LogiSteps Final Report
May 17th, 2019

Mitchell Larson
Gunther Huebler
James Windorff
Zachary Oberbroeckling
Brandon Reed

Advised by Dr. Gerald Thomas

Table of Contents

1. Project Summary

1.1 Project Overview

LogiSteps is a personal fitness data collection device that is housed inside the sole of a shoe. The data collection and transmission components are housed inside the sole and the device is supplied power through a passive power supply originating from piezo-electric sensors that are placed in the front and back of the sole. Step data is collected and transmitted over Bluetooth to a smart phone that has an app to collect and display data. The data is then further transmitted to a web server that displays the data in more advanced ways. The web display can show foot pressure, various displays for step counts, and location maps.

Team Omicron decided to pursue this project based on both the customer size and the problems that it would solve. LogiSteps is based on the personal fitness tracking industry which is an enormous industry that is expected to continue growing. In addition, LogiSteps was based on solving problems with current personal fitness technology including inaccurate data collection, poor user experience, and most importantly, having to supply power. The LogiSteps device aims to solve these problems by collecting data with the sole as well as passively providing power with the user's own movement so that the user doesn't have to recharge the device.

1.2 Team Accomplishments and Deliverables

Fall 2018:

- Week 2: Project Proposal
- Week 5: Technology and Concept Investigation Report
- Week 7: System Requirements and Interface Report
- Week 10: Introductory presentation

Winter 2018-2019:

- Week 2: Prototype for insole
- Week 6: Prototype for microcontroller
- Week 6: Design documentation for all components
- Week 7: Functionality for mobile application
- Week 9: Working web application
- Week 10: Status update presentation

Spring 2019

- Week 3: Power optimized and shoe design finalized
- Week 4: Bluetooth connection from phone to microcontroller
- Week 5: Full integration of all components
- Week 6: Web server online

- Week 10: Final report and poster
- Week 11: Senior design show presentation

1.3 Plan Summary

Section 1.2 describes the various milestones that our group had, as well as the time of completion. These milestones were measurable goals that greatly helped our project stay on pace, as well as allow us to keep track of progress. Overall, this project plan was followed well with only a couple exceptions of milestones being turned in late. Even when milestones were late, they were still turned in within a week of the project plan so that the project could stay on track.

In terms of project time expended, each team member was expected to contribute 10 hours a week. Throughout the project, there were weeks in which members did less than 10 hours, but there were just as many weeks in which members did more than 10. Therefore, the average time spent on the project per week was near 10 hours. This adds up to 50 hours per week for the whole team, and since the project ran for 30 weeks (not including time spent over breaks), the overall time expended on the project for the last year was 1500 hours. This is equal to 62.5 days of work, or nearly 9 weeks of work without breaks.

1.4 Engineering Problem

1.4.1 Business case

LogiSteps has based its business case upon the fact that no other product on the market is able to provide wearable devices that are powered by the user's movement. The wearable devices industry is a large industry, that is continuing to grow, and is projected to grow by 4.2% in the next year. Providing a product that is easier to use than any other product on the market should allow LogiSteps to gain market share and obtain healthy amounts of revenue.

In addition to this, LogiSteps aims to provide a better user experience to consumers by increasing the accuracy in which data is collecting. Existing products on the market use accelerometers in other parts of the body to estimate steps. By moving the data collection process to the shoe, LogiSteps will be able to provide data that is more accurate and provides a better picture of a user's true fitness.

LogiSteps believes that by creating a device that a user can place into their shoe, without the need for removal at any time to recharge the device, users will be inclined to choose LogiSteps over competitors because of its ease of use. Users simply purchase the smartsoles, do an initial Bluetooth pair with their mobile device, and allow the devices to work on their own. This, in combination with the potential for increased levels of accuracy gives LogiSteps a step up on its competition in an industry where 120 million wearable devices were sold in 2017 alone.

1.4.2 Project context

LogiSteps aims to be a long-lasting product by making use of relatively new technology. The project makes use of piezoelectric sensors as well as pressure sensors which are both new enough to not become outdated soon, but also old enough to be purchased within our budget. The mobile application

as well as the web application both make use of current development tools to create applications that will be supported in the future.

Legal context is also important for the LogiSteps project, as the physical device needs to meet health and safety standards. To ensure safe use of the physical component, all electrical devices and components are housed inside of the insole. This is to isolate any electrical hazards from the user, as well as shielding from environmental conditions.

1.4.3 Background domain information

In order to begin working on the LogiSteps project, the team needed to learn more about the background domains relating to the project. Specifically, fitness trackers and shoe insoles had to be studied so that the LogiSteps team could create and implement both things.

First, fitness trackers had to be researched. Various other already existing fitness trackers were researched to determine how to record data, as well as what kind of information should be displayed for the LogiSteps web application. After looking into other fitness trackers, it was decided that the web application should display a recent chart, a daily chart, a weekday breakdown, weekly activity, a pressure map, and a location map.

Secondly, shoe insoles had to be researched. Research was done to determine the average shoe size for both males and females, and the LogiSteps insole had to be sized to fit the components into an average shoe size. In addition, shoe insoles had to be researched to determine what materials could be used to create a comfortable shoe insole.

1.5 Skills and Constraints

The LogiStep development team was diverse in its skill sets, and to complete the project in an efficient manner, project components were split up according to developer skills. The LogiSteps insole itself utilizes previous knowledge from mechanical courses in making the design for the shoe as well as from electrical courses for creating the energy harvesting circuit and providing power. The data collection component of the project is done with a microcontroller, which requires knowledge on working with an embedded system which was provided through multiple computer hardware courses. The mobile application portion of the project requires software knowledge, specifically with Android, and this was learned through various software courses such as Software Tools & Practices and Software development 1-2. Finally, the web application portion of the project requires extensive knowledge of web infrastructure, and this was developed through various internships. To summarize, LogiSteps leveraged previous knowledge from electrical, software, and computer hardware courses, allowing for a variety of different areas to work on.

LogiSteps is also able to incorporate realistic constraints in the project. Since LogiSteps is aimed to be a marketable product, there is an economic constraint on the project. The project aims to spend as little money as possible on the various components so that it can later be sold at a reasonable price. In addition, since the LogiSteps insole is a physical device, it has environmental, sustainability, manufacturability, and safety constraints. The LogiSteps insole must remain sustainable in its expected

environment, as well as in any natural extreme environment (cold, hot, wet, etc.). The insole also must be safe and easy to manufacture as well as always being safe for the user in all possible conditions. These constraints are described in much more detail in the System Requirements and Specifications section.

2. Project Proposal

2.1 Project Description

2.1.1 Intro

Team Omicron has decided to formally pursue a project involving the creation of a personal fitness data collection device which is to be housed in the sole of a shoe. Yet to be determined is the whether all the data collection and transmission will be placed in the sole of the shoe, or if the data transmission hardware will be housed outside the sole. This device will be supplied with power through passive power supply originating from piezo-electronic materials. Data will then be transmitted over Bluetooth from the data collection device to a smart phone running a service to further transmit the data to a remote server. This server will serve a web page for users to view their data. Team Omicron aims to track foot pressure, steps, and weight through the use of this device. Other possible data attributes may include stride length, distance, and cadence.

Initial market research has indicated that the fitness wearables industry is experiencing strong growth, with a forecast for this to continue. While the smartwatch segment has seen lots of competition the past few years, fitness tracking devices which are embedded into the shoe have just begun to show up. Due to the massive size of this market, its predicted growth, and the relatively untapped “smart shoe” market segment, the wearables (for personal fitness) use case has been chosen for this project. The proceeding document will provide greater insight into the project proposal.

2.1.2 Customer/User

Team Omicron originally identified several possible customers and/or users that could use a fitness tracking device embedded into the sole of a shoe. Some of the possibilities consisted of:

- Runners
- Pedometer users
- Health Insurance
- Companies (for liability protection)
- Medical Professionals (for back/joint diagnosis)
- Personal Health

After performing research into each market, the team found that several of the categories could be collapsed into a single category. That is, runners, pedometer users, and personal health users could all be collapsed into a single category named “Personal Fitness”.

This left narrowed the decision to a factor of 3, all of which showed strong potential. All 3 industries had hundreds of millions of users globally, with total revenues in the billions of dollars. Due to the ease of entry into the personal fitness market, this was the user the team decided the product would best serve.

2.1.2.1 Market Size

As mentioned earlier, the wearables industry for personal fitness tracking is an enormous industry. Several sources indicate that hundreds of millions of users purchase personal fitness wearables annually, which translates to revenues in the billions of dollars. In summary, market research found:

- 125.5 million wearable devices were sold in 2017 (Lamkin, 2017)
- Smart clothing shipments (including "smart shoes") are forecasted to increase by a total of 3.3 million (21.6%) by the year 2021 (Lamkin, 2017)
- The expected annual growth rate is 4.2% (Statista, 2018)
- The average revenue per use in the US is $77.62 and is expected to rise (Statista, 2018)
- 1/3 of the global population used a mobile app or fitness tracking device to track health in 2016 (Statista, 2016)

The fitness tracking wearables market continues to grow and serves as a great entry point into the market. Furthermore, unlike the smartwatch segment, the "smart shoe" segment is relatively unsaturated and presents and even better opportunity for market entry.

2.1.2.2 Similar Products

Based on customer discovery and market research, the team found that in the wearable fitness tracking market, there are several large competitors that dominate. While this is true, it mostly applies to categories outside of the "smart shoe" segment. The major companies (such as Apple, Fitbit, Garmin, Nike) have instead exceled in segments such as smartwatches, smartphones, mobile apps, etc. This further highlights the possibility of tapping into a market segment that has not been fully tapped by the major competitors.

While the largest fitness companies have yet to fully enter the "smart shoe" market segment, smaller companies and startups have begun experimenting with this technology. Some of the existing early products/prototypes that are similar to team Omicron's proposed project are listed below.

Sensoria Fitness Shoe

- The Sensoria Fitness Shoe is "Embedded textile pressure sensors at the plantar area of your foot and [a] detachable electronic device"
- Connected to a mobile app
- Tracks pace, speed, ascent/descent, cadence, contact time, foot-landing technique, and impact
- Not available publicly yet

Source: https://preorder.sensoriafitness.com/

Adidas miCoach Speed Cell

- A detachable shoe sensor that attaches to heel
- Tracks top speed, burst speed, distance, and game time
- Onboard memory for 7-8 hours

- Syncs to mobile devices over Bluetooth
- Tracks soccer, basketball, tennis, rugby, handball, and football

Source: https://www.soccerone.com/micoach_speed_cell_bluetooth_smart_compatible_p_2082.html

Digitsole

- Startup aimed at providing smart soles
- Controllable through smartphone app
- One concept was a sole that could dynamically heat feet
- Tracks steps, distance, calories
- Other varieties tracked 3D position of feet and stride

Source: https://gadgetsandwearables.com/2018/07/13/trackers-feet/#Digitsole

Altra IQ Sports Shoes

- Syncs shoe to smartphone
- Analyzes impact of foot with ground
- Measures stride, speed, distance, ground contact time, and cadence
- Real time suggestions

Source: https://gadgetsandwearables.com/2018/07/13/trackers-feet/#Digitsole

Lethal Smart Insoles

- Pairs with phone GPS to provide vibration and patterns indicating direction to travel
- Tracks steps, calories, distance, etc.
- Syncs with phone app
- Battery life lasts 15 days

Source: https://www.amazon.com/Lechal-Navigation-Fitness-Tracking-Insoles/dp/B01GFWQRY4?tag=healthand0fb0-20

2.1.3 Problem Solved

This project aims to solve several problems that exist in both smartwatch and phone technology, but also problems that exist the in the currently existing options for smart insoles and shoes.

2.1.3.1 Problems

1. Inaccurate data collection (particularly step count)

   Many current pedometers are inaccurate in their tracking of steps. A study found that on average, even for major competitors such as Fitbit, step counts for a distance of 400 meters were off by 40 steps. (Husted & Llewellyn, 2017) In addition, a paper published in the *Journal of Personalized Medicine* found that while major smartwatch manufacturers were bad at measuring calories burned during activities. (Dusheck, 2017)

2. Injury due to poor posture

   While smartwatches are capable of measuring GPS data, steps, heart rate, and more, they typically are not able to accurately measure posture other than differentiating standing, sitting, and lying down. By levering the piezo-electric materials in the proposed project's insoles, a pressure map could be collected and used to identify incorrect posture, poor form, and more.

3. Power Supply

   Current smart equipment available on the market, whether that be smartwatches or shoes, typically use rechargeable batteries as a power source. This not only is inconvenient, but it adds significant weight to the product which is important for many users. The proposed project aims to use passive power supply from the piezo-electric material.

4. User Experience

   Current wearable technologies typically are bulky, heavy, and obstruct a user's normal range of motion. The proposed project would dramatically reduce this, as the device would be embedded into the sole of a shoe.

5. Weight Tracking

   Current wearable technology typically cannot track weight. By utilizing the piezo-electric properties, the proposed project will attempt to accurately measure weight.

2.2 Project Technologies

The technology required for this project will require both hardware and software heavily. In addition to these broader categories, the project will need a few other miscellaneous materials to complete prototyping and design. While it's impossible to forecast every required material at this time, the following list of materials has been assembled to help guide planning.

2.2.1 Hardware

2.2.1.1 Technology Required

- Pressure sensors

  These will be used in the shoe to do a majority of the sensing. At this time, the team is leaning towards using piezoelectric materials due to their ability to generate electrical signals and power when a force is applied to them.

- Microcontroller

  A microcontroller will be required for collecting data from the sensors, temporarily storing it, and transmitting it to a paired device using a Bluetooth connection. This microcontroller will, at the least, require ultra-low power consumption, Bluetooth transmitters, AD converters, and be a small size.

- Mobile phone

  This will be required to receive data from the shoe's embedded microcontroller to a remote server capable of large capacity storage.

- Web server

  This is required to receive data from mobile phones and store it permanently. Additionally, this sever is necessary for processing the data for analytical insights presented to users through a web application.

- Miscellaneous components

  This project will require small, basic electrical components such as wires, capacitors, resistors, etc.

2.2.1.2 Familiarity

- Zach Oberbroeckling:
  - Familiar with microcontroller hardware (Embedded Systems courses)
  - Familiar with electrical components
  - Not familiar with mobile phone or web server hardware, but willing to learn.
- Brandon Reed:
  - Experienced with development on microcontrollers and embedded design.
  - Familiar with basic electrical components and circuit design. Unfamiliar with pressure sensor technology.
  - Unfamiliar with web server and mobile phone development.
- James Windorff
  - Experienced with embedded systems through classwork.
  - Familiar with circuits and circuit design.
  - Have a basic understanding of web design and html.
- Mitchell Larson:
  - Experienced with development on embedded systems
  - Familiar with interfacing electrical components with microcontrollers
  - Unfamiliar with mobile phone deployment
  - Unfamiliar with piezoelectric sensors
- Gunther Huebler
  - Electrical Physics/Mathematics (Atomic level to Transient level)
  - Circuit Design/Analysis (Spice Programming, Electrical Debugging)
  - Embedded Systems (Architecture to component design)

2.2.1.3 Assistance Needed

To complete this work, team Omicron will likely need access to MSOE's hardware laboratories for performing both prototype and design work for circuitry.

2.2.2 Software

2.2.2.1 Technology Required

- Web framework

This will be required to receive data from services running on mobile phones, process the data for analytical purposes, and serve a web page to clients. Initial ideas are the Angular 4 framework.

- Embedded software development tools

  This is needed to not only write the software for the chosen microcontroller, but also download the program to the embedded systems program memory.

- Android development environment

  This will be required to write the service responsible for transmitting data from the microcontroller to the web server.

- Bluetooth Protocol
- Project Management Software

2.2.2.2 Familiarity

- Zach Oberbroeckling:
  - Familiar with embedded software development.
  - Not very familiar with web framework, Android development, or Bluetooth Protocol, but willing to learn.
  - Not familiar with project management software but could research and learn.
- Brandon Reed:
  - Experienced with embedded software development.
  - Unfamiliar with web design, android development, Bluetooth, and project management software.
- James Windorff
  - Understanding of the android development environment.
  - Unfamiliar with Bluetooth.
- Mitchell Larson
  - Familiar with web development using the Angular framework and other tools (knockoutJS, LESS, Jade, etc.).
  - Familiar with development in languages meant for embedded systems (C, C++, ARM assembly).
  - Familiar with using project management software though internships.
  - Familiar with several application level protocols, but unfamiliar with Bluetooth.
  - Unfamiliar with mobile application development.
- Gunther Huebler
  - Embedded Systems (Assembly, C, VHDL)
  - Experience with various communication protocols (I2C, USART, component specific), no Bluetooth knowledge however
  - General Programming (C++, Java, Fortran, bash, python, MATLAB)

2.2.2.3 Assistance Needed

As of right now, team Omicron does not foresee needed assistance for software.

2.2.3 Other

2.2.3.1 Technology Required

- Shoes

  Shoes will be needed for housing the electrical components and/or smart sole.

- e-textile materials

  This will be required for assembling the smart soles which will be placed in the shoe. The e-textile material has not yet been decided.

2.2.3.2 Familiarity

- Zach Oberbroeckling:
  - Not familiar with shoe or e-textile technology, but willing to learn.
- Brandon Reed:
  - Have cursory knowledge of e-textiles, and ready to learn more.
- James Windorff
  - Would like to learn more about E-textiles and how they work.
- Mitchell Larson:
  - Not familiar with shoe or e-textile technology, but willing to learn.
- Gunther Huebler
  - Has shoes, no further shoe or e-textile knowledge

2.2.4 Roles

Based on project needs and team member familiarity, the following roles will be assigned. Note that just because someone has a project role, it does not limit a team members scope to only that role. Roles are just a team members primary focus.

- Zach Oberbroeckling – Embedded system development and interfacing
- Brandon Reed – Embedded system development and interfacing
- James – Android development and Bluetooth interfacing
- Mitchell Larson – Web application development
- Gunther Huebler – Sensors, power, and e-textiles.

2.2.4.1 Assistance Needed

Team Omicron may require the rapid prototyping lab for assembling and constructing our e-textile materials and fabricating them into the shoe.

2.3 Project Management

Project management will be done using a cloud instance of JIRA, managed and paid for by Team Omicron. Jira will be used to assign tasks, track progress, track time, and aid in minor organizational tasks. Project documentation will be placed in a publicly visible GitHub repository under the

SeniorDesignTeamOmicron GitHub organization. Additional repositories will be created under this organization for version control of software related any web development and embedded controls.

3. System Requirements and Specifications Report

3.1. Introduction

3.1.1 System Purpose

LogiSteps is a full stack application that is designed to collect, process, and display user fitness data in a seamless, self-powered construct. LogiSteps allows a user to pair their Bluetooth enabled smart sole with their mobile device and stream data to the cloud in a manner that is unobtrusive and relies very little on the user. By using this system, users can enhance, monitor, and improve their personal fitness without the need of energy demanding equipment that is often bulky and uncomfortable.

3.1.2 System Scope

To satisfy the needs of customers, LogiSteps is scoped to provide data collection abilities that will expose user data using a cloud-based web application. Prior market research and analysis of customer needs identified a need for a wearable device that users could use without constant maintenance and recharging. To solve this, LogiSteps will provide several essential capabilities to make it possible for customers to track personal fitness in an invisible, self-sustaining manner. As a result, the system will use the impact of users' steps to both collect data and power the system, harvesting raw data in an electronic device embedded into the sole of a shoe. This embedded device will transmit the data it collects to a nearby mobile device, which will then relay the data to a cloud-based web application.

To provide valuable insights into personal fitness, the system will store collected data in a long-term storage medium that will make it possible for performing powerful queries and aggregate calculations that will be served to users through a flexible web-based interface. This enables a user to view their data from any platform, regardless of the device that is used to relay data from the smart sole to the web server. The storage medium chosen will be designed to perform well under a high rate of data insertions and very little deletions. Furthermore, LogiSteps will enable users to closely monitor and track personal fitness progress over long periods of time, helping achieve milestones and fitness goals. While the system is will provide tools for measuring personal fitness, LogiSteps is not meant to, and will not provide medical diagnosis, fitness training, and other high-skill analysis. The physical construction of LogiSteps will be designed to integrate into a user's shoe with virtually no impact on comfort, allowing customers to setup and 'forget' about the device.

3.1.5 System Overview

In summary, the LogiSteps system will be designed to provide fitness data for health-conscious users wishing to track their data in non-obtrusive, self-sufficient means. The system will be compromised of a smart sole embedded with electronic communication and data collection technology, a mobile device, and a cloud-based web application capable of running across all systems using popular web browsers. This document will layout the black box system requirements for LogiSteps.

3.2. General System Description

3.2.1 System Context

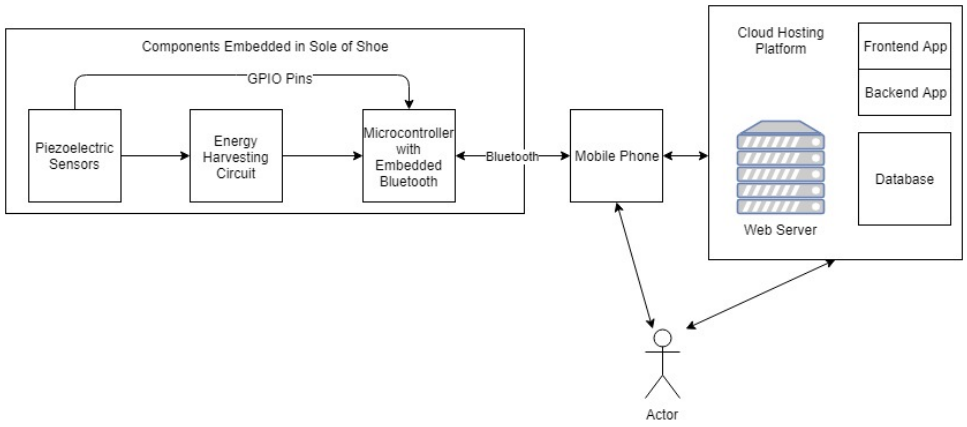

Figure 1 - System architecture of the LogiSteps system. Note the three major components present in the system - Shoe sole, mobile device, and web server.

The proposed system has 7 major components – each with an important role in facilitating data flow from its raw source to the web interface. The system will obtain both data and power using piezoelectric materials that will be embedded into the sole of a user's shoe. Voltages from the piezoelectric sensors will then provide input to a microcontroller unit and an energy harvesting circuit that is designed to provide a stable power source to the microcontroller unit. The microcontroller unit will use an embedded Bluetooth controller to transmit all data to a nearby mobile device, which will provide a light user interface for the user, and then relay data for long term storage and heavier processing. The web application will then serve a lightweight, but powerful user interface that will provide rich graphics for exploring fitness data.

The LogiSteps system exposes three interfaces that cross system boundaries. These interfaces provide the means for collecting and portraying data to/from external forces and actors. These interfaces include:

1. **Environment to Insole**

   The transfer of mechanical energy created from the impact between a user's shoe and the ground is converted into electrical energy which can be measured and collected. This interface is invisible to the user and requires no interaction other than typical movement.

2. **Mobile Application to User**

   Intermediary data, prior to heavy processing, is relayed to the user using a simple user interface native to a mobile device. Additionally, this interface will provide status information regarding connections to the microcontroller unit, and the web server. To initiate these connections, the user interface will provide a means for initiating the connections through Bluetooth pairing and account creation/login.

3. **Web Application to User**

Advanced tables, charts, graphs, and statuses are relayed to the user through a web-based user interface accessible to any device capable of running a web browser. This interface will provide a complex and rich user interaction and can be accessed from any geographical location or technical platform.

3.2.2 System Modes and States

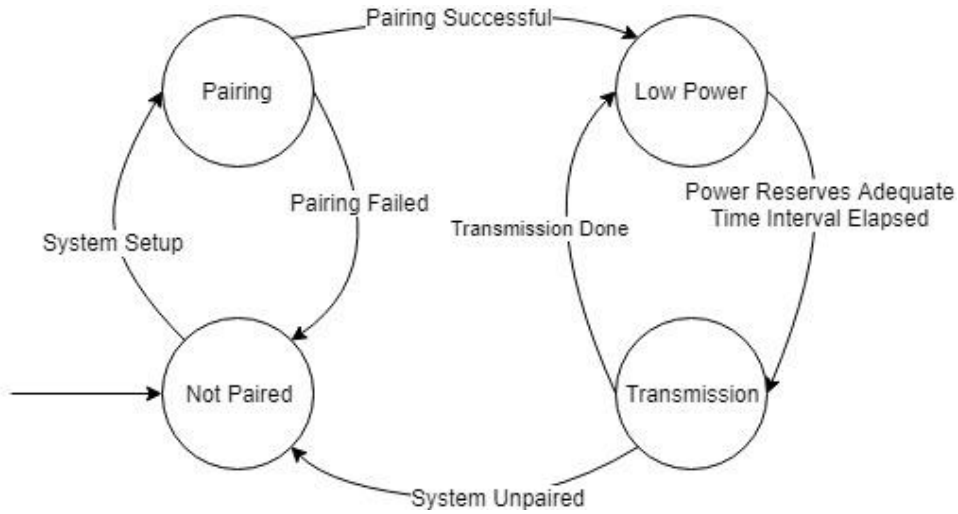

Figure 2 - System states that represent the LogiSteps system.

The system is required to operate under four different states: an unpaired state, a pairing state, a low power state, and transmission state.

When the system is bought, it must be shipped and supplied to customers in the 'Not Paired' state. During this state, no collection must occur, and no data transmission must occur. The system should be incapable of performing normal system functions until a user successfully pairs the embedded device with their mobile device.

When a user begins the process of pairing their embedded smart sole with their mobile device, the system should enter a pairing state. In this state, all power must be reserved for the Bluetooth pairing process. Prior to performing this, user's will be required to walk with their shoes on to build up enough power to perform the Bluetooth intensive task. No data collection or data transmission (other than that required for Bluetooth transmission) should occur when the system is in this state.

During the low power state, the system will minimize the power consumption of essential system functions and shut down any functions that are not critical to system function to save power. Because the system will operate with limited access to power, having a low power state is essential to long term operability. During this state, the system must not perform data transmission, and instead only capture data input as it occurs.

During the transmission state, the system should increase power consumption to transmit collected data points to the paired Bluetooth enabled mobile device. To do so, the system must temporarily supply power to the embedded Bluetooth chip. The system must draw from power reserves built up during the low power state and re-enter the low power state as quickly as possible after data

transmission of all data points has occurred. Upon exit of the transmission state, non-essential functions and capabilities will be shut down.

3.2.3 Major System Capabilities

This subclause should provide diagrams and accompanying narrative to show major capability groupings of the requirements.

The system needs to be able to gather data pertaining to the user's activity, store the data permanently, and provide a way of displaying that data to the user in various ways. The system also needs to be robust enough to handle multiple common scenarios the user may be found in and not affect the user's comfort in any way.

Physical Capabilities:
The physical capabilities of the device pertain to concepts such as how the electrical components and sensors are housed, the durability of the insole, and how the insole affects the user's comfort.

Form:
The insole part of the system needs to be a single piece, requiring no assembly, housed entirely in the user's shoe. There will be no parts of the insole contained outside of the shoe and no parts of the insole crucial to operation that can be removed or broken off under normal conditions.

Robustness:
The insole needs to be capable of not being damaged in any way that affects the ability of the system to operate under normal circumstances. Normal circumstances will be discussed in the operational scenarios of this report.

Comfort Criteria:
The insole itself needs to not affect the comfort of the user while it is in use. This requires ensuring that the device has a small footprint and be made of a material that is similar to others currently existing in shoe devices.

Non-Physical Capabilities:
The non-physical capabilities of the system pertain to the data gathered by the system and the software the user can interact with.

Data:
The data will be gathered from the user via a physical device to be worn inside the user's shoe. This data will be produced while the user takes steps at any rate and relayed wirelessly at all points. The insole will relay the data first to the user's mobile device, then the mobile device will relay the pressure data from the piezoelectric materials, along with GPS data from the mobile device, to a server for permanent storage and further processing.

Data Storage:
The data collected will not require the user to own any other personal storage device nor permanently take up any storage on the user's mobile device. The data gathered from the insole and relayed to the

user's mobile device will be temporarily stored on the user's mobile device until connection to the server can be established, at which time the data will be wirelessly offloaded to be permanently stored elsewhere.

Data presentation:
The data gathered during operation needs to be made available to the user to view and interact with. A portion of the data will be made available to the user via an application installed on the user's mobile device and the entirety of the data will be made available via a web application the user can access on any web browsing capable device.

3.2.4 Major System Conditions

The system needs to be able to gather data only in the intended scenarios described in the operational scenarios section. All these scenarios require that the insole part of the system be in the user's shoe while the user is wearing their shoe and taking steps. During each of these scenarios the insole must be collecting and transmitting data. While the user is not actively producing collectable data (e.g. the insole laying idle somewhere) the system is not required to collect data of any kind.

For data to be relayed from the insole to the user's mobile device, the insole and the user's mobile device must be in range to communicate. This range need only be the distance of an average room. While the user's mobile device and the insole are significantly far apart, data does not need to be collected and may be lost.

For data to be relayed from the user's mobile device to the server for permanent storage, the user's mobile device must have an active internet connection. If no internet connection is available, data may be temporarily stored on the user's mobile device until an internet connection is reestablished.

3.2.5 Major System Constraints

Constraints on the system require that the physical system not be abused in any way. Misusing the insole device may result in decreased operational ability or result in physical system failure. Specific constraints of system operation will be discussed in: section *3 - System Capabilities, Conditions, and Constraints* and section *3.5 - System Operations.*

3.2.6 User Characteristics

LogiSteps is a system targeted towards users who are casually health conscious and wish to gain further insight into data related to their movement. As a result, the system will be designed to meet requirements targeted towards satisfying the needs of this consumer group. There exist other market segment groups which may use the LogiSteps system however, with differing needs and expectations. Each user group may use the system in a different manner, and as a result, they have been recorded. Each group is defined according to their function, location, relative size, and nature of use.

Casual Health Conscious Users (Target Group)

- *Function*: Users under this category have average activity rates and health. These users casually track their fitness data using devices like Fitbits and Apple Watches. Precision is not key to these users, but availability and ease of access to data is.
- *Location*: Distribution evenly spread. Climate varies but users do not spend great amounts of time outdoors in harsh conditions.
- *Size*: This is the largest user group who may use LogiSteps (~1/3 of all Americans).
- *Nature of Use:* Users in this category will require that the system can withstand normal walking conditions, and the impact that occurs when walking up and down stairs. These users will occasionally expose their shoes to harsh elements such as rain, snow, and salt, but generally attempt to avoid such conditions. This user group will be primarily interested in step count and calories burned.

Runners

- Function: Users under this category have above activity rates. These users typically use wearable technology to track statistics and progress of a run. To these users' precision and accuracy is usually essential.
- Location: Distribution evenly spread. Climate varies, and users typically spend lots of time outdoors in harsh conditions.
- Size: ~65 million joggers/runners in the United States.
- Nature of Use: Although LogiSteps is not targeted to satisfy all the needs of a runner, many runners may use LogiSteps in place of, or in complement to, other wearable devices.

Users with Back/Joint Problems

- Function: Users under this category have back/joint problems, and as a result, closely monitor their posture. Users generally see a certified professional for help improving posture.
- Location: Distribution evenly spread. Climate varies, and users typically spend little time in harsh conditions.
- Size: ~31 million people have back problems in the United States.
- Nature of Use: Although LogiSteps is not scoped to provide medical aid in diagnosing and monitoring posture, users in this group may use LogiSteps in aiding their prognosis of the ailment.

Tech Enthusiasts

- Function: Users under this category are excited by new technology and take pride in owning experimental and cutting-edge technology.
- Location: Distribution evenly spread. Climate varies.
- Size: Difficult to estimate quantitatively. ~2 million technology jobs in the US
- Nature of Use: Users in this group will likely use the system sparingly, trying it out for a small amount of time, before focusing on the latest piece of technology.

3.2.7 Assumptions and Dependencies

For successful completion of the system, a few assumptions are made regarding the resources available to the user. The assumptions are made with the thought that they do not provide a difficult barrier to product use and are readily available to all users considering purchase of the LogiSteps system. Additionally, LogiSteps depends on a couple of services from third-party resources to meet the requirements of the system. The assumptions and dependencies that the LogiSteps system will incorporate into design are described in the following list.

Assumptions

- Mobile device support/Internet connectivity

  The user is assumed to have an android mobile device for phase 1 of the project. Future expansion of the project may include support for additional mobile operating systems/devices. The mobile device must have the ability to access the internet using Wi-Fi or LTE. Additionally, LogiSteps will assume that the mobile device has BLE for communication with the insole.

- Closed toed shoes

  LogiSteps assumes that the user has closed-toed shoes, and thus, LogiSteps will not provide, or design for open-toed shoes.

Dependencies

- Web Hosting Services

  The LogiSteps system will rely on a web hosting service to provide necessary load balancing, scaling, security, and other web-based services that are outside the scope of the project but are a requirement for proper operation.

- Authentication

  The LogiSteps system will depend on libraries which belong to the back-end framework for user authentication and authorization.

- Graphics

  The LogiSteps system will depend on a third-party library for rendering graphical representation of user data.

3.2.8 Operational Scenarios

Due to the mobile nature of the LogiSteps system, users may experience several different operational scenarios when using the system. To help provide further insight into the requirements and behavior of the system, a few examples are provided below.

Walking

As a user begins to walk, the system will begin collecting the energy absorbed by the impact between their foot and the ground. As soon as the system has gathered enough energy, the system will move into

its low energy state, continuing to harvest energy, while recording each measurement from the sensors. After the system has acquired enough energy and buffered data for the required amount of time, the system will transmit the data to a nearby mobile device using the BLE protocol. Data will continuously be buffered and sent in this manner periodically. Data will then be buffered on the mobile device, while the mobile device UI is simultaneously updated to reflect the buffered data. Data will finally be sent to a web server for long term storage, where advanced graphics can be viewed.

Running

When a user runs, operation will occur in a manner similar to when a user is walking, with one exception. When a user is running, the amount of data being generated by the user (as well as energy) will increase, and as a result, the system may need transmit data at a faster rate to avoid buffer overflow.

Sitting

A common scenario that may arise when a customer is using the LogiSteps system is where they are elevated off the ground in a sitting position. In such a situation, little to no pressure will be placed on the insole of the shoe. Such a situation will lead to very little energy generation, as well as very little data collection. As a result, the system will remain in a low energy state until the user begins to behave in a way like the walking and running scenario. In this operational scenario, the system will be mostly idle and attempt to conserve as much energy as possible.

Standing

The LogiSteps system may exist in a scenario where a user is applying pressure to the insoles of their shoes but remains stationary. Similar to behavior while a user is sitting, very little energy will be generated, and the sensors will not be able to detect steps occurring. When this happens, the system will remain in a low energy state, withholding data transitions until the user begins to move – either walking or running.

3.3. System Capabilities, Conditions, and Constraints

3.3.1 Physical

3.3.1.1 Construction and Durability

Since an important part of LogiSteps is the physical construct of the system, there are a handful of physical conditions and constraints that the system must meet. The physical aspect of LogiSteps includes a shoe insole that measures pressure put on it. Because the insole will be located inside of a shoe, there are various requirements for it to work properly.

Construction

The LogiSteps system should be able to support the weight of the average person (around 170 pounds) and has a maximum weight limit of 300 pounds. For the safety of the user and to avoid hardware malfunctions, the LogiSteps system should not expose any electrical components or contacts. To avoid exposed components, all necessary electrical components should be housed inside of the LogiSteps

insole. The LogiSteps insole should properly fit inside the size range of adult men's 8 and above without causing any discomfort to the user. This distribution of shoe sizes should fit most users.

Durability

When in use, the LogiSteps insole should run properly in the shoe environment, regardless of any moisture created in the environment. The LogiSteps insole should properly function after going through a cleaning process. This process would consist of lightly wiping with a rag containing cleaning chemicals (e.g. disinfecting wipe). The LogiSteps insole is not guaranteed to properly function after an excessive cleaning process (e.g. submerged in water, washing machine).

Summary of Requirements

- The system must support up to 300 lbs.
- No electrical components or contacts may be exposed.
- Components must be housed inside the sole of a shoe.
    - Conditions: LogiSteps must support shoe sizes of Adult 8 (and the women's equivalent) and up.
- The system must withstand the moisture created by a user's foot.
- The system must withstand the moisture of a cleaning process.
    - Conditions: wiping with rag using cleaning materials
        - Constraint: Not submerged in water (e.g. washing machine)

3.3.1.2 Adaptability

As a physical project, the LogiSteps system should provide adaptability. As a product, the LogiSteps system should provide expansion and growth as new technology is made available, and new user interface features are requested and developed.

Data processing

The LogiSteps system should be designed with the ability to expand upon the data processing features. The data collection and transmitting will most likely stay the same, but as LogiSteps enters the market, the system must be developed in a modular approach which allows new data processing features to be easily integrated into the data pipeline. Additionally, the system should be designed to scale up the number of sensors used for data collection and energy generation.

Mobile devices

An important aspect of the LogiSteps system is the mobile device interface. Since the LogiSteps system will communicate with a web server using a mobile device, the LogiSteps system should be able to support changes and growth of mobile devices. The LogiSteps system should continue to work properly with mobile devices as long as Bluetooth connections are still enabled and provide backwards compatibility as new operating systems for mobile devices are unrolled.

Summary of Requirements

- The system should support expansion of data processing features.
- The system should support expansion and enrichment of user experience.
- The system should support future growth of mobile devices.
    - The system should support changing mobile devices.
    - The system should support the addition of sensors.

3.3.1.3 Environmental Conditions

One of the most important aspects of a physical system is the environmental conditions it can endure. The LogiSteps physical system should be able to withstand all possible conditions that are caused by the environment in which it is used. Because the LogiSteps insole will be placed inside of a shoe, which often encounters harsh conditions, the system must be built to withstand many extreme environmental conditions that most electronic systems are not subject to.

Pressure

Since the LogiSteps insole will be put in the user's shoe, the LogiSteps system must be able to withstand large amounts of pressure. The system must be able to withstand pressure caused by day to day actions of the user (e.g. running, jumping, lifting weight, etc.). The system is not guaranteed to work properly when excessive force, such as car accident or high fall, is applied.

Protection

Given the environment in which the LogiSteps insole will reside, the system must be able to protect against water and dust. To accomplish this, the system will have an Ingress Protection rating of 67. This means that it will be protected against all solids and will be protected from temporary submersion in water (e.g. stepping in a puddle). It will not be protected against continuous submersion in water.

In addition, the system must be protected against electricity that could be naturally applied to it (e.g. static electricity from friction).

Temperature

Given the environment of the LogiSteps insole, the system must also be able to withstand temperature changes caused by the environment. When used outdoors, the LogiSteps system must withstand a wide range of temperatures. The microcontroller has an operating temperature range of –25 to 75 degrees Celsius and the system should function properly in any environment that meets this temperature range. The system should not be guaranteed to work properly when exposed to extreme heat or cold outside of this range.

Communication

The LogiSteps system must be able to communicate properly to a mobile device when background noise is present. The system must also ensure that the system is able to properly communicate through the insole material as well as any other obstacles between the LogiSteps system and the mobile device.

Summary of Requirements

- The system must withstand additional impulses and pressure caused by contact between the user and their environment:
    - Conditions: This includes behavior that ranges from typical walking, to running, jumping, and lifting heavy weights.
    - Constraints: The system should not support impacts due to abnormal collisions such as car accidents and high falls.
- The system must be IP67 rated for water and dust.
- The system must withstand environmental temperature conditions:
    - Conditions: temperature range (MCU scale: -45-85 degrees Celsius)
- The system must withstand heat radiated from a user's foot, as well as heat generated from foot contact with the ground.
- The system must not be adversely affected by static created when a user's foot contacts any surface (such as concrete, carpet, and much more).
- The system must be able to successfully communicate under regular environmental background noise.
- System communication must be able to permeate through the form factor of a user's shoe and body and reach a mobile device within 0 and 3 meters of distance.

