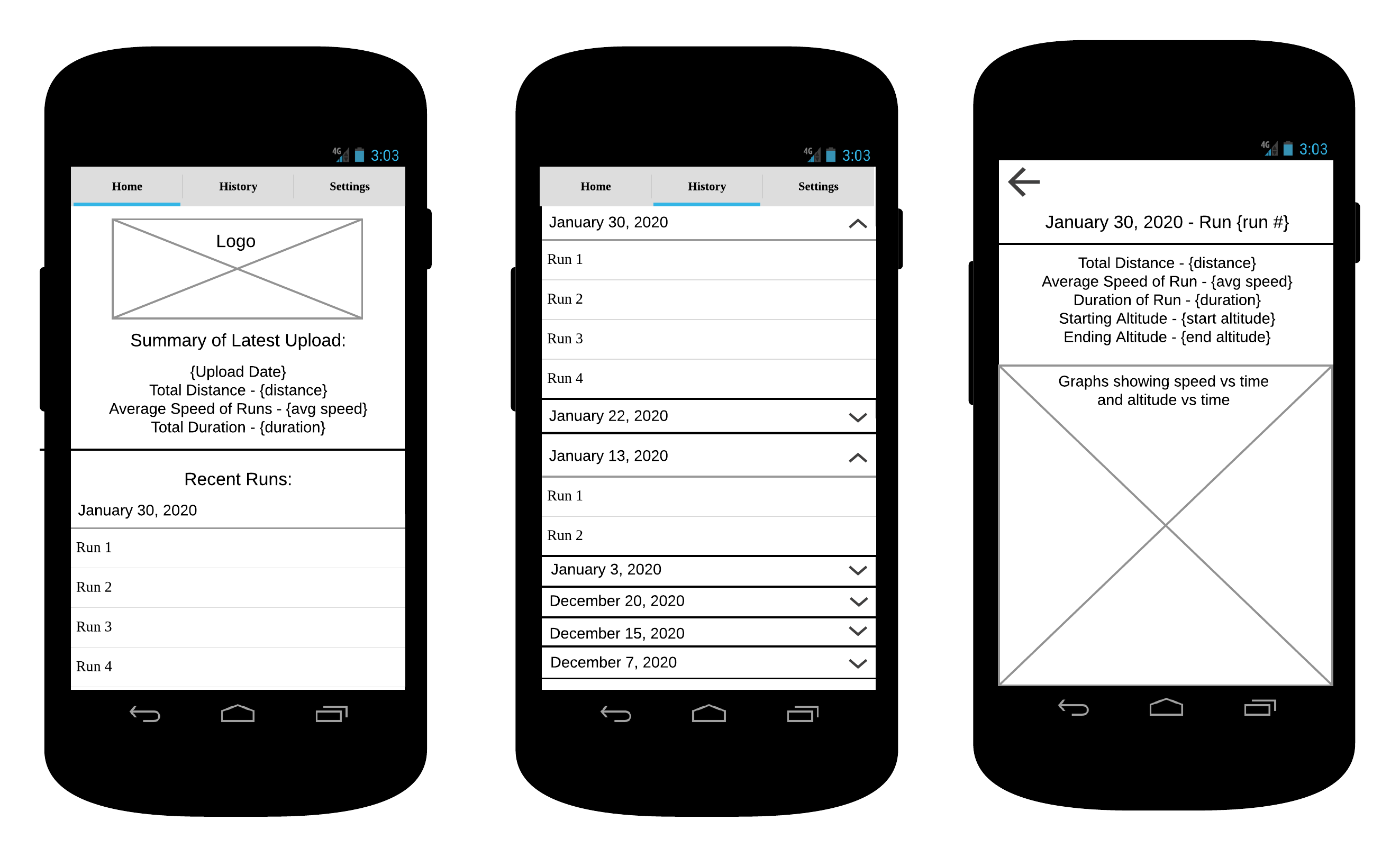
Figure 3 - UI Mockup for Mobile App Home Page