3.3.2 System Performance Characteristics

There are some expected minimums the LogiSteps system will meet in regard to its performance. It has specific performance requirements for each stage of the system.

Startup

When the LogiSteps system begins functioning (when it is turned off and the user begins walking), it is expected that the user will be able to pair the insole to their mobile device and begin receiving their fitness data on that mobile device within thirty seconds. They should also be able to view their processed fitness data on the web application within sixty seconds of beginning usage.

Post-Startup

After the LogiSteps insole has gathered enough energy for the initial pairing with the user's device, and has paired with the user's device, it has entered its post-startup stage. In the post-startup it is expected that the user will be able to view all of their basic data gathered from the insole on their mobile application in real time. And they will be able to view all processed data from the insole on the web application within sixty seconds of seeing it on their mobile device. It is also expected that the insole will gather and transmit user fitness data at a consistent rate, regardless of the user's cadence. That is, no matter how slowly or quickly the user moves, the system will gather all pertinent fitness data from their movements. This of course makes the assumption that the user's cadence will fall into a normal human's range that can be expected in the day to day activities of a casual user, or the heightened activities of a serious user.

Lifetime

Regarding the LogiSteps system lifetime there are some expectations as to how long the insole should last. As the insole will be put into a user's shoe, and always kept in their shoe, the system is expected to last a minimum of 1 year.

3.3.3 System Security

As the LogiSteps system will require pairing with a user's mobile device, the transmission of user data over the internet, and the storage of user data on the cloud, there are several considerations that have to be made for LogiSteps security. The security required can be looked at for each individual stage of the system.

Insole to Mobile Device

Neither the LogiSteps embedded system in the insole, or the wireless transmission from the insole to the user's mobile device will have to be secured in any way. This is due to the fact that all of the information that will be on the embedded system, and transmitted over Bluetooth, will contain no confidential information. It will consist solely of raw sensor data, and the associated time data.

Mobile Application to Cloud

The LogiSteps mobile application on the user's mobile device will require some security to ensure only the correct user can connect to their insole, and to the cloud. To achieve this, the mobile application must implement login functionality to ensure the user's data on the mobile application can only be accessed by the correct user. This login functionality will also be used in the connection to the cloud to provide user authorization to send and receive data from the mobile application. Because data transmitted from users' mobile device will contain sensitive GPS data, the connection between the mobile application and the cloud must be secured using HTTPS.

Cloud Web Application

LogiSteps data will be stored on a server on the cloud. As this data will contain personal and confidential user information, the web server must implement proper security protocols to ensure all user data contained within it will be protected from hacking attempts and data stealing. Also, for the user to access their data on the web application, login functionality will be used again to enforce user authentication to access their data stored on the web server. The web application will also ensure a secure connection between itself and the user using HTTPS. Additionally, the web application must keep all data used for global statistical analysis anonymous to other users.

3.3.4 Information management

LogiSteps is a data driven application that will acquire large amounts of user data, which will increase proportionally with both time and the number of users using the system. To ensure that the system can provide the necessary capabilities to users, while responsibly and safely storing data, the following requirements have been drafted. The LogiSteps system must be designed to meet the following information management requirements.

- Users must have access to their data through the user of interactive graphical user interfaces
- User data must not be sold or given to any third party without the express written consent of the effected user.
- The system must store data for long-term retrieval and statistical analysis on a database decoupled from the user's mobile device
- The system must initially support up to 6 months of user data and provide means for scaling up the amount of stored user data for future expansion.
- The system should allow users to permanently delete their connection to their data, while maintaining the data in an anonymous format for aggregation of global data.
- The storage medium used for long-term storage must be capable of scaling well with time and users.
- The system should delete all user data off mobile devices upon deletion of the mobile application.
- User data used for aggregate global analysis must be anonymized to protect users' identities.
- If a connection cannot be established, or a connection is dropped between the insole and a user's mobile phone, data should be discarded upon attempted transmission.
- If a connection cannot be established, or a connection is dropped between a user's mobile phone and the remote web-server, a circular buffer of 20 MB should be established, and data should be temporarily held on the mobile device until it can be successfully offloaded to the server.

3.3.5 System Operations

3.3.5.1 System Human Factors

Because part of the LogiSteps system is the physical insole, the insole must be sensitive to human factors and limitations. Ultimately, the LogiSteps insole must be able to properly read and create data based on the pressure applied within the capabilities of a human. The LogiSteps insole system must be able to detect when small amounts of pressure are applied and will be able to differentiate between the pressure of an average step and small amounts applied naturally without taking a step.

Durability

The LogiSteps system must be durable enough to withstand pressure that is naturally applied to it in a natural setting. The LogiSteps insole must be able to withstand the pressure caused by day to day activities that are performed by the user, such as running, walking, jumping, and heavy lifting. The LogiSteps system should not be guaranteed to withstand any pressure caused outside of human capabilities.

In addition, the LogiSteps insole system must withstand any bending that is applied to it within its natural environment. The LogiSteps insole will be designed to withstand any bending as a result of force applied by the human foot. The LogiSteps insole should not be guaranteed to withstand any bending caused by excessive forces beyond the human capability.

Summary of Requirements

- The system must sustain operation under the pressure generated from a user's step under reasonable conditions
    - Conditions: Running, walking, heavy lifting, etc.
    - Constraints: Applying excessive force (running over with car, smashing with something, etc.)
- Must be able to handle any bends that are a consequence of human foot flexibility

3.3.5.2 System Maintainability

LogiSteps must be a self-sustaining system. The hardware should be encased so that the user is not able to change any of it. The solution to hardware failure will require changing out the component. The hardware will be expected to last a minimum of one year.

The only maintenance required by the user should be the connections between pieces of hardware. The user must be able to disconnect and re-establish Bluetooth connection to the insole from the user's mobile device. This should only be required in worst case failures or change in hardware.

The mobile application will have the potential for updates and for preventative maintenance. As the product gains popularity, it is possible that flaws in the software may be discovered. This should be monitored on a weekly basis and if updated if required.

3.3.5.3 System Reliability

LogiSteps has multiple systems within itself that make up the whole. The reliability of the system is dependent on the parts being reliable. The system must maintain data transfer from the insole to the web server with an uptime greater than 99%.

Server

The server must have an uptime greater than 99%. This will allow the mobile app to not have to store a lot of data on itself. The use of a cloud-based web-server should prevent the user from noticing any substantial slowdown of their mobile device.

Mobile app

The mobile app will have a Bluetooth connection that will be required to communicate with devices between zero and three meters of the user. BLE supports a maximum range of one-hundred meters when operating under full power. Due to the low power requirements of the LogiSteps system, a maximum Bluetooth range of 100 meters will not be supported.

The second part of the mobile app is the connection to the server. The connection to the server must be available if the user is connected to their Wi-Fi or LTE network with an uptime greater than 99% in order to ensure a seamless user experience and reduce the amount of data that will need to be cached on a user's mobile device.

3.3.6 Policy and Regulation

LogiSteps has defined two major policy and regulation requirements to ensure that the system meets the needs of its users and fulfills ethical design requirements.

Health and Safety

LogiSteps must ensure that any user of the LogiSteps system will not suffer any adverse physical affects due to the LogiSteps system. The LogiSteps system should be non-flammable and its electrical components should be unable to harm the user in any way. The insole should also be made of a safe material to ensure users will not suffer from adverse effects such as allergic reactions, rashes, etc.

User Data

The user's data must, first and foremost, be safe and secure wherever it is stored, or transmitted. The requirements to satisfy this have already been defined in sections *3.3* and *3.4* of this system requirements report. The user data will, however, be owned by LogiSteps. This data will not be sold or given to any third party without user consent. Should the user wish to delete their account and associated data, all personally identifiable and confidential data associated with that user will be permanently deleted and removed from the LogiSteps servers; the data however, will continue to reside on LogiSteps servers to provide global analytics and aggregate calculations.

3.3.7 System Life Cycle Sustainment

In terms of system life cycle sustainment, the LogiSteps system must plan for sustainment before release as well as provide continuous sustainment after being released.

Prior to release:
In planning the LogiSteps system, Team Omicron must plan for the system to be updated on a regular basis. The main form of updates for the system will be updates to the user interface. To accomplish this, the user interface design must plan for continuous updating and should be able to be updated without causing errors.

Also prior to release, field testing must be done to ensure proper operation of the LogiSteps system. Through testing of the system, issues can be found and resolved before finalizing the system design and operation.

Ongoing:

In providing ongoing life cycle sustainment, the LogiSteps system must be able to ensure proper operation. The system must be able to detect any anomalies with data and provide solutions to fixing the data. The software will be updated in the case of any bugs or hardware will be replaced if not functioning properly.

In addition, user feedback will be an important aspect of ongoing sustainment. The LogiSteps user interface and software should be continuously updated to provide users with requested features and bug fixes, and a conduit for submitting user feedback should be established.

3.4. System Interfaces

3.4.1 Environment to insole

The way the environment interacts with the sensors needs to be carefully considered. The energy produced by the piezoelectric sensors needs to be great enough to power the electrical components inside of the insole device. For piezoelectric material to produce energy it needs to be deformed, and this deformation will be caused by the intended environment.

Definition of Intended Environment:

The intended environment is the area immediately surrounding the insole while the system is in use: the sole of the shoe and the user's foot. When a user takes a step, their foot applies downward pressure, compressing the insole in the direction of the sole of the shoe.

Capability:

The interaction between the environment and the insole needs to be capable of producing enough energy to power the electrical components within the shoe. This energy is created by a deformation of the piezoelectric sensors embedded in the insole and this deformation will be caused by the design of the insole.

Condition:

This interaction need only meet requirements while the insole is within the intended environment. It is not required that the insole interact with any other environment enough to power any component. However, the insole will still be capable of working outside of the intended environment if the user wishes to simulate footsteps for testing/troubleshooting.

Constraints:

The intended environment must not be capable of producing any amount of stress to the insole that would cause damage to the hardware housed inside. This involves ensuring that the user cannot apply too much pressure during normal activities.

Future Modifications:

The interface between the environment and the insole must have room for improvement in the future. The insole part of the system is very dependent on low power consumption of the electrical components, limiting the choices for insole hardware and software features. This creates a significant demand for the ability to harvest as much power as possible, since doing so will allow for as many hardware and software options as possible: electrical components with a higher energy demand can be considered, transmitting data at higher rates becomes an option, and more powerful real time processing can be done inside the insole itself rather than the mobile device or server. Because of this, optimizations to sensor activating techniques need to be explored in the future as new technology is developed.

3.4.2 Mobile Device to User

The next major system interface is the mobile device app. There are three aspects to the mobile device app: the Bluetooth to the insoles, the data connection to the server, and the user interface. The three are the middle point to the system as a whole to bus data between the devices. The Bluetooth connection should transfer data from the LogiSteps device to the mobile app at an optimal speed of 100kb/s with a maximum of 1Mb/s. The maximum is the maximum speed that Bluetooth Low Energy can achieve, while the optimal is dependent on how large the data is that is collected from the insole. Additionally, the mobile app must show the user real time data that the mobile device has temporarily stored as it prepares to send it to the server. The mobile app will move the data at an optimal speed of 1Mb/s to the server with a maximum of 12Mb/s and a minimum of 300 kb/s. The maximum and minimum are dependent on the mobile device's LTE network where the optimal is for burst sending data from the mobile device. This is required to save the user's data so LogiSteps does not need a constant connection and consume unnecessary amounts of data.

The user interface should have two screens - the log in screen and the summary view. The log in screen is very basic. LogiSteps allows must provide a conduit for a user to log in through Facebook and Google if they choose, or, they should be able to create an account through LogiSteps' services.

Basic View Interface Requirements:

- View step count.
- View a step projection for the day.
- Connect and disconnect from the insoles individually.
- View steps in the past hour.
- Show the Bluetooth status for each insole.
- Show the status of the connection to the server.
- Show a projection of steps for the day.
- Show a user defined step goal.
- Main screen is one page without scrolling

Login Screen

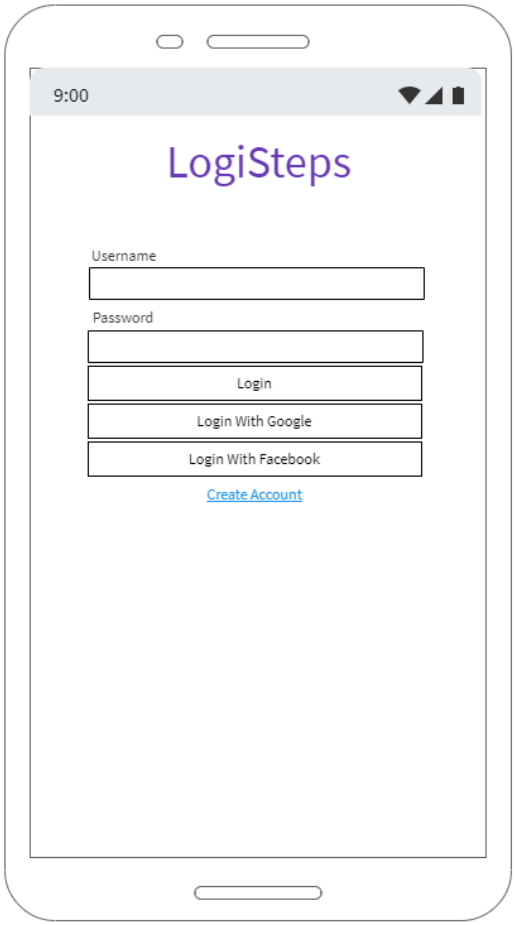

Figure 3 - Initial mockup demonstrating system requirements for a log on screen on the system's mobile device.

The login screen should be a basic screen. The authentication with the username and password fields should be managed by the LogiSteps system, and if the user chooses, they should be capable of logging in using services such as Facebook or Google by clicking their corresponding button. The create account link should not change the layout and instead act as an alternate to the login button. It should take what is in the username and password field and create an account using the entered information.

Main Screen

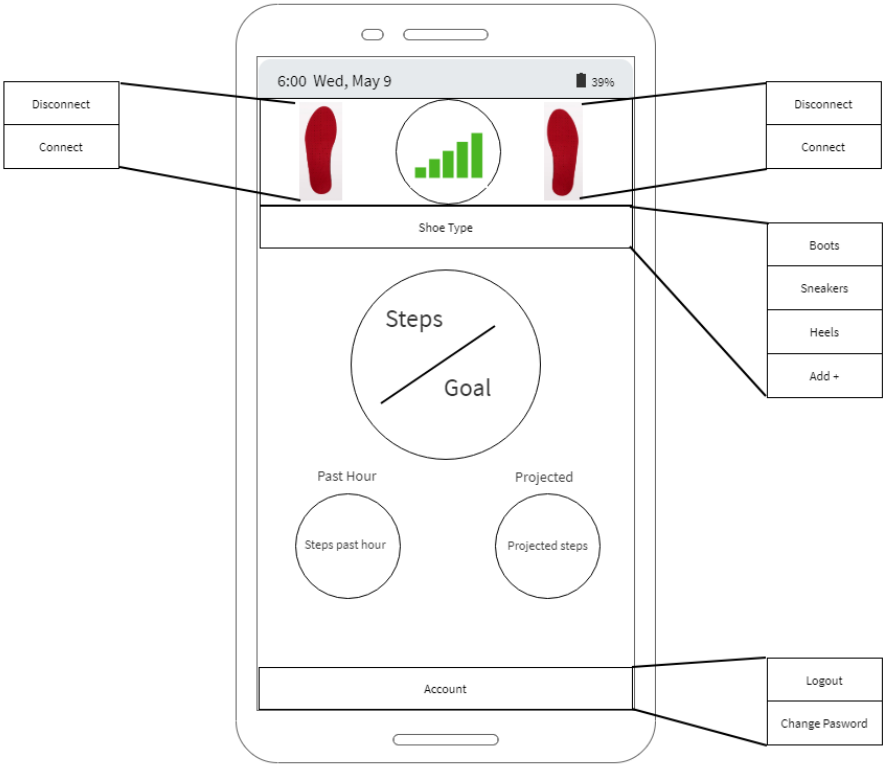

Figure 4 - Mockup of potential screen that could satisfy system requirements for showing simple connection statuses and user statistics.

Upon a successful login, the user should be able to view the screen shown above. The pop-outs shown are buttons that show options for each item selected. The app should be one page and allow the user to have a basic screen without a lot of clutter. The feet logos should either be green or red, depending on connection status to each insole.

3.4.3 Server to user

The last major interface between the LogiSteps system and the outside world is interaction that will occur between the web server and the user. The purpose of this interface is primarily to view and interact with user data that was generated from the insoles. While there is another user interface that will be available to users through their mobile device, the user interface that is served from the web server will provide more detailed analytics and provide a more complete experience. This interface will represent a bidirectional flow of information, presenting data to the user, while also accepting input related to authorization, navigation, and user account changes. This interface will represent several different view components, with their own requirements. These views, as well as the requirements for the interface will be defined in this section.

Interface Requirements

- The interface should provide a login screen to provide user authentication and protection of data.
- The interface should provide a method to create an account for users that have not been registered yet.
- The interface should not allow access to any application features until a user is authenticated.
- The interface should navigate a user to a main landing page upon successful authentication which will display a summary of user data from the past two days.
- The interface should provide a settings icon which will allow a user to change profile settings and logout of the application.
- The interface should provide a section for the user's data, global data, and a connections tab which may be implemented in future version of the product.
- The interface should provide a list of tabs on the left-hand portion of the interface to navigate between different statistical analysis pages and graphics.
- The interface should present statistical analysis and graphics pages for recent activity, steps over time, steps by weekday, activity levels by week, pressure, and a map view of the user's activity.
- The interface should protect user data during communications using HTTPS to encrypt data over the internet.

The specific statistical analysis and graphics pages each have their own set of requirements as well, which can be grouped better by identifying them on a feature by feature basis. A few wireframe mockups have been provided to illustrate what a view that fulfills the requirements may look like. Note that not all views have been mocked, but an implementation of the view should still fulfill its defined requirements.

Login Screen

Figure 5 - Potential login screen that could meet system requirements for securing user data on the web platform.

The login screen is required to provide fields for providing user authentication using a username and password. Additionally, the login page must provide a link for registering a new user. This page must be

the only page presented to a user until a successful user authentication entering a correct username and password and logging in.

Main Page/Recent Tab

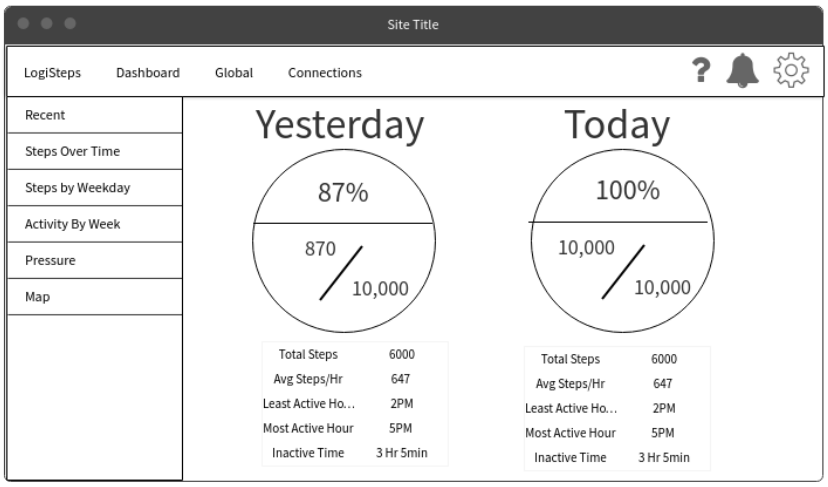

Figure 6 - Potential view that could satisfy the requirements for a main landing page and recent activity tab.

After a successful login attempt, the web application must navigate a user to a view displaying a summary of their past two days of activity. The minimum requirements of this view are:

- The view must display a user's goal completion status for the current day and the previous day.
- The view must display a user's step count against their goal for the current day and the previous day near the goal completion status.
- The goal completion status and steps vs goal visuals must be the dominating visual on the view.
- A summary of supporting statistics such as step total, average steps per hour, least active hour, most active hour, and inactive time must be displayed under the primary visuals.

Steps over Time Tab

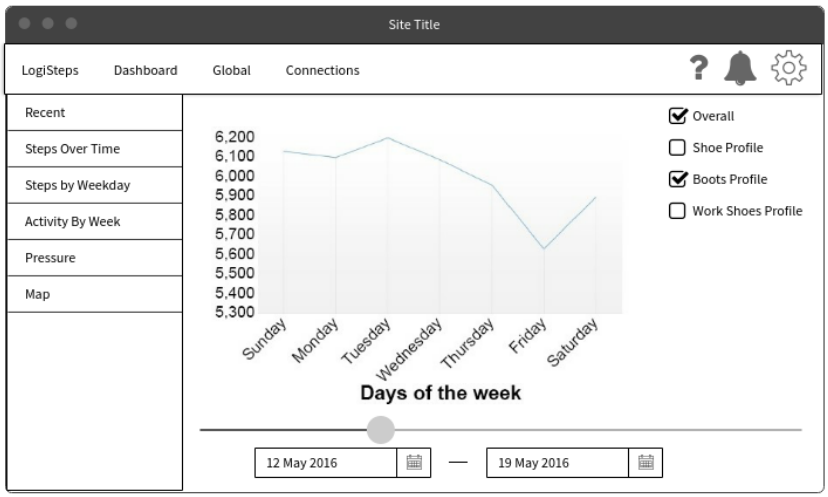

Figure 7 - Potential view that could satisfy the requirements for a steps over time page.

Upon navigating to the steps over time page, a user should be greeted with a visual representation of their steps over time. The type of graphical representation is not specified but must meet the criteria for the view.

- The view must display the number of steps taken in a day over the timespan of a week.
- The view must provide a means for viewing and selecting other weeks to view.
- The view must allow users to select different profiles in the case that they have multiple shoes using the LogiSteps technology.

Steps by Weekday

When a user navigates to the "Steps by Weekday" tab they should be presented with a view that summarizes their total step count over the past 6 months broken down by day of the week. For a view to satisfy this requirement, it must meet the following requirements.

- A graph must be shown which presents days of the week on the x-axis and total step count on the y-axis.
- All of the step data for a user should be used when displaying step count.

Activity by Week

The "activity by week" view should present a user with a summary of their activity vs. inactivity statistics for a given week. To meet this requirement, the view must meet the following criteria

- A user should be presented with a graphic for each of the seven days of the week.
- Each graphic for a day of the week should visually present active time percentage against inactive time percentage (such as stacked bar charts, pie charts).
- A user must be able to select different weeks to display data for.

Pressure

The pressure tab should present a view which will summarize the pressure statistics for a user's feet while using the LogiSteps system. This view should provide a means of viewing changes over time that could result from changes in posture and fitness. To satisfy this requirement, the view must meet the following criteria

- Present a graphical visualization of a user's pressure they place on certain parts of their feet using a heatmap placed over an image of an insole.
- The view should show data for the past day, past week, and past month to communicate changes in behavior.
- Each heatmap should use an aggregation of the data collected during the timespan that is being conveyed.

Map

The map tab should use GPS data to visualize a user's step data by geographical area. To satisfy this requirement, the view should satisfy the following criteria.

- The view should use a map library, such as Google Maps, to display a map behind any data points.

- Steps should be plotted on the Map to convey a user's geographic movement.
- Only a single day of step data should be shown at once.
- A filter should be implemented to allow users to select data for different days.

4. Technology and Concept Report

4.1 Introduction

The purpose of this document is to document possible approaches for Team Omicron to successfully implement the LogiSteps project, as proposed in the formal project proposal document. Team Omicron first theorized a general data flow for the system, identifying key components and aspects of the design. In-depth analysis of market solutions and compatible technologies that could be used for successful development of the project was performed, helping to identify the best technologies to be used in the system design. In this document, the general system (without specific hardware and software components) will be discussed, familiarizing readers with the general structure of the project. After Identifying the basic project requirements, component technologies that help achieve the respective system requirements will be discussed in depth, ending with a conclusion of the best available option. The possible modularity of this project's design makes it possible to choose many technologies independent of other components in the project. Lastly, the final approach will be discussed; this will be the optimal design using the selected component technologies based on team Omicron's research.

4.2 General Approach

4.2.1 General System Requirements

LogiSteps, as introduced in Team Omicron's proposal, is a full stack product designed to track user's fitness activity. To be invisible to the user, providing a seamless user experience, the fitness tracking device is going to be embedded into a user's shoe. This device will use sensors tracking the pressure between a user's foot and the ground, feeding data into a data collection device that is also located on the user's shoe. In addition to the sensors, whether provided by the sensors or not, the data collection device will require enough power to collect sensor data and also transmit the data off the device. The data collection device will need to transmit data to a receiving device, whether that be in close proximity to the data collection device or not. Upon receiving data, the receiver will need to store and the process the data for long term retrieval, indexed by time. A final system component will be required to present data to a user. In short, the general requirements of the system, regardless of actual implementation and technology selection, are those shown in figure 8.

4.2.2 General Block Diagram

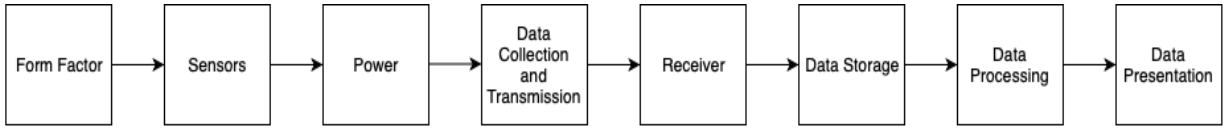

Figure 8 - -General System Requirements for achieving LogiStep's functionality, as described in the project proposal.

4.3 Component Technologies

Each component shown in figure 8 is high-level, only describing the general function of the specific stage in LogiStep's design approach, satisfying the high-level requirements of the project. By drilling further into each stage, specific technologies can be identified which may provide the tools needed to achieve successful completion of the project. In the following sections, several hardware and software technologies will be discussed, accompanied with deep analysis and comparisons to other alternative technologies. After discussing the advantages and disadvantages of varying technologies as they pertain

to design of LogiSteps, a technology that best suits the need of the specific stage in the design will be chosen. Using the conclusion of each technological component, a final system design will be discussed.

4.3.1 Form Factor

The form factor of the device is one of the most important aspects. Ideally it will be an insole that requires no maintenance and is thin enough so as to not reduce comfort. Additionally, the material will need to be flexible and temperature/pressure resistant to not be damaged under normal use. The best option for this seems to be some type of rubber. The material also needs to be malleable enough easily shape and work with to fit other design requirements.

Other than the material used, the characteristics of the device need to be considered. In a perfect world, the insole will be as flat as possible and contained entirely in the shoe. However, due to other components required for the device, it may be necessary to have a thicker sole with the main electrical components attached outside of the shoe.

4.3.1.1 Options

Silicone Rubber

Silicon rubber can be cast and cured into any shape (including thin ones); it is also very robust making it a desirable choice. The rubber would be used to help activate the sensors and house the electrical components. Activating the sensors may require deformation which can be easily done with a shape that is producible with silicone rubber. Housing the electrical components requires a minimum amount of stress on them and that they remain safe from static. This can be achieved using silicone rubber since it is both an insulating material and can absorb much of the shock/stress put one them.

Cost

The cost of silicone rubber depends on the quality, quantity, and form it comes in. The ideal form is a type of castable liquid. A sizeable amount of this costs $20-$50.

3D Printed Flexible Rubber

The main benefit to this approach over silicone rubber is that it can be designed with more precision. It has all the same benefits as silicone rubber as well. A drawback is that it requires a more complex design since it would not be trivial to get the necessary electrical components inside.

Cost

Assuming free access to a 3D printer, the cost of this would be the cost of the rubber filament itself. There are a few options for an ideal filament and all are within $30-$40.

The labor cost of this should be noted as well. It is a more complex process and design resulting in potentially a significant amount more time devoted.

Polycarbonate/Plastic Sheets

This option is the thinnest of them all. The concept here would be that nothing other than the flat sensors are contained in the shoe and the electrical components are housed outside the shoe. The benefit to this is a maximum amount of comfort since the insole is paper thin. The major drawback however is the second part of the device outside of the shoe that is required as well.

Cost

There are many options to the type of material, vendors, sizes, and cost. This option would cost as little as $10 and no more than $20.

4.3.1.2 Other Form Considerations

The layout of the sensors and electrical components needs to be considered in any form factor. The sensor layout depends largely on the number of sensors to be used, which depends on power requirements and size of electrical components. But the simplest and most desired concept will be the fully insole version of the device, with at least two sensors on the heel and forefoot to get pressure differences between those areas. The electrical components will need to be housed in an area that has the least amount of pressure; this would be the bridge of the foot. A concept of this layout is shown in figure 9.

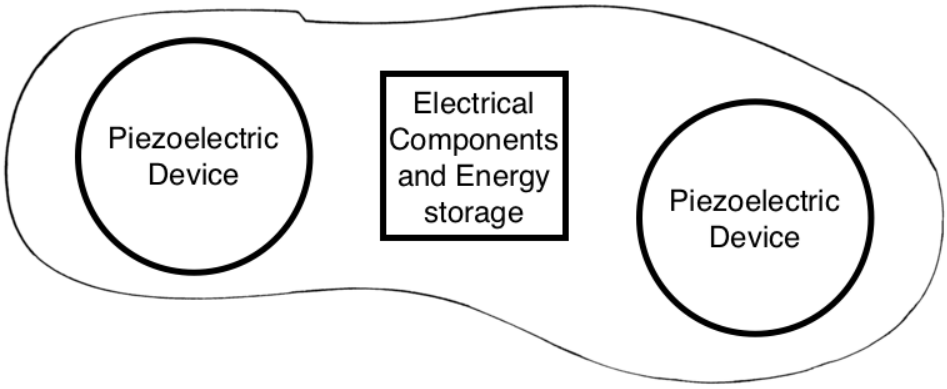

Figure 9 - Theorized physical form of the embedded LogiSteps application.

4.3.1.3 Form Factor Conclusions

The most attractive approach seems to be the 3D printing of flexible rubber. It allows detailed design which may be important for sensor development, and while it is more complex than silicone rubber, the increased quality control may end up saving time and effort.

The thin polycarbonate /plastic material would the most different design. It would require housing the electrical components somewhere other than the insole, which would most likely be outside of the shoe entirely. It would be the simplest option because it requires no complexity in design and it also allows more freedom with electrical component sizes but less

freedom with sensor choices. Because of these things, the reason to go with this option would most likely be due to other factors of the sensors and electrical components.

4.3.2 Sensors

The most important requirement for the sensors is that they remain thin and robust. Ideally, power would be obtained from these sensors as well, removing the need to charge the device. If possible, the sensors would be capable of detecting specific amounts of pressure so that factors such as weight can be determined.

4.3.2.1 Options

Piezoelectric Material

Piezoelectric sensors appear to be a great choice; they're thin, scalable to meet system requirements, produce power when stress is applied, and certain types of piezoelectric material are very robust. The material used for this would ideally be PVDF (polyvinylidene difluoride) since it is the thinnest type of piezoelectric material that is commonly used and can handle the most stress. The only problem with PVDF is that it is generally expensive compared to the next best option, piezoceramic. Piezoceramic is capable of producing more power, it's cheaper and most easily accessible, but it is quite brittle. Either option will require the form factor to be one capable of deforming the material significantly enough to produce power but not enough to damage the sensors. The largest downside to piezo material, besides that cost, is that it requires extensive work to calibrate the sensors well enough to detect specific pressure.

Cost

The best cost option for the sensors would be piezoceramic deposited on copper. This can be as cheap as $1 per sensor up to $3 per sensor. This would also make prototyping very easy since they can be damaged without too much loss.

The best option functionality-wise would be a piezo PVDF film. This option is ideal because it is extremely thin (µm range) and can be layered for voltage and material strength. This option comes at a large price however - the best options seem to be around $133 per 8''x11'' sheet.

Pressure Dependent Resistive Sensors

Pressure dependent resistive sensors are the common approach to detecting pressure. These sensors have a bassline resistance that changes based on the pressure applied to them. This would allow for very specific measurement of pressure and there are far more choices (most are cheaper then piezo material). The downside to the resistive sensor approach is that power cannot be harvested from them. Instead, it would require additional power to measure them. Another benefit to this approach would be that the sensors would not require an activator, meaning the insole could be entirely thinner.

Cost

Reasonable quality sensors start at around $5-10 each but can be up to $100 each for very high-quality ones. The high-quality ones are usually made for industrial type applications so those are

not only out of the price range, but unnecessary for the application. $5 sensors should be suffice.

4.3.2.2 Sensors Conclusions

The PVDF film is the top choice for the sensors due to its power harvesting ability. The goal of the LogiSteps is to never require a charge and force resistive sensors would require the use of a battery with significant energy density (far from ideal). On the other hand, the cost of piezoelectronics, as well as their complexity, means that other power options need to be explored in case as a backup if piezoelectronic sensors fail. Based on this research however, Team Omicron concludes piezoelectronic sensors to be the best technology for measuring fitness data.

4.3.3 Power

A goal of the device is to never need to be charged. This requires either a battery or supercapacitor capable of sustaining the device for its entire lifetime, or the ability of the system to harvest energy from human locomotion. The harvesting of energy can only be done with piezoelectric materials. Other types of energy harvesting were considered such as triboelectricity, thermal energy harvesting, and even RF energy harvesting, but the investigation into those options proved them to not be viable.

4.3.3.1 Options

Battery

The battery option is the least desired since batteries tend to not do well under pressure or when exposed to heat. This would require the form factor of the device to be partially outside of the shoe to avoid danger risks. The largest benefit of the battery however is that the device could potentially be powered for its entire lifetime on a single charge. At a rate of sending 20+ sensor values every second, using a low footprint battery, the device could last several years before needing to be recharged.

Cost

Thin film Lithium Ion Polymer batteries cost as little as $5 each and don't get much above $20 each. The number of options and sizes here are quite significant as well.

The cost of design needs to be mentioned as well. Designing the battery power supplying system would be nearly trivial compared to the piezoelectric power delivery system, so it may be the better option based simply on time constraints and labor required.

Supercapacitor

The main benefit to a super capacitor over a battery is that a super capacitor does perfectly fine under pressure, is not significantly affected by heat, and there are less risks. The main drawback is that they have a much lower energy density. A supercapacitor would only be able to power the device for a matter of months. A small drawback is the fewer amount of supercapacitor options, but another benefit is that charging the device would take only a matter of seconds.

Cost

Thin film, high capacity supercapacitors are generally all around $5-$8. The DMHA14R5V353M4ATA is currently the best seeming option due to its size, capacity, and voltage range; They're $7 each. Due to their size, it's possible to even have multiple in each shoe, potentially extending the battery life to lifetime of device capable ranges, but at a higher cost.

Piezoelectric Energy Harvesting

By far the most attractive option because this system never requires a charge, nor any type of costly or dangerous components. It is however, also by far, the most difficult system to develop. When piezoelectric material is deformed, it produces a voltage. That voltage can be harvested to power the rest of the device.

Piezoelectricity Harvesting Design

Since energy needs to be controlled to power the device at proper and consistent voltage/current limits, other components will be needed. These other components would be diodes, capacitors, and a voltage regulating chip. The diodes and capacitors depend largely on the voltage and amount of energy in general that the piezo material can produce. A good option for a voltage regulating chip this is the LTC3588-1; this is a silicon-based chip that can maintain proper voltage limits while being extremely efficient. The design of the energy harvesting circuit is shown figure 10.

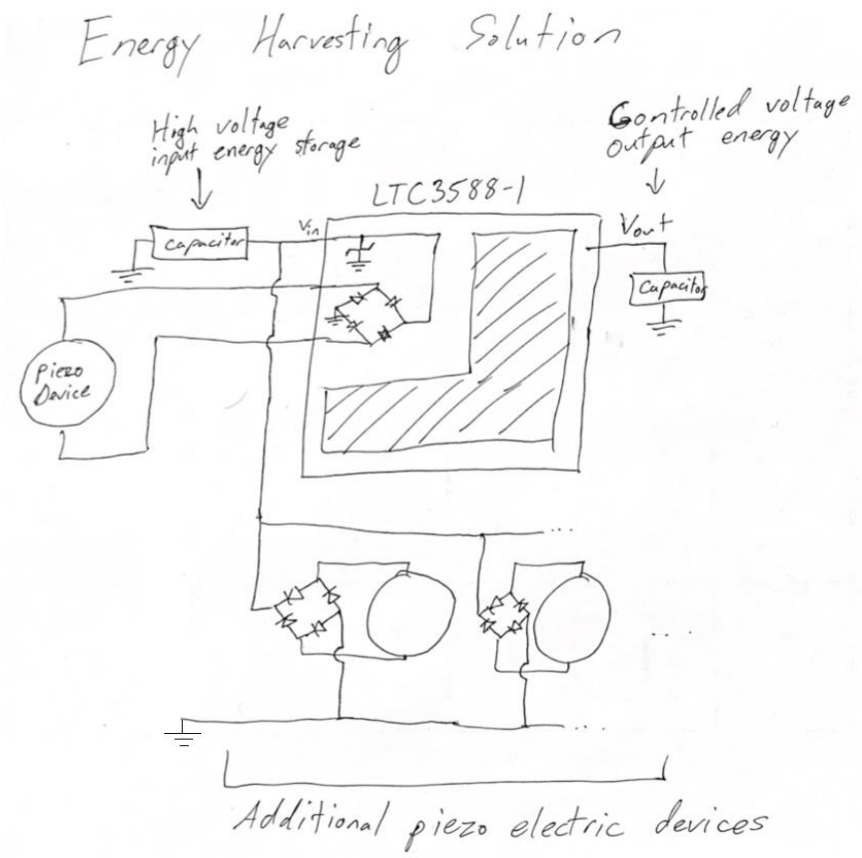

Figure 10 - Prototype circuit design for the energy harvesting circuitry.

The voltage generated by the piezo devices will alternate, thus full bridge rectifiers will be used to polarize the voltage, so it can be stored on a capacitor. The uncontrollable nature of the voltage requires that a buck regulator or similar type of voltage controller be used so a voltage level adequate to power a microcontroller can be output. The LTC3588 has voltage output controlling components, and high voltage energy can be stored on the input capacitor to be used to power the output capacitor at a controlled voltage. This system requires two capacitors and a full bridge rectifier circuit for every additional piezo device used, giving the ability to scale the system to meet requirements as long as the additional components can fit on the shoe.

The piezoelectric device would also require an activator. This activator would need to have the ability to deform the material to produce a voltage. This activator would be built into the form factor of the insole. An example of an activator is shown in figure 11.

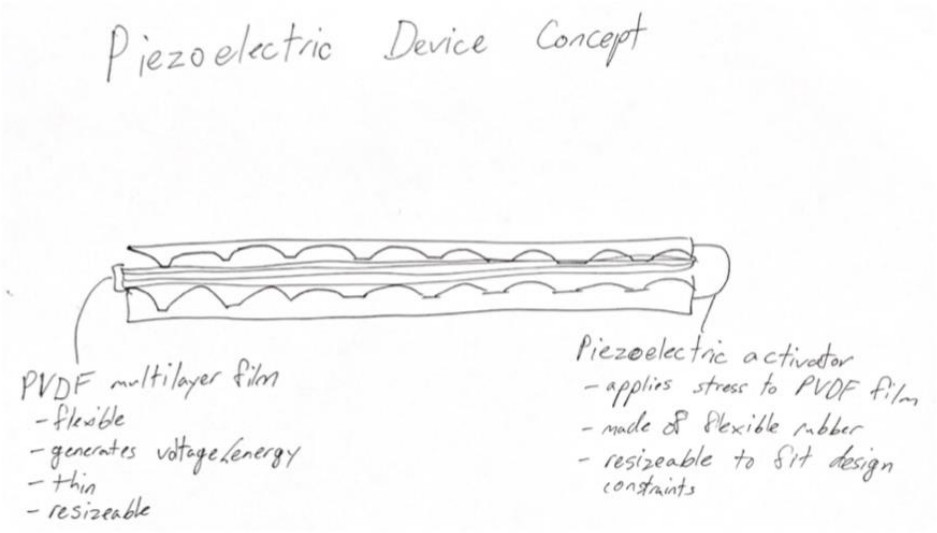

Figure 11 - Theorized design for the piezoelectric sensors. Note the rigid edges in the design, enhancing voltage output.

Cost

The piezo material itself is the costliest aspect of this approach, which was previously. The LTC3588-1 is the next most costly component at about $7 each. The diodes and capacitors are all in the range of tens of cents, so until mass production begins, their cost will be considered negligible.

4.3.3.2 Power Conclusions

The piezoelectric energy harvesting option is the most desirable for many reasons, including cost, but comes at the price of difficult development. The supercapacitor option is the next most preferable due to its robust nature and scalability. The battery is the least desirable option because of danger risks, but it is also the one that requires the least development, so it is still a viable alternative.

It is also possible to combine the supercapacitor and piezoelectric ideas. If a supercapacitor is used, it can supply the device with power for a number of months, and when combined with piezoelectric energy harvesting, is much more likely to be able to supply the device with power for its entire lifetime. This approach would be taken if the energy harvesting alone is found to be incapable of supplying enough energy to fully power the device but may be enough to extend the lifetime of the device when supplied with a significant enough amount of initial energy. The only drawback to this approach seems to be that a supercapacitor as well as the energy harvesting circuit adds to the cost and footprint of the system.

4.3.4 Data Collection and Transmission

A microcontroller will be used to collect sensor data and transmit to a receiver. Since there are many options for microcontrollers to use, research was performed to find a microcontroller that had the necessary components to read sensors and transmit data over Bluetooth. Specific requirements the microcontroller include: ADC count, power, size, cost, lead time, programmability, and documentation.

After researching microcontrollers, a handful of microcontrollers with Bluetooth capabilities embedded in them were found. It was decided to choose a microcontroller with embedded Bluetooth rather than using an external component, which would add additional complexity to the system.

4.3.4.1 Options

By researching, a handful of microcontroller options were found that could be used for the project. Each microcontroller is embedded with Bluetooth and meets each of the specific requirements mentioned above. The first microcontroller option is the CC2540 chip from Texas Instruments. The second microcontroller option is the nRF52832 chip from Nordic Semiconductor. The third option is the ESP32 chip from Espressif Systems. Each microcontroller option has its own advantages and disadvantages, and they will be explored by reviewing the specific requirements mentioned above for each option.

ADC count

The ADC count of the microcontrollers is important because there needs to be enough channels to read each sensor input. Both the CC2540 and the nRF52832 have 8 channels for a 12-bit ADC. The ESP32 excels in ADC count, having 18 channels for a 12-bit ADC. Although it seems that the ESP32 is the best choice in terms of ADC count, a simpler design in terms of sensors is desired. Therefore, the microcontroller won't require more than a few ADC channels, so the large number of channels with the ESP32 is not very important.

Power

The power requirement for the microcontroller is one of the more important requirements, since the hope is to be able to power the microcontroller with the voltage created by the sensors. The CC2540 has a supply voltage range of 2 – 3.6 volts, the ESP32 has a recommended supply voltage between 1.8 – 3.3 volts, and the nRF52832 has a supply voltage range of 1.7 – 3.6 volts.

Overall, the nRF52832 chip has the best power options. Not only does it have the smallest voltage requirement, it also provides a low power mode, and allows the microcontroller to drift in and out of a powered state.

Size

Since the space for housing electrical components will be limited, small size for any microcontroller option is essential.

SparkFun provides development platforms for microcontrollers and provides a platform for both the nRF52832 and the ESP32 microcontrollers. Both development boards are the same size: slightly longer and less wide than a quarter. The CC2540 development kit is a lot larger than the SparkFun development platforms.

Overall, the nRF52832 and ESP32 development boards are both very size efficient. They would be small enough to use in the system design, while also being large enough to properly work with and connect the sensor inputs.

Cost

During development, it was determined that it would be best to have multiple microcontrollers to work on at the same time, if necessary. Also, it would be best to have an extra microcontroller in case there are any problems or malfunctions with the ones that are worked with. Therefore, cost is an important factor for the project. The development boards mentioned in the size section both sell for $19.95, while the CC2540 development kit sells for around $50.

In terms of cost, the development boards provided from SparkFun Electronics are the cheaper options.

Leadtime

Leadtime is an important aspect of this development process, since the project is expected to be finished by the end of spring 2019. Therefore, there is a need to choose a microcontroller that is both readily available and ships out quickly. Both SparkFun Electronics development boards are in supply and can be shipped within 3 – 11 business days (or faster with a cost increase). The development kit for the CC2540 microcontroller is also readily available and will ship in 4-7 business days.

Overall, the availability and lead time for each microcontroller chip is about the same, so it isn't a very important deciding factor.

Programmability

Programmability is one of the most important aspects of a microcontroller for this project, since the preference is to have a microcontroller that is as simple to work with as possible. The programmability for each microcontroller can be determined by looking at documentation provided for the development boards that would be worked with.

The CC2540 programming kit provides a datasheet for the microcontroller kit. It provides information about the microcontroller itself like a regular datasheet while also providing information about software available to use. It doesn't provide much in terms of program examples or troubleshooting. The nRF52832 and ESP32 development boards both provide a hookup guide to get started with their board. They both provide example circuits and programs that can be used to ensure that the microcontroller and board is working too. The difference between the two is the documentation for the microcontroller itself. The ESP32 provides a well-

documented IDF along with a few example applications. The Nordic website provides numerous example applications as well as an open forum for continuous support when working with the nRF52832 microcontroller.

In terms of programmability, the provided example applications and code for the ESP32 and the nRF52832 are extremely helpful for programmers looking to use the product. Beyond sample applications, the nRF52832 microcontroller has an extensive amount of support and application examples for it, making it most likely the easiest product to work with.

Documentation

The documentation for a microcontroller is an important aspect for anyone wanting to use it.

The smallest datasheet is the CC2540 datasheet with 33 pages. It is mostly made up of schematic diagrams and characteristic information. The next smallest datasheet is the ESP32 datasheet with 43 pages. It is mostly made up of pin definitions and peripheral information. The largest datasheet is the nRF52832 datasheet with 555 pages. This is far and away the most detailed and most helpful datasheet out of the microcontroller choices. It contains a very detailed table of contents that makes it easy to navigate to a certain topic of interest.

In terms of documentation, the nRF52832 microcontroller has the best documentation provided for users. The datasheet covers nearly every possible topic of interest, making it very easy to find answers for any questions that need to be answered. Not only is the datasheet much better than the other choices, but the Nordic Semiconductor website provides an extensive amount of helpful documentation including starting guides, product specifications, and software user manuals.

4.3.4.2 Microcontroller Conclusions

After looking at the specific requirements for each microcontroller option, it could be seen that each would be a viable option for this project. The ADC count, power requirement, size, and lead time for each microcontroller is acceptable for what this project would need. Therefore, choosing a microcontroller came down to the programmability, cost, and documentation.

Our group decided that the Nordic Semiconductor's nRF52832 microcontroller is the best option. Not only does SparkFun Electronics provide an incredibly useful development kit with very detailed instructions, but the Nordic website provides much more documentation and support than the other microcontroller options. The nRF52832 should be the most accessible microcontroller for this project and should be the least challenging to work with.

The cost of purchasing a single nRF52832 microcontroller development board is around $27 including shipping. The cost of purchasing multiple microcontrollers to work with (and have one as backup) would be around $70 total.

4.3.5 Receiver

How data is transported through LogiSteps, and how the user views it quickly, is what makes the product useful. Using a mobile app on a mobile phone is not the only way to transport the data to the server, but it minimizes the amount of power needed to get data from the microcontroller to the server. Some other advantages of using a mobile phone with a lightweight app to relay fitness data to the server are as follows:

- **Power Saving –** The mobile app allows the of use Bluetooth on the LogiSteps device. Bluetooth is much lower power than the alternative which is LTE. This will save power on the device which further enable the device to be passively powered.
- **Ease of Use –** The mobile phone is something everyone has. Almost everyone carries it with them everywhere. This allows for ease of use and no need to store data on the LogiSteps device or the phone (provided the phone has LTE turned on or is connected to WIFI).
- **Small Application –** Because the web server is storing the data, and is the primary method of displaying the data, the app only needs to transfer the received data to the web server. It also means that the app does not need to do any heavy processing of the data. This will keep the app small for those who have limited space on their phones.
- **Updates –** The use of the app store allows the pushing of updates to be seamless. Yes, the application needs to be pushed out to anyone who owns LogiSteps. However, the update does not have to come directly from development. The update can be developed outside of the store, then pushed when ready.
- **Additional data** – The mobile phone can provide additional data points such as GPS data.

There can be two parts to the receiver - the aspect that receives data from the microcontroller over Bluetooth, and the aspect that relays the data to the web server if data processing and UI is not being supplied by the mobile phone.

4.3.5.1 Data Receiving

The method for receiving data from the microcontroller has a few different options: LTE or Bluetooth. Between these options, only one is acceptable for the project's constraints: Bluetooth. Bluetooth has a low energy mode, so it does not take up too much power. Power is a large constraint for LogiSteps to be passively powered. LTE has much higher power requirments, ruling it out

Android has a library already in place for Bluetooth connections. It also has protocols in place to store the key for the product, so a connection can be done automatically after the pairing is done. The libraries are stored in three different permissions. These libraries are very well documented on the android developer page.

4.3.5.2 Data Transmission

The LogiSteps data, if it is assumed to be sent to a server, sent straight from the embedded device to LTE, or from the Android device and onto the internet, will need to encapsulate the

data in a data protocol. There are many data protocols that can be used to send data over the internet, either over LTE from the embedded system, or from a mobile phone app.

MQTT

MQTT is a Message Queueing Telemetry Transport. It is a lightweight (low bandwidth), publish/subscribe protocol that runs over TCP/IP. It is ideal for one-to-many information delivery. MQTT offers three QoS, or qualities of service, for its messages. At most once, the information might be lost. At least once, duplicate messages might occur. And Exactly once, no loss or duplicity. So, there is low bandwidth usage, medium, and high. Of note is its message header, which is a fixed 2 bytes in length. With these features MQTT was designed for constrained, low bandwidth devices, while assuring reliability in its messages. MQTT can also ensure the security of its data using SSL, or other data encryption methods.

Advantages of MQTT for LogiSteps are low bandwidth, reliability, and low power consumption since it is lighter weight than HTTP and will consume less of the user's phone battery.

A Stephen Nicholas performed a power profiling comparing HTTPS and MQTT. What he found was that for a connection receiving sporadically MQTT used about 70% as much power as HTTPS over 3G, and over WIFI only used about 10% as much power. Sending as fast as possible saw MQTT using 2% less battery/hour over 3G. For sending data he found that MQTT used 1% less battery/hour over 3G, and 2% less battery/hour over WIFI. MQTT also had a better rate of successful packets. So, for LogiSteps, MQTT's clear advantage is it will use less energy. The expected usage of LogiSteps and its app could lead to a considerable saving of battery life for the user, were MQTT to be the chosen data protocol.

For MQTT on android there are a few libraries out there that can be used. There is the Moquette library on GitHub which has almost one-thousand stars. Moquette also uses Netty for encoding/decoding. This library would be fairly easy to begin using. There is also an MQTT client by IBM using either IBM®MessageSight or IBM WebSphere® MQ, which has a lot of tutorial/development help. It also has a C library, Mosquitto, which would enable development directly on the embedded system.

CoAP

The next protocol of note is CoAP, or Constrained Application Protocol. Like MQTT CoAP is designed for IoT. CoAP is defined by RFC7252, written by the Internet Engineering Task Force (IETF). CoAP's big points are that it is built for nodes with small amounts of memory, and networks with high error rate, which can be translated to a phone app environment, and it is built for IoT and many, many nodes. Its lightweight attributes include a small header of 4-bytes and requiring as little as 10KB of RAM and 100KB of code space. CoAP is also built on the REST model, and uses a request/response, so being very similar to HTTP can easily interface with HTTP services. And of course, CoAP data can be secured, with the default being 3072-bit RSA keys. It should also be noted that that the power saving aspects of MQTT will likely also apply to

CoAP, maybe even more so, which would be important for any battery-operated device using the protocol.

Like MQTT, CoAP has some open libraries that could be used for LogiSteps. The first is Californium. Californium is under the Eclipse Foundation and is well documented and supported. It implements CoAP (RFC 7252), the Observe draft (RFC 7641), the block wise transfers draft (RFC 7959), and DTLS1.2 for security. It provides CoAP-HTTP cross-proxy support. And it is designed as a scalable model for IoT. Another implementation is SpitFireFox on GitHub, which is notable for, like Moquette of MQTT, encapsulating CoAP in Netty, the asynchronous, event-driven framework. This would be a fairly easy implementation to set up and begin using for LogiSteps. CoAP also has a C library, libcoap, that would enable development of CoAP directly on the embedded system.

HTTP

Of course, there is HTTP, HTTP/2, and HTTPS. They are the standards of internet communication. HTTP, Hyper Text Transfer Protocol, used by the Internet. HTTP is a stateless protocol, meaning there is no knowledge of what previously occurred on the network, that uses request-response protocol to communicate between client and server, and it operates on TCP/IP. HTTP is not designed with constraints in mind and has very large and complex headers. Another fault is HTTP is not a secure method of data transfer and communication.

However, there are the HTTP upgrades HTTPS and HTTP/2. Both of them are improvements and widely used. HTTPS secures HTTP by using either Transport Lay Security (TSL), or the Secure Sockets Layer (SSL). This and adds protection and privacy to the data, which is something LogiSteps will most likely want for its data. HTTP/2 was created with several goals to improve HTTP. It adds the ability to allow clients and servers to choose their data protocol. It maintains compatibility with HTTP 1.1. And has improved upon the large overhead of HTTP, thus reducing latency. Of note is that HTTP/2 itself is not secure but does support HTTPS using TLS.

As HTTP is the world standard for internet communication there are many more libraries and clients for Android that LogiSteps could utilize for itself. For example, there is the Google HTTP Client Library for Java, supporting Android 1.5 and higher. There is OkHTTP which works as an HTTP & HTTP/2 client for android and has almost 30,000 stars on GitHub. And there is Retrofit which turns the HTTP API into a Java interface. Retrofit also has almost 30,000 stars on GitHub. Due to the wide spread usage of HTTP and all the libraries there are for it.

Data Protocol Conclusions

MQTT and CoAP have their advantages as communication protocols for IoT. However, HTTP is a more versatile protocol, and due to being the global standard there are many libraries that make using HTTP very simple and easy to use. Should LogiSteps remove the Android phone app from the design, and instead send the data directly from the embedded device to the web service over LTE, CoAP would be the optimal choice for its IoT benefits and HTTP similarity. However, as LogiSteps does not require the IoT benefits provided by MQTT and CoAP specifically for communication from an Android device, LogiSteps will use HTTP on the Android application due to the benefits provided by HTTP's widespread use, documentation, and libraries.

4.3.5.3 Receiver Conclusions

While receiving data from the microcontroller embedded into the shoe could theoretically be done using LTE or wired connections, a mobile application receiving data over Bluetooth offers significant advantages in terms of power savings and usability. Based on the system that processes and presents the data, a mobile app receiving data could then relay the data to a final destination using higher level communication protocols.

4.3.6 Data Storage

The LogiSteps product will generate data pertaining to a user's movement originating from their shoe. Data will be relayed from the shoe's sensors, to a microcontroller, where actions will need to be taken to move data into long term memory for analysis and presentation. The medium of the long-term storage could be a database able to perform efficient storage and querying on timeseries data, or the data couple simply be placed into a hard drive on a web server or mobile device. Application data is being considered timeseries data since it will likely be indexed and queried using time.

Choosing the right medium for long term storage of application data is highly dependent on the structure of the data, it's rate of generation, how the data is being used, and other factors such as the ability to quickly access and perform analysis on the data. For LogiSteps, the data storage medium needs to scale well as the number of rows increases with little or no deletions. Additionally, the LogiSteps application will need to query large amounts of data based on the time that the data was generated. A good storage medium for the LogiSteps application is one which allows for easy, efficient querying of data indexed optimized for timeseries data.

Some popular storage technologies that exist for long term storage are databased such as MongoDB, MySQL, PostgreSQL, and simpler technologies like hard drives. Several options for long term storage of LogiSteps application data are compared in the following sections, followed by a conclusion on the best storage technology for LogiSteps.

4.3.6.1 SQL Databases

SQL databases, such as MySQL and PostgreSQL, are advantageous in that they make use of powerful, long established standards for querying and manipulating data. As a result, users of the database can access and view subsets and supersets of the database, helping answer analytical questions and present applicable data to a user. Additionally, SQL databases are accessed using a standard SQL language, which can make it easier to access data without having to write a lot of code. While SQL databases can excel in querying and keeping consistent data, they often do not scale well. An experiment done by a TimeScale engineer showed that the insertion rate for a PostgreSQL database decreased at a near linear rate when increasing the number of records in the database, beginning to level out near 400 million rows. This could present a problem as LogiSteps scales up, increasing the number of users logging data into the database. Another disadvantage of using a SQL database is the inability to store complex data structures such as objects; a SQL database requires all columns to be a single scalar attribute.

While this likely won't impact LogiSteps, the inability to scale with dataset size presents a problem.

Cost

Pricing can be complicated and depends largely on the database being used and it's use. For development, most SQL database options are free, and only require payment for commercial uses. As a result, the cost of using a SQL database for senior design would be free. Cost would then be dependent on hosting options if the application were hosted on a web server. Even then, most hosting services provide a free development period.

4.3.6.2 NoSQL Databases

A popular database alternative that scales well are noSQL databases such as MongoDB. noSQL databases have taken off in popularity in previous years due to their various advantages over SQL databases such as

- Ability to handle well structured, semi-structured, and unstructured data
- Quick prototyping and development (structure does not need to be well defined)
- Ability to store objects
- Efficient, scale-out architectures

While these advantages can be beneficial, LogiSteps only needs the ability to scale up, as well as quick development and prototyping. In addition to the advantages of a noSQL database,noSQL present several disadvantages as well. Most notably, noSQL databases lack many of the business intelligence and analytic features that SQL databases are able offer or paired with. While MongoDB provides a service for timeseries data, it lags behind more specialized solutions designed for timeseries data.

Cost

Like SQL databases, most noSQL databases such as MongoDB provide a free version for developers. Once a product is taken public, MongoDB will charge a price, depending on the needs of the platform. For the length of senior design, noSQL databases should remain free to develop with.

4.3.6.3 TimescaleDB

Timescale is a SQL based database which is designed specifically for timeseries data. Specifically, Timescale is built on PostgreSQL, and as a result, is compatible with any tools that work with PostgreSQL. Being built on a SQL database allows Timescale data to benefit from the same advantages of traditional SQL databases, but it is designed to scale like a noSQL database. Essentially, Timescale attempts to bridge the gap and offer some of the most notable benefits of a traditional SQL database and a noSQL database. To achieve scalability similar to that of a noSQL database, Timescale partitions data into time-based chunks which allows for faster performance at scale. Some of the most notable advantages of this database are

- SQL database designed to be compatible with SQL compatible tools

- Scales like a noSQL database
- Comes with built in timeseries specific analytical functions
- Abstracts data as one continuous table for simplified queries.
- Can be managed like a traditional PostgreSQL database
- TimescaleDB integrates directly into the PostgreSQL query planner and execution engine

Cost

Timescale offers an installation that does not involve payment. This means that TimescaleDB can be used to develop for free. Additionally, TimescaleDB can be installed on most of the popular dynamic hosting services such as Google Cloud Platform and AWS. TimescaleDB does offer an Enterprise addition of its database, which provides custom deployment assistance and proprietary functions. For its use in senior design, the free install of TimescaleDB should suffice.

4.3.6.4 Data Storage Conclusions

LogiSteps aims to provide a user interface for customers to interact with and view their fitness data. As a result, a powerful query language will be needed that will make it possible to easily calculate aggregate data and find subsets of their data based on certain parameters. Additionally, the storage medium used to store user data will likely need to interface with a third-party visualization library or an object relational mapper. Such tools are often designed to interface with SQL databases due to their standardization. These needs are well suited by a SQL database; however, the SQL database will likely not scale well for LogiStep's timeseries data. A noSQL database would likely be best for scalability but would sacrifice the benefits of a SQL database. To fulfill both requirements, TimescaleDB appears to be the best option for LogiSteps. Timescale DB is a database designed specifically for timeseries data that is built to scale, but is built on, and takes advantage of, the SQL properties in PostgreSQL. Other timeseries databases with similar features, such as influx, may be used if needed.

4.3.7 Data Processing

Data processing could potentially be performed either on a web server, centralizing all processing. Another option for LogiSteps is to perform all data processing on the mobile phone of each user.

4.3.7.1 Mobile Phone

Data processing on a mobile phone has a few advantages, but many disadvantages. Using a mobile app to perform processing of fitness data has a few advantages:

- Always on you - the data will always be there and able to be processed, even when not on LTE
- Clean data – the data sent to a server would already be formatted
- Display faster – the app would not have to pull data to display general information

There are many disadvantages as well. The mobile app could prove to be detrimental in these ways:

- Battery use – processing the data on the mobile app will use more of the user's phone battery

- Slowdown app – processing the data in the app would slow down the app being running
- App size – the app would greatly increase in size if processing of the data was on it
- Updates – the updates are not mandatory, therefore not doing it could cause problems
- Development effort – Team Omicron would need to develop the same app for multiple platforms

These factors alone may drive users to not use or not want LogiSteps. Those who have older phones and need a charge every few hours due to a bad battery would more than likely not use LogiSteps at all. Additionally, those who do not have a lot of storage space may be unable to download the app.

4.3.7.2 Web Server

Processing all data using a web server offers several advantages over performing all data processing on a mobile phone. Many of these advantages are similar to the advantages of using a web interface for presenting data to users. By processing data on a server, LogiSteps could take advantage of the following:

- Single Platform – no need to develop for multiple platforms
- Installation-less – Users do not need to install an application on their phone
- Updates – Updates only need to be published to LogiStep's web servers
- Administrative overhead – No need to gain approval from companies such as apple
- More powerful hardware – Not limited to processing power of mobile device

To assist with developing back-end servers for processing data in web applications, several back-end frameworks exist to aid in development. A few of the best options available for LogiStep's project are discussed.

Express

Express is a framework that uses NodeJS for development. NodeJS is an event driven, non-blocking I/O language, which makes it ideal for applications driven by user interaction and events. LogiSteps is predicted to be mostly a data driven application, with I/O events occurring in regular, infrequent intervals. Due to this, NodeJS is most likely not an idea language for writing the LogiSteps web application.

Express is also a lightweight framework that comes with support for both SQL and NoSQL databases, allowing easier integration of data for applications. Additionally, Express offers features such as routing, view caching, and middleware chaining. Essentially, express is a minimalist web framework that has as little functionality as possible, while providing a series of middleware function calls for executing code, changing request and response objects, ending the request/response cycle, and calling the next middleware in the pipeline. This gives applications using Express a log of flexibility, without bogging down an application in express specific code. Additionally, by using express and NodeJS, developers have access to NPM – the largest open source library in the world.

Django

Django differs from express in a multitude of ways, providing more structure, and utilizing a completely different programming language. Django is a framework that helps build back-end applications using Python, a common language used for data driven applications. While Express was a framework with little requirements for structure, Django enforces rigid application structure, following the MVC pattern for representing objects and state. While a stricter structure requirement can limit the possibilities of an application, Django provides several features that frameworks such as Express and Flask do not provide. In particular, Django has a relational database interface that is built into the framework. This provides access to a build in object-relational mapper, support for SQL database managers, the ability to quickly switch between different DBMS, and allows the application to be closely coupled with the backend database.

Additionally, Django advertised several more features that frameworks such as Express and Flask do not advertise. Some of these features include built in Security (SQL injection, cross-site scripting, cross-site request forgery, clickjacking), user authentication, administrative controls, scalability, and more. Django provides an all-in-one solution, and this helps assist in quicker application development. In frameworks such as Express and Django, these features may only be available through third party plugins and libraries. Django is designed for building data driven applications, such as the web application for LogiSteps.

Flask

Flask is a back-end framework for applications written in Python (similar to Django), but with limited functionality (similar to Express). Flask was designed to provide a web framework that is focused on simplicity, minimalism, and fine grain control. As a result, the Flask framework has a smaller community than Django, but is less restrictive. Further, due to Flask's simplicity, it does not contain an object-relational mapper, which allows for greater flexibility, but more work implementing data driven applications. While Flask is considered a "microframework", it still provides some of the fundamental features that web frameworks provide, such as minor security, routing, middleware, and scalability.

Cost

All the mentioned frameworks are free of cost.

4.3.7.3 Web Server Conclusions

Performing data processing on a web server offers the best user experience for users of LogiSteps, and significantly reduces development work for team Omicron. By completing processing on a server, LogiSteps will have minimal impact on a user's phone, and development can be focused on a single platform, with better performance. When considering the best backend framework to use for the development of LogiSteps, several factors were considered.

Express is built on NodeJS and is designed for efficient event driven programs. Express also integrates well with NoSQL databases such as MongoDB, which may be beneficial for storing complex data structures. Django and Flask, on the other hand, are built on Python, which offers a core API much more equipped for data processing. While Django and Flask share a common underlying programming language, they are quite different. Django offers a "batteries included" approach, providing several tools for quick and efficient prototyping. Additionally, Django specializes in providing an interface for integrating relational databases into an application. This reduces the effort required for performing CRUD operations and querying for data. Flask on the other hand, offers a bare-bones, minimalism approach. This offers inherit benefits, such as increased freedom and size, but can make it more difficult to rapidly prototype and develop. This provides increased flexibility for the type of database and management systems being used for storing and querying data, but increases the effort required for integrating databases into the application. This project will mostly be a data driven application, serving data visualization views to clients. Because of this, it makes more sense for the back-end technology to utilize the Django framework, which assists in easy data mapping and database querying using the relational object mapper. Using Django, relational databases can be easily integrated into the application, Python can be used for efficient data manipulation, and other Django features such as user authentication, security, and more can be used to support development.

4.3.8 Data Presentation

4.3.8.1 Mobile App

One option for displaying user data is implementing a full UI on a user's mobile phone. Displaying all the data on the phone offers a few advantages.

- The application has easier access to personal user data
- The user does not need multiple views to see all their data; it is all in one place

Displaying all the data also offers multiple disadvantages.

- The application size goes up
- The application slows down
- The user is not required to update. Therefore, if the format of the data changes, the user view could break due to the server having a different format
- User interfaces must be designed for dozens of different types of mobile phones.

Rather than building a full feature user interface on users' mobile devices, another option is to display a very basic UI, displaying only a few basic statistics and connection status. This will:

- Help the user save recourses on their phone
- Greatly reduce the development needed.

The only disadvantage to this presentation style would be that the user would have to go online to view their complete data.

4.3.8.2 Web Application

An important aspect to LogiSteps is the ability to portray user data generated from their shoes in a clean, easy to use manner. While there may be several approaches for displaying the data (mobile application, native application, etc.), one of the most dynamic and flexible options is using an HTML based web user interface - doing so presents numerous advantages over other display options. While the advantages are numerous and vary from use case to user case, a few important advantages to consider are the following:

- **Cross Platform** – by serving user interfaces from a web server, it allows the application to be designed without worrying about compatibility across Windows, MacOS, and Linux. UX code will render and run in the browser.
- **Installation-less** – Since the user interface is being served from a web server, there is no need to design software for installing and booting the user interface. End users only need to know the right URL for accessing the application.
- **Updates** – Updates can be easily deployed to a centralized location without the need to push updates software to numerous distributed users.
- **Administrative Overhead** – by deploying the end user application to a web server, this project will be able to mitigate much of the administrative overhead placed by companies such as Apple.
- **Fast Prototyping/Development** – The abundant availability of front-end frameworks help get the project into deployable state without needing to design UI components from the ground up.

Of course, there are more advantages than the ones considered, however they help illustrate the case for using a web application for presenting data to users.

While there are numerous advantages to this approach, one must also consider valid disadvantages, and factor them into the decision-making process. The use of a web server for processing and displaying user data lacks behind mobile and native apps when considering the following issues:

- **Processing Overhead** – Using front end and back end frameworks for building an interactive application for a user comes with a lot of extra files. Web applications often require an extensive number of files for rendering and processing data, much of which may not be used if the user does not navigate to that respective part of the application. Additionally, since the application is abstracted from the system, it often runs slower than native desktop and mobile apps. The speed of the application also becomes dependent on the web browser being used, an aspect of design that the developer cannot control.
- **Programming Language** – Since the front-end portion of the application is being run in a browser, development is limited to JavaScript, HTML, and CSS. Additionally, the back-end server will also be limited to a language supported by web frameworks and web servers – typically NodeJS, Python, Ruby, etc.
- **Resources** – The application will not have access to many of the client's system resources. This will somewhat limit the design flexibility for displaying information to the user.

- **Security** – Resources will be more viable to attack and probe from outside sources. Additionally, data being transmitted over the public internet may be vulnerable to spies. Security becomes a much larger problem when using the public internet to serve and present resources to users.

One last thing to note that by presenting information to users using a web application, the application becomes decoupled, with a distinct separation of client side and server-side software. This can be an advantage if designed correctly, but the server and client may not be able to easily access resources that could be if the application were closely coupled (like a native mobile/desktop app).

Choosing a front-end web framework for a web application can be a difficult task due to the vast multitude of available frameworks – each claiming to be better than the next. After doing initial technology research, team Omicron discovered that three of the most popular front-end web frameworks available to developers are Angular, ReactJS, and VueJS. Other notable front end frameworks worth mentioning are Ember, Elm, knockout, and more.

When considering front-end frameworks, their attributes were closely examined to help determine the framework that would best suit LogiSteps. Each of the comparable attributes are examined in finer detail below.

Documentation

Angular, React, and Vue are three of the most popular front-end frameworks in the world. As a result, each has an extensive amount of rich, detailed documentation and tutorials. All three frameworks have the necessary documentation to begin development and reference if there is any trouble in development. Of all three, it is difficult to choose which framework has the best documentation, but it appears that React may lag behind both Angular and Vue in documentation, with simple explanations of the software. This claim appears to be backed up by several members of the software development community, with many claiming that React lacks in some aspects of its documentation. Having little experience in web development, documentation is an essential attribute for Team Omicron to consider when choosing a proper front-end framework.

Architecture

Architecture of a framework has a large impact on how flexible a developer's application can be. Angular is highly opinionated on how a developer's application should be structured, with React and Vue providing a much more flexible approach, allowing developers to structure their application in a manner that best fits it's needs. This has much to do with the fact that React and Angular are only View layer libraries, while Angular uses a model, view, view model structure with structured components and services.

Having a strict/strongly opinionated framework can make design work simpler by limiting the developer's decisions, but it limits the flexibility of the application, while also adding overhead that is often unnecessary. This makes Angular a much heavier framework than Vue and React,

introducing a lot of complexity that is likely unneeded. On the other hand, having a structure to develop from can be beneficial in implementing an application. Due to this, both Vue and React have templates and third-party plugins for modeling client-side data and structuring an application. This makes it possible to integrate model layers into a front-end application without requiring an application to adhere by a strict set of rules (as it is in Angular). The benefits of structure can be exploited without dependency on it.

Data Binding

Data binding is a property that maps underlying data to the view layer of the application. This attribute relates to the architecture of the framework being used. In frameworks that are primarily the view layer of an application, data modeling is typically one way, from the model to the user interface. Vue and React follow this pattern; data has a downward flow, and in React child elements don't have any effect on parent data. This can be advantageous because it keeps the application logic simple – data and the state of the data lives in only one place. The disadvantage of this attribute is that it makes it more difficult to directly change the state of data from user interaction. Other tools and features of the framework must be exploited to update the underlying data.

Angular differs from Vue and React in that it offers two-way data binding. Two-way data binding is a powerful feature that allows actions by the user to directly update and change the state of underlying data models. This becomes possible in Angular because of its model and view model layers underneath the view layer of the framework. Although this feature can be powerful in applications, it can lead to bugs that are difficult to test, find, and fix. It also adds extra logic to the application. For applications where data is primarily viewed, with no great need for user manipulation, this feature of the framework is less useful.

Since LogiSteps will be using the user interface to primarily display user data, two-way data binding is likely an unnecessary feature. One-way data binding will most likely provide a cleaner conduit for conveying data to the end user.

Performance and Load Times

A noticeable metric for an end user of a web application is the performance of the web page and how quick the initial load time is. John Hannon did a study on the performance and load times for various front-end web frameworks, and he found that for keyed implementations of the frameworks, Vue beat out Angular and React significantly in both typical operations as well as initial load times. Angular and React appeared to perform similarly for typical operations, with React slightly edging out Angular in most metrics. When comparing initial load times, React was significantly faster than Angular, but still significantly slower than Vue.

The web application for LogiSteps will not be an enormous web application performing complex algorithms. Due to this, there is an expectation that the web page should be fast and respond to users quickly. The page should also load quickly when a user navigates to the page to view their statistics. For this metric, Vue beats both Angular and React.

Syntax

Syntax is an important aspect of a framework, as it directly correlates to the learning curve associated with the framework; the more unfamiliar a syntax is, the longer it will take to begin development of the application. Angular and React present syntaxes unlike others found in other languages and frameworks. React represents all UI elements using JSX and rendering functions. JSX is an XML like syntax that is used within JavaScript. While JSX can be powerful, allowing JavaScript to be interwoven with UI, it means that using React would require learning a new language for expressing UI elements, presenting a steep learning curve to all members of Team Omicron. Vue, on the other hand, supports JSX, but allows UI elements to be expressed with standard HTML, CSS, and JavaScript.

Angular, like React, provides a rich syntax that is complex in nature. Additionally, Angular makes use of Typescript rather than JavaScript. This means developers using Angular may need to be learn the Angular specific syntax for element expression, as well as Typescript for data modeling. This presents a large learning curve to developers on Team Omicron. Vue keeps syntax simple, using HTML, CSS, and JavaScript for UI element expression, while also picking a select few expressions used in AngularJS for more powerful UI expression.

Vue provides a simple syntax for its core functionality, while expanding upon the strengths of other frameworks to allow for more powerful element expression and HTML based templates.

Community Support

An important aspect of a front-end framework is the amount of third party and community support that the framework has. This metric was checked by team Omicron by comparing GitHub statistics. Angular, React, and Vue have the following community activity.

Metric	Angular	React	Vue
GitHub Stars	40,903	111,824	114,636
GitHub Contributors	731	1,241	193
Stack Overflow Tags	132,505	103,665	23,029

Chart 1 - Community Support Statistics for Angular, React, and Vue.

From the metrics collected in table 1, Angular and React appear to have massive following in the developer community. Angular has a significantly lower GitHub star count than the others, but this is likely a consequence of Angular migrating versions. Both Angular and React also have a large amount of questions on stack overflow, meaning that many potential questions likely have answers available on stack overflow. Vue has a significantly lower number of GitHub contributors and stack overflow questions, possibly making it more difficult to find answers to questions during the development process.

Other

Angular, React, and Vue all provide several other features as well which have been considered when performing technology research. All three frameworks provide view templating routing, and expressive element expression. When considering framework transfer size, Vue wins, beating both React and Angular based on a study performed by Jacek Schae.

Costs

All three options (Angular, React, and Vue) are free to use and develop with.

4.3.8.3 Data Presentation Conclusions

Presenting user data using a web-based interface offers the most benefit to LogiSteps. Not only does it simplify the development effort required, but it minimizes the amount of resources that the application would require from a user's phone. A web-based app makes it easier to build cross-platform interfaces, mitigate much of administrative tasks to publish an app to an app store, develop for multiple screen sizes, and more.

The best front-end framework for development of the web-based user interface was decided using the following criteria. All 3 front end frameworks provide similar speed performance, however, Angular typically has a longer initial load time. In terms of learning curve, both Angular and React have steep learning curves, with Angular having complex syntax, and React using JSX for UI development. Vue offers the smallest learning curve by using native JavaScript, CSS, and HTML to implement rich, expressive user interfaces. Vue also has advantage in terms of size - due to Angular's heavy use of libraries, complex syntax, and other features, it becomes bloated. Both Vue and React are much smaller in size, with libraries being included as they are required. Angular provides more structure, but this structure can be restrictive at times; React and Vue allow for much greater flexibility. The flexibility of Vue and React can sometimes make it more difficult to start projects, but Vue offers a powerful CLI that helps setup projects with various configuration settings available. Vue is currently less popular than Angular and React - leading to a smaller supporting community and a smaller collection of libraries, but Vue is quickly growing in community size and popularity. In the end, Vue likely offers the best option for the project due to its small size, and smaller learning curve. Team Omicron's front-end UI is meant to be simple, providing a portal for viewing statistics. The added size of Angular, and complexity of React and Angular make them a less attractive option for the project.

4.3.9 Web Hosting Overview

Some of the options for processing, storing, and presenting data to users would rely on a publicly hosted website. A critical piece for creating the web applications is hosting the application publicly so that any user across the world can access it. While it is technically feasible to host a dynamic website on a development PC or server, it would be vulnerable to a multitude of security concerns and would not be able to scale well. To solve this issue, there are a multitude of web hosting services that allow a developer to host their database, web server, and website publicly. Some of the top competitors in this

industry are Google Cloud Platform, Amazon Web Services, and Heroku. All 3 of these options offer free versions for active development.

Google Cloud Platform and Amazon Web Services are similar in that they offer infrastructure for hosting web applications as a service. This allows a developer to choose the correct plan based on the needs of the application and offers limited free trials for initial development. Amazon web services have been around the longest of all 3 options (Heroku actually uses Amazon Web Services), but often it can be difficult to predict the price for hosting a web application on Amazon. Additionally, based on experimentation, Amazon Web Services appear to be the most difficult of the three options for navigating and finding all features. AWS claims to have free tier with the following features:

- Amazon Cognito – Mobile user identification and synchronization
- Amazon DynamoDB – 25 GB NoSQL database
- Amazon EC2 – 750 hours of cloud compute capacity per month
- AmazonMQ – 750 hours of broker service for Apache ActiveMQ per month
- AmazonRDS – 750 hours of Managed relation database service for MySQL, PostgreSQL, etc.
- Elastic Load Balancing – 750 hours of traffic distribution

The Google Cloud Platform also has a free tier that gives a 12-month $300 credit to developers. Google Cloud offers many of the same features as AWS, but with a more intuitive interface for developers. One of Google Cloud's most valuable features is its Compute Engine, which provides scalable, high performance virtual machines. Using this, custom software can be installed and used to host a web application. Google Cloud also offers services for load balancing applications, removing a necessary piece of deployment. Additionally, Google Cloud offers several predefined production environments that a web application can be deployed to, removing the necessary administrative tasks involved with a custom web stack. While Google offers a $300 credit to its customers, the following services are always free

- Google App Engine – platform for building scalable web applications
- Highly-scalable NoSQL databases
- Google Compute Engine – scalable, high performance virtual machines (web hosting too)
    - 30 GB of HHD
    - 1 f1-micro instance per month
- Google Cloud Pub/Sub – real-time and reliable messaging and streaming data
- Cluster Management
- Google Stackdriver – monitoring, logging, and diagnostics for applications on Cloud Platform
- GCP Marketplace – pre-configured free production grade solutions

4.3.9.1 Pricing

Google cloud will likely be free for the duration of Senior Design due to the requirements of the web application. If LogiSteps were scaled up, pricing would be heavily dependent on the number of users. A server with 4 cores and 15GB of memory would cost $97.09 per month, and a server with 1 core and 3.75GB memory would cost $24.27 per month.