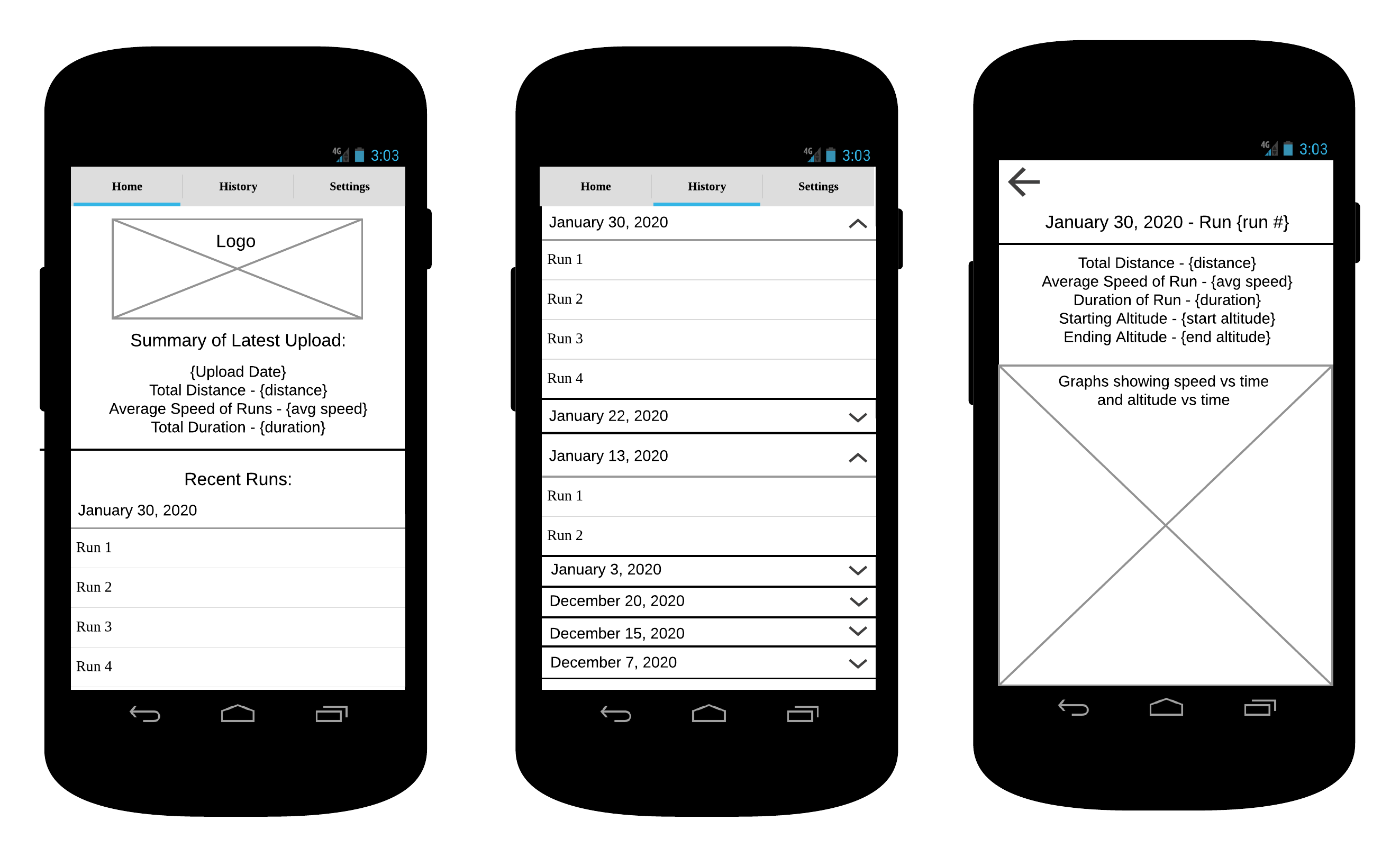
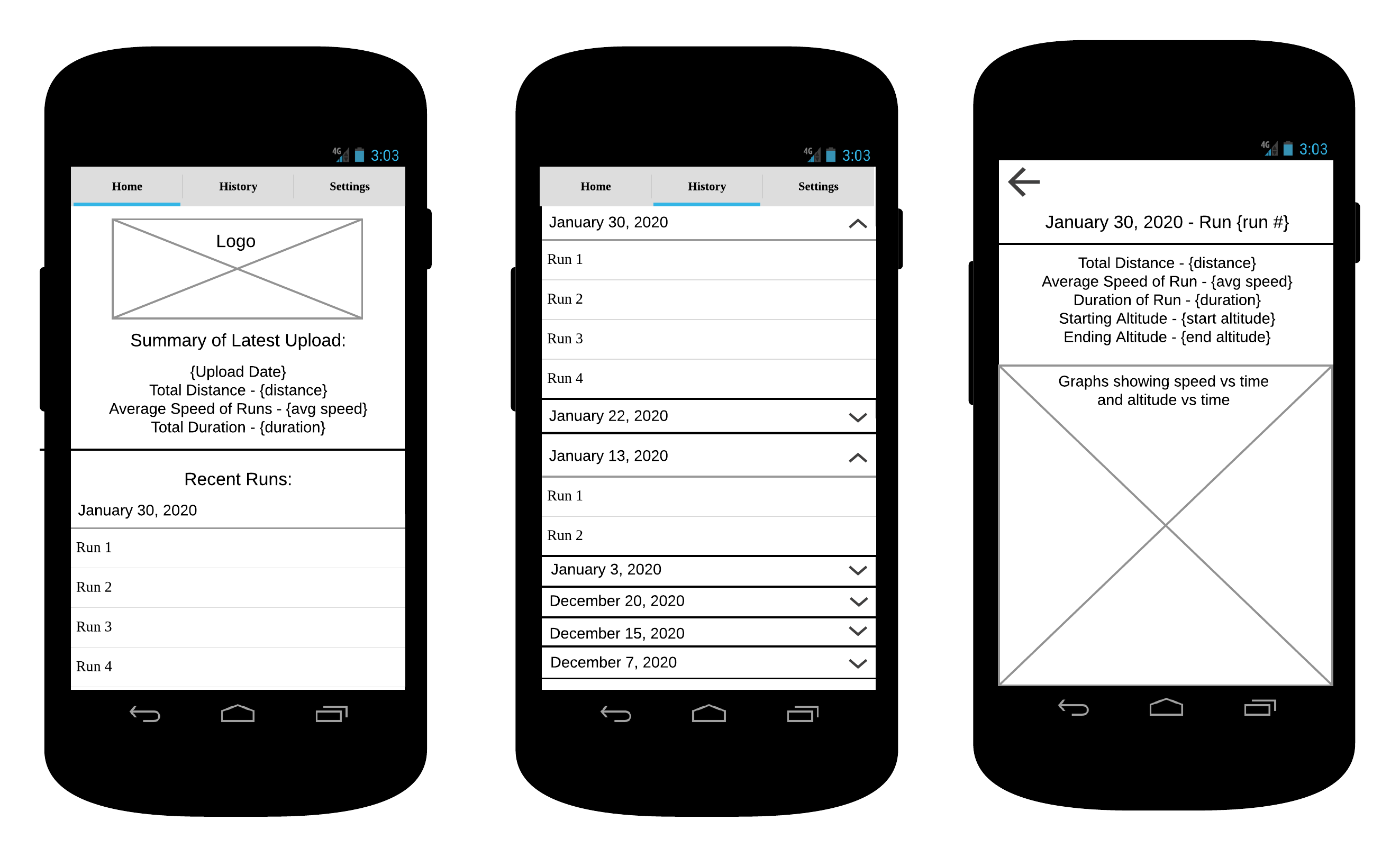
Figure 4 - UI Mockup for Mobile App Run History Page