Heroku is a cloud application platform that supports building, deploying, and managing apps. Heroku allows apps to be run inside of what they call dynos, which are fully managed runtime environments for applications. Heroku is designed to be as easy as possible for developers to easily, quickly, and frequently deploy applications; Heroku then provides a dashboard for managing all applications. Heroku helps handle scale, similar to Amazon Web Services and Google Could Platform. Heroku offers several different database solutions such as Heroku PostgreSQL but requires extensions and add-ons to work with other database systems, making it slightly more difficult to deploy data driven applications. Heroku is likely an excellent source for active development but appears to lack the extensive functionality that can be achieved using a Google Cloud Compute virtual machine. Heroku also sleeps after inactivity for free accounts, leading to large latency for requests made when the dyno is sleeping.

- Free - $0
    - Core platform features
    - Sleeps after 30 minutes of inactivity
    - 512 MB RAM | 1 web/1 worker
- Hobby Level - $7/dyno/month (pay for time used to the second)
    - Core platform features
    - Never sleeps
    - Free SSL & Automated Certificate Management
    - 512 MB RAM | 10 Process Types
- Standard - $25 - $500/dyno/month (pay for time used to the second)
    - All hobby features+
    - Horizontal scalability
    - Threshold alerts
    - Pre-boot
    - 512 MB RAM or 1 GB RAM
- Performance - $25 - $500/dyno/month (pay for time used to the second)
    - All standard features+
    - Dedicated
    - 2.5GB RAM or 14GB RAM
    - Infinite process types

4.3.9.2 Web Hosting Conclusions

For the development of LogiSteps, Google Cloud Platform will likely be the best option. GCP not only offers a free plan to begin development, but it also offers a robust administrative web interface for managing the deployment. Google Cloud also presents a more straightforward pricing model which coincides with a pricing calculator to help determine cost of hosting, depending on the needs of the web application. Additionally, Google's compute Engine allows a customized web stack to be developed and deployed without having to worry about compatibility. This presents a huge advantage over services such as Heroku. It is worth noting that there are several other web hosting services which appear to be extremely cheap up front (such as HostGator), but these services often attempt to lock customers into long term deals and offer limited hosting options, severely limiting the web stack.

4.3.10 Block Diagram with Available Component Technologies

Analyzing how the different approaches to each section may affect each other, a more complex block diagram has been developed to help visualize possible permutations of each technology that would satisfy design requirements. This diagram is shown in figure 12.

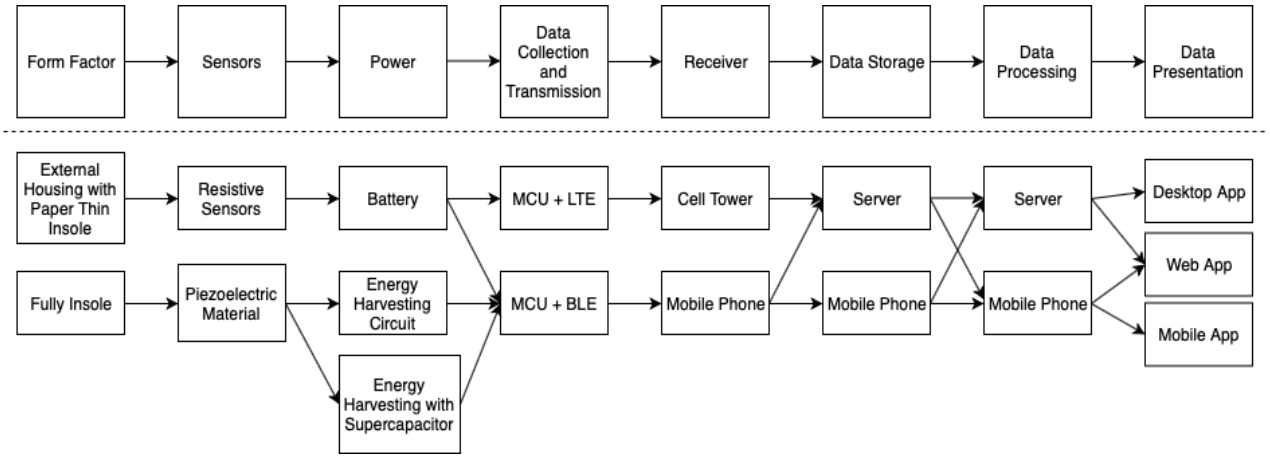

Figure 12 - General system requirements with technological components that make it possible to achieve system requirements. Several options are shown for each system requirement.

4.4 Final Design Approach

The final design has taken on a very modular approach, allowing for some freedom between design choices should changes need to be made down the road. Some of the choices affect others, such as resistive sensors requiring an energy dense power source rather than an energy harvesting solution, or LTE to server data storage technique making a mobile app no longer useful. Overall, the top choices in each category work together and are most likely to offer the best solution, and issues to be encountered won't affect other choices in a breaking manner. A full stack view of the system using the preferred approaches for each system requirement is shown in figure 13.

4.4.1 Final Block Diagram

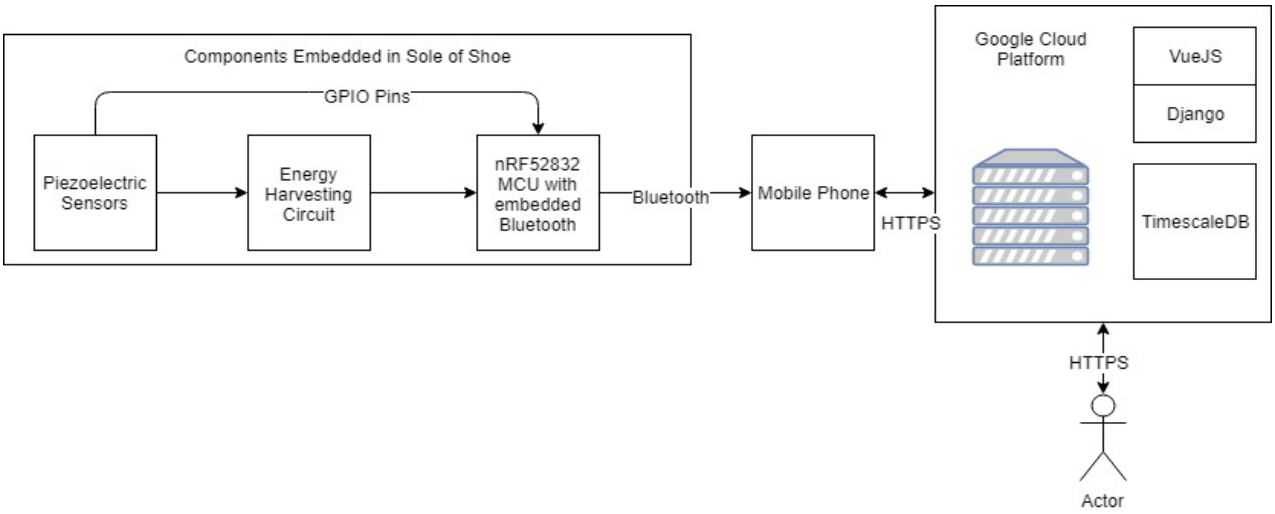

Figure 13 - Final design approach, using the technologies discussed in previous sections of this report that were concluded to be the best options for meeting system requirements.

4.4.2 System Requirement Satisfaction

The solutions that have been chosen to be pursued will entirely meet design requirements as previously specified. The piezoelectric sensors will be able to provide useful data pertaining to the users' steps, including pressure sensitive readings. Those type of sensors will also be able to power the system through an efficient energy harvesting technique, and with the thin, fully insole form factor the user experience will be as seamless as possible. The MCU and BLE capabilities will allow the data to be collected and transmitted from the shoe to a receiver in real time. The phone acting as the receiver allows the system to require no more than the insole alone, and it enables a simple solution to transmit data to a server for permanent storage and processing. The timescale database solution allows for efficient permanent storage of user data over time. Choosing Django as the server backend allows for efficient processing of data and choosing VueJS as the front end lets the processed data be professionally and simply presented to the user.

4.4.3 Modularity

Rather than highlighting 3 specific approaches that the project could have, the project design was split into a more modular view. Since the project contains a lot of different modules working together, it was decided it would be better to give information about the various options for each module.

Having a modular design for the project will help as prototyping is begun, and progress made with the project. If any problems are encountered with the design and a need to change the approach arises, it will be able to be switched to another option for that module without having to drastically change the rest of the project design. Since the project has a lot of pieces working together, it was desired to make it easy to change the individual approaches if necessary.

4.4.4 Total Cost

The development costs were estimated using the minimum and maximum costs determined by the analysis of each approach. The preferred approach costs were determined using the preferences for each module's solution, in addition to conservative prototyping plans.

Development	**Min ($)**	**Max ($)**	**Preferred Approach ($)**
Form Factor	40	70	70
Sensors	30	400	133
Energy Components (pure harvesting)	20	20	20
Energy Components (general)	10	40	40
Microcontroller/BLE	60	150	60
Server Costs	0	0	0
Solution Totals	160	680	323
Labor	55,500	111,000	111,000

Table 2 - Summary of development costs for LogiSteps.

Should this product be taken to market, production costs would need to be estimated as well. The component costs would go down significantly in each category per device due to whole sale prices, but some additional costs would arise such as licensing and hosting fees. The hosting fees are difficult to determine due to the plethora of options available, more would need to be known about the customer base and specific service requirements to get a good estimate of necessary processing power and storage needs. The Bluetooth branding fee is estimated to be $8000. The Google App store license fee is $20. The cost of mass production also needs to be considered, this would require outsourcing the production of the final product design to a manufacturer and distributor.

4.5 Conclusion

In conclusion, the final system approach discussed in the previous section satisfies all system requirements. In addition, thorough research has been performed for each component of the system, so if an approach is found to be unfeasible, the Team Omicron will be able to quickly change one piece of the system design, without needing to perform a complete system redesign. Each aspect of the final approach is defended by the research documented in the component technologies of this report. Team Omicron is optimistic that using the technology and concepts discussed in this report will result in a switch, and successful prototype and design cycle.

5. Project Design

5.1 Insole design documentation

5.1.1 Insole Overview

The insole is the component which is inserted in the user's shoe. It is a standalone system requiring no additional assembly or user serviceable parts. The insole is designed to house all hardware, including the micro-controller, energy harvesting system, energy storage devices, force sensors, and wire connections.

The insole is made up of a number of independent components, assembled using no adhesives or machined parts. The components are either 3D printed or come from a third part manufacturer. Once assembled, the insole will be sealed with a liquid rubber to solidify the design to meet the robustness criteria.

The insole will be capable of harvesting the energy from user's steps, provide power to the micro-controller, provide the micro-controller with the tools to measure pressure on different parts of the user's foot, and meet all other criteria specified in the system requirements documentation.

In order to harvest the energy of user's steps, the insole will implement two brass discs coated with a piezo ceramic material. This material, when deformed, produces an electrical potential difference between the brass and ceramic material that can be polarized, collected on a capacitor, then transformed to an ideal voltage range to be stored on another capacitor. This is done using a piezo-ceramic bender, housed in a component known as the activator, and connected to a third-party energy harvesting device, an LTC3588. The LTC3588 collects the energy from the piezo ceramic on an input capacitor, then uses a low power buck converted to efficiently convert that input voltage to the desired 3.6V stored on a larger output capacitor. This output capacitor is to be the voltage source that powers all other components within the insole.

To measure force, the insole uses two sensors, force sensitive resistors. These sensors decrease in resistance as more force is applied to them. For simplicity of design, the sensors are located in the same place as the piezoelectric buzzers, one on the users heel and the other on the forefoot. The contacts to these resistors will be exposed to the micro-controller, allowing it to determine an estimate for the amount of force the user is applying to that part of their foot.

The other criteria the insole needs to meet deals with robustness and comfort. All components that make up the insole are resistant to heat within the specified range, can withstand all forces that are capable of being applied to it under intended use cases, and is sufficiently waterproof. This is accomplished by using multiple materials to create the components and finalizing the design by sealing it with a liquid rubber.

5.1.2 Electrical Operation

The component that makes it possible to harvest energy from a user's step is the piezoelectric buzzers, manufactured by a third party. The buzzers consist of a thin brass disc coated with piezo-ceramic on both sides. The buzzers to be used are known as AB4113B-LW100-R, manufactured by PUI Audio, shown in figure 14.

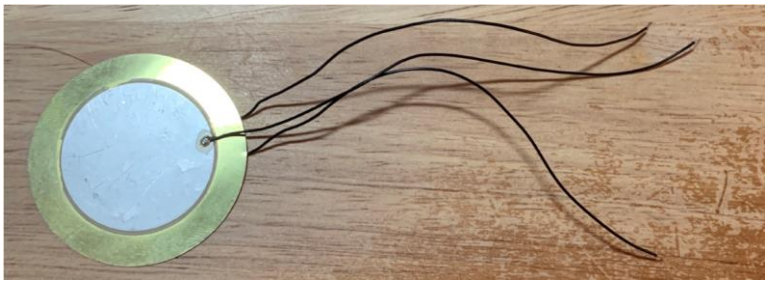

Figure 14 - The piezo buzzer.

When the buzzer is deformed, it produces an electrical potential difference between the ceramic and the brass. Under any deformation, each ceramic coating produces a potential difference with the same polarity relative to the brass. The waveform shown in figure 15 shows an example of this.

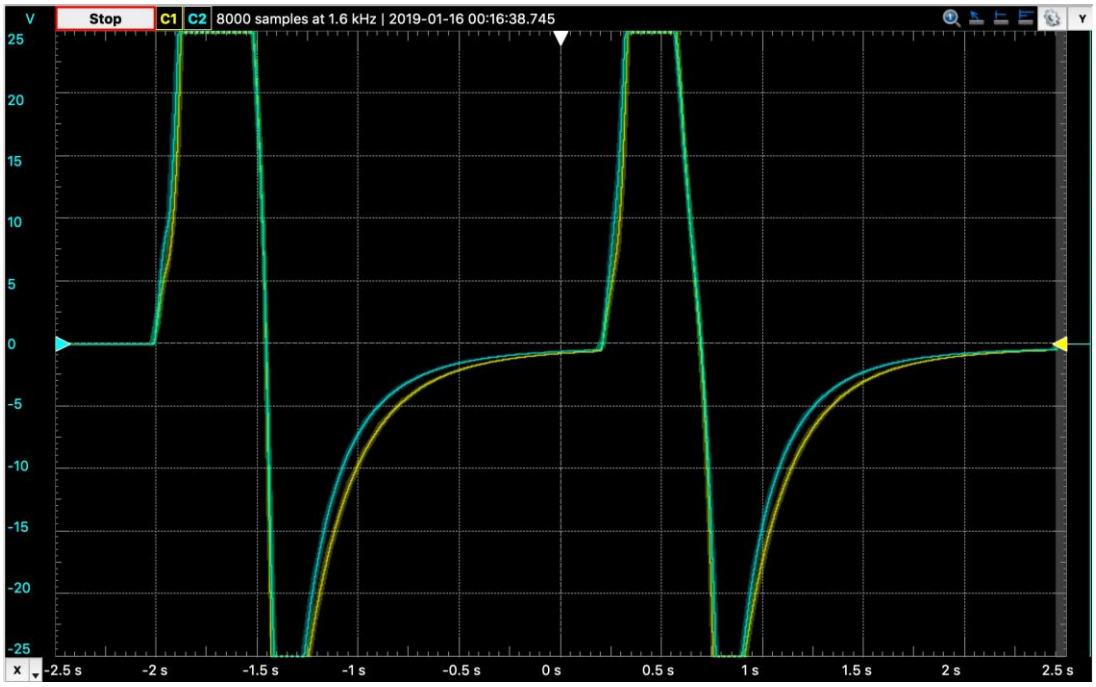

Figure 15 - Piezoelectric buzzer waveform when deformed. The voltages shown are produced by the ceramic plates relative to the brass disc, one yellow, one blue.

To use this voltage, it is polarized using a full-bridge rectifier and stored on a capacitor.

Because the ceramic plates produce the same polarity voltage relative to the brass, the lead wires of the ceramic can be connected to each other, allowing for a simple two inputs to the rectifier.

When the voltage is stored on the capacitor, it has a potential maximum voltage equivalent to the max voltage that can be produced by the buzzers. This voltage is too great to power the micro-controller however and needs to be transformed to a proper range first. The voltage the buzzers can produce is >30V while the micro-controller requires voltages in the range of 1.7-3.6V. To transform the voltage, a third-party energy harvesting chip is used, the LTC3588, manufactured by linear technology.

The LTC3588 has a full-bridge rectifier on the chip and stores the voltage on a capacitor (C_in), then uses a low power buck converter to transform the voltage and stores it on an output capacitor (C_out). This output capacitor is then used to provide the micro-controller. The circuit diagram including the buzzers are shown in figure 16.

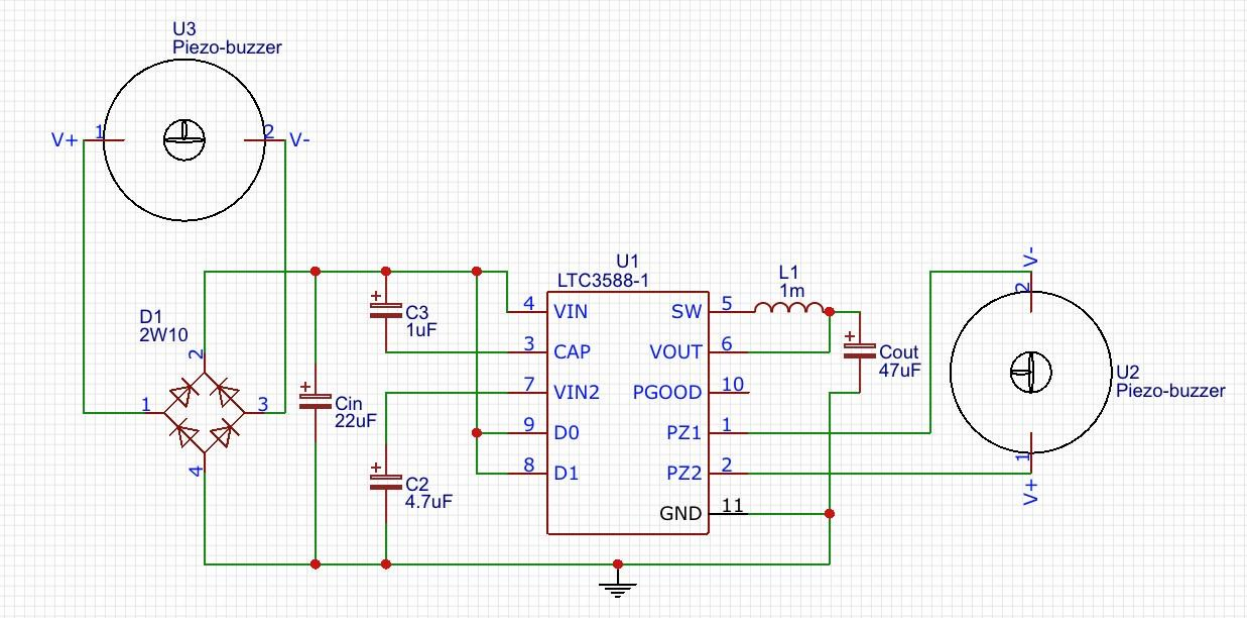

Figure 16 - Piezoelectric buzzers connected to the LTC3588 chip.

Because the LTC3588 has only one full-bridge rectifier on it, a second rectifier must be used for the second buzzer. The output of both rectifiers are connected to the same capacitor however, C_in. The capacitors C_in and C_out can be sized to meet electrical requirements and the output voltage can be any of the following values: 1.8V, 2.5V, 3.3V, 3.6V.

The energy stored on capacitors is calculated using the equation:

$$E_{cap} = \frac{1}{2} CV^2$$

To store the most energy on the capacitors, the voltage should be maximized. The microcontroller requires voltages in the range of 1.7-3.6V, so the LTC3588 will be configured to supply the output capacitor with a voltage of 3.6V. The input capacitor can be up to 20V, regulated by a shunt diode to prevent over voltage. The equation to determine the maximum energy capacity of the capacitors is given by:

$$E_{max} = E_{in,max} + E_{out,max} = 12 (C_{in} *V_{in}^2 + C_{out} *V_{out}^2)$$

The capacitors to be used are aluminum electrolytic capacitors. Sufficient capacitance for C_in have been determined to be 22uF and 47uF for C_out. This gives a maximum energy capacity of:

$$E_{max} = 1/2 (22uF)(3.6V)^2 + 1/2(47uF)*(3.6V)^2 = 4700uJ$$

This maximum energy capacity provides the micro-controller with more than enough power to continue operation uninterrupted between steps. This is necessary because there is no input power between steps and thus the micro-controller will need to exhaust this energy capacity sparingly in such cases. This satisfies the energy requirements determined in the micro-controller section, and if needed, the capacitors can be resized to power up the system faster with less energy capacity, or slower with greater energy capacity.

The test plan described at the end of this insole section will determine how many steps a user needs to take in order to power up the micro-controller, and how much data collection, manipulation, and transferring the micro-controller is capable of per step. This is done by determining the energy the system gains per step and comparing that value to the energy requirements determined in the micro-controller section.

5.1.3 Piezo Buzzer Usage

Energy is produced by the buzzer in the form of a voltage difference between the ceramic plates and the brass disc when the buzzer is deformed, and the more deformation, the greater the energy produced. Harvesting the energy from the piezo buzzer requires that the design be optimized to produce as much of a deformity as possible without damaging the buzzer.

The component that houses the buzzer consists of two parts shown in figure 17 and figure 18.

The assembled components with the piezo buzzer included is shown in figure 19.

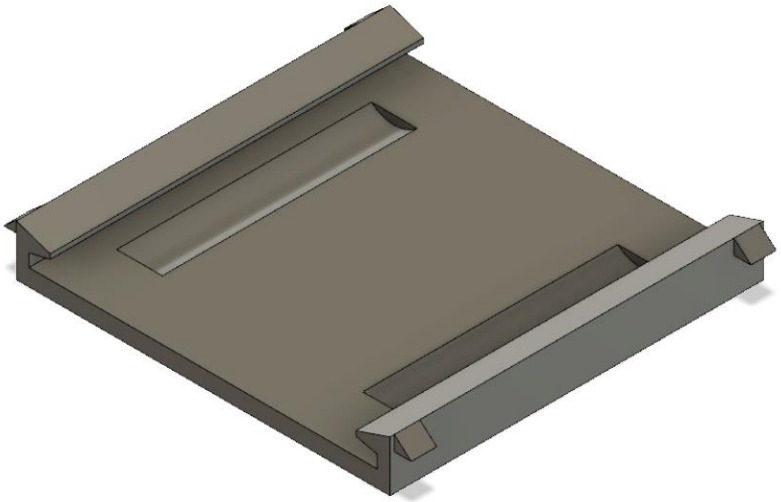

Figure 17. This component is made from PLA using a 3D printer. PLA is a hard plastic. This is to control the maximum amount of bend that can occur on the piezo buzzer, and will be referred to as the buzzer base

Figure 18. This component is made from TPU using a 3D printer as well. This is a flexible plastic. This is used to deform the buzzer in a direction opposite to the deformity produced by the users foot, returning the buzzer to its original state after being activated. This will be referred to as the buzzer decompressor.

Two buzzer decompressors are placed in the semi-circular rivets in the buzzer base. The buzzer is then inserted above the decompressors while being contained in the volume of the base. This will be referred to as the piezo component.

Assembled piezo component - initial state

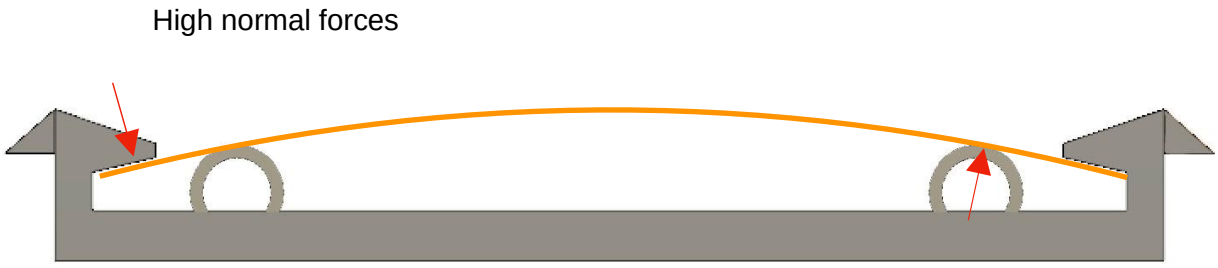

Figure 19 – The decompressor applies an upward force more toward the center of the buzzer than where it is held in place. This causes the center of the buzzer to budge upward.

Assembled piezo component - force applied

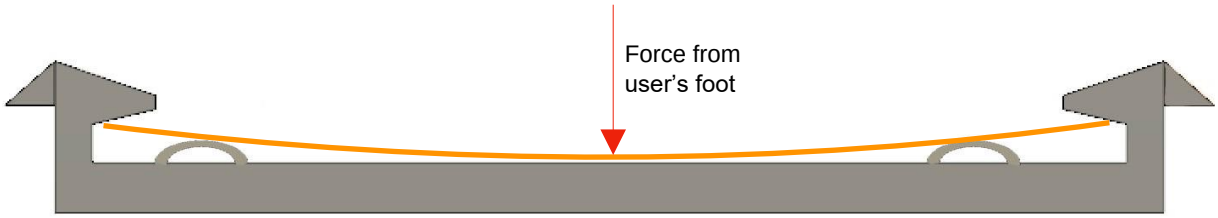

Figure 20 – The force from the user's foot compresses the decompressors and allows the buzzer to deform in the opposite direction.

This method of allowing the piezo buzzer to deform in either direction, up or down, manages to maximize the deformity with minimal space required. The normal forces applied to the edges of the buzzer and the rivets for the decompressor keep every component from moving or rotating, thus no adhesives are required for the piezo component.

The hard plastic the base is made out of prevents the buzzer from being deformed too much, protecting it from damage due to the user's foot.

For design flexibility, the radius of the decompressors can be adjusted. Reducing the radius will have the effect of decreasing the initial bend, reducing the energy it is capable of producing when stepped on. Increasing the radius could increase the initial bend and producible energy but comes at the risk of deforming the bender too much and causing damage under the no downward force condition.

5.1.4 Force Sensor

The sensor used to determine the amount of pressure the user is applying is a force resistor sensor. As the user applies more pressure, the resistance decreases. To use this requires a voltage division technique and an analog to digital voltage conversion to be done by the micro-controller. The circuit technique for this is shown in figure 21.

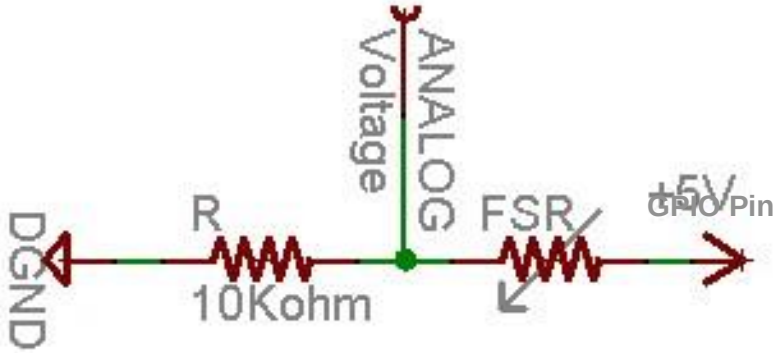

Figure 21 – The FSR Circuit.

Measuring the voltage between the force resistive sensor (FSR) and the reference resistor allows for the resistance of the FSR to be calculated with the following equation

$$RFSR = (Vgpio/VADC - 1)*Rref$$

Allowing the voltage to be applied with a GPIO pin lets the amount of power dissipated be controlled by the micro-controller. Applying the voltage only during periods of measurement will minimize the energy required by this circuit. Additionally, the reference resistor can be sized to modify the power dissipated during measurement. Increasing the size of the resistance will decrease the power dissipated, but this causes the measured voltage to be less sensitive to the resistance of the FSR, potentially increasing the error of the measurement.

5.1.5 Physical Assembly

The entire assembly will consist of the base insole, two piezo components, two force sensors, the micro-controller, and energy harvesting/storage hardware previously discussed. Once assembled, all cracks and exposed parts are to be sealed with silicone rubber in order to solidify the design and make it water/dust proof. The assembled insole will be capable of providing the micro-controller with enough power to function and allow it to collect pressure measurements from multiple points. The assembly process will be described here.

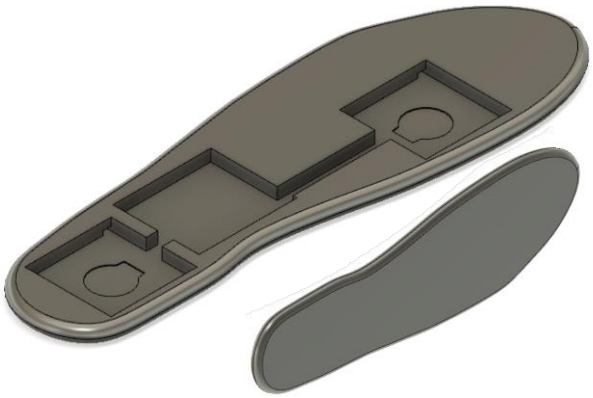

Figure 22. The insole, bottom view left. This component is made of TPU, a flexible plastic. All other components of the insole will be housed within this.

Figure 23. A force resistive sensor. It is thin and flexible. Capable of accurately measuring any pressure above 10g.

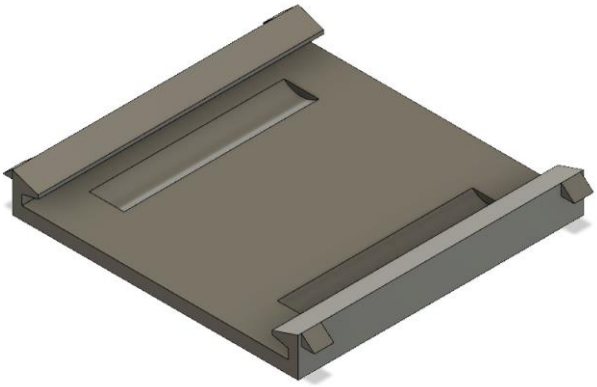

Figure 24. Previously seen piezo base, further discussed. The sharp corners protruding from the sides of this base are used to anchor into the insole, making it difficult to remove.

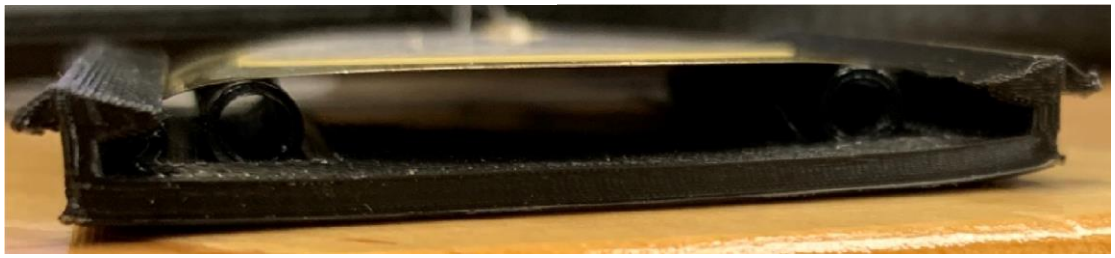

Figure 25 - The physical construction of the design.

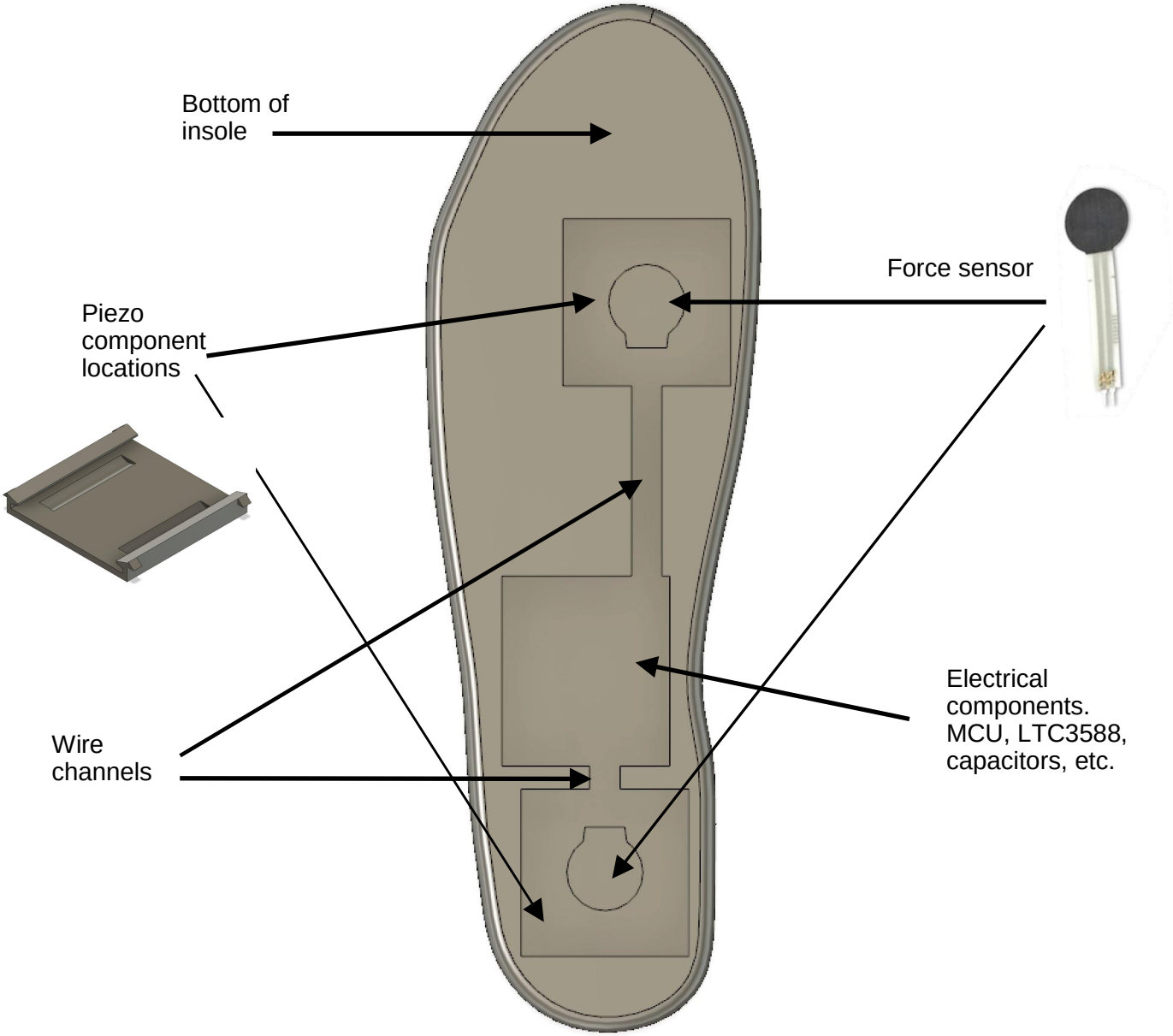

Figure 26 - Full design diagram for the insole design.

None of the components needs to be assembled with any adhesives or tools. The wires and electrical components do need to be soldered together however. The force sensors lay underneath the piezo components in this picture. The square section in the middle that houses the electrical components will be filled in with silicone rubber, preventing water for gaining access to these components.

5.2 Embedded application documentation

5.2.1 Overview

This document explains the basic design of the nRF52 app, and the various 'modules' that compartmentalize the work done to achieve the apps goal of staying low power, reading step data, and transmitting it to a mobile device over Bluetooth Low Energy.

The sequence diagram for the nRF52 embedded app is shown on the right. It shows the high-level logic flow of the application. It begins with Main querying the Energy module to ensure there is enough energy stored to begin. Once there is enough power Main initializes the modules, and the BLE module begins advertising. Once a device has connected Main begins the Trigger module to wait for an event to be registered. From there three things can happen. In normal function an event occurs, the Energy module is queried to ascertain the current power levels of the system. Based on the power levels the saadc samples at a specified frequency until the event ends. Then the BLE transmits the collected data, and the system goes back into a waiting state. The second option is that the connected device disconnects, in which case the system reverts back to the beginning of operation. The third option is that the system is running out of power, in which case main attempts to gracefully shut everything down.

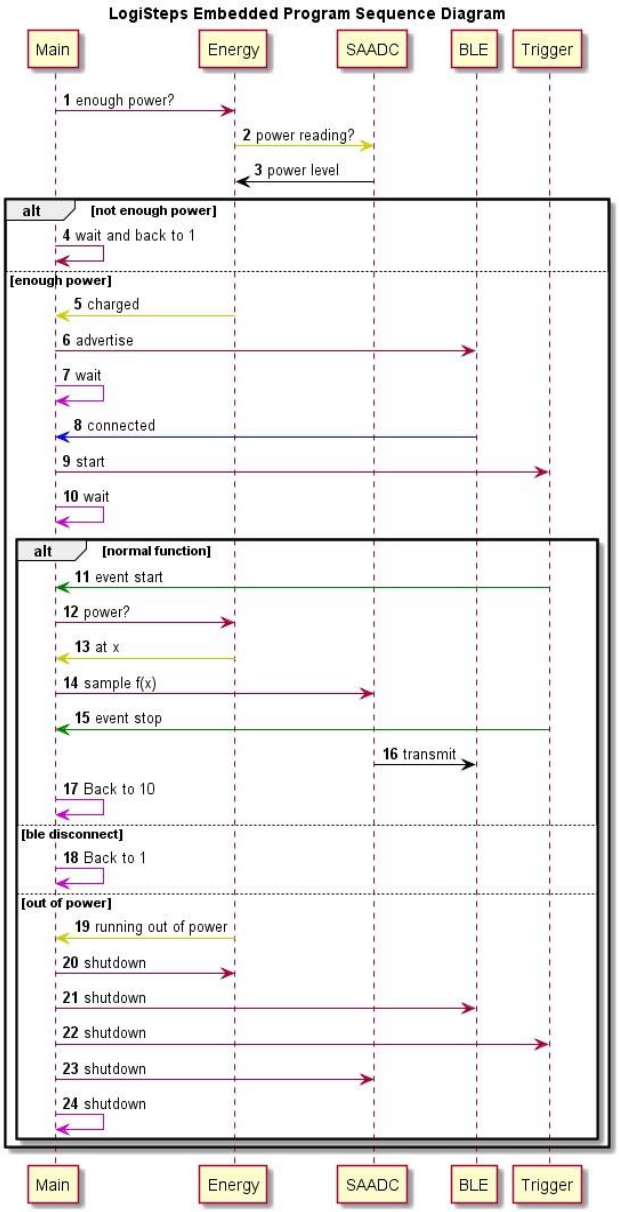

Figure 27 - Sequence Diagram of the nRF52 app.

5.2.2 Modules

Main

The purpose of Main in our nRF52 app is to handle the initialization of all other modules and ensure proper logic flow of the program. Main also begins low power management and puts the device into sleep mode when it is not working to save power.

Energy

The purpose of the Energy module is to monitor the system's energy supply. With that in mind it has two jobs. It answers queries about the current supply when asked by main and alerts main when power drops too low.

SAADC

The saadc module handles all the systems saadc operations, which entail the measurement of all the systems sensors, and the system's specific energy level. It provides the data so that it can be sent to BLE module to allow it to transmit the data to the device the system is connected to.

BLE

The BLE module handles all BLE operations required for the nRF52 app. It advertises the system, so the user can connect, it receives the necessary data from the user's phone to continue operation, and it transmits the sensor data to the user's device for use. Specifically, the BLE handles one Bluetooth service which holds two Bluetooth characteristics. For a device to work with the app, on connection it must transmit the current time to the time characteristic. Then whenever there is data to send the data characteristic notifies the device with the sensor data, and the time characteristic notifies the device with the systems time pairing for the data.

Trigger

The trigger module simply handles the event that the system uses to begin gathering data from the sensors. After it is initialized by main it will wait for a signal to register an interrupt which will alert main to begin reading sensor data.

5.2.3 Interfaces

Sensors

For the nRF52 app to interface with the sensor hardware, the input power provided by the energy harvesting circuit is expected to be in a voltage range between 1.8V and 3.6V as the microcontroller specifications require. The sensor inputs, and other inputs from the energy harvesting circuit, are expected to be in a voltage range between zero and the input power as the saadc specification requires.

With those voltage ranges decided the nRF52 app will work with the sensors by watching a p-good signal from the energy harvesting circuit at all times to ensure the power supply is sufficient. It will also read the voltage from one of the energy harvesting capacitors to determine the actual energy levels for use in the application. The last way it will interface is with the sensors, which depending on the energy levels, it will sample using the saadc read the sensor levels.

Mobile Device

For the nRF52 app to interface with a user's mobile device there are a few things of importance. First is that the nRF52 uses one custom BLE service. This service has two characteristics, one of which is a time characteristic and one is a sensor data characteristic.

When the device connects it must send the current time over the time characteristic so the app knows the current time, so it can pair it with sensor data. From that point on whenever there is sensor data to transmit, the data characteristic will be updated with a 16-bit signed integer detailing the sensor data. At the same time the time characteristic will update with the time paired with that sensor data.

5.2.4 Bill of materials

There were many different pieces of hardware equipment that were required to program the microcontroller for our project. Luckily, they were all easily accessible and only two different vendors were needed to buy what we needed.

The two main components needed for our project were the nRF52832 SparkFun breakout board and the nRF52832 hardware development kit. The hardware development kit is available through various electronics vendors, but we purchased two of them from DigiKey for $39 each with $7 shipping for a total of around $90 with tax included. The SparkFun breakout board was purchased through the SparkFun website and we bought 3 of them for $20 each plus $7 shipping. In addition to the breakout board itself, an FTDI basic module was needed to allow for USB connection to program and power the breakout board. We purchased three of these for $15 each, making the total for the breakout boards around $112 and an overall total of around $202 for the microcontroller hardware.

In addition to the two development boards, various other electrical components were needed. A micro USB cord was needed to power the hardware development kit, and an additional micro USB cord (or a mini USB cord depending on which FTDI module was used) was needed to power and program the breakout board. The breakout board also required pins to be soldered onto the chip to allow for connections to be made with the GPIO pins. The breakout board would then need to be placed on a breadboard to allow for connections to be made. Finally, various wires are needed to make connections, and specifically, M/F wires are needed to connect the breakout board to the hardware development kit.

Luckily, our project team already had all of these electrical components from earlier classes or from home and they didn't need to be purchased. However, if a person didn't have any of these components, they could all be purchased on the SparkFun website along with the breakout board and FTDI basic module. To buy all of the various components needed to program the microcontroller for this project, a person would have to pay, at a maximum, around $120.

5.3 Mobile Application

5.3.1 Overview

This section describes a high-level design approach for the LogiSteps mobile application. The mobile application is required to act as a data bridge between the web server and the insole devices. Exponential averaged sensor data is streamed from each insole, and the mobile application applies an algorithm to determine if a step has occurred. When a step is detected, the mobile application then

sends an instance of a step to the mobile REST API running on the web server, which permanently stores the data. The mobile application is designed to be narrowly focused on step detection and providing a basic user interface for users to view basic summary statistics regarding their daily movement and monitor/create/repair Bluetooth connections.

5.3.2 Architecture

The app uses a model, view model, model infrastructure (MVVM), with a repository pattern for data persistence and data retrieval. Figure 28 below shows a block level abstraction of the LogiSteps MVVM structure.

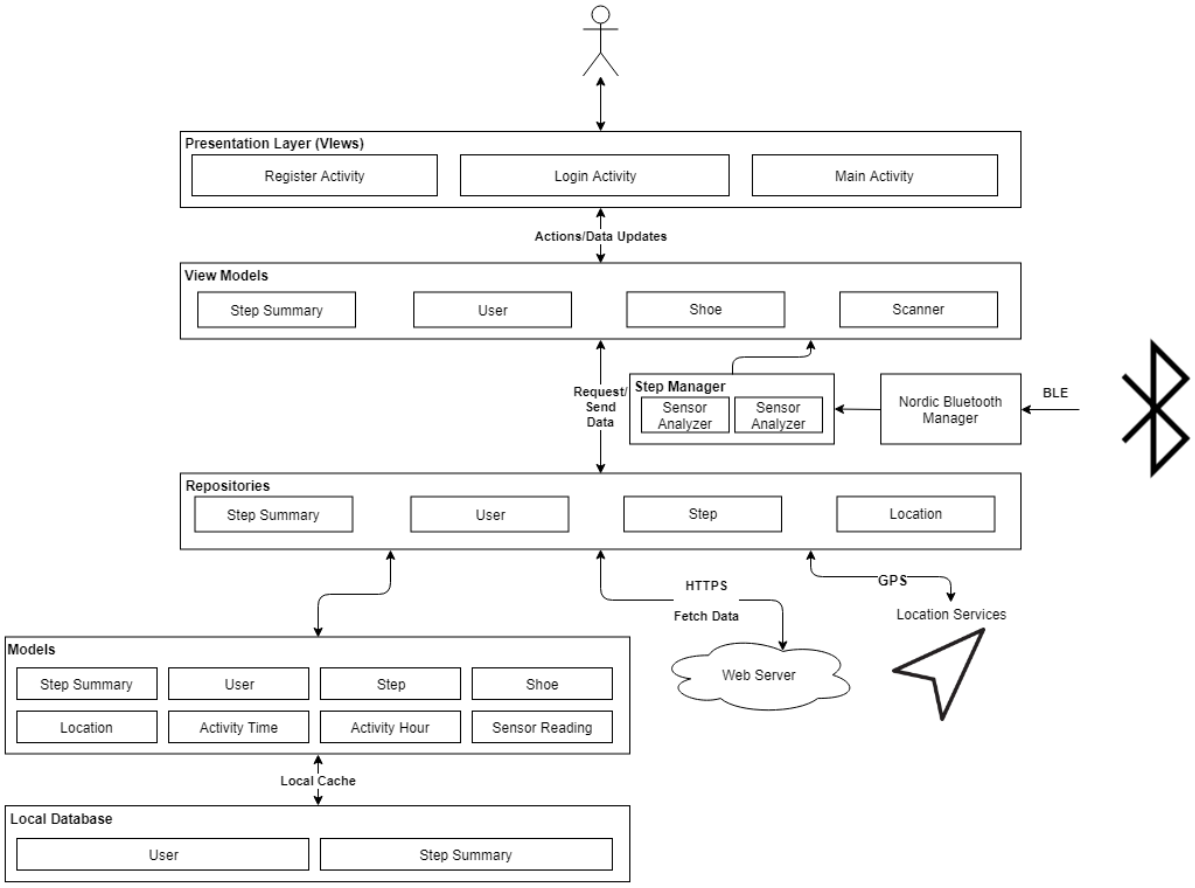

Figure 28 - Android App's MVVM block diagram.

With this structure, the views simply present data, and make UI changes based on observed objects in their view models and react to user input. All actions that a user create are passed to their respective view models, and the view models perform all logic needed to update data models or query for additional data. The view models communicate with a single model class and use the models to save and retrieve data. In this construct, each layer has a distinct and narrow responsibility, helping reduce error, increasing organization, increasing scalability of the app, and increasing the feasibility of testing.

Data models are only responsible for saving/retrieving data and representing the object and should not be responsible for the implementations in which data is retrieved/saved and should be agnostic to the

methods and mediums in which data is stored/retrieved. To satisfy this requirement, the repository architecture that Android recommends is used.

Using this pattern, when a view model requests updated data or wishes to save data, it uses a repository interface, which decides whether to retrieve the object from a local database, local cache, or a webservice. This pattern was implemented to improve performance, reduce data usage on metered networks, and decouple data services from the view model and model layers, which increases the testability of the application.

5.3.3 Data Format from Microcontroller

When the data is received in the Android application, it is received as a Data object[1], which has no primitive types, or metadata related to the format of the received data. It is up to the designer to implement a way to decode the data into appropriate Android primitive data types.

To make data parsing as simple as possible in the Android application, the LogiSteps development team decided that data should be packed together in the embedded source code prior to being transmitted over Bluetooth. What this means is that ADC sensor readings are placed into a single 8-bit array of length 15. When the embedded application senses that a user is placing pressure on the sensor, it increases its sampling rate, and fills the buffer with 15 individual ADC sensor readings. When little (or no) pressure is being placed on the piezo-electric sensors, only 5 ADC readings are done, and the rest of the buffer is filled with copies of the 5 individual sensor readings. This always ensures that exactly 15 8-bit data pieces are in the buffer. This is done for each sensor, and these readings are sent on a fixed interval using separate Bluetooth data characteristics. The format of the data is shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sr0	sr0	sr0	sr1	sr1	sr1	sr2	sr2	sr2	sr3	sr3	sr3	sr4	sr4	sr4

Table 3 - Data format of raw sensor readings when no pressure is detected.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sr0	sr1	sr2	sr3	sr4	sr5	sr6	sr7	sr8	sr9	sr10	sr11	sr12	sr13	sr14

Table 4 - Data format of raw sensor readings when pressure is detected.

Using this format, regardless of the user's movement or the data characteristic, the Android application does not need to account for different length arrays of sensor readings.

5.3.4 StepManager

A step manager object is needed to help facilitate the detection of legitimate user steps. Data is streamed in chunks from the embedded firmware running the microcontrollers, and an algorithm must be applied to determine consecutive samples that constitute an actual user step.

[1] https://github.com/NordicSemiconductor/Android-BLE-Library/blob/master/ble/src/main/java/no/nordicsemi/android/ble/data/Data.java

Determining a Valid Step

The step manager object should register two callback functions that are able to receive incoming Bluetooth data from insole devices. These callbacks will be called from the Nordic Bluetooth Manager. In addition, the step manager should be capable of instantiating a step instance and passing the step instance to the correct shoe view model. Additionally, the step manager should hold a sensor analyzer instance for each sensor. The sensor analyzers should continuously watch for consecutive samples of data above a certain threshold, and then alert the step manager when a pattern has been detected. Upon pattern detection, the step manager should check if the sensor pattern indicated that a step has been taken. There are several different sensor pattern detection scenarios that constitute a step:

- When a pattern has been detected for both the top and bottom sensors of an insole within a short amount of time, a single step has occurred.
- When two consecutive sensor patterns correspond to the same sensor, a single step has occurred.
- When there is a long period of time between a pattern in the top and bottom sensor, two steps have occurred.

In order to do this, a cache of the most recent sensor patterns must be maintained.

The following diagram illustrates the algorithm for detecting valid steps based on incoming streams of pressure data. Note, the algorithms are dependent on thresholds that can be adjusted to improve accuracy.

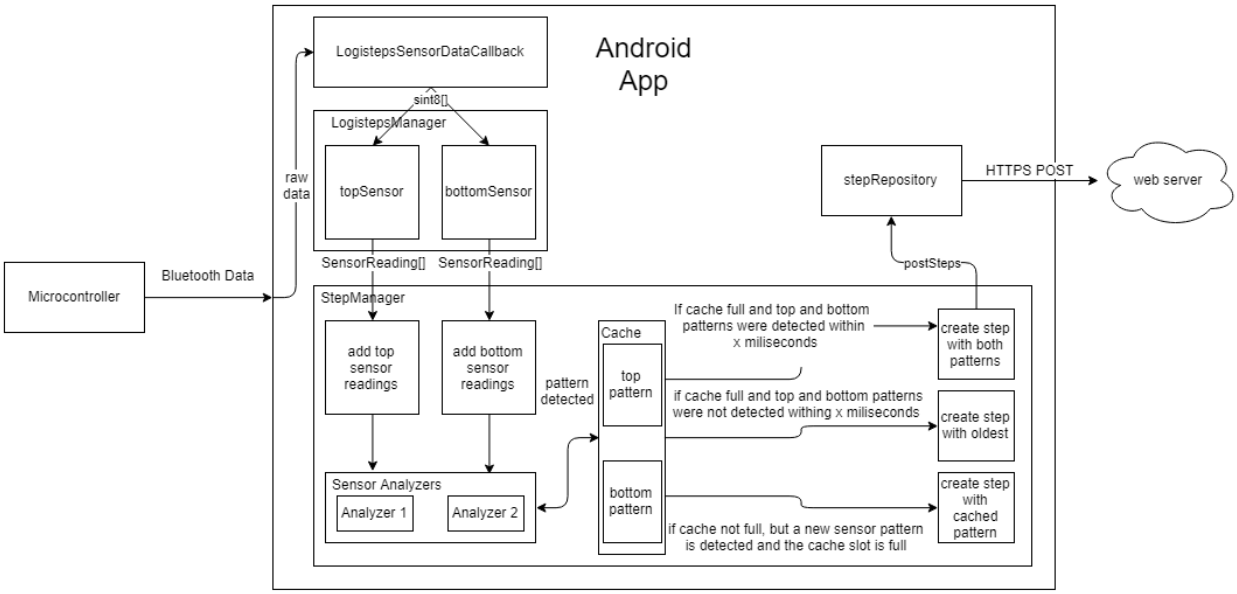

Figure 29 - Logical flow for the LogiSteps step detection algorithm.

5.4 Web Application

5.4.1 High-Level Architecture

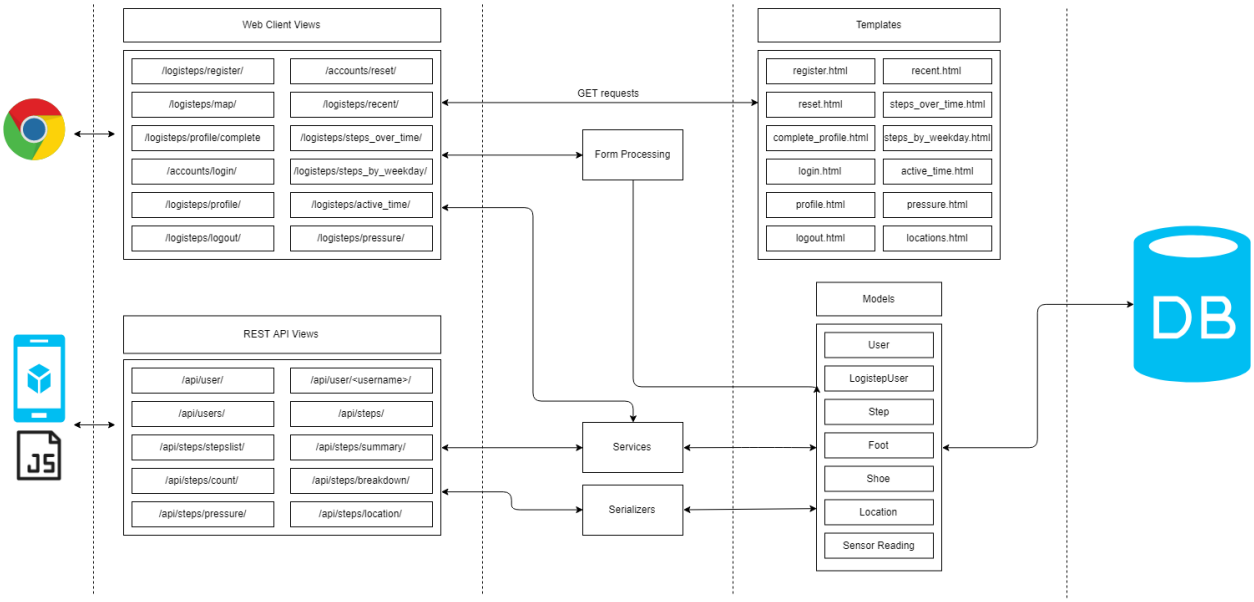

Figure 30 - High-Level diagram of web application architecture.

5.4.2 Sequence Diagrams

Get a Statistics Object

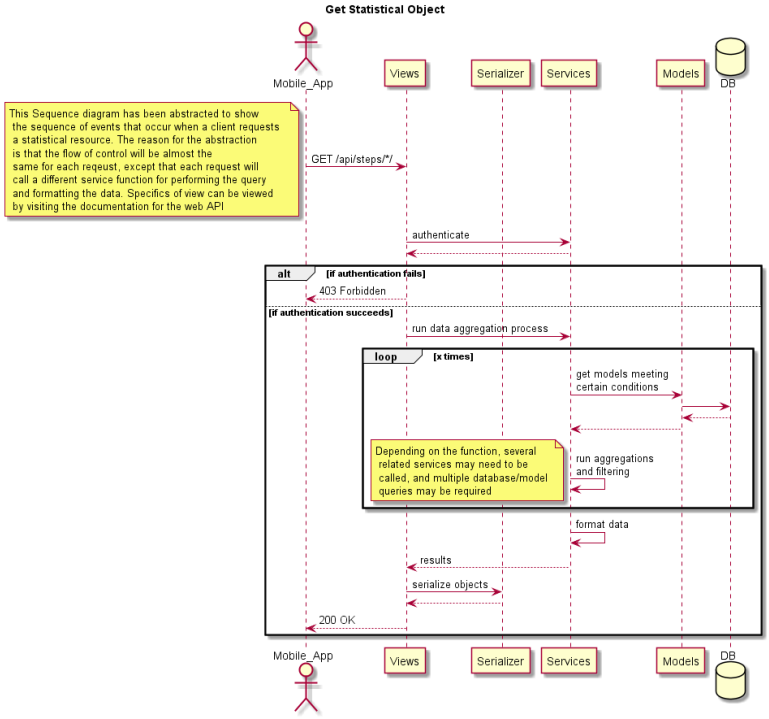

Figure 31 - Sequence Diagram for obtaining a statistical object from the mobile REST API.

Post Steps

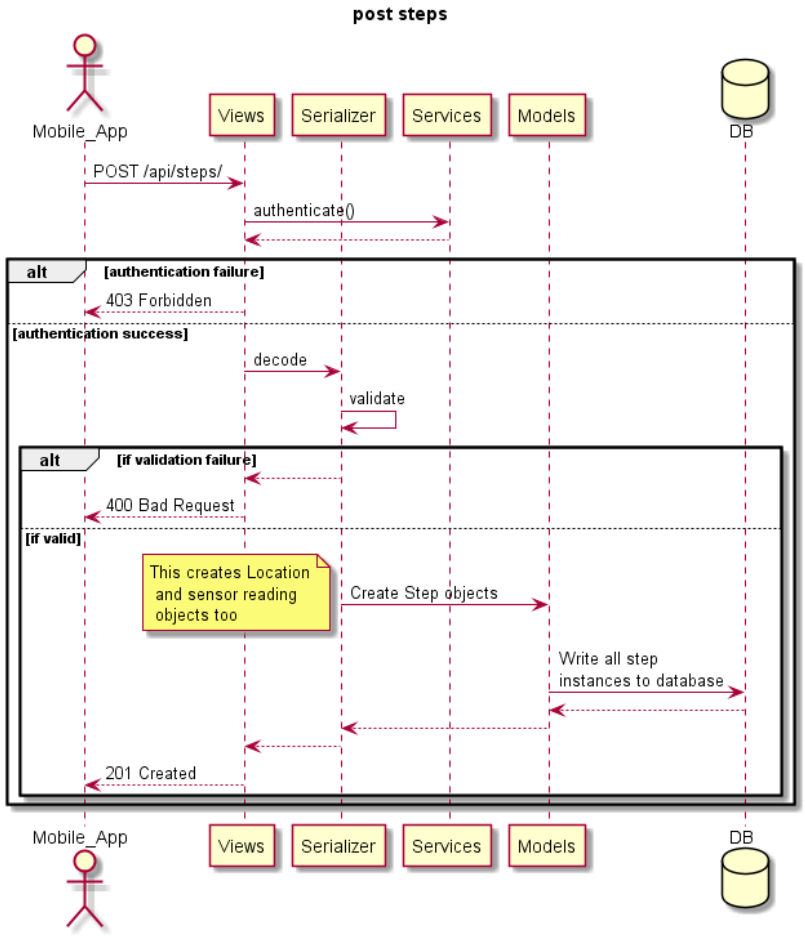

Figure 32 - Sequence diagram for posting step data to the web server.

Mobile Authentication

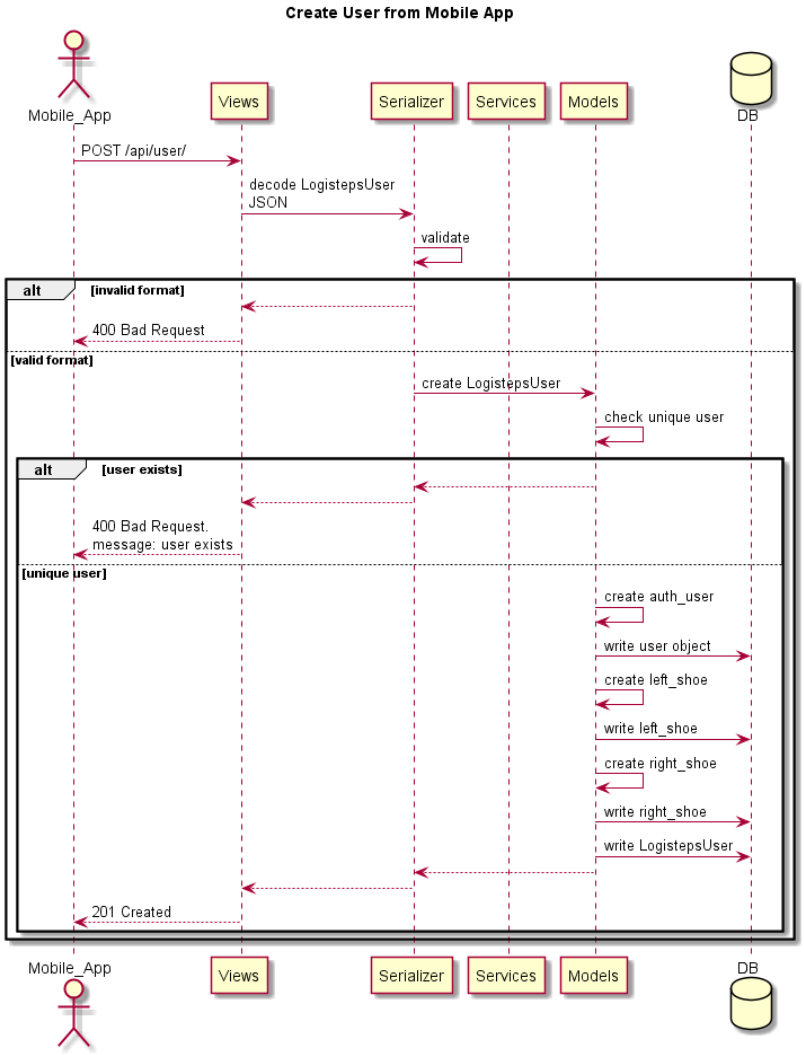

Figure 33 - Sequence diagram for creating a new user from the mobile app.

5.4.2 LogiSteps Models

Overview

This section documents the models required to satisfactorily represent LogiSteps data objects. The purpose of each model is to represent an in-memory instance of a database row that can be easily manipulated using the Django/Python web application infrastructure. Each model declaration abstracts a database table, and a row is represented by creating an instance of the model. This represents one key piece of the model-view-controller (MVC) architecture used by the LogiSteps web application. Models are intended to be data structures with only essential fields and functions. Most functionality for manipulating data is performed at a different layer of the web application.

More information pertaining to models can be found in the Django documentation.

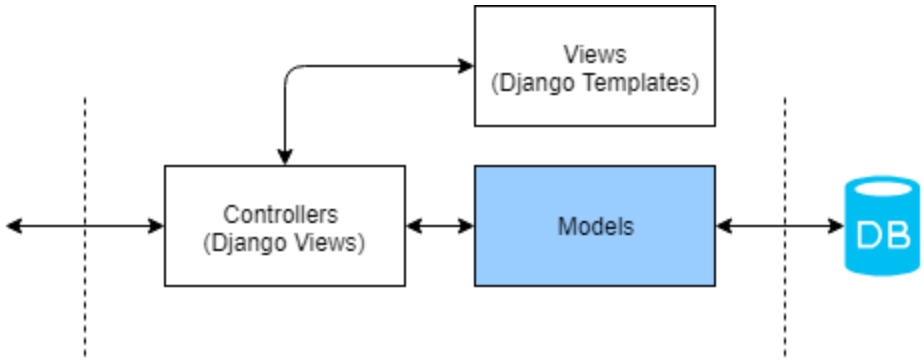

Figure 34 - Models act as the interface between the database and controllers.

It is essential that each model declaration resembles the database table as close as possible in order to reduce conflicts in communication with the database, as well as to keep a clear separation of function. Django will use the models do dynamically configure the underlying database.

Model Design

LogiStepsUser

logistepUser.py

Purpose

The purpose of this model is to represent each unique user for the LogiSteps products. This model should contain fields related to a user's physical attributes, assets, and authentication data.

Fields

Name	Type	Constraints	Description
user	Django Model	Unique	One-to-one field that references a user's default Django user object. Represents a database foreign key.
height	IntegerField	> 0	Height of the user in centimeters.
weight	IntegerField	> 0	Weight of the user in lbs.
step_goal	IntegerField	> 0	User's step goal that they wish to achieve daily.
left_shoe	Shoe Model	-	One-to-one field that references a user's left show object. Represents a database foreign key.
right_shoe	Shoe Model	-	One-to-one field that references a user's right show object. Represents a database foreign key.

Functions

The following functions should expose additional functionality that may be leveraged to retrieve additional data and provide customized manipulation. This is required because only models should make reads and writes to the database.

__str__

- Purpose: Should stringify the LogiSteps user for display purposes.
- Returns: String

user__username

- Purpose: Exposes the username of the LogiStepsUser. May be required for advanced queries.
- Returns: String

fullname

- Purpose: Returns a string representing the user's full name.
- Returns: String

height_ft

- Purpose: Get the height of the user, truncated to the nearest foot.
- Returns: Integer

height_in

- Purpose: Get the inches remaining of the user's height after truncating to the nearest foot. Intended to be used with height_ft
- Returns: Integer

delete

- Purpose: Override the default behavior for deleting object from database. Needs to handle proper deletion of one-to-one fields.
- Returns: void

Step

step.py

Purpose

Represents a unique step taken by a user. Instances are frequently created by both database reads and web API posts.

Fields

Name	Type	Constraints	Description
datetime	datetime object		Python datetime object representing both the date and time that a step was taken.
location	Location Model		One-to-one field that corresponds to the Django model object holding location data.
user	LogiStepsUser Model		ForeignKey object that points to the user in which the step belongs to. Not required but assists in writing concise queries.

Functions

__str__

- Purpose: Creates a string representation of the step object for display purposes.
- Returns: String

delete

- Purpose: Override default deletion behavior. Need to delete associated one-to-one fields.

- Returns: void

Shoe

shoe.py

Purpose

Represents a LogiSteps designed insole that belongs to a user's shoe. This should be a relatively small model, mainly used to differentiate steps between a user's left and right foot.

Fields

Name	Type	Constraints	Description
FOOT_CHOICES	List of tuples	Right or left	This is a list used by the Django templates and views that define available foot choices. The tuple should provide a mapping for decoding json.
size	DecimalField	max_digits=3, decimal_places=1	Represents the size of a user's insole/foot.
foot	CharField	max_length=1, choices=FOOT_CHOICES	A single character ("R" or "L") that represents which foot the insole should be associated with.

Functions

__str__

- Purpose: Creates a string representation of the foot for display purposes.
- Returns: String

SensorReading

sensorReading.py

Purpose

Simple data model to represent a single sensor reading. Includes the raw data read when recording the step. Sensor readings can take place on either the top or bottom of the insole.

Fields

Name	Type	Constraints	Description
LOCATION_CHOICES	List of tuples	top or bottom	This is a list used by the Django templates and views that define available sensor locations. The tuple should provide a mapping for decoding json.
pressure	FloatField		Raw pressure reading associated with step.
location	CharField	"T" or "B"	Indicates which sensor in the insole recorded the step.
shoe	ForeignKey		Foreign key to the shoe model that the sensor belongs to.

step	Step Model		Foreign key to the step that the sensor reading logically belongs to.

Functions

__str__

- Purpose: Creates a string representation of the sensor reading for display purposes.
- Returns: String

Location

location.py

Purpose

Represents a single location in which a step occurred.

Fields

Name	Type	Constraints	Description
latitude	FloatField	-180 <= latitude <= 180	Latitude of user when step was taken.
longitude	FloatField	-180 <= longitude <= 180	Longitude of user when step was taken

Functions

__str__

- Purpose: Creates a string representation of the location for display purposes.
- Returns: String

5.4.3 LogiSteps Controllers

Overview

This section provides design details regarding the web application's controllers. Each controller, known in Django as a View, handles requests by querying and processing data. Once the request has been fulfilled, the controller sends a response back to the requesting client. Consequently, there a controller must be defined for each valid URL – whether that is for retrieving web pages or manipulating/accessing data models. Since the LogiSteps web application must handle requests from the mobile app client (for getting/creating user data and posting step data) in addition to serving web pages to users wishing to view their data, there are two logically separated categories of controllers – web page views and rest framework views. This document will document design for all controllers necessary to fulfill system requirements.

The controllers sit facilitate communication between external agents, templates/views, and data models. More information regarding controllers (Django Views) can be found on their website.

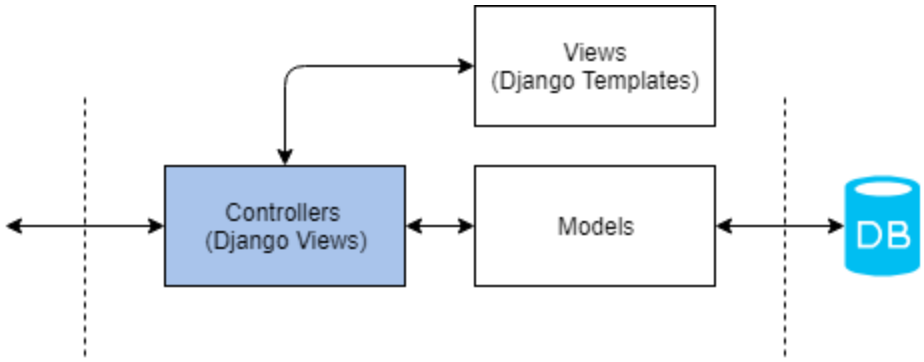

Figure 35 - Controllers handle requests and return responses.

Web Page Controllers

This set of vies provides should provide the functionality for displaying different web pages to a user. When users navigate to a URL in their browser, the controllers defined in this section will handle the GET requests and respond with the appropriate rendered HTML file.

For example, when a user navigates to /logisteps/login.html, the Login controller should handle the GET request for serving the web page, as well as handle the POST request that occurs when the user submits the form data.

Controllers

Register

Purpose

Provide an interface for users to register new accounts if they have never logged into the system before. This controller should serve an HTML page for submitting form data if a GET request is made and should handle data submitted from the client in a POST request.

URL

/logisteps/register/

Authentication

Login not required

Supported HTTP Methods

GET

- Action: Serve the registration template to the client. This should include a form for posting user data.
- Parameters:

  None

POST

- Action: Handle form data from client and create a new LogiStepsUser model.
- Parameters:

Name	Type	Constraints	Description
username	string	unique	Username for new account
email	string		Email for user. Should be validated client side
first_name	string	max_length=50	User's first name
last_name	string	max_length=50	User's last name
password	string		User's password. Password validation should be performed client side.

Complete Profile

Purpose

After completing registration on the registration page, a Django User model will be saved in the database, but a LogiStepsUser model cannot be created because there is not enough information to create the model yet. This controller provides an interface for completing a user profile.

URL

/logisteps/profile/complete/

Authentication

Login required.

Supported HTTP Methods

GET

- Action: Serve the template for completing a user profile. This should include a form for posting user data.
- Parameters:

  None

POST

- Action: Handle the form data from the client and create a new LogiStepsUser model.
- Parameters:

Name	Type	Constraints	Description
left_shoe	decimal	4 <= value <= 16 max_digits=3 max_decimal=1	Size of the user's left foot/insole. Validation should be performed client side.
right_shoe	decimal	4 <= value <= 16 max_digits=3 max_decimal=1	Size of the user's right foot/insole. Validation should be performed client side.
height_feet	integer	value > 0	User's height in feet.
height_inches	integer	value > 0	User's height in inches.
weight	integer	value > 0	User's weight in lbs.
step_goal	integer	value >= 0	User's daily step goal.

Login

Purpose

This controller provides a login page to a client and handles form data from the login screen when form data is posted. Upon successful login, this controller should automatically redirect a user to the landing page of the LogiSteps web application.

Note: Django provides a default controller which can handle this functionality. The only thing that needs to be done is implementing the login.html template.

URL

/accounts/login/

Authentication

Login not required

Supported HTTP Methods

GET

- Action: Serve the login.html template for authenticating a user. Should provide a form for posting required data.
- Parameters
  None

POST

- Action: Handle for data and check if user credentials are valid. Redirect to landing page if successful; show error message if unsuccessful.
- Parameters

Name	Type	Constraints	Description
username	string		User's username for their account
password	string		User's password associated with their account.

Profile

Purpose

This controller should provide a means for users to view and update their profile information. In particular, the controller should serve a form for updating LogiStepsUser fields, and then handle any POST request when the form is submitted by the user. Upon successful processing of the POST, users will see their updated information immediately.

URL

/logisteps/profile/

Authentication

Login required

Supported HTTP Methods

GET

- Action: Serve profile.html to the user with a form for updating their information.
- Parameters:
  None

POST

- Action: Handle user profile data submitted from the form and update the user's LogiStepsUser model. Should reload the page after success, and display error if there was an error.
- Parameters:

Name	Type	Constraints	Description
left_shoe	decimal	4 <= value <= 16 max_digits=3 max_decimal=1	Size of the user's left foot/insole. Validation should be performed clientside.
right_shoe	decimal	4 <= value <= 16 max_digits=3 max_decimal=1	Size of the user's right foot/insole. Validation should be performed clientside.
height_feet	integer	value > 0	User's height in feet.
height_inches	integer	value > 0	User's height in inches.
weight	integer	value > 0	User's weight in lbs.
step_goal	integer	value >= 0	User's daily step goal.

Logout

Purpose

This controller should provide a logged_out.html template to the client indicating that the user has been successfully logged out, with a link for logging back into the application.

Note: Django provides a default logout controller. A template will be provided to the default controller.

URL

/logisteps/logout/

Authentication

Login not required, but is a often a precursor to logout.

Supported HTTP Methods

GET

- Action: Should return a HTML document to the client indicating that they have successfully logged out.

Forgot Password

Purpose

This controller should allow a user to reset their password by serving a form to the client and handling subsequent POST requests. This controller will require users to reference resources outside of the webpage to verify identity.

URL

/accounts/reset/

Authentication

Login is not required since the user has forgotten their password, but they will be required to verify their identity using their email address that they provided when creating a profile.

Supported HTTP Methods

GET

- Action: Serves a webpage with a form for resetting a password.
- Parameters:
  None

POST

- Action: Should process request and change the user's password if user verification was successful. If verification fails, the user should be notified and prompted to try again.
- Parameters:

Name	Type	Constraints	Description
username	string		User's username that they user to login
email	string		User's email that they used to register
verification_code	string		Code sent to user's email to verify identity

---

Recent

Purpose

This controller should handle returning the HTML template that displays a user's step summary from the current day and the previous page. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API for populating graph data.

URL

/logisteps/recent/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the user's step summary for the previous two days, as defined by the system requirements.

Steps Over Time

Purpose

This controller should handle returning the HTML template that displays a user's steps over time for the current week. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API. The returned page will have an adjustable filter but changing the filter will be handled by the embedded JavaScript/REST framework.

URL

/logisteps/steps_over_time/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the user's steps over a week, as defined by the system requirements.

Steps by *Weekday*

Purpose

This controller should handle returning the HTML template that displays a cumulative steps per weekday graphic. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API to populate the graph data.

URL

/logisteps/steps_by_weekday/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the user's steps per weekday, as defined by the system requirements.

Activity by *Week*

Purpose

This controller should handle returning the HTML template that displays a user's activity time vs inactive time. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API to populate the graph data. The returned page will have an adjustable filter but changing the filter will be handled by the embedded JavaScript/REST framework.

URL

/logisteps/active_time/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the active time vs their inactive time, as defined by the system requirements.

Pressure

Purpose

This controller should handle returning the HTML template that displays a user's foot pressure over time. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API to populate the graph data. The returned page will have an adjustable filter but changing the filter will be handled by the embedded JavaScript/REST framework.

URL

/logisteps/pressure/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the user's pressure over time, as defined by the system requirements.

Map

Purpose

This controller should handle returning the HTML template that displays a step location for the past day. Due to the nature of Django databinding and JavaScript graphing libraries, the actual data for the

graphics will not be queried and processed in this controller. Rather, this controller will respond with an HTML page that has embedded JavaScript which makes calls to the rest API to populate the graph data. The returned page will have an adjustable filter but changing the filter will be handled by the embedded JavaScript/REST framework.

URL

/logisteps/map/

Authorization

Login Required

Supported HTTP Methods

GET

- Action: Return a rendered HTML page for displaying the user's step location over the past day, as defined by the system requirements.

REST API Controllers

The web page controllers' primary responsibilities are rendering and returning interactive HTML and JS files to the client as a way for a user to view and manipulate data. For controllers that return HTML documents with forms, they also implement functionality for processing data in POST requests which create/update LogiSteps models. REST API Controllers differ greatly from the web page controllers. Rather than serving web pages and handling responses, the primary responsibility for the REST API controllers is to return JSON documents representing LogiSteps models, statistics, and queries. In addition to returning JSON documents, the other primary responsibility for the REST API controllers is to manipulate and create models by interpreting JSON documents sent to the web server via POST, PUT, and DELETE HTTP methods.

The REST API Controllers expose data that would normally not be exposed by Django and are meant to be used by the mobile app and JavaScript embedded in the Django HTML templates. Most REST API controllers require basic authentication.

The REST API is documented in detail in the API documentation. Developers wishing to use the API should consult the API documentation found in section 11.1 of this document.

6. Project Developer's Guide

6.1 Required Tools

- Postman
- Android Studio
- Visual Studio Code
- Git
- GitHub
- PostgreSQL
- SEGGER
- 3D Printer

6.2 Repository Layout and Instructions

The LogiSteps project is split into several different architectural components, and each has been partitioned into its own GitHub repository. In particular, there are GitHub repositories for all of the hardware design files, the embedded application deployed on the microcontroller, the Android application, and the web technology. In addition to this, LogiSteps has a single GitHub repository that holds all the documentation for the project in a single centralized location. All of the GitHub repositories are held in a single GitHub organization.