Figure 5 - UI Mockup for Mobile App Run Info Page

### 2.1.3 Hardware Interfaces

The wearable product will run on an embedded board and will read sensors to obtain the data needed to generate the metrics being tracked.

The selection for a board is the SparkFun ESP32 Thing. The board comes with built-in Bluetooth and WIFI capabilities, integrated LiPo battery charging, 4MB flash memory, and 520kB SRAM. (SparkFun, 2020)

The selection for the first sensor is the SparkFun ESP32 Thing Motion Shield. This sensor tracks angular velocity, acceleration and heading. It also has a micro SD slot and a JST port for connecting a GPS. (SparkFun, 2020)

The selection for the second sensor is the Adafruit Ultimate GPS Breakout. This GPS has -165 dBm sensitivity, 10 Hz updates, 66 channels, and a built-in antenna. (Adafruit, 2020)

The selection for the final sensor is the MPL3115A2 I2C Barometric Pressure/Temperature/Altitude Sensor Breakout Board. This sensor will primarily be used to track altitude. (RobotShop, 2020)

### 2.1.4 Software Interfaces

The product will interact with a REST API created by Coachable in order to send data to Coachable’s current product. The wearable will communicate using Bluetooth tethering to a mobile device that has a WIFI connection.

### 2.1.5 Communication Interfaces

The wearable will use HTTPS to communicate with Coachable’s REST API. The wearable will verify and send the data to the REST API.

### 2.1.6 Memory Constraints

The memory constraints for the wearable are directly linked to the hardware that is used. This means with our current board selection, the SparkFun ESP32 Thing (mentioned in Section 2.1.3), the initial memory constraints would be 4MB flash memory and 520kB SRAM (SparkFun, 2020). Since we will also be using the SparkFun ESP32 Thing Motion Shield (also mentioned in Section 2.1.3), we will also have access to a micro SD slot allow us to add more flash memory as needed (SparkFun, 2020).

Overall, because of the micro SD slot on the SparkFun ESP32 Thing Motion Shield, running out of memory should not be an issue.

### 2.1.7 Operations

The final product will run on a default mode since there will only be one type of user that will use the product. Since the wearable only collects skiing metrics and sends the metrics to Coachable’s ski coaching product, there is no need for multiple user types.

### 2.1.8 Site Adaptation Requirements

Since the wearable is an embedded system, it will need to run on its own without any external dependencies. Also, the accompanying data viewing website should only depend on the Coachable API to retrieve the data that will be displayed.

## 2.2 Product Functions

With the wearable device the coaches will be able to see the wide variety of metrics detailing how each run went. The metrics that will be displayed are run duration, max speed, max altitude, distance travelled, average speed, number of laps, and fastest run. These metrics will help provide the coach with insight on what the student needs to improve on to get better.