The GitHub organization that holds all the GitHub repositories can be found at the following link.

https://GitHub.com/SeniorDesignTeamOmicron

Each individual repository is structured according to the requirements of the development tools to implement the project. Due to this, each repository is in a state where it can easily be imported into the development tools documented in section 6.1. For example, the mobile phone repository can simply be cloned locally to a computer and imported directly into Android Studio. Android Studio will then proceed to build the project.

For information regarding the structure required for each technology, refer to the online documentation for each respectively.

6.3 Deployment Process

The deployment process refers to work necessary for developers/technicians to perform for end users to successfully use the LogiSteps technology. End users are required to download the mobile application to their devices, but all other technological setup should be decoupled from the end user. This means that the embedded software must be deployed to insole technology prior to usage, and all web technology should be hosted on publicly accessible servers that LogiSteps monitors and maintains. For development, the LogiSteps team used Google Cloud VMs to host the online database as well as the web server. This section will document the process for deploying the embedded system software to the Nordic microcontroller and deployment of the web technology to Google Cloud services.

6.3.1 Embedded System Software Deployment

When deploying software to the nRF52832 microcontroller, the team found it was easiest to connect the nRF52832 hardware development kit to microcontroller and directly program it via the SWD protocol.

Prior programming onto the development kit hardware, the nRF software development kit (SDK) must be installed. The SDK offers a variety of drivers, libraries, and examples for programming and can be downloaded as a .zip archive. Since the SDK is contained in a .zip archive, various IDE and compilers can be used to run code. We chose to use SEGGER embedded studio as our IDE. In addition to downloading the SDK and the IDE, a SoftDevice is needed for Bluetooth capability.

To deploy the LogiSteps application to the microcontroller, SEGGER embedded studio is used to build and download onto the hardware development kit.

6.3.2 Google Cloud Deployment

Google Cloud SQL Instance

In order to save user data, including user-specific details and step data being streamed from a remote device, a database technology must be established to put the data into long-term storage that is easily accessible for consumption by technologies such as web servers and mobile applications. If the database was not hosted online, each end user would be required to setup their own SQL server, and expose it publicly to allow for mobile application communication. This user experience would be unacceptable, considering the target audience; to adequately satisfy the requirements of the end user, LogiSteps needs to deploy and manage its own cloud instances of the database technology.

The Google Cloud Platform provides the functionality needed by the LogiSteps application, while also providing an abundant number of additional resources and advantages, such as free tier development, scalability, security, etc. The first step in web technology deployment is making the database available online for use by the server written using the Django framework. The database is a dependency of the web application, so the database needs to be deployed *prior* to the deployment of the web application.

Moving the database to the Google Cloud Platform is a relatively simple process, and the following steps describe the process of doing so.

Creation of the SQL Instance

The first step in making the database online is the creation of a Google Cloud SQL instance. This is done by navigating to https://console.cloud.google.com/sql/instances and clicking the “Create Instance” button, as shown in figure 36 below.

Note: If a Google Cloud Account has not yet been created, this should be done prior to deploying the web technology.

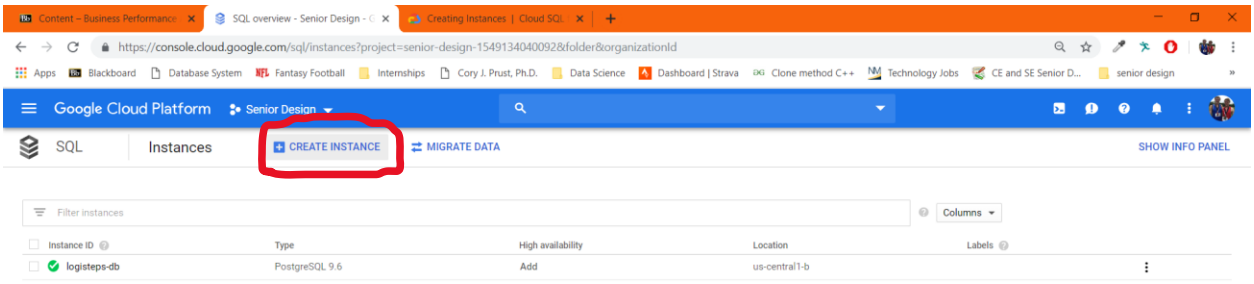

Figure 36 - Creation of GCP SQL Instance.

After clicking the create instance button, a screen asks whether the database should be a MySQL or PostgreSQL instance. To work with timescaleDB, which provides a scalable SQL implementation, the PostgreSQL engine should be chosen. Next, the configuration screen shown in figure 37 is presented.

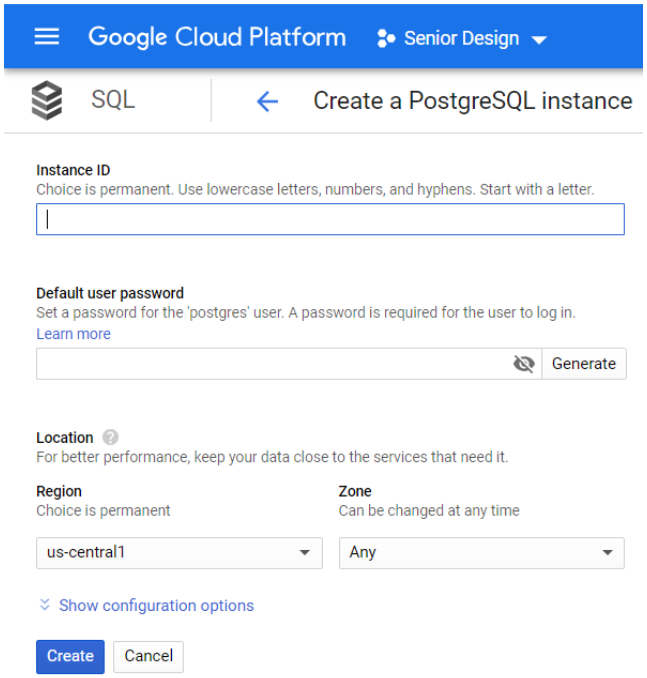

Figure 37 - GCP PostgreSQL configuration.

For development of the LogiSteps application, an instance ID of Logisteps_db was chosen, a secret password was created, and the rest of the options were left as their default values. Using a “us-central1” region ensures that the database is hosted in a geographical area near the Midwest, ensuring fast connection speeds. This should be adjusted based on the location of the targeted users.

After clicking the “create” button, creation of the PostgreSQL instance begins, and should complete after a couple minutes. At this point, an empty database should exist, but it still needs a database and user instance. To create a user for the LogiSteps application, a user can be created by navigating to https://console.cloud.google.com/sql/instances/logisteps-db/users (substitute appropriate DB name)

and clicking the “create user account” button. As shown in figure 38, a username and password are required.

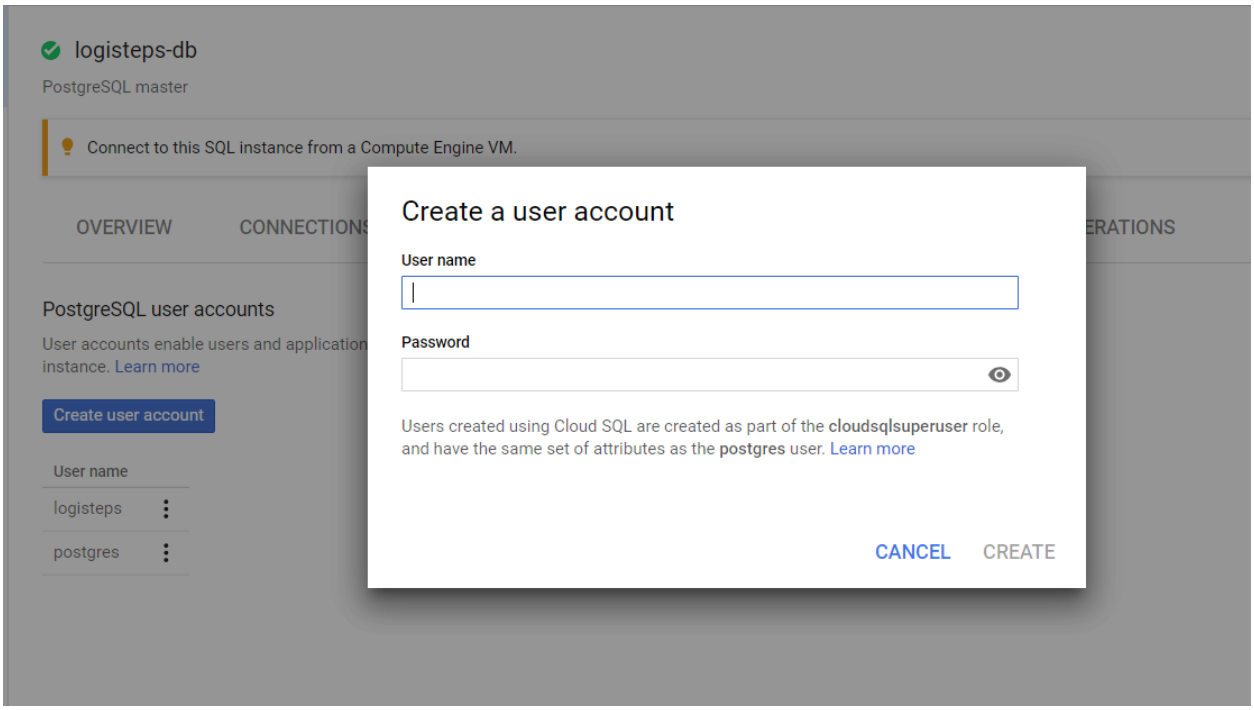

Figure 38 - Creation of a GCP SQL user.

The next step requires the creation of a database other than the default database created when the PostgreSQL instance was created. A new database can be created by navigating to https://console.cloud.google.com/sql/instances/logisteps-db/databases (again, substitute appropriate DB name) and clicking the “create database” button. A configuration screen, as shown in figure 39, asks for the name of the new database.

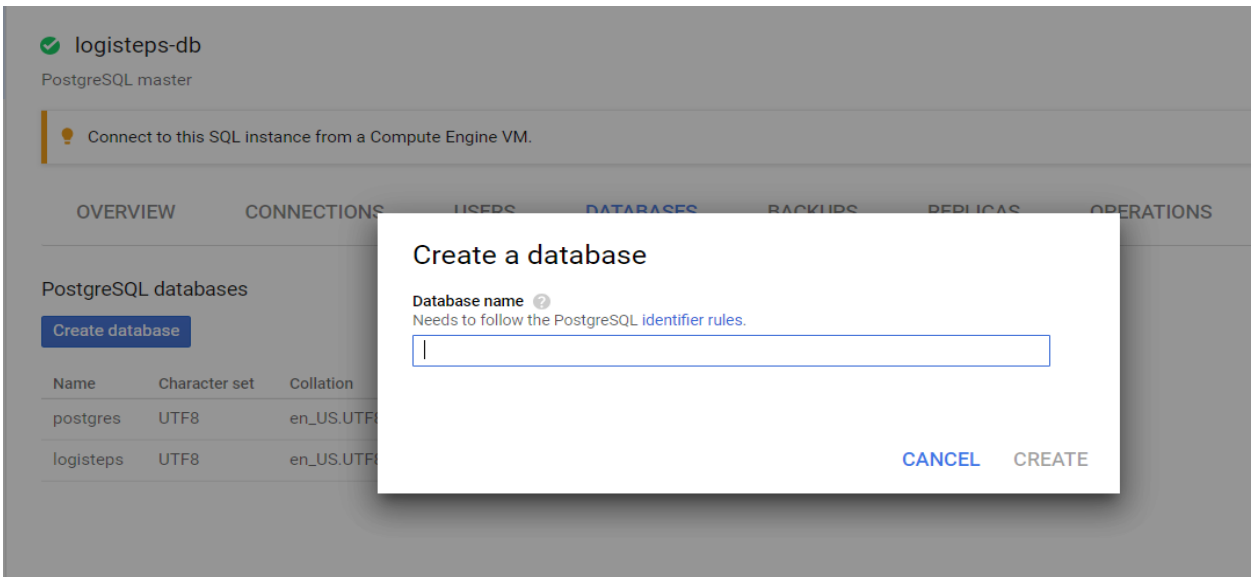

Figure 39 - Creation of a CGP SQL database.

At this point, a GCP PostgreSQL instance should be created, with a LogiSteps user and database. The last remaining step is creating the database tables required by the LogiSteps application. Django has a feature which creates database schema based off the Django data models, which can be used to create the required database tables. This step requires migrating the locally run Django web application from a local PostgreSQL instance to the cloud instance. Doing this requires the use of a Google tool known as the Google SQL Proxy. This tool makes it possible to connect a locally running application to a GCP SQL instance.

The first step involves enabling the API required by the Cloud Proxy. This can be done by navigating to https://console.cloud.google.com/flows/enableapi?apiid=sqladmin&redirect=https://console.cloud.google.com&_ga=2.247154700.-982376191.1548382701 and selecting the correct Google project. Upon successful enablement of the API, the message shown in figure 40 should be shown.

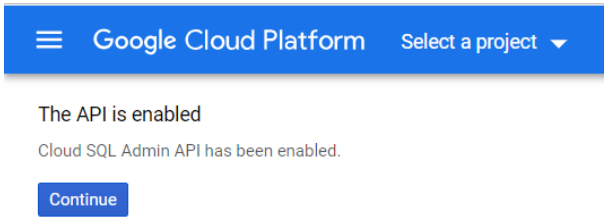

Figure 40 - Cloud SQL Admin API enablement.

Once the API had been enabled, the Cloud Proxy should be downloaded from https://dl.google.com/cloudsql/cloud_sql_proxy_x64.exe (assuming Windows) and moved to the root of the web application. Using the Google Cloud SDK Shell, the proxy should be run using the following command.

```
./cloud_sql_proxy -instances=senior-design-1549134040092:us-
central1:logisteps-db=tcp:3307
```

At this point, the Cloud Proxy is serving the cloud database on local port 5432, making it possible to run the web server locally, while maintaining a connection to the cloud instance. This made it possible to run Django migrations, which configures the database according to the Django data models.

The last step required prior to running Django migrations is configuring the web servers settings.py file to properly connect and authenticate with the Cloud Proxy. To do so, the DATABASES object should be changed to the following (enter the secret password in the PASSWORD field):

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql',
            'HOST': '127.0.0.1',
            'NAME': 'logisteps',
            'USER': 'logisteps',
            'PASSWORD': '••••••••',
            'PORT': '3307'
        }
```

```
    }
```

At this point, the web application can be run locally with a connection to the cloud database. The next step is running database migrations. To do this, the following commands should be run in the root of the web server project.

```
py /manage.py makemigrations
```

```
py /manage.py migrate
```

Following successful execution of these commands, the GCP SQL instance will be properly configured for use by LogiSteps applications.

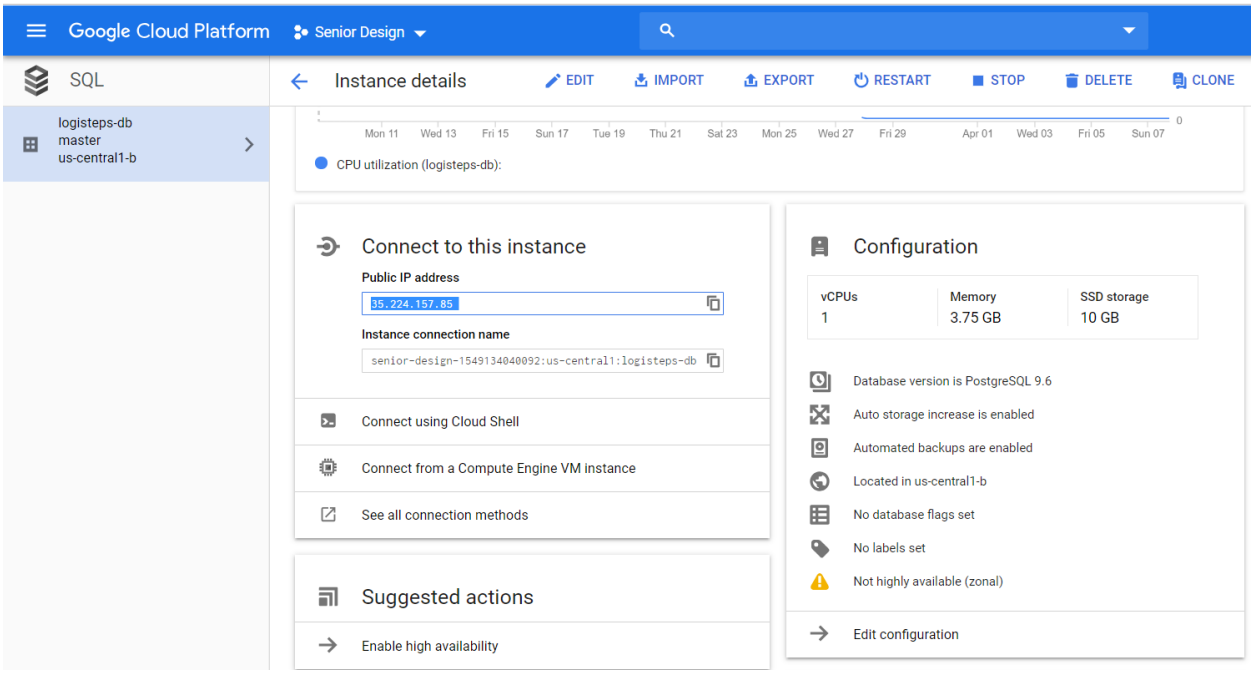

Figure 41 - Overview of CGP SQL instance.

Google Cloud Compute Engine

After the LogiSteps databases has been successfully deployed to the Google Cloud infrastructure, the next step requires deploying the rest of the LogiSteps web technology. This step involves a significantly larger number of configuration steps and configuration files. The following sub-sections describe the process to follow when deploying the web application and its related features.

The technology used to host the web application is the Google Cloud Compute Engine. The compute engine is a Google Cloud technology that allows execution of custom software in a virtual machine. The virtual machine is configured to only operate while actively serving requests and go to sleep when no longer needed to reduce power consumption and cost.

Configuration Files

The first step in deploying the LogiSteps web technologies is creating the configuration files required by the Google Cloud Platform. The configuration files help the Google VMs determine the proper settings to use so that the custom application is properly executed. The first file that should be created is a configuration file named app.yaml. This file should be placed in the root of the project configured as shown below.

Note: These configuration files were obtained by following the tutorial located at **https://medium.com/@BennettGarner/deploying-a-django-application-to-google-app-engine-f9c91a30bd35.**

```
# [START django_app]
runtime: python37
handlers:
# This configures Google App Engine to serve the files in the app's
# static directory.
- url: /static
  static_dir: static/
# This handler routes all requests not caught above to the main app.
# It is required when static routes are defined, but can be omitted
# (along with the entire handlers section) when there are no static
# files defined.
- url: /.*
  script: auto
# [END django_app]
```

The next step involves creating a main.py file in the root of the project. This file helps the Google App Engine start the application. The file needs to appear similar to the following code snippet.

```
from mysite.wsgi import application
# App Engine by default looks for a main.py file at the root of the app
# directory with a WSGI-compatible object called app.
# This file imports the WSGI-compatible object of the Django app,
# application from mysite/wsgi.py and renames it app so it is
# discoverable by App Engine without additional configuration.
# Alternatively, you can add a custom entrypoint field in your app.yaml:
# entrypoint: gunicorn -b :$PORT mysite.wsgi
app = application
```

Next, a requirements.txt file needs to be generated to list the dependencies of the Django app. This file is needed so that the Google Compute Engine can install the software libraries used by the LogiSteps application. At the creation of this document, only the following requirements are needed for the Django app.

```
Django==2.1.7
```

```
djangorestframework==3.9.0
psycopg2==2.7.6.1
pytz==2018.6
```

Note: These dependencies should be updated as future releases are available.

Next, the settings.py file needs to be updated to tell the Google Compute Engine where to find the web applications static files, such as HTML, javascript, CSS, etc. To do this, a STATIC_ROOT config was added to the settings.py file, as shown in the following code snippet.

```
STATIC_ROOT = 'static'
```

The last step in configuration file creation requires verifying the contents of the /mysite/wsgi.py file. The file should be verified to ensure it holds the following content.

```
"""
WSGI config for mysite project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')

application = get_wsgi_application()
```

Following this, all of the configuration files required by the GCP should be created.

Collecting Static Files

During development of the Django web application, dozens of static files were created to render content for client browsers. While all static content is placed in static folders, the folders were separated among several different Django projects. The three projects created during development of the LogiSteps application were the default “mysite” project, the “logisteps” project, and the “logisteps_api” project.

Prior to deploying the web technology, all static files need to be located in a single static folder for the Google Compute Engine to access. Django provides a convenient command for locating and gathering all static files in a single folder. To do this, the following command should be run from the root of the web application project.

```
py /manage.py collectstatic
```

At this point, all static files should now be located in a single static folder, located in the root of the project, as shown in figure 42.

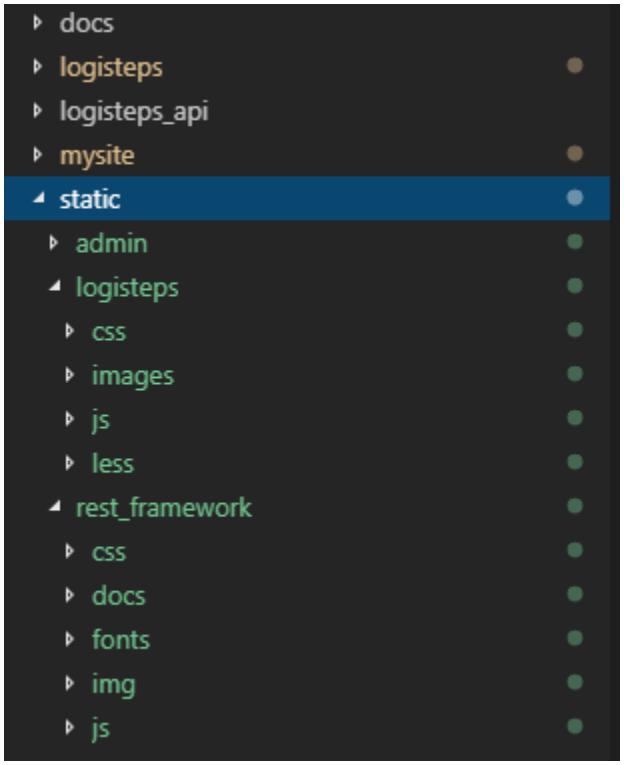

Figure 42 - static files located in the root of the project.

Application Deployment

At this point, all configuration should be complete, and nearly ready for deployment. Prior to deploying the application, the DATABASES object in settings.py should be changed to reference the cloud database directly, rather than referencing it through the Cloud Proxy. The following code snippet shows the configuration required for this.

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql',
            'HOST': '/cloudsql/senior-design-1549134040092:us-central1:logisteps-db',
            'USER': 'logisteps',
            'PASSWORD': '••••••••',
            'NAME': 'logisteps'
        }
    }
```

After this, the application should be deployed using the Google Cloud SDK Shell by running the following command.

```
gcloud app deploy
```

Upon completion of this command, all the web technology should be hosted in a Google Cloud VM. This includes.

- Web Server
- Mobile App REST API
- Static client files
- User data

6.5 Example Feature Extension Scenario

In the future, there will be more features added to LogiSteps. An example that might be added is a more precise foot map. The foot map can be found on the web application once signed in. currently it shows only two locations. In the future, if more sensors are added to the insole module, the foot map could become more precise and more intuitive. It might show high-pressure areas as different colors than the low-pressure areas based on readings from each sensor. All that would be needed is more sensors in the insole, tags for each one so the server knows which is sending data, and coding on the server to allow for more than two sensors.

Another addition to LogiSteps might be to add a real time weight measurement to the mobile application and web server. That can be done with a little coding on the server's end, and with a layout update on the mobile application end. The server would have to take the sensor reading from both sensors, convert it to weight with a calculation on the sensor's datasheet, then average the two to get a live reading of the user's weight. It could then pass it to the mobile application for the user to easily view in real time.

7. Test Plans and Results

7.1 Insole

Energy: 180uJ per step

The energy per step is defined as the increase in energy stored on the capacitors within the insole every step. This is measured by counting the number of steps required to charge the input capacitor up to 15V, and the output up to 3.6V, then dividing the energy stored on the capacitors at that voltage by the number of steps.

Water/Dust Resistance: N/A

Water and dust resistance in the future will be tested by submerging the device in a small bucket of water. This iteration though has no waterproofing whatsoever yet.

FSR Sensitivity: ~7kOhms Heel / ~8kOhms Forefoot

FRS sensitivity measures the lowest resistance measured be the force sensor during a brief walk.

Flexibility Result: No damage

The flexibility test was run by bending the insole 90° and analyzing the damage.

Dimensions: 280mm x 100mm x 8mm

These are the largest dimensions of the insole. It's 8mm in-depth all the way around. The thinness combined with the flexibility makes it very comfortable. The components could however fit into a shoe as small as a 10 male and potentially any larger size.

7.2 Web Server

7.2.1 Overview

This document serves as a test plan for verifying correct behavior of the web server. This test plan will test both the user facing web serving capabilities of the application, as well as the REST API used by a mobile client and XHR JavaScript requests.

This test plan covers higher level integration/systems tests. To test individual functions of the application, unit tests should be written and executed.

7.2.2 Test Setup

Prior to beginning the test, the tester should generate a JSON file with a week's worth of step data. This known data will be used to verify that endpoints are returning the expected content, whether that be a dynamic HTML page or a JSON documents.

7.2.3 Begin Test

1. Verify that the backend database has been emptied. Run the development web server by executing the command `py manage.py runserver`. **Indicate pass or fail: pass**

2. Attempt to create a new user by sending the following body in a POST request to /api/user/

```
{
    "user": {
        "username": "test8",
        "email": "test@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson",
        "password": "test"
    },
    "right_shoe": {
        "foot": "R",
        "size": 8.5
    },
    "left_shoe":{
        "foot": "L",
        "size": 8.5
    },
    "height": 67,
    "weight": 165,
    "step_goal": 10000
}
```

   - Verify that a 201-response code was returned. **Indicate pass or fail: pass**
   - Verify that the same JSON object was return, with the password removed and an id number present.
     Indicate pass or fail: pass
   - Verify that a User object was added to the auth_user table.
     Indicate pass or fail: pass
   - Verify that two shoe objects were added to the Shoe table.
     Indicate pass or fail: pass
   - Verify that a LogiStepsUser object was added to the LogiStepsUser table.
     Indicate pass or fail: pass

3. Attempt to create the same user again by sending the same HTTP request. Verify that the server responds with a 400-response, indicating that the user exists. **Indicate pass or fail: pass**

4. Attempt to create a user a field omitted. Verify that the server responds with a 400-reponse, indicating that the data is invalid. **Indicate pass or fail: pass**

5. Attempt to retrieve the user without providing valid credentials by making a GET request to /api/user/test8/. Verify that the server response with a 403-response, indicating in the response body that an invalid username/password was provided. **Indicate pass or fail: pass**

6. Repeat the same request but provide the user's username and password in the request header for authorization. The server should respond with a 200-response, providing the user object shown below in the JSON body. **Indicate pass or fail: pass**


```
{
    "user": {
        "id": 7,
        "username": "test8",
        "email": "test@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson"
    },
    "left_shoe": {
        "size": "8.5",
        "foot": "L"
    },
    "right_shoe": {
        "size": "8.5",
        "foot": "R"
    },
    "height": 67,
    "weight": 165,
    "step_goal": 10000
}
```


7. Attempt to update a user's profile by making a PUT request to /api/user/test8/, providing the correct authorizations in the header of the request. Use the following JSON object as the body of the request.


```
{
    "user": {
        "username": "test9",
        "email": "test@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson",
        "password": "test"
    },
    "right_shoe": {
```

```
        "foot": "R",
        "size": 8.5
    },
    "left_shoe":{
        "foot": "L",
        "size": 8.5
    },
    "height": 67,
    "weight": 165,
    "step_goal": 10000
}
```

Verify that the server response with a 400-response, indicating that the user's username cannot be changed, and verify that the user has not been updated in the LogiStepsUser database table. **Indicate pass or fail: pass**

8. Repeat the same request as step 7, but omit the username field, and change the user's step goal to 50. Verify that the server responses with a 200-response and that the user object has been updated in the LogiStepsUser table in the database. **Indicate pass or fail: pass**

9. Attempt to post step data for the user created in the previous steps by making a POST request to /api/steps/. The user's username and password should be provided in the header of the request, and the JSON object generated in the test setup should be provided as the body of the request.

    - Verify that the server responds with a 201-reponse containing the step data posted to the server. **Indicate pass or fail: pass**
    - Verify that the Step database table now holds rows corresponding to the posted step data. **Indicate pass or fail: pass**

10. Attempt to get a list of all step data that the user generated for a given day in the step data generated in the test setup by making a GET request to `/api/steps/steplist/?date=`, providing the given day at the end of the url in mm-dd-yyyy format. Verify that the server responds with a 200-response, proving an array of step data that occurred for the given day. **Indicate pass or fail: pass**

11. Repeat the same request but omit the GET parameters (/api/steps/steplist). Verify the server responds with a 200-response, providing an array of step data that occurred on the present day. **Indicate pass or fail: pass**

12. Repeat the same request as step 10 but provide a malformed date string. Verify that the server responds with a 400-responds. **Indicate pass or fail: pass**

13. Request a step summary for a given day in the data generated in the test setup by making a GET request to '/api/steps/summary/?date=', providing the given day at the end of the url in mm-dd-yyyy format. Verify that the server responds with a 200-response, with a body containing a JSON object with format shown below. **Indicate pass or fail: pass**


```
{
    "steps": 33,
    "goal": 49,
    "percent": 67.3469387755102,
    "least_active": {
        "hour": 2,
        "steps": 1
    },
    "most_active": {
        "hour": 0,
        "steps": 5
    },
    "inactive_time": {
        "hours": 23,
        "minutes": 29
    },
    "steps_per_hour": 1.375
}
```


    - Manually verify that the calculated statistics match with the data generated in the test setup. **Indicate pass or fail: pass**

14. Repeat the same request but omit the GET parameters. Verify that the statistics return match with manual calculations for step data posted for the current day. **Indicate pass or fail: pass**

15. Request a step count for the range of dates covered by the data generated in the test setup by making a GET request to /api/steps/count/?start=<startdate>;end=<enddate>. Provide the dates in mm-dd-yyyy format. Verify that the server responds with a 200-response containing a JSON object with the following format. **Indicate pass or fail: pass**


```
{
    "range": {
        "start": "10-22-2018",
        "end": "10-26-2018"
    },
    "counts": {
        "10-22-2018": 9435,
```

```
        "10-23-2018": 2345,
        "10-24-2018": 5555,
        "10-25-2018": 6483,
        "10-26-2018": 5093
    }
}
```

- Verify that the start date and end dates match the GET parameters.
  Indicate pass or fail: pass
- Verify that there is a key/value pair for each date in the range.
  Indicate pass or fail: pass
- Manually verify that the correct step counts are calculated.
  Indicate pass or fail: pass

16. Request a breakdown for the number of steps taken, grouped by week. This can be done by making a GET request to /api/steps/breakdown/. Provide the GET parameters "groupby=weekly". Verify that the server responds with a 200-response in the format shown below. **Indicate pass or fail: pass**

```
{
    "groupby": "weekly",
    "week": [
        {
            "day": 0,
            "steps": 45235,
            "active time": 3542,
            "inactive time": 3245
        },
        {
            "day": 1,
            "steps": 45235,
            "active time": 3542,
            "inactive time": 3245
        },
        {
            "day": 2,
            "steps": 45235,
            "active time": 3542,
            "inactive time": 3245
        },
        …
```

- Perform a manual calculation based on the setup data, and verify that the step count, active time, and inactive time match. **Indicate pass or fail: pass**

17. Request a pressure breakdown from the server by making a GET request to /api/steps/pressure, and append the GET a date parameter for a given day. Verify that the server responds with a 200-response containing a body with the following structure. **Indicate pass or fail: pass**


```
{
    "query_date": "04-30-2019",
    "pressure": {
        "past_day": {
            "left_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 0
                },
                {
                    "location": "B",
                    "avg_pressure": 0
                }
            ],
            "right_shoe": [
                {
                    "location": "B",
                    "avg_pressure": 31.1591
                },
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                }
            ]
        },
        "past_week": {
            "left_shoe": [
                {
                    "location": "B",
                    "avg_pressure": 96.4787878787879
                },
                {
                    "location": "T",
                    "avg_pressure": 96.7030303030303
                }
            ],
            "right_shoe": [
                {
                    "location": "B",
                    "avg_pressure": 31.1591
```

```
                },
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                }
            ]
        },
        "past_month": {
            "left_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 8.30329144225015
                },
                {
                    "location": "B",
                    "avg_pressure": 96.4787878787879
                }
            ],
            "right_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                },
                {
                    "location": "B",
                    "avg_pressure": 31.1591
                }
            ]
        }
    }
}
```


- Manually calculate, based on the data generated in the test setup, the average pressure for the past day, past week, and past month. Verify that the manual calculations match the response from the server. **Indicate pass or fail: pass**
- Verify the server returns a query_date the same as the date parameter appended to the URL. **Indicate pass or fail: pass**

18. Perform the same query but omit the date parameter. Verify that the server returns data for the previous day, week, and month from the current day. **Indicate pass or fail: pass**

19. Request the user's location data for a given day by making a GET request to /api/steps/location/, appending a date parameter to the URL. Verify that the server returns a

200-response with an array of JSON objects containing latitude and longitude key pairs for the date requested. **Indicate pass or fail: pass**

This concludes the tests for verifying proper behavior of the REST web API. The next tests will verify proper user facing web behavior. The pages served to a user use the same REST API that was previously documented, so the following tests aim to verify the proper elements are present in the page, and that the user's path through the application is correct.

20. Navigate to /logisteps/ and verify that the web server redirects the browser to /accounts/login/, displaying a form with a username field, a password field, and a login button.
    Indicate pass or fail: pass

21. Enter invalid credentials into the form and submit it to the web server by clicking login. Verify that an error message indicating that the username or password was incorrect, and that the web client remained at the login screen. **Indicate pass or fail: pass**

22. Re-enter the user's correct credentials, and verify that the web client is redirected to /logisteps/
    Indicate pass or fail: pass

23. Upon Login, verify that the user is presented with the recent view by default.
    Indicate pass or fail: pass

24. Verify that there is a navigation bar present spanning the top of the screen, allowing a user to navigate between their dashboard, connections, and global data.
    Indicate pass or fail: pass

25. Verify that a side navigation bar is present which allows a user to navigate between the recent tab, "steps over time" tab, a "steps by weekday" tab, an "activity by week" tab, a "pressure" tab, and a "map" tab. **Indicate pass or fail: pass**

26. Verify that the recent view presents the user with the following data:
    - A progress graphic for the current date and the previous date, displaying how close they are to meeting their daily step goal. **Indicate pass or fail: pass**
    - Verify that under each graphic, a table is present with total steps, average steps per hour, least active hour, most active hour, and inactive time. **Indicate pass or fail: pass**

27. Navigate to the steps over time page and verify that the following elements are present:
    - Verify that a line graph is present, with days of the week on the x-axis, and step count on the y-axis. **Indicate pass or fail: pass**

- Verify that there is a filter at the bottom of the page allowing a user to adjust the viewing/data window. **Indicate pass or fail: pass**
- Verify that the graph re-renders when the filter is changed. **Indicate pass or fail: pass**

28. Navigate to the Activity by Week page and verify that the following elements are present:
    - Verify that inactive time vs active time is displayed using stacked bar charts with days of the week as the x-axis, and time as the y-axis. **Indicate pass or fail: pass**
    - Verify that a interactive filter is present, allowing a user to change the week displayed week. **Indicate pass or fail: pass**
    - Verify that the graph re-renders when a different week is selected. **Indicate pass or fail: pass**

29. Navigate to the pressure tab and verify that the following elements are present:
    - Verify that there are 3 elements displaying average pressure. One for the past day, past week, and past month. **Indicate pass or fail: pass**
    - Verify that there is an interactive graphic present for changing the date. **Indicate pass or fail: pass**
    - Verify that the graphics re-render when the filter is changed. **Indicate pass or fail: pass**

30. Navigate to the Location tab and verify that the following elements are present:
    - Verify that a map is displayed, with a user's location data overlaid onto the graph. **Indicate pass or fail: pass**
    - Verify that the user can change the displayed day using an interactive filter. **Indicate pass or fail: pass**
    - Verify that when the date is changed, the graph re-renders with updated user data. **Indicate pass or fail: pass**

31. Navigate to /logisteps/profile/ and verify that a form is present allowing a user to update their profile information. **Indicate pass or fail: pass**

32. Change a user's first name in the form and press submit. Verify that the user profile was updated in the database by inspecting the auth_user table. **Indicate pass or fail: pass**

33. Press the logout button in the upper-right corner of the screen and verify that the user is redirected to /logisteps/logout.html. **Indicate pass or fail: pass**

34. Lastly, use an HTTP client to send a DELETE request to /logisteps/user/test8/, providing the correct credentials, and verify that the user was successfully deleted from the auth_user and LogiStepsUser database. **Indicate pass or fail: pass**

7.2 Embedded Application

1. Power each board

The first step to programming the breakout board to fit our project needs was to successfully connect and power each board. The SparkFun breakout board had to have pins soldered onto it and connected to a breadboard to allow for any connections. In addition, a separate FTDI module had to be used with either a micro USB or mini USB cable connection. Similarly, the hardware development kit was powered via a micro USB connection.

Initially, there was a problem connecting with the hardware development kit board. Instead of the green power LED turning on, it had continuously flashed. After a bit of troubleshooting, it was discovered that the USB cable didn't allow for the necessary connections, and the board connected properly with a different cable.

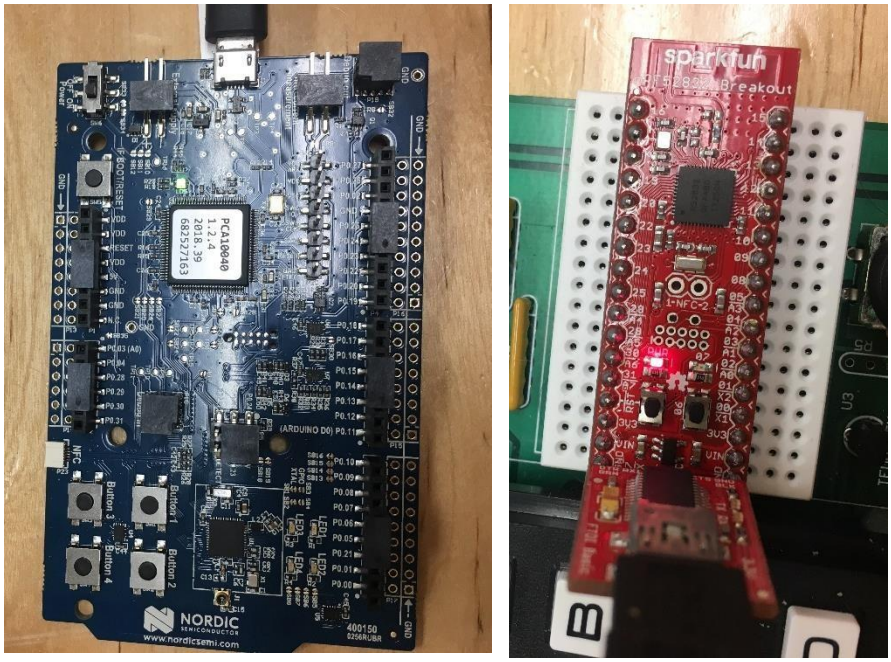

Figure 43 - Both boards are properly powered on and connected as shown by the power LED light being active (Green on left, red on right).

2. Connect boards together

The next step to programming the breakout board is to make the necessary connections between the hardware development kit and the breakout board. This allows for the Nordic SDK programs to be flashed onto the breakout board directly. This is done with the SWD protocol, and the SWDIO and SWDCLK from the hardware development kit is connected to the breakout board.

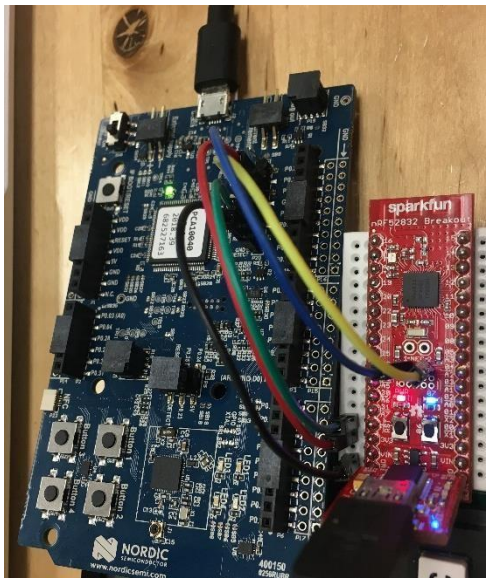

Figure 44 - The connections mentioned in the set up section are made to allow for direct programming onto the breakout board.

3. Run example programs on each board

Next, simple example programs were run on each board to ensure that they could properly run code. The hardware development kit was programmed directly separate from the breakout board and only needed the micro USB connection. The breakout board had to be connected to the hardware development kit as well as powered on with the mini USB cable. After the program is flashed to the breakout board, it no longer has to be connected to run the program.

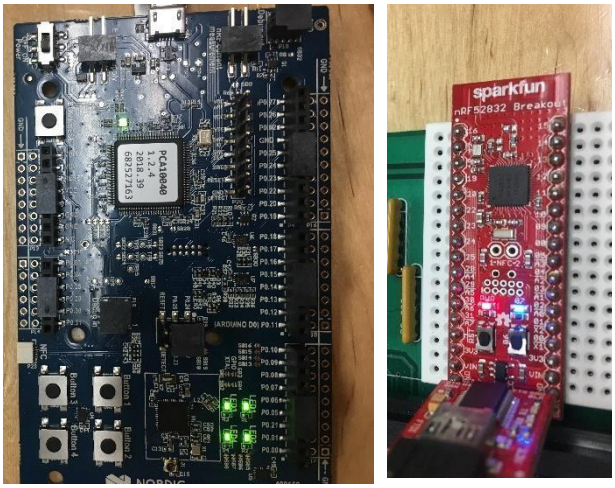

Figure 45 - Example programs are written to each board. Both of them toggle LED's on and off using a timer (green on left, blue on right).

4. Example program with ADC peripheral

An important aspect of our overall project is to be able to read voltages created from the sensors in the shoe. To do this, the sensors can be connected to an ADC unit on the microcontroller. To begin working with the ADC units, an example program was created to make sure that the ADC unit was reading voltages correctly.

A custom program was written using the ADC and the LED light. The LED light would turn on when voltage was applied to the ADC and would turn off when no voltage was being applied. We initially had problems with toggling the LED light but quickly realized the LED was active low and we had to adjust accordingly.

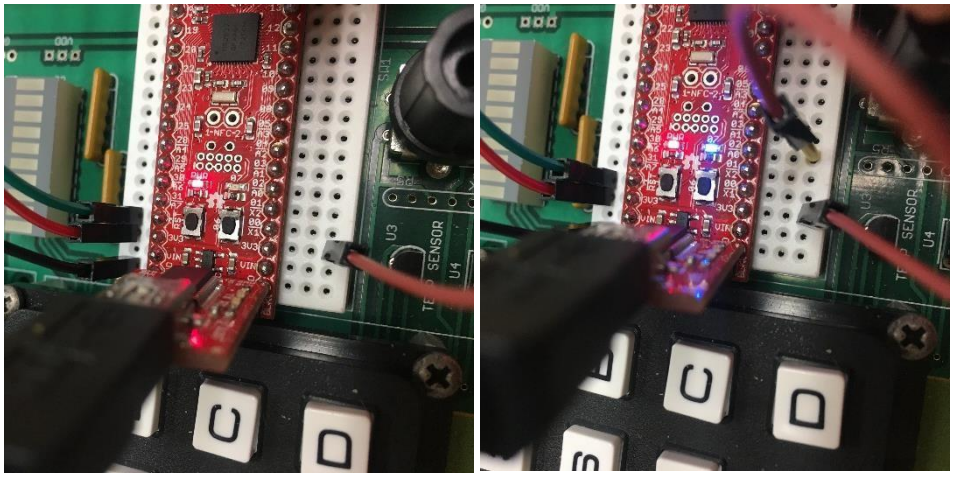

Figure 46 - The ADC sample program. The brown wire is connected to a 3.3-volt output. When the ADC is not connected to any voltage the LED is off (left). When a voltage is applied to the ADC, the LED turns on (right).

5. Example program with BLE peripheral

Another important aspect of our overall project is to be able to send data via Bluetooth to a mobile device. This will allow the microcontroller to continuously transmit step data as long as the microcontroller remains powered. Luckily, Nordic provides many example programs and libraries for Bluetooth capabilities.

To connect with the breakout board through Bluetooth, a Bluetooth connection app needed to be downloaded. Nordic provides an nRF Connect application to connect with any nRF microcontroller and is available on desktop and mobile devices. For displaying purposes, we used the desktop version of nRF Connect. The desktop version connected through the hardware development kit and was able to search for and connect with our breakout board.

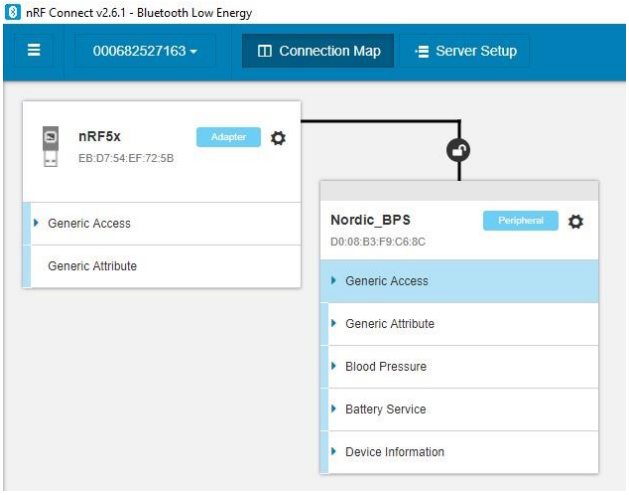

Figure 47 - An example Bluetooth program that successfully connects with our breakout board. Various descriptions of our breakout board can be seen through the Bluetooth connection.

6. Our program (sends ADC data over BLE)

Finally, the last step for the microcontroller to work for our project was for the microcontroller to be able to send ADC data over a Bluetooth connection to the mobile device. A circuit was built to allow for the force sensor to create a voltage that would then be read by the ADC.

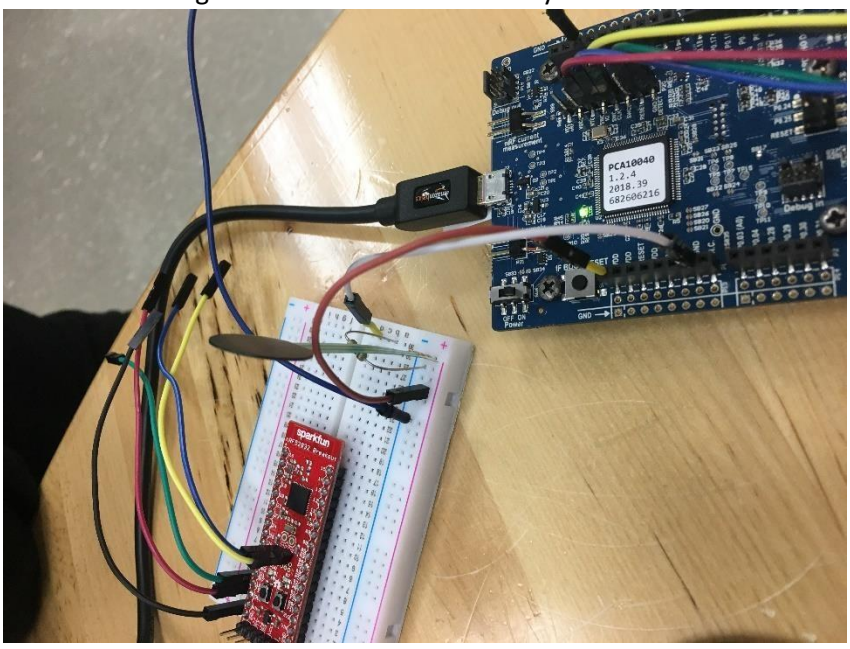

Figure 48 - The circuit created to allow the force sensor to create voltages. The force sensor is connected to a voltage source and a 10k ohm resistor, and the resistor is connected to ground. The ADC is then connected between the force sensor and the resistor to read voltages created by the sensor.

After the circuit was built to allow for the ADC to read voltages, the ADC values had to be sent over Bluetooth. A Bluetooth connection was established between the breakout board and the nRF Connect app, and values from the ADC peripheral were continuously sent over Bluetooth.

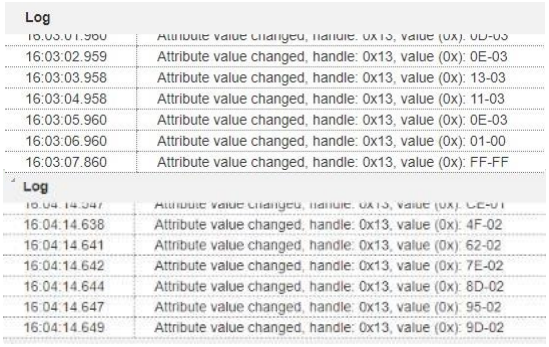

Figure 49 -ADC through the nRF Connect app. The ADC values 11 A few examples of values being read from the change based on the amount of force applied to the force sensor.

There were many problems encountered when we attempted to combine the ADC examples with the Bluetooth examples to meet our project needs. In general, there were many problems that had to do with the sdkconfig.h file. That file declares various things about the nRF52 microchip such as, for example, what timers are enabled, or the functions enabled of that timer. There were many things that came up that were not immediately obvious that had to do with settings in the sdkconfig.h.

One major problem that occurred had to do with a timer being used by multiple microchip systems. What happened was the SoftDevice, which controls the nRF52 Bluetooth capabilities, used a timer, and when the program attempted to use the same timer for the ADC peripheral, the system would say “Error” and reset itself. This was not noted anywhere easily visible.

Another error came in with making our own Bluetooth service and characteristics. Every Bluetooth service needs its own UUID, or universal unique identifier. Many are already defined by the Bluetooth SIG, and are 16-bit, but those that aren’t, like ours, need to have a special 128-bit one created for them. Nordic’s sdk allows for the creation of them, but they don’t easily tell everything that has to be changed to make them work. In order to make the Bluetooth vendor specific ID, there are two things outside of the source code that need to be changed. The first is in the sdkconfig.h file, where you need set the actual number of vendor specific IDs allowed. Then you need to change go into the project settings, use the common configuration, and change the linker memory placement macros to allow for space in memory being taken up by your vendor specific IDs. This was also not easily visible in the documentation used.

7.3 Mobile Application

1. Bluetooth Connection:

The first step in the mobile app is to get Bluetooth connecting. This is a main component of the mobile app and integrating it with the physical hardware of the LogiSteps system. Once this is completed, then when the main board of the device is completed, more testing can occur. For now, the connection is being established with a Windows 10 PC.

Initially, there is a problem with this method. The error that was coming up was one that made little sense. With a little internet searching, it is determined that the UUID of the device was the issue. All the documentation on the Android website said that the way the UUID is found originally is the correct way. However, in a deeper internet search, it is found that the documentation is out of date and a different method needed to be used. After much time and searching, the connection is established.

2. Server Connection:

The next step is to get the server connected. Meeting up with a teammate and getting a local version of the server installed is the first part of this. After that is done, being able to connect from an external device is next. The documentation for the type of server it is provides that answer. It is done when starting the server by putting in 0.0.0.0:port at the end of the command. This links the server to the IP address on the device it is being run. Once the command is run, using a different device and inputting the IP of the server, followed by the port, followed by /logisteps, will allow that device connection to the server. This is, if they are on the same network.

3. Server Interaction (Log in):

Since the connection is now set up, now interaction with needs to be tested. This needs to be done inside the app itself. The most basic way to interact with the server is to verify an account. The computer app Postman is used to create an account with the server. Then, in the mobile app in debugging mode, the request is made to the server to get the account. Luckily, Postman has a function that converts their requests into code that is useable for the app. This makes testing simple. Once the request is made, the debugger stops after the response is received and tells the user if there was a response and if so, what it is. If the response code is 200, everything went right.

4. Device Interaction:

The next step is to test the interaction with the physical device. This is later in the test plan because it is waiting on the device to be ready. Once the device is ready there are two things to test. The first is the initial connection. Upon initial connection, the app must send the time in milliseconds from 1970. To test that, the board will send back an ack of some sort when it receives it.

The second part is getting data from the device. To test this, the app will be put in debug mode, and set to stop when it receives over Bluetooth. This will inform the developer of the format that the data is coming in and that data is coming in. Once the data is received and the developer knows the format of the data, the data can be parsed and used to update the UI and send to the server.

5. Server Interaction (Post step):

The last thing to test in the mobile app is posting a step to the server. This will be tested similarly to the log in function in that the Postman helps create the request and the debugger is used to check the response. The difference is the code will be 201 if working. Once this is tested and the server shows 201, the mobile app will have all functionality. All that will be needed after this will be to shine it up and make it smoother.

8. Software Installation Guide

LogiSteps was designed to be as simple to the user as possible. The only software that needs to be installed is the mobile app. As of right now, the app is only deployable through android studio as it is still in the beta phase. Android studio can be found here: https://developer.android.com/studio/. This gives you the latest version. Once downloaded and installed, you need to clone the project through git or download the project from GitHub the app is on GitHub here: https://GitHub.com/SeniorDesignTeamOmicron/Mobile-Phone.git.

To clone the project git needs to be installed on your computer. Git can be found here: https://git-scm.com/.

1. Once git is installed, navigate to your file explorer where you would like to place the project on your computer.
2. Then right click and select open git GUI here
3. Click clone an existing repository
4. Copy the link to the repository from GitHub in the source location box
5. Click browse on the target directory box and then select okay
6. Go to the end of the file path in the target directory box and add /LogiSteps to the end
7. Click clone and wait for the git GUI to run

To download the project from GitHub, navigate to the GitHub page listed above with the mobile application in it.

1. Click clone or download
2. Click download zip
3. Navigate to your Download folder on your desktop
4. Right click the downloaded file named Mobile-Phone-Master.zip
5. Click Extract All. This will extract into a folder called Mobile-Phone-Master in your Downloads Folder

Once the project is on your computer, open Android Studio. Click Open Project and navigate to where the project is stored. Then click open.

Once the project is open, the android phone needs to be connected to the computer through its charging cord and USB. The phone also needs to be in developer mode.

1. Open the settings app
2. Select System
3. Scroll to the bottom and select About Phone
4. Select build number seven times
5. Return to previous screen and find developer options at the bottom
6. Turn on Developer Options
7. Scroll down until you see USB Debugging and turn that on as well

To continue installing the app, return to android studio and in the upper right had corner click the green play button. A screen will appear and your phone should be an option in the screen. Select your phone and click Okay. Once the app opens on your phone you have it installed!

You can now use the LogiSteps mobile application. Thank you for following along. In the future, the app will be deployed in the Google Play Store. Instructions on how to install from there are below.

1. Navigate on your android phone to the Play Store
2. Once there, search LogiSteps. It should be the first one on the list
3. Click on the app, then click Install
4. To ensure the install went correctly navigate on your phone to the application and open it, once it is open the application is fully installed

9. User Manual

LogiSteps was designed to ease the burden of installation and maintenance for the user. As a result, users only need to perform simple initial setup, and then use the web application to view their analytics. This user manual will document the necessary steps users must take to setup the device, as well as how to use the mobile app and web app.

9.1 Initial Setup

When users purchase their LogiSteps technology, they will receive two separate insole devices. These insoles are designed with custom hardware and software to track user fitness data for each foot. To begin, users should place the left insole into their left shoe, and the right insole into their right shoe. By completing this, users have completed all necessary hardware setup. The insoles should never need to be removed, except for repair, as they are self-powered using the energy of a user's movement.

An essential aspect of the LogiSteps application infrastructure is the mobile application. This allows LogiSteps to transfer the data from the smart-insole devices to permanent cloud storage. For the technology to work correctly, the mobile application should be downloaded to a mobile device. This can be done using Android Studio or the Google Play Store for Android devices, or the App Store for iOS devices (in the future). Upon successful installation, all the necessary software has been downloaded.

The next setup step involves creation of a user account. To do this, launch the LogiSteps mobile application and select the "create account" link, as shown in the figure 50.

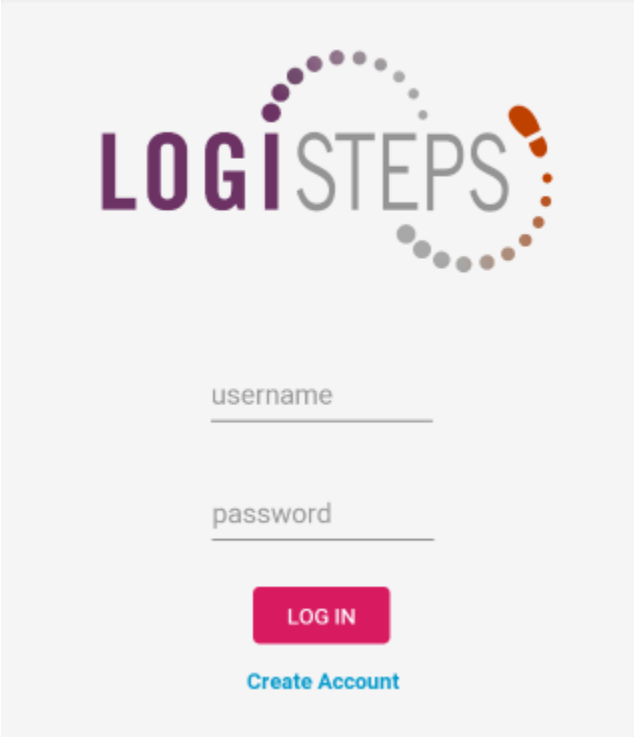

Figure 50 - Login screen for mobile app.

A new page will open with a form for user creation. Create a new user by filling out the form and clicking the create button. If a username has already been taken, the app will ask for a different username; otherwise, the app should open the main page. At this point, a user account has been created which can be used for both the mobile application as well as the web application.

One the main page has opened, the smart-sole devices should be paired using Bluetooth to the mobile device. Begin by clicking the “connect” button in the upper-left portion of the screen. Available Bluetooth devices will appear and select the insole for the left shoe.

Note: If no Bluetooth devices appear, try walking around the room. To be able to pair, the devices must be powered, and energy to power the smart-sole devices is harvested from steps.

Once the left insole Bluetooth device appears, click on the device. Pairing should begin and upon completion, the list of Bluetooth devices should disappear, and the connection status text for the left shoe should say “connected”. Repeat this process for the right shoe.

Once this has been completed, all initial setup is complete. As walking begins, the smart-sole devices should begin recording data and sending it to the mobile app, which will relay the data to the cloud. No further installation or configuration is needed.

9.2 General Use

General use consists of using the mobile app and web app to view step data and analysis. The applications are designed to satisfy different use cases, with the mobile app displaying status information, and the web application disapplying more advanced analytics and summary statistics. While the applications are vastly different, user data is synchronized across the platforms.

9.2.1 Mobile Application

After initial setup, the mobile application is used to simply display summary statistics for the current day. This allows a user to grab a quick snapshot of their fitness for the day but limits the complexity of the application. To obtain a quick summary of activity for a given day, the user can log into the app and view the data on the main screen.

In addition to this, the mobile application can be used to verify Bluetooth connection status and repair/disconnect any connections as required.

9.2.2 Web Application

Login

To view statistics and summary data, users should navigate to the web application running at https://senior-design-1549134040092.appspot.com/. When the page loads, a login screen will be presented, and the credentials created during initial setup can be used to login. If an account has not been created yet, the “create account” link should be used to register a new user.

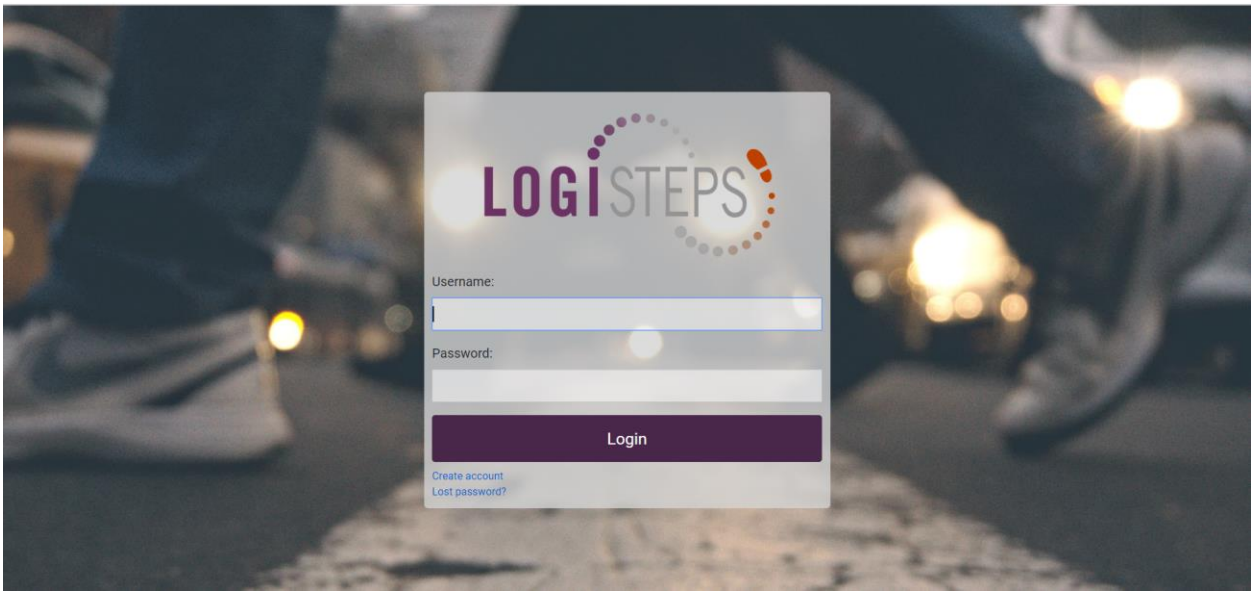

Figure 51 - Login Screen.

Logout

Users may log out of the LogiSteps web application by clicking the logout button located in the upper right portion of the page, as shown in figure 52. Upon success, the user will be navigated to the login page.

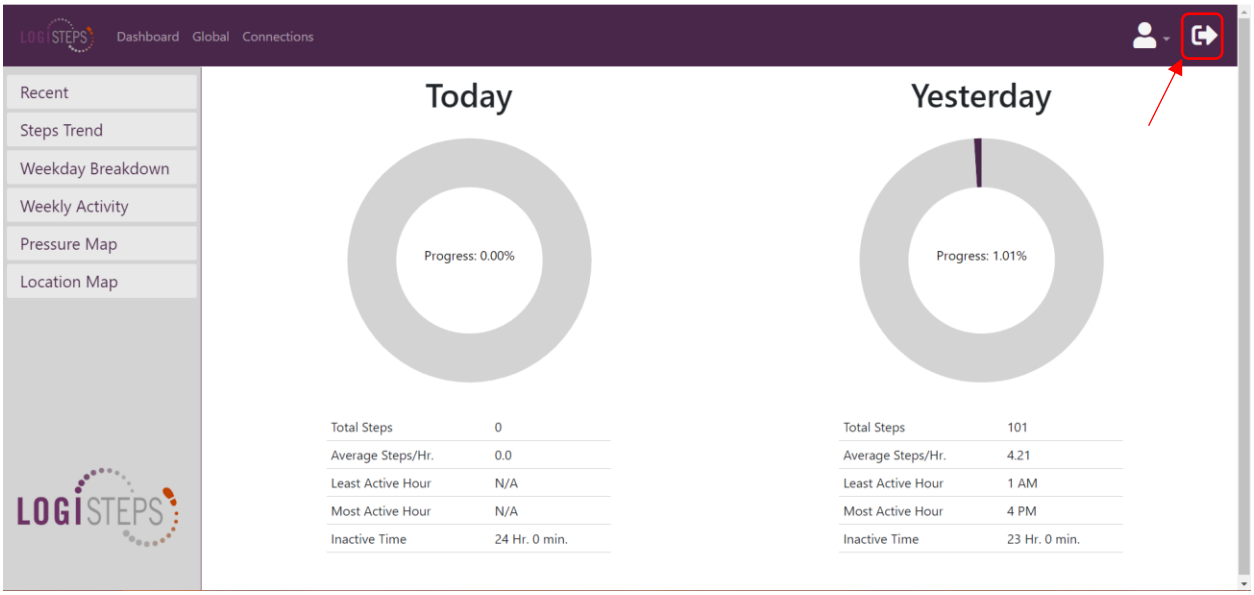

Figure 52 - Users may logout using the logout icon.

Edit Account

Users may edit their account details using the dropdown menu at the upper-right part of the screen. The dropdown will display basic user information, as well as a button that can be used to open an edit account page.

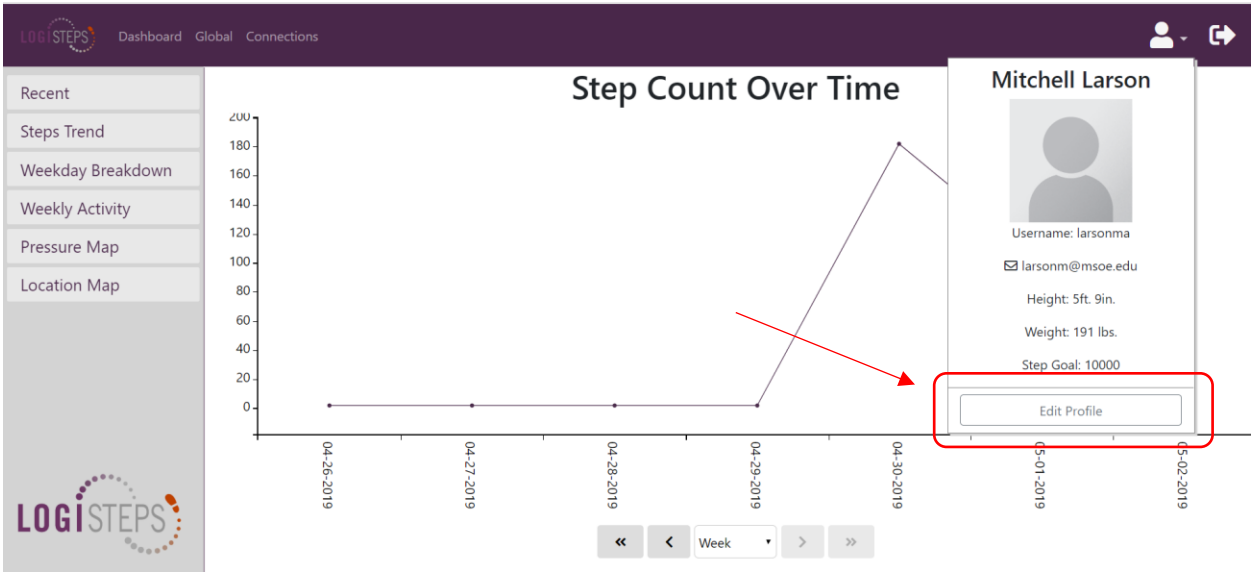

Figure 53 - User dropdown window.

Once the button has been clicked, the page shown in figure 54 will open, and users may edit their data.

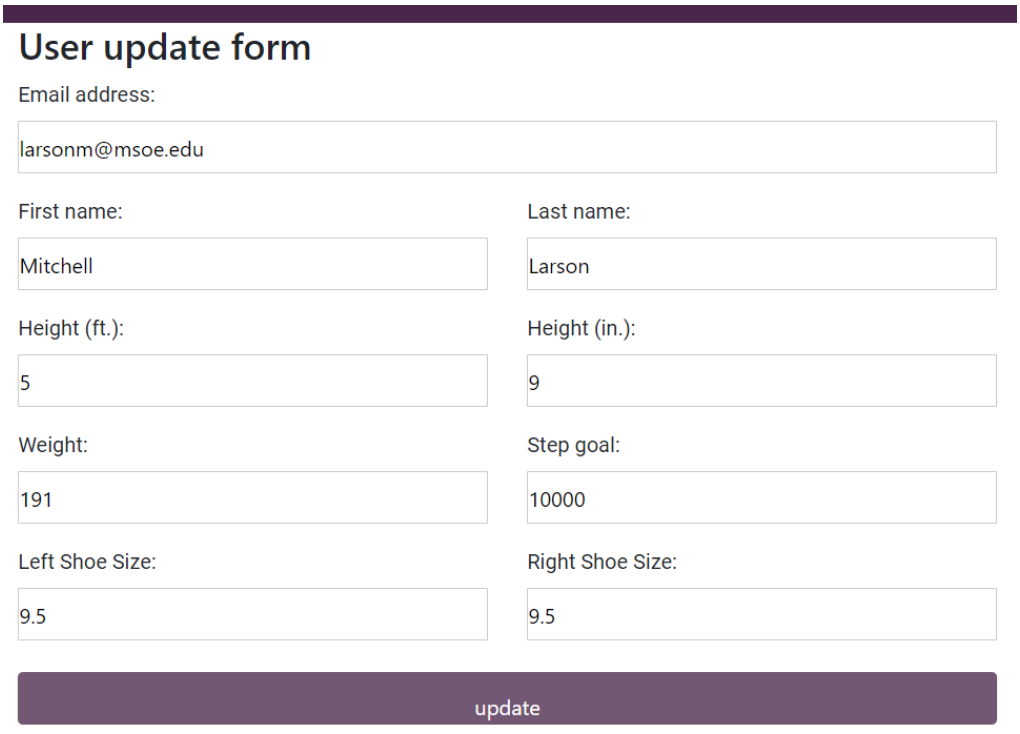

Figure 54 - User account update form.

When finished, the “update” button should be used to permanently save user data to the cloud.

Activity Summary

Users may access a summary of their activity for the past two days by clicking the “Recent” tab on the left side of the screen. This will present a page, shown in figure 55, which displays a user’s progress towards their step goal, as well as other statistics such as their most active hour and their average steps per hour. Data for the previous day is shown as well to help users compare their movement relative to the previous day.

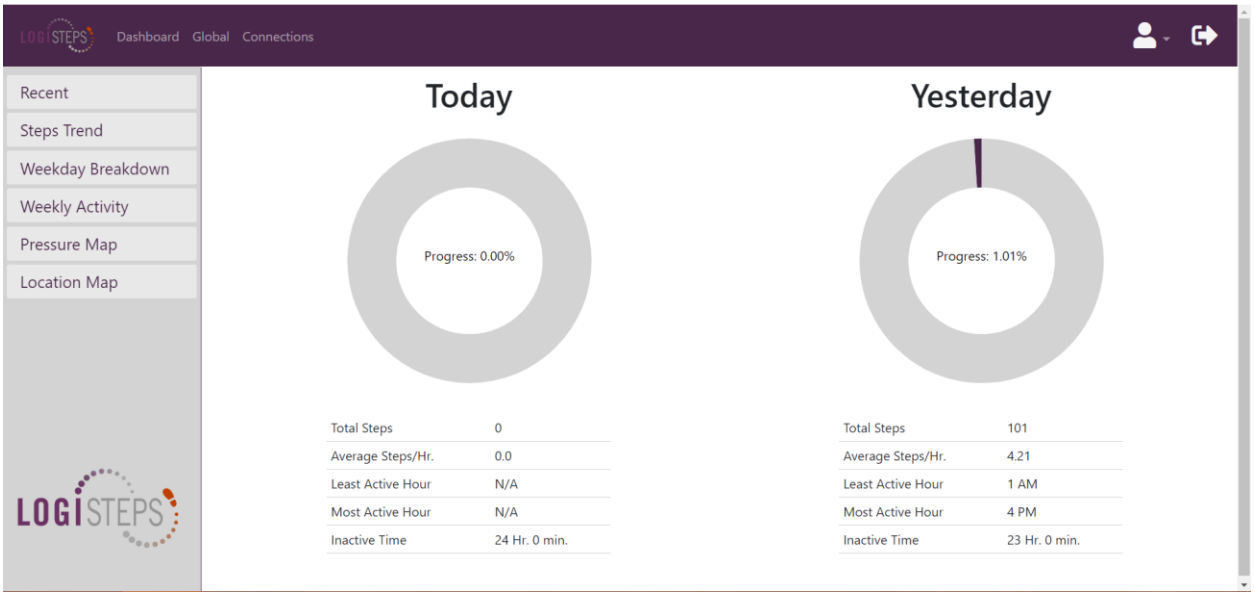

Figure 55 - Page displayed when navigating to the "Recent" tab.

Activity Trend

Users can view their activity trend by clicking the “Steps Trend” tab on the menu to the left of the screen. As shown in figure 56, this screen allows a user to view their total number of steps taken over a fixed period of time. The window can be shifted to the right or the left using the buttons at the bottom of the screen, and the window size can be adjusted using the drop-down menu.

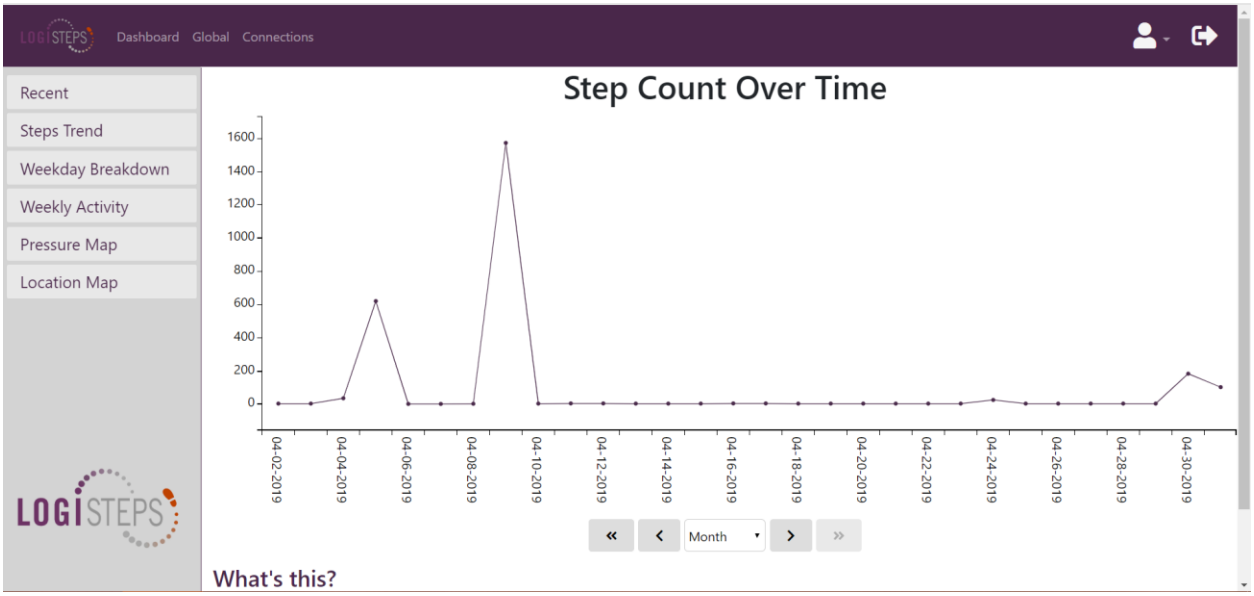

Figure 56 - Step count plotted over a fixed period of time.

Weekday Breakdown

By navigating to the Weekday Breakdown tab, users can obtain a graphical representation of their activity for the current week. The graphic, as shown in figure 57, compares the number of steps taken each day on the left axis, and the graphic displays a ratio of active minutes to inactive minutes using the right axis.

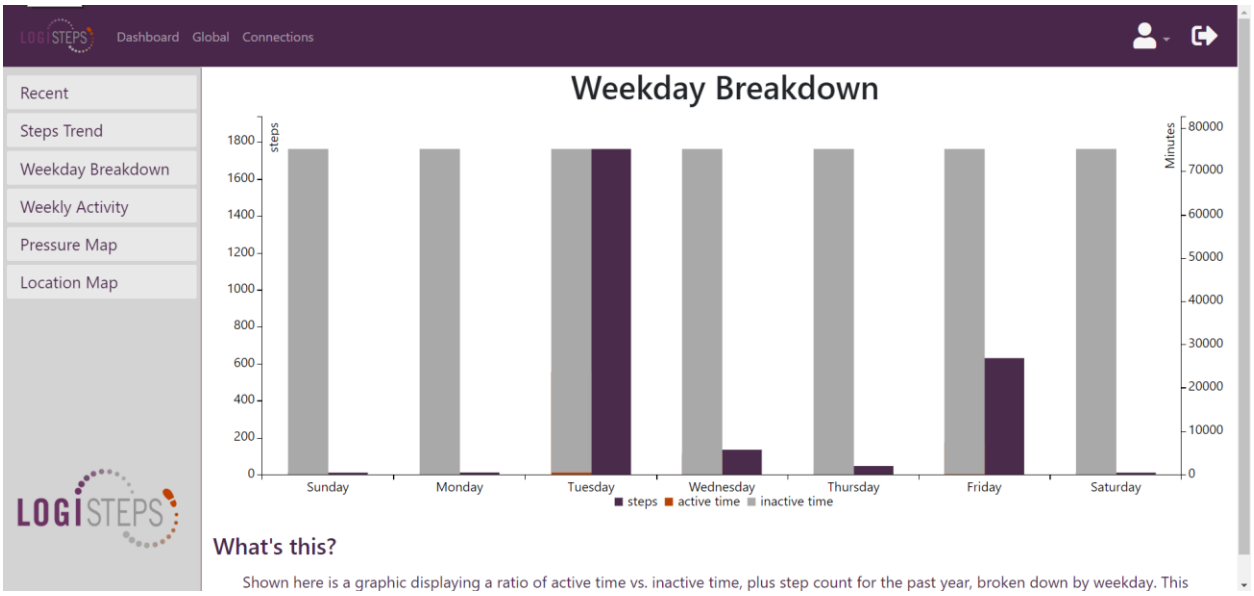

Figure 57 - Weekday Breakdown graphic.

Weekly Activity

Users can obtain a graphic, shown in figure 58, displaying their progress toward their step goal, as well as their inactive to active ratio for any week by navigating to the weekly activity tab. To change weeks, users can click the calendar icon next to the displayed dates to select a different week.