### 2.2.1 Wearable

#### 2.2.1.1 Tracking Skiing Metrics

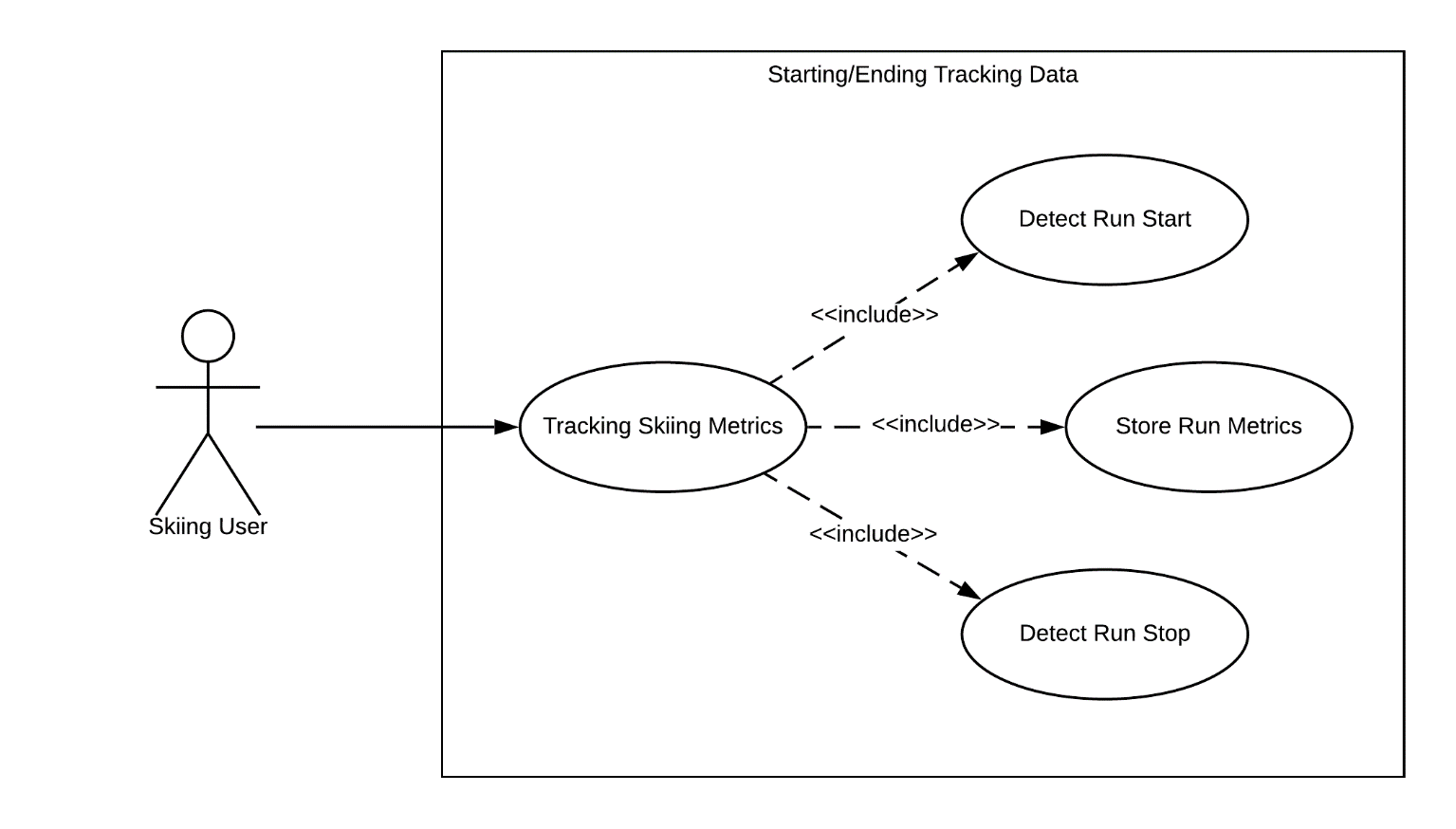
This section describes the use case of the user interacting with the wearable device to start/end tracking user statistics. When the user turns on the device, the tracking system begins. This system detects the start of a run, gathers metrics for the duration of the run, and detects the end of the run.

Figure 6 – UC Diagram for Tracking Metrics

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Track skiing metrics | |
| Scenario: | Track skiing metrics while the user is skiing | |
| Triggering Event: | User turns on the wearable. | |
| Brief Description: | Track a user’s skiing metrics. This includes determining when a run starts and ends. The user must turn the device on before they begin and turn it off after they finish skiing. | |
| Actors: | Skiing User | |
| Related Use Cases: | Upload skiing metrics, View skiing metrics | |
| Flow of Activities: | Actor | System |
| 1. User turns on the device  2. User begins skiing  3. User finishes a skiing run  4. User turns off the device | 2.1 Wait to detect skiing has started (high altitude, altitude descending, wearable angled downwards, velocity building)  2.2 Record the data during the run  3.1 Detect when the user has completed a run (velocity lowers, wearable returns to level angle, altitude stops descending)  3.2 Save the data from the run |
| Exception Conditions: | 1.1 If the wearable gets turned on but the user is not skiing, detect that the wearable is not being used and preserve power.  4.1 If the device gets turned off while recording a run, scrap the incomplete data set | |

Table 2 – UC Description for Tracking Metrics

#### 2.2.1.1 Uploading Data

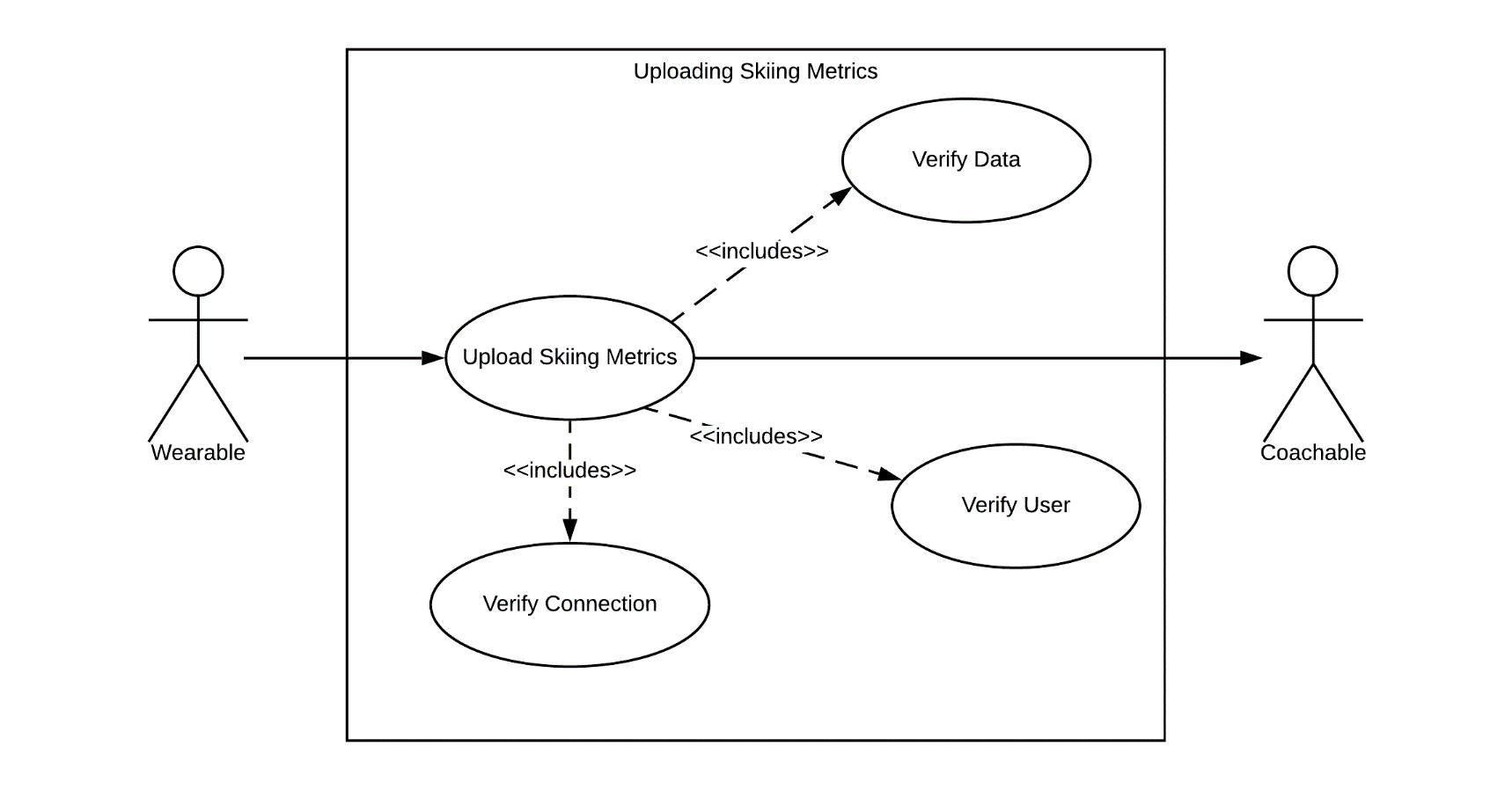
This section describes the use case of uploading data from the wearable device to the Coachable API. When the device is Bluetooth tethered to a mobile device that has a WIFI connection, the wearable attempts to verify the user to ensure they are a valid member of Coachable. After this, the data itself is validated and if successful, the metrics are sent to the Coachable API.

Figure 7 – UC Diagram for Uploading Data

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Upload skiing metrics | |
| Scenario: | The user is done skiing and wants to upload their metrics to Coachable | |
| Triggering Event: | User Bluetooth tethers the wearable to a mobile device that has a WIFI connection. | |
| Brief Description: | Upload a set of recorded skiing metrics to Coachable. | |
| Actors: | Skiing User | |
| Related Use Cases: | Track skiing metrics, View skiing metrics | |
| Flow of Activities: | Actor | System |
| 1. User Bluetooth tethers the wearable to a mobile device that has a WIFI connection | 1.1 Read the skiing data from the wearable  1.2 Verify the user’s identity to ensure data is uploaded to the correct user  1.3 Send the data to the Coachable REST API  1.4 Remove the data from the wearable |
| Exception Conditions: | 1.2.1 If the user’s identity cannot be verified, notify the user and cancel the upload  1.3.1 If the skiing data is invalid or corrupted, notify the user and remove the invalid/corrupt data  1.3.2 If a connection cannot be established with the Coachable REST API, notify the user and cancel the upload | |

Table 3 – UC Description for Uploading Data

### 2.2.2 Website

This section describes the use cases of the user interacting with the website.