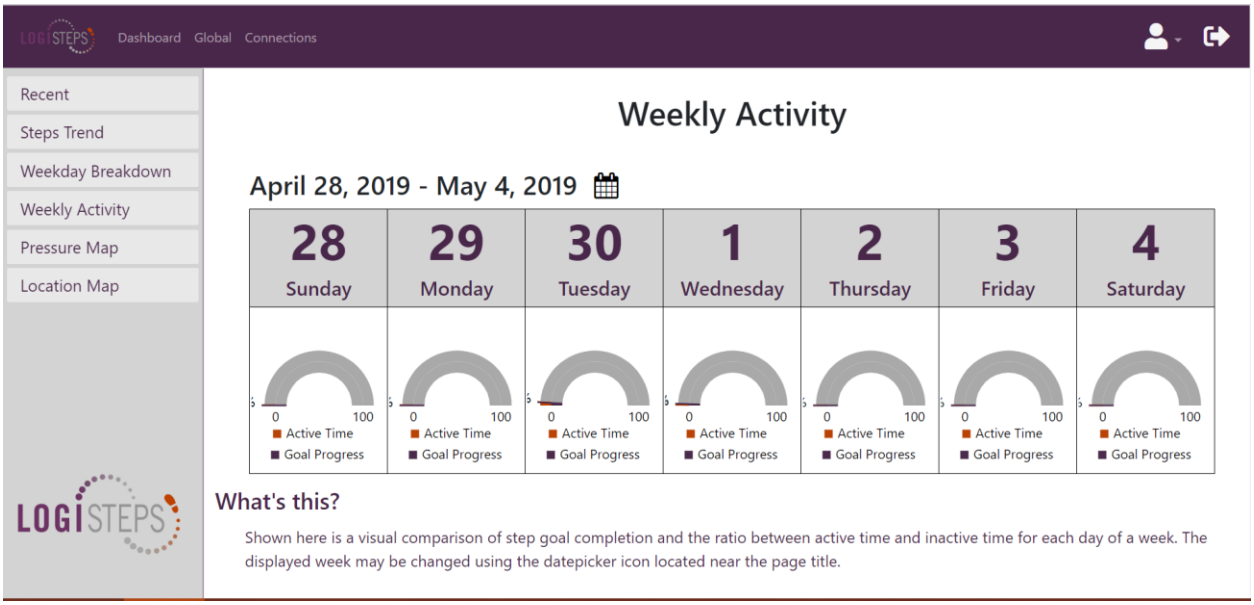

Figure 58 - Goal progress and activity ratio for a given week.

Pressure Breakdown

By navigating to the pressure breakdown tab, users can obtain a gradient map of the pressure they place on their feet, average over the past day, past week, and past month. These graphics can be used to analyze changes in the way a user places pressure on their feet. Applicable scenario as attempting to improve posture, changing running form, etc.

Location Map

By navigating to the location map, users can obtain a graphical representation of their geographic movement for the current day. This works by overlaying GPS points connected via lines onto a map provided by the Google Maps API.

10. Project Retrospective

10.1 Technical Issues

Throughout the course of the project, the team ran into several technical issues that proved be significantly difficult. These were issues that took several days, or even several weeks to figure out. Since much of the technology used in this project was new to the team, the team turned to outside sources to research possible approaches to handling the issues. One of the very first issues that the team encountered was learning how to use the Django web framework. Development on the web server began at the beginning of the winter quarter, and no member of the team had ever done any development using the Django web framework. In retrospect, the Django framework made is easy to get the web server up and running, but the framework was often inflexible, and it was difficult to implement certain algorithms or patterns without performing a great amount of research on public forums.

Another aspect of the project that began in the early part of the winter quarter was prototyping models for the insole, and to do this, the team purchased a 3D printer. Since no one on the team had any 3D printing experience, the team had to teach itself how to use 3D printing CAD software, and the process for deploying the design files to the 3D printing technology. This involved a lot of trial and error, and at the beginning of this process, the 3D printer didn't work. It took a significant amount of time to troubleshoot the 3D printer and get it up and running.

The team also ran into significant issues when it began development using the Sparkfun development board. The microcontroller used by the Sparkfun board advertised its great low power capabilities, but it turned out that the Sparkfun board did not provide the necessary circuitry to take advantage of the low power characteristics of the Nordic microcontroller. To resolve this issue, the team was forced to purchase new breakout boards that allowed the team to take advantage of the lower power capabilities of the Nordic board.

Once work began on the Android application, the team ran into new technical issues completing the communication channel between the microcontroller and the phone. The team had made the false assumption that implementing Bluetooth Low Energy communication would be easy. Since most communication protocols had libraries implemented for easy communication, the team thought that it could use an easy-to-use BLE library. This did not turn out to be the case, and it took several weeks of research and experimentation to send the first byte of data from the microcontroller firmware to the Android application. The team ended up using a Nordic Bluetooth Manager and used an example application as reference when implementing the LogiSteps system.

Further, the application makes use of libraries such as Retrofit and OkHTTP to perform network communication in the Android application. To send the data, an instance of each library must be used, but the issue was those libraries were needed in multiple different parts of the code. Instantiating a new instance for every request would have been really inefficient, so the team decided to use some kind of dependency injection technology. The team settled on a technology called Dagger2. This helped solve the inefficiency, but the library was new to the team, and proved difficult to use. Aspects of the library were difficult to use, and it led to several hours of lost time during the development of the project. The team read countless tutorials, and eventually, the library was used for its intended purposes.

Although the team encountered several issues while designing and implementing the project, each issue was eventually resolved, and the team learned a lot in the process of solving the issues. The encountered technical issues led to a better understanding of the system as a whole, and helped the team understand the libraries and tools that were used in the development of the project.

10.2 Skills Learned

For the insole portion of our project, the shoe insole itself was 3D printed. This required us to learn more about 3D printing in general. We were able to purchase a personal 3D printer and learn how to use it. More specifically, software had to be used to create 3D models. Throughout the project, the insole design continued to be more optimized by creating better designs and using different materials. Aside from the insole itself, the power circuit inside the insole had to be optimized for power. Therefore, a lot was learned about how to optimize an electrical circuit in general.

For the embedded code on the microcontroller, there was a lot of experience gained with using a large, open sourced C library. In addition, there was experience gained in reading and understanding code when using the Nordic libraries and example code. The most experience gained from embedded coding was with the Bluetooth low energy module and energy management. Interfacing between the microcontroller and the mobile app through Bluetooth was an important skill learned from the project.

For the mobile application, a lot of experience was gained with Android development. Primarily, a lot of work was done to allow for a Bluetooth connection between the microcontroller and mobile app for data transmission. In addition, experience was gained with using Dagger2 to allow for dependency injection in Android.

For the web application, a lot of experience was gained with front-end development in general. In particular, Django was used and learned to create an efficient web application. In terms of coding, experience was gained with using Python.

Overall, a lot of experience was gained with project planning and managing. In previous courses, projects would last only a week or two, so it was interesting to have a nearly yearlong project to work on and plan out. Our team was able to plan out milestones with due dates and stay on schedule. We also gained experience with meeting with an advisor and giving progress reports and project updates.

10.3 Project Planning and Evolution

Throughout the project process, every major component of LogiSteps underwent distinct design choice alterations, some major and some minor. As is mentioned elsewhere the major components of LogiSteps are: the insole hardware, the Bluetooth enabled microcontroller application, the android mobile application, and the cloud-based web service.

Hardware

The insole hardware for the most part underwent a gradual evolution of improvement but did include a major design change. At the start of the project the insole was intended to be a 3-D printed object, in the shape of a standard shoe insole, in which there would a few piezo-electric discs, through which energy would be gathered, and the force of the footsteps measured. In the center of the insole would be the energy harvesting circuity, as well the Bluetooth capable microcontroller. As with all mechanical objects the insole went through many small design improvements over time, each iteration simply being

newly 3-D printed. This included changes such as altering the infill to improve comfort and altering the piezo-electric holder to improve its ability to gather energy. The one major change the insole hardware underwent was the decision to not use the piezo-electric discs as force sensors, but instead to place resistive-force sensors with the piezo-electric discs in order to gain a better force reading.

Microcontroller

From the beginning LogiSteps decided that the microcontroller to be used would one of Nordic Semiconductors Bluetooth capable nRF line of processors based off of the ARM M4. Specifically, the nRF52832 was chosen. Initially it had been decided that LogiSteps would use the SparkFun nRF52832 breakout board, and program it is using the Arduino ecosystem and BLE libraries. However, there would come to be two major changes to this plan. It was quickly discovered that the Arduino ecosystem, and the BLE libraries available to program the nRF52832 would not be sufficient to meet the demands of the system. The largest reasons for that being the lack of ability to control the power consumption in the Arduino libraries, and the lack of support for other components such as the RTC, and ADC. So, the project switched to using the Nordic recommended SEGGER development environment and utilizing Nordics nRF SDK. The second major change came with the realization that there were fatal flaws in the design of SparkFun's breakout board concerning power consumption. A replacement breakout board was found in the Espruino MDBT42Q breakout board, which was both smaller, and fixed the power consumption issues.

Mobile Application

At the start of the project it was decided that device that would receive Bluetooth data from the insole would be a simple intermediary to the web service with a simple user interface. It was decided that an Android mobile application would be developed, as Android apps: have the largest user base, are easier to develop, and have more support in development. It was initially decided that structure of the application would be of solely LogiSteps design using only standard Android libraries and resources. However, it was realized that Bluetooth development was far more complicated than expected. So the decision was made restructure the mobile application to create a Bluetooth Manager based off of the design created by Nordic for their open source Android Bluetooth applications.

Web Application

Initially, the plan was to use Vue for front end design. In the end, the team did not end up using this technology due to the steep learning curve. The team was already learning an abundance of new technology, and this would have slowed down progression of the project substantially. Rather, client-side user interface was written using vanilla HTML, CSS, and javascript. Additionally, the team settled on using C3.js for providing dynamic charts. Other than this, the team stuck closely to initial plans for web development.

10.4 Project Management

10.4.1 Things That Went Well

In the project, the team had multiple aspects that came together in the right way. They were what drove the team to the finish line in the long run. The fact that the team stuck to the milestones that were created was the biggest driving force in keeping the project moving. There were weeks where members would be doing extra time in order to hit the milestones when a problem arose.

Even though there were the times that hitting a milestone caused extra work, the work was always there. There were no stagnant times in the project that caused any member to not have something to do. There was always something for each member. That fell back on what were set for the milestones. They were designed so that there were multiple things happening at once.

The project had no funding from an outside source, so the members had to fund all the equipment. The project managed to stay at a low cost for materials. That also includes all backup materials as well. The low cost allowed the members to have more time to work on the milestones rather than needing to fund raise.

The last thing that went well with the project was the step algorithm. It was updated multiple times and the final result was an algorithm that misses very few steps. The algorithm is what makes LogiSteps function properly and provide a user with their full fitness profile.

10.4.2 Areas for Improvement

There were times when members of our team would miss meetings with either last minute or no warning at all. This may have been caused by the lack of definitive standards we were follow regarding meeting attendance or punctuality. We also never implemented any repercussions for missing meetings because we didn't know how to punish each other. We should've developed a better understanding of how to handle meeting attendance.

During the start of the project, our communication with each other was far from stellar. We were still in the process of developing our standards of communication and it caused some confusion as to what duties people had. After we finalized our medium for communication and developed a solid understanding of duties, communication wasn't much of a problem.

Our project was full stack, from hardware to database storage, which allows for a lot of room for bug to creep up. We developed the project in parts so that they could be worked on individually, and we met some resistance when putting parts together. For example, the power harvesting method and microcontroller worked well in theory and mathematics, but when implemented in the insole together caused large problems. This may have been solved with more rigorous testing methods with a more well-defined interface between the power harvesting circuit and microcontroller.

Another issue that may have played a part was the lack of code review. Due to the compartmentalized nature of the project development, we generally worked on our parts separately. This resulted in little to no code review by each other and it was impossible for any one member of the group to have a solid understanding of the full stack operation. This made problems somewhat difficult to solve collaboratively and had to generally be tackled individually.

11. Appendices

11.1 API Documentation

11.1.1 Overview

This document describes high level documentation for accessing and manipulating data for the LogiSteps application. All permanent data is stored and accessed via the LogiSteps backend webserver, which makes reads and writes to the LogiSteps database. Database schema details can be found in other documentation.

11.1.2 Current Version

This the first version of the LogiSteps rest API. Updates to the rest API, including modifications, additions, and deletions will be documented in subsequent versions. The latest version of this documents can be determined by accessing the following file:

https://GitHub.com/SeniorDesignTeamOmicron/Documentation/blob/master/Project%20Design/web%20application/current/current%20version.txt

11.1.3 Schema

All access to the rest API will be performed using the HTTPS protocol at /api. All data sent to the server and data returned from the server will sent using the JSON format.

```
curl -X GET \
>   http://127.0.0.1:8000/api/user/larsonma/ \
>   -H 'Authorization: Basic bGFyc29ubWE6dGVzdA==' \
>   -H 'cache-control: no-cache'

HTTP/1.1 200 OK
Date: Sat, 12 Jan 2019 22:26:24 GMT
Server: WSGIServer/0.2 CPython/3.7.0
Content-Type: application/json
Vary: Accept, Cookie
Allow: GET, PUT, DELETE, HEAD, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 229
```

All datetime objects are to be sent and received using the ISO-8601 format using UTC time with offsets for time zone differences:

```
2019-01-12T19:02:19+00:00
```

11.1.4 Authentication

Several API endpoints require authentication to both view and manipulate data. In the initial design of the API, only one form of authentication is supported: basic authentication. This will require clients to temporarily store a username password pair and pass them to the API in each request.

Basic Authentication

In order to access protected endpoints, basic authentication must be provided in the header of the request, encoded using base 64.

```
curl -X GET \ http://127.0.0.1:8000/api/user/larsonma/ \
 > -H Authorization: Basic bGFyc29ubWE6dGVzdA==
```

The authorization data is decoded on the server and used to validate against current user profiles.

Failed Authentication

If a client attempts to access a resource which requires authentication, and the client provides incorrect credentials, a `403 Forbidden` response from the server will be returned with a JSON object indicating that the username or password was incorrect.

```
curl -i -X GET \
>   http://127.0.0.1:8000/api/user/larsonma/ \
>   -H 'Authorization: Basic bGFyc29ubWE6dGVzdHQ='

HTTP/1.1 403 Forbidden
Date: Sat, 12 Jan 2019 22:49:19 GMT
Server: WSGIServer/0.2 CPython/3.7.0
Content-Type: application/json
Vary: Accept, Cookie
Allow: GET, PUT, DELETE, HEAD, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 39

{"detail":"Invalid username/password."}
```

Failed Login Limit

There is currently no failed login limit, but this may be implemented in a future version.

11.1.5 Parameters

Some requests to the rest API allow for optional parameters. When sending GET requests, the parameters should be appended to the end of the URL following a '?'. Each parameter should be a key value pair.

```
curl -i http://127.0.0.1:8000/api/steps/steplist/?date=07-04-2018
```

In this GET request, a date parameter of 07-04-2018 is send by appending to the end of the URL.

For POST, PUT, and DELETE requests, parameters should be passed using a JSON object in the request's body.

11.1.6 Client Errors

Processing errors that are the result of a client mistake will always send a 400 Bad Request response. The body of the response will contain details regarding the source of the response.

```
HTTP/1.1 400 Bad Request
Content-Length: 55

[{"location":{"latitude":["This field is required."]}}]
```

11.1.7 Users

This is a readable and writeable API for LogiSteps users. This API can be leveraged to create, update, get, and delete user objects. This API provides clients the ability to register new users, authenticate users, and retrieve/update details.

Create a User

Any client may register a new user using this endpoint, as it requires no authentication. Creating a user creates a new LogiStepsUser instance in the backend database.

New users must have a unique username not present in the database. Attempting to create a user using a username that already exists will result in a 400 Bad Request with a JSON body describing the issue.

```
POST /api/user/
```

Authorization

Not Required

Parameters

Name	Type	Description
user	Object	JSON object containing user details
user.username	String	Unique username for identifying user
user.email	String	Valid email that user can be contacted at

user.first_name	String	First name of the user
user.last_name	String	Last name of the user
user.password	String	Plain text password for user. Will be hashed prior to being stored
right_shoe	Object	JSON object containing details for the user's right shoe
right_shoe.foot	Char	Character representing which foot the shoe belongs to. Valid options are "R" or "L"
right_shoe.size	Float	Size of the user's right foot. Precision should be limited to 1 decimal. Valid sizes are between 4 and 16.
left_shoe	Object	JSON object containing details for the user's left shoe
left_shoe.foot	Char	Character representing which foot the shoe belongs to. Valid options are "R" or "L"
left_shoe.size	Float	Size of the user's left foot. Precision should be limited to 1 decimal. Valid sizes are between 4 and 16.
height	Integer	Height of the user in inches
Weight	Integer	Weight of the user in pounds
step_goal	Integer	User's daily step goal. Must be greater than or equal to 0.

Response

```
Status: 201 Created

{
    "user": {
        "id": 5,
        "username": "larsonm22",
        "email": "test@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson"
    },
    "left_shoe": {
        "size": "8.5",
        "foot": "L"
    },
    "right_shoe": {
        "size": "8.5",
        "foot": "R"
    },
    "height": 67,
    "weight": 165,
    "step_goal": 10000
```

```
}
```

Get a User

This endpoint allows a client to retrieve a user object. This may be useful for loading user profile details or verifying user credentials at login. If valid credentials are provided, the entire user object, except the password, is returned in the response.

GET /api/user/<username>/

Authorization

Required

Parameters

None

Response

```
Status: 200 OK

{
    "user": {
        "id": 5,
        "username": "larsonm22",
        "email": "test@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson"
    },
    "left_shoe": {
        "size": "8.5",
        "foot": "L"
    },
    "right_shoe": {
        "size": "8.5",
        "foot": "R"
    },
    "height": 67,
    "weight": 165,
    "step_goal": 10000
}
```

Get List of All Users

Allows a client to obtain a list of all user profiles in the LogiSteps database. This endpoint is intended for administrative purposes and requires special privileges for access.

GET /api/users/

Authorization

Requires admin privileges. The client must be a user registered with is staff set to true. It is not currently possible to create a user with such privileges using the API. Users with this privilege must be created using the command line interface.

Parameters

None

Response

```
Status: 200 OK

[{
    "user": {
        "id": 3,
        "username": "larsonma",
        "email": "larsonma@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson"
    },
    "left_shoe": {
        "size": "9.5",
        "foot": "L"
    },
    "right_shoe": {
        "size": "9.5",
        "foot": "R"
    },
    "height": 67,
    "weight": 185,
    "step_goal": 49
},
…,
]
```

Update a User

Allows a client to update a user's profile. This is useful in the case that a user may want to update their step goal, change their weight, or even change shoe details.

Note: to update a user, the entire user object must be sent to the server. PATCH is not currently supported.

```
PUT /api/user/<username>/
```

Authentication

Required

Parameters

Name	Type	Description
user	Object	JSON object containing user details
user.email	String	Valid email that user can be contacted at
user.first_name	String	First name of the user
user.last_name	String	Last name of the user
user.password	String	Plain text password for user. Will be hashed prior to being stored
right_shoe	Object	JSON object containing details for the user's right shoe
right_shoe.foot	Char	Character representing which foot the shoe belongs to. Valid options are "R" or "L"
right_shoe.size	Float	Size of the user's right foot. Precision should be limited to 1 decimal. Valid sizes are between 4 and 16.
left_shoe	Object	JSON object containing details for the user's left shoe
left_shoe.foot	Char	Character representing which foot the shoe belongs to. Valid options are "R" or "L"
left_shoe.size	Float	Size of the user's left foot. Precision should be limited to 1 decimal. Valid sizes are between 4 and 16.
height	Integer	Height of the user in inches
Weight	Integer	Weight of the user in pounds
step_goal	Integer	User's daily step goal. Must be greater than or equal to 0.

Response

```
Status: 200 OK
```

```
{
    "user": {
        "id": 3,
        "username": "larsonma",
        "email": "larsonma@aol.com",
        "first_name": "Mitchell",
        "last_name": "Larson"
    },
    "left_shoe": {
        "size": "9.5",
        "foot": "L"
    },
    "right_shoe": {
        "size": "9.5",
        "foot": "R"
    },
    "height": 67,
    "weight": 185,
    "step_goal": 49
}
```


Delete a User

Allows a user to be deleted from the LogiSteps database. While this will delete a user's profile from the application, this will not delete a user's step data. Their step data will remain in the database with an anonymous user.

```
DELETE /api/user/<username>/
```

Authentication

Required

Parameters

None

Response

```
Status: 204 No Content
```

11.1.8 Steps

This is a readable and writeable API for creating and reading step data pertaining to a user. This API also provides a method for obtaining summary statistics regarding a user's step data.

Post Step Data

Allows a client to post step data for a user. This endpoint is designed to allow multiple instances of step data to be posted in a single request, reducing the amount of overhead needed to post multiple samples. This endpoint will determine which user to save the step data for based on the authorization header of the request.

```
POST /api/steps/
```

Authorization

Required

Parameters

Parameters detailed below should be provided as a JSON object

Name	Type	Description
	Array	Unnamed JSON array of step data instances
	Object	Unnamed JSON object representing step data
datetime	String	String representation of an ISO-8601 datetime object with timezone offset.
sensor_readings	Array	JSON array containing sensor reading objects
	Object	Unammed JSON object representing a sensor reading
sensor_reading.location	Char	Character representing the location of the sensor on the shoe. "T" for top, "B" for bottom.
sensor_reading.pressure	Float	Pressure recorded for the step
sensor_reading.shoe	String	Indication of which shoe the step was taken. "right" for the right shoe, "left" for the left shoe.
location	Object	JSON object containing details for the user's location when the step was taken.
location.latitude	Float	Latitude of user's location
location.longitude	Float	Longitude of user's location

Response

```
Status: 201 Created

[
    {
        "datetime": "2019-03-28T21:59:55+0000",
        "shoe": "left",
        "sensor_readings": [
            {
                "location": "B",
                "pressure": 28.0183
            }
        ],
        "location": {
            "latitude": 178.92323,
            "longitude": -9.040912
        }
    },
    {
        "datetime": "2019-03-28T21:59:55+0000",
        "shoe": "right",
        "sensor_readings": [
            {
                "location": "B",
                "pressure": 30.0183
            },
            {
                "location": "T",
                "pressure": 31.0183
            }
        ],
        "location": {
            "latitude": 178.92323,
            "longitude": -9.040912
        }
    }
]
```

Get Step Data for a Day

Allows a client to retrieve step data for a particular day.

```
GET /api/steps/stepslist/?date=mm-dd-yyyy
```

Authorization

Required

Parameters

An optional date parameter can be appended to the end of the URL to specify which date the client would like to receive step data for. If no parameter is provided, the default behavior will return the step data for the current day.

Name	Type	Description
[date]	String	The day that should be used to filter steps. Required format is mm-dd-yyyy

Response

```
Status: 200 OK

[
    {
        "datetime": "2019-04-30T01:26:20-05:00",
        "sensor_readings": [
            {
                "pressure": 33.3683,
                "location": "T"
            }
        ],
        "location": {
            "id": 715,
            "latitude": 70.346833,
            "longitude": 151.618907
        }
    }
]
```

Get Step Summary

Allows a client of obtain a summary of steps statistics for a given day. Clients can specify the date in which they would like a step summary by appending a date parameter to the end of the URL.

```
GET /api/steps/summary/?date=mm-dd-yyyy
```

Authentication

Required

Parameters

An optional date parameter can be appended to the end of the URL to specify which date the client would like to receive step data for. If no parameter is provided, the default behavior will return the step data for the current day.

Name	Type	Description
[date]	String	The day that should be used to filter steps. Required format is mm-dd-yyyy

Response

```
Status: 200 OK

{
    "steps": 33,
    "goal": 49,
    "percent": 67.3469387755102,
    "least_active": {
        "hour": 2,
        "steps": 1
    },
    "most_active": {
        "hour": 0,
        "steps": 5
    },
    "inactive_time": {
        "hours": 23,
        "minutes": 29
    },
    "steps_per_hour": 1.375
}
```

Get Step Count(s)

Allows a client to obtain the total steps taken by a user for a range of time. This will only return the total amount of steps taken by a user for each day in a range of days and will not return the instances of the step data.

GET /api/steps/count/?start=mm-dd-yyyy;end=mm-dd-yyyy

Authentication

Required

Parameters

This endpoint accepts two optional parameters. When using this endpoint, either the parameters should both be omitted, or both included; a client should not supply only one of the two optional parameters. The end date should not be in the future. There should also be no more than 365 days between the start and end date. If no parameters are provided, the endpoint will return the step count for the current day.

Additionally, the start date must occur before the end date.

Name	Type	Description
[start]	String	The start day that should be used to filter steps. Required format is mm-dd-yyyy
[end]	String	The end day that should be used to filter steps. Required format is mm-dd-yyyy

Response

```
Status: 200 OK

{
    "range": {
        "start": "10-22-2018",
        "end": "10-26-2018"
    },
    "counts": {
        "10-22-2018": 9435,
        "10-23-2018": 2345,
        "10-24-2018": 5555,
        "10-25-2018": 6483,
        "10-26-2018": 5093
    }
}
```

Get Breakdown

Allows a client to obtain a breakdown of a user's activity for the current year. The client can request the data to be grouped by week, or by month. Currently, this will return step count grouped by the granularity specified and active time vs. inactive time grouped by the granularity.

GET /api/steps/breakdown/?groupby=weekly

Authorization

Required

Parameters

An optional group by parameter should be provided. If this is not specified, it defaults to weekly. This indicated to the server how the data should be grouped by in its query.

Name	Type	Description
[groupby]	String	Groupby parameter. Weekly or Monthly.

Response

```
Status: 200 OK

[
    {
        "weekday": 2,
        "count": 9,
        "inactive_minutes": 75076.71428571429,
        "active_minutes": 9
    },
    {
        "weekday": 6,
        "count": 628,
        "inactive_minutes": 74913.71428571429,
        "active_minutes": 172
    },
    {
        "weekday": 5,
        "count": 44,
        "inactive_minutes": 75064.71428571429,
        "active_minutes": 21
    },
    {
        "weekday": 4,
        "count": 31,
        "inactive_minutes": 75056.71428571429,
        "active_minutes": 29
    },
```

```
    {
        "weekday": 1,
        "count": 8,
        "inactive_minutes": 75077.71428571429,
        "active_minutes": 8
    },
    {
        "weekday": 3,
        "count": 1579,
        "inactive_minutes": 74663.71428571429,
        "active_minutes": 422
    },
    {
        "weekday": 7,
        "count": 8,
        "inactive_minutes": 75077.71428571429,
        "active_minutes": 8
    }
]
```


Get Pressure Snapshot

Allows a client to obtain data aimed at describing changes in pressure a user places on certain points of the foot, over the course of the past month, week, and day.

LogiSteps only currently supports two sensors per foot currently, but this endpoint is designed to account for future expansion of sensor count. Data returned is an aggregate average.

GET /api/steps/pressure/?date=mm-dd-yyyy

Authorization

Required

Parameters

The only parameter is an optional date parameter. If not present, the date defaults to the current day.

Name	Type	Description
[date]	String	Day to use when calculating average over the period of time. For example, a date of 07-22-2018 would return the

		average pressure for the week of 07-16-2018. Required format is mm-dd-yyyy

Response

```
Status: 200 OK

{
    "query_date": "04-30-2019",
    "pressure": {
        "past_day": {
            "left_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 0
                },
                {
                    "location": "B",
                    "avg_pressure": 0
                }
            ],
            "right_shoe": [
                {
                    "location": "B",
                    "avg_pressure": 31.1591
                },
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                }
            ]
        },
        "past_week": {
            "left_shoe": [
                {
                    "location": "B",
                    "avg_pressure": 96.4787878787879
                },
                {
                    "location": "T",
                    "avg_pressure": 96.7030303030303
                }
            ],
            "right_shoe": [
                {
```

```
                    "location": "B",
                    "avg_pressure": 31.1591
                },
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                }
            ]
        },
        "past_month": {
            "left_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 8.30329144225015
                },
                {
                    "location": "B",
                    "avg_pressure": 96.4787878787879
                }
            ],
            "right_shoe": [
                {
                    "location": "T",
                    "avg_pressure": 33.3683
                },
                {
                    "location": "B",
                    "avg_pressure": 31.1591
                }
            ]
        }
    }
}
```


Get Location Data

This endpoint allows a client to request a user's location data for a single day. Clients should expect an array containing longitude, latitude pairs to be returned. This endpoint will currently not return time data with location data to protect the privacy of users, but this is subject to change in future API revisions.

```
GET /api/steps/location/?date=mm-dd-yyyy
```

Authorization

Required

Parameters

An optional date parameter can be appended to the end of the URL to specify which date the client would like to receive step location data for. If no parameter is provided, the default behavior will return the step location data for the current day.

Name	Type	Description
[date]	String	The day that should be used to filter steps location. Required format is mm-dd-yyyy

Response

```
Status: 200 OK
{
    "query_date": "07-18-2018",
    "locations": [
        {
            "latitude": 178.92323,
            "longitude": -9.040912
        },
        {
            "latitude": 178.92323,
            "longitude": -9.040912
        },
        {
            "latitude": 178.92323,
            "longitude": -9.040912
        },
        {
            "latitude": 178.92323,
            "longitude": -9.040912
        },
        {
            "latitude": 178.92323,
            "longitude": -9.040912
        }
    ]
}
```

11.2 Senior Design Show Poster

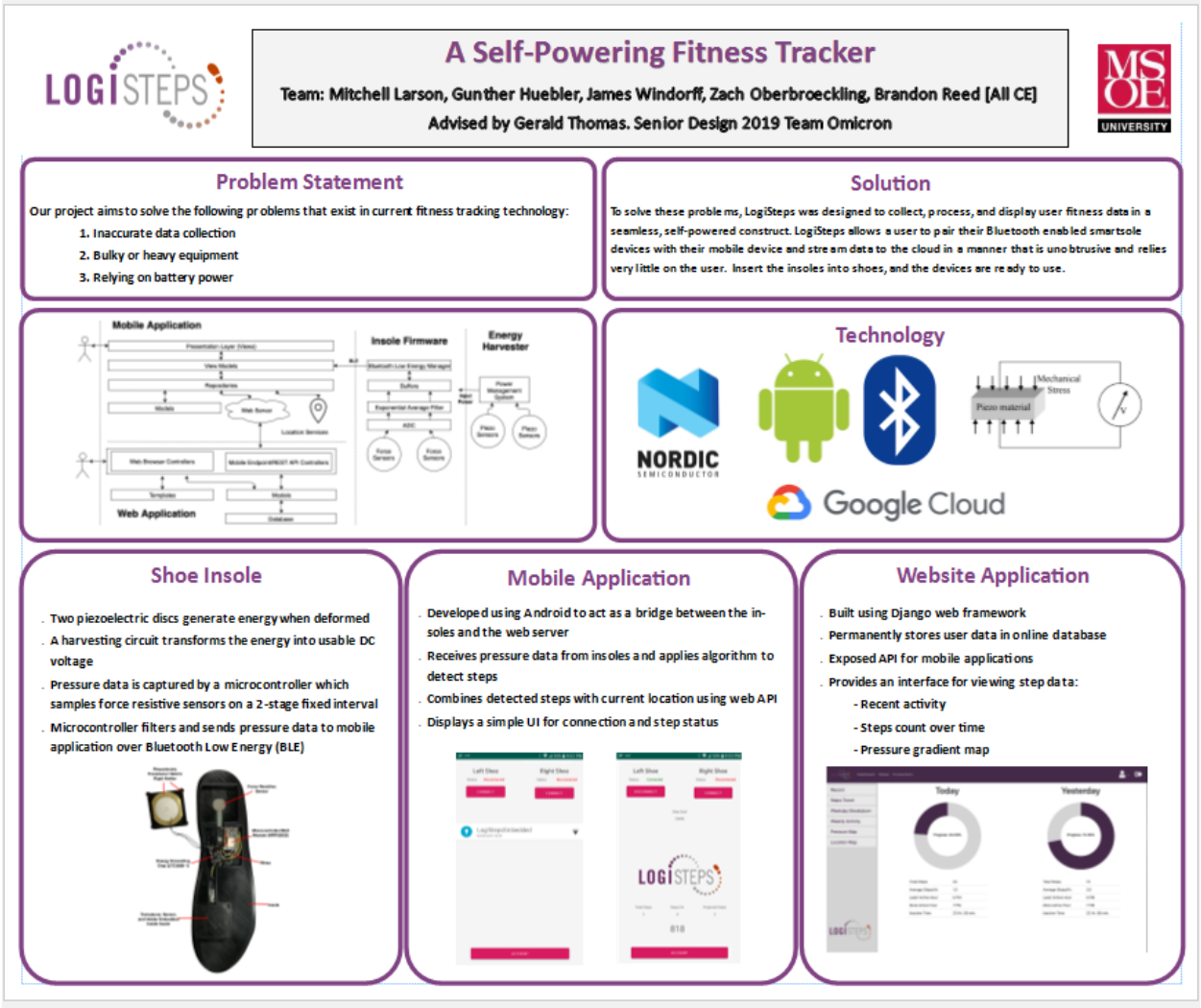

Figure 59 - Senior design poster

11.3 Hardware

The component expenses will be analyzed for all parts considered to not be negligible. Negligible parts include capacitors, wires, resistors, and solder. Component tables will be created with a single insole in mind.

3D Printed Parts

	Print Time	Material Type	# Required
Buzzer Base	55 minutes	PLA (hard)	2
Decompressor	17 minutes	TPU (flexible)	4
Insole	9 hours	TPU (flexible)	1

PLA Print Settings:

- 80mm/s
- 210*C Nozzle
- 80*C Bed
- Retraction Enabled
- Support above 80*

TPU Print Settings:

- 40mm/s
- 225*C Nozzle
- 70*C Bed
- No retraction
- No support
- Infill 20% triangles

Third Party Manufactured Purchased Parts

	Manufacturer	Model Name/#	Cost per Item	# Required
Buzzer	PUI Audio	AB4113B-LW100-R	$2.28	2
Energy Harvester	Linear Technology	LTC3588-1	$6.00	1
Force Sensor	Adafruit	Interlink 402	$7.00	2
Silicone Rubber	Silicone Depot	Pro Grade RTV	$3.49	1

11.4 Glossary

AD converter – A circuit that translates an analog signal into a digital binary format.

Android – An open-source operating system for mobile devices.

ARM assembly – low level language used for writing software on ARM architectures.

Bash – Unix shell and command language.

BLE – Bluetooth Low Energy

Bluetooth – a standard for the short-range wireless interconnection of mobile phones, computers, and other electronic devices.

C – General purpose programming language ideal for development on embedded systems.

C++ – High level object-oriented programming language that extends C.

Cadence – the total number of steps you take per minute.

Embedded Systems – A combination of hardware and software designed for a specific application, often small in size.

e-textiles – fabrics that enable digital components and electronics to be embedded in them.

Fortran – high level programming language used for scientific computation.

I2C – synchronous, multi-master, multi-slave serial computer bus protocol.

Jade – Template engine for generating powerful HTML user interfaces.

Java – High level, object-oriented programming language with high portability using the Java Virtual Machine

knockoutJS – A JavaScript library to help create responsive user interfaces with underlying data models.

LESS – Extension of cascading style sheets.

MATLAB – programming language used for complex numerical analysis.

Microcontroller – A small computer or integrated circuit that contains memory and input/output peripherals.

Noise – a summation of unwanted or disturbing energy from natural and sometimes man-made sources.

Piezoelectric – electricity or electric polarity due to pressure especially in a crystalline substance.

Python – high-level general-purpose programming language.

Server – a computer or computer program that manages access to a centralized resource or service in a network.

Spice – software for simulation of electronic circuits.

USART – Universal Synchronous/Asynchronous Receiver/Transmitter

VHDL – Hardware descriptive language.

Wearables – an item that can be worn.

Web application – an application that is stored on a remote server and served to a user over the internet in a web browser.

11.5 References

Amazon. (2018). *Amazon EC2 Pricing.* Retrieved from https://aws.amazon.com/ec2/pricing/on-demand/

Amazon. (2018). *AWS Free Tier.* Retrieved from https://aws.amazon.com/free/?awsf.Free%20Tier%20Types=categories%23featured

American Chiropractic Association. (2018). *Back Pain Facts and Statistics.* Retrieved from https://www.acatoday.org/Patients/Health-Wellness-Information/Back-Pain-Facts-and-Statistics

Android. (2018). *Bluetooth overview.* Retrieved from https://developer.android.com/guide/topics/connectivity/bluetooth

Android. (2018). *Connect to the network.* Retrieved from https://developer.android.com/training/basics/network-ops/connecting

Angular. (2018). *What is Angular?* Retrieved from https://angular.io/docs

Boshell, B. (2017). *Average App File Size: Data for Android and iOS Mobile Apps.* Retrieved from https://sweetpricing.com/blog/2017/02/average-app-file-size/

Django. (2018). *Django Homepage.* Retrieved from https://www.djangoproject.com/

Dusheck, J. (2017). *Fitness trackers accurately measure heart rate but not calories burned.* Stanford's departments of Medicine. Stanford's departments of Medicine,. Retrieved from https://med.stanford.edu/news/all-news/2017/05/fitness-trackers-accurately-measure-heart-rate-but-not-calories-burned.html

Eclipse. (2018). *Eclipse Californium (Cf) CoAP Framework.* Retrieved from https://projects.eclipse.org/projects/technology.californium

Eclipse. (2018). *Eclipse Mosquitto™.* Retrieved from https://mosquitto.org/

eSun. (2018). *eSUN eLastic TPE 85A 1.75mm Flexible 3D Printer Filament Natural 1kg (2.2lbs) Spool, Natural White.* Amazon. Retrieved from https://www.amazon.com/eSUN-eLastic-Flexible-Printer-Filament/dp/B01A4WP4AY/ref=sr_1_1_sspa?ie=UTF8&qid=1538631121&sr=8-1-spons&keywords=esun+elastic&psc=1

Express. (2018). *Express Homepage.* Retrieved from https://expressjs.com/

Flask. (2018). *Welcome to Flask.* Retrieved from http://flask.pocoo.org/docs/1.0/

Google. (2018). *Google Cloud Platform Free Tier.* Retrieved from https://cloud.google.com/free/

Google. (2018). *Google Compute Engine Pricing.* Retrieved from https://cloud.google.com/compute/pricing

Google. (2018). *Using the Google HTTP Client Library for Java on Android.* Retrieved from https://developers.google.com/api-client-library/java/google-http-java-client/android

Hannah, J. (2017). *Which JavaScript Frameworks Are the Fastest?* Javascript Report. Retrieved from https://javascriptreport.com/js-frameworks-fastest/

Heroku. (2018). *Simple, flexible pricing.* Retrieved from https://www.heroku.com/pricing

Husted, H. M., & Llewellyn, T. L. (2017). *The Accuracy of Pedometers in Measuring Walking Steps on a Treadmill in College Students.* Nebraska Wesleyan University, Health and Human Performance. Lincoln: NCBI. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5214549/

IBM. (2018). *Introduction to MQTT.* Retrieved from https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc00000_.htm

IETF. (2014). *The Constrained Application Protocol (CoAP).* Retrieved from https://tools.ietf.org/html/rfc7252

Instruments, T. (2019). *2.4-GHz Bluetooth low energy System-on-Chip.* Texas Instruments. Retrieved from http://www.ti.com/lit/ds/symlink/cc2540.pdf

Kiefer, R. (2017). *TimescaleDB vs. Postgres for time-series: 20x higher inserts, 2000x faster deletes, 1.2x-14,000x faster queries.* Timescale. Retrieved from https://blog.timescale.com/timescaledb-vs-6a696248104e

Kiefer, R. (2018). *How to store time-series data in MongoDB, and why that's a bad idea.* Timescale. Retrieved from https://blog.timescale.com/how-to-store-time-series-data-mongodb-vs-timescaledb-postgresql-a73939734016

Kleine, O. (2018). *CoAP Client for Android based on nCoAP.* Github. Retrieved from https://github.com/okleine/spitfirefox

Lamkin, P. (2017). *Wearable Tech Market To Double By 2021.* Forbes. Retrieved from https://www.forbes.com/sites/paullamkin/2017/06/22/wearable