#### 2.2.2.1 Viewing Data as Guardian

This section describes the use of a guardian viewing their children’s data in the Coachable database. After being verified by the site, the system shall return all relevant information such as their children’s run metrics, team and event information.

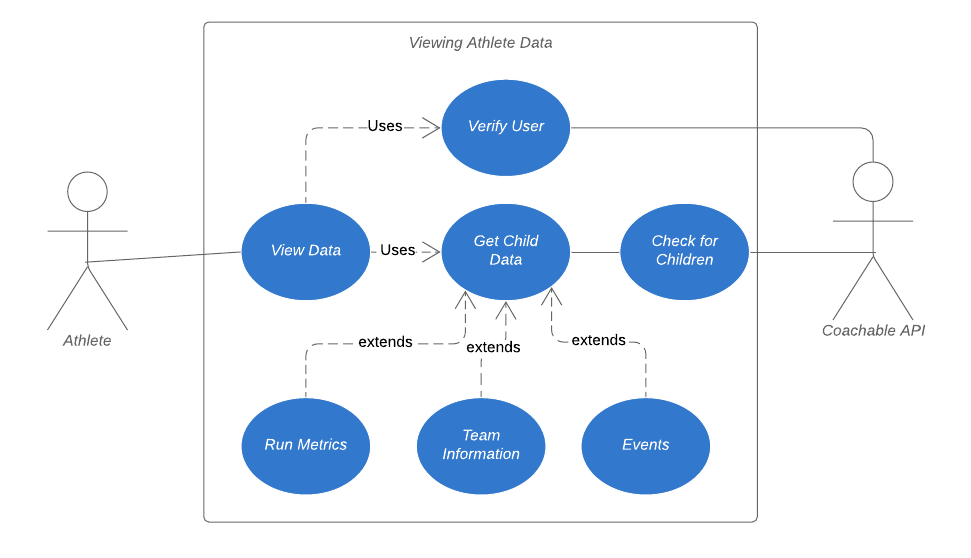


Figure 8 – UC Diagram for Viewing Data as Guardian

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Viewing Data as Guardian | |
| Scenario: | Guardian wants to check their children’s skiing information | |
| Triggering Event: | Guardian attempts to login to the website | |
| Brief Description: | Guardian wants to check their athlete’s information. | |
| Actors: | Guardian | |
| Related Use Cases: | None | |
| Flow of Activities: | Actor | System |
| 1. Guardian attempts to log into the website  2. Guardian waits until their information is verified  3. Guardian views data presented | 2.1. Verify if user is in the database.  2.2. Obtain all children’s information and return to the client. |
| Exception Conditions: | 1.1 If the guardian does not have internet connection.  1.2 If the guardian is not registered on the database.  1.3 If the database is down. | |

Table 4 – UC Description for Viewing Data as Guardian

#### 2.2.2.2 Viewing Data as Athlete

This section describes when an athlete wants to check their statistics. Upon the system verifying the user, they will be presented with all relevant data such as run metrics, team information, seasonal information, and event information.

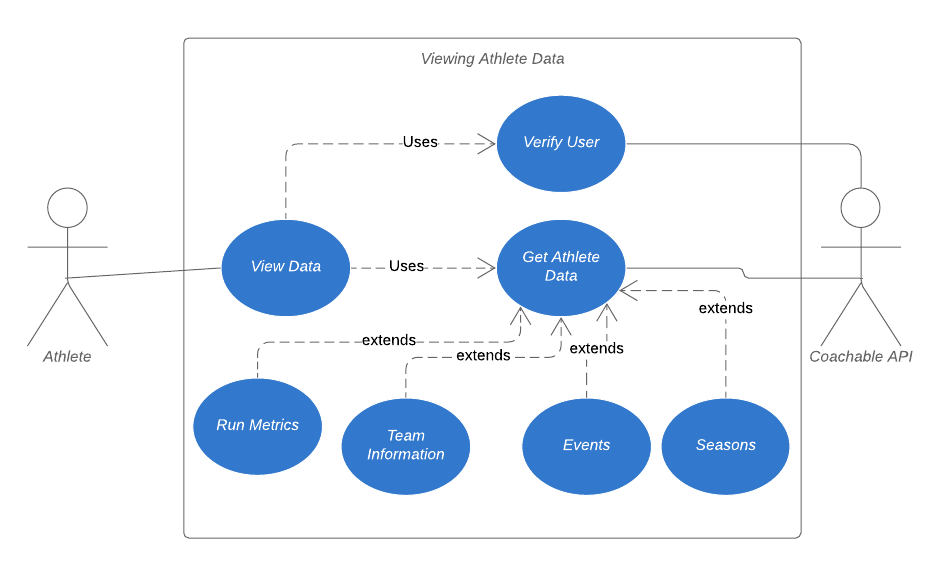


Figure 9 – UC Diagram for Viewing Data as Athlete

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Viewing Data as Athlete | |
| Scenario: | An athlete wants to check up their general information | |
| Triggering Event: | An athlete attempts to login to the website | |
| Brief Description: | Athlete wants to check up their metrics and general information | |
| Actors: | Athlete | |
| Related Use Cases: | None | |
| Flow of Activities: | Actor | System |
| 1. Athlete attempts to log into the website  2. Athlete waits until their information is verified  3. Athlete views data presented | 2.1. Verify if user is in the database.  2.2. Obtain all athlete’s information and return to them. |
| Exception Conditions: | 1.1 If the athlete does not have internet connection.  1.2 If the athlete is not registered on the database.  1.3 If the database is not available / down. | |

Table 5 – UC Description for Viewing Data as Athlete

#### 2.2.2.3 Viewing Team Data as Coach

This section describes the scenario where a coach wants to login and check all their team’s information. After being verified, the system shall search all teams that the coach is apart of and return their information such as members, events, and seasons.

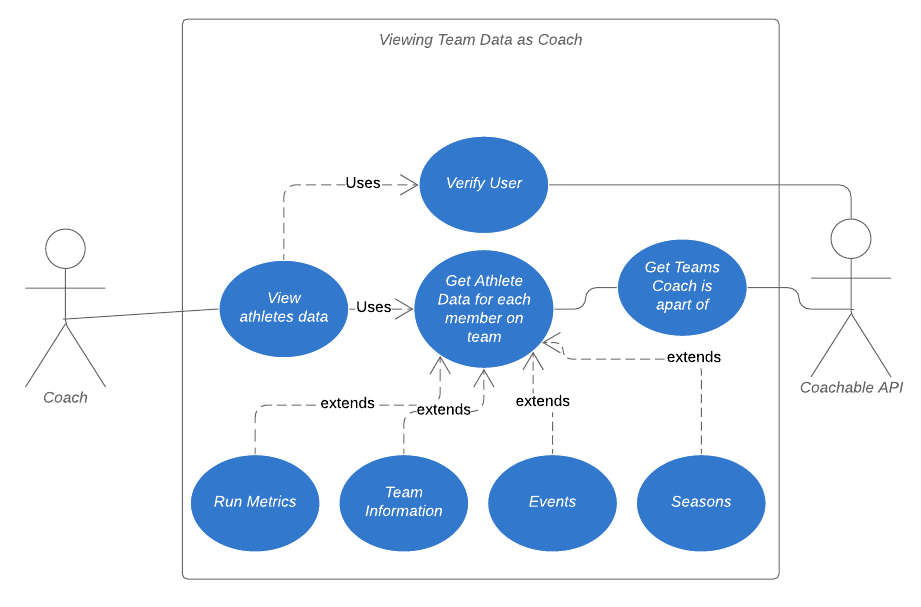
****

Figure 10 – UC Diagram for Viewing Data as Coach

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Viewing Team Data as Coach | |
| Scenario: | A coach wants to check up their team general information | |
| Triggering Event: | Coach goes to the website to check meaningful information about their team members | |
| Brief Description: | Coach wants to check up their metrics, season, events and general information | |
| Actors: | Coach | |
| Related Use Cases: | None | |
| Flow of Activities: | Actor | System |
| 1. Coach logs into the website  2. Coach waits until their information is being verified  3. Coach can now view information such as metrics, team, events depending on the season, plus team members list | 2.1 Detecting role on the database  2.2 Database checks to see which teams the coach is on, then returns all athlete data associated with each member. |
| Exception Conditions: | 1.1 If the coach does not have internet connection.  1.2 If the coach is not registered on the database or does not have that specific role.  1.3 If the database is not available / down. | |

Table 6 – UC Description for Viewing Data as Coach

#### 2.2.2.3 Viewing Team Data as Head Coach

This section describes when the head coach wants to check meaningful and important information about the organization. After the verification stage, the system shall return key information like events and seasons happening within each team in their organization.

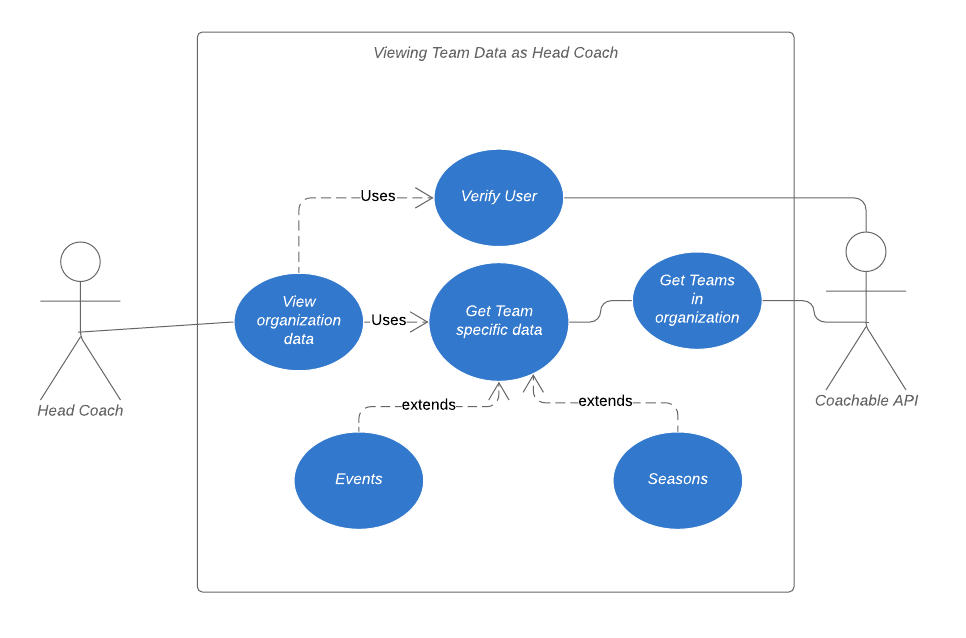


Figure 11 – UC Diagram for Viewing Data as Head Coach

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Viewing Data as Head Coach | |
| Scenario: | A head coach wants to check up general information about their organization | |
| Triggering Event: | Head coach attempts to login to website | |
| Brief Description: | Head coach wants to check up their seasonal events made by their organization | |
| Actors: | Head Coach | |
| Related Use Cases: | None | |
| Flow of Activities: | Actor | System |
| 1. Head coach logs into the website  2. Head coach waits until their information is being verified  3. Head coach can now view information such as teams in org alongside their events and seasons. | 2.1 Detecting role on the database  2.2 Database checks to see which teams are in the organization ran by the head coach and returns specific data to user. |
| Exception Conditions: | 1.1 If the head coach does not have internet connection.  1.2 If the head coach is not registered on the database or does not have that role.  1.3 If the database is not available / down. | |

Table 7 – UC Description for Viewing Data as Head Coach

## 2.3 User Characteristics

Since the product is designed for people that practice skiing at beginner, professional, and entertainment levels, we cannot make assumptions about the user’s education level or technical expertise. This is because, the user could be anyone from a child beginning to learn to ski to an athlete trying to improve their skiing performance.

## 2.4 Constraints

The main constraint for the wearable will be form factor. Since the users will be skiing with the device attached to their boot, we will need to keep the wearable small and light to avoid hindering the user’s skiing.

## 2.5 Assumptions and Dependencies

If a switch in development occurs, this document will be updated and republished. That being said, it is assumed that the user will also have access to a mobile device to connect to the wearable by Bluetooth tethering in order to upload their data. The project relies on the acquisition of hardware and sensors along with the safekeeping of said hardware and sensors. A major dependency is the act of testing, some costs may be incurred to get proper tests done. If the hardware and sensors are not sent on time, this will create a backup of scheduled milestones. We must make sure we get the required hardware and sensors within reasonable time.

## 2.6 Apportioning of Requirements

At this current time, we will be apportioning the creation of the housing of the wearable, as this goes into a field that we are currently not trained to deal with; but given time, housing for the device will be a future release.

## 2.7 Software System Attributes

### 2.7.1 Reliability

The wearable product will work in high altitudes, under cold conditions, and needs to avoid condensation when transitioning between a cold environment to a warmer environment. In addition, the battery life will need to be very good to work for the entire time the user is skiing.

### 2.7.2 Availability

Since the wearable will be collecting skiing metrics and writing them to memory, if the device were to lose power all currently written data would be available when power is regained. The device will be able to immediately begin collecting data as soon as the device is turned on and collection requirements are met (ex. altitude begins decreasing).

### 2.7.3 Security

The final product will send the data to the REST API, and the user will need to Bluetooth tether the wearable to a mobile device with a WIFI connection. We will need to ensure that there is no way malicious data will be sent through the API.

Modification of the product will be advised against, as is uploading data to the wearable itself for malicious purposes.

### 2.7.4 Maintainability

Since the wearable will be an embedded system, being able to perform maintenance is likely not an option. The software will need to be designed with proper error and exception handling practices. Also, extensive testing will be required to ensure that all possible errors and exceptions are accounted for.

### 2.7.5 Portability

Due to the wearable being an embedded system, porting the software to another system may require changes. First, the software will be written in the MicroPython language so the other system would need to be able to execute MicroPython code. Second, if different sensors are used, the methods used to retrieve data from the sensors may need to be changed to comply with the new sensors. However, the portion of the software that manages how often data is collected from sensors, and handles storing the data would need very few changes since it will be designed with portability and scalability in mind.

Overall, porting to another system is unlikely to occur but could be done as long as the new system supports MicroPython code. The only required changes would be dependent on what sensors are used in the new system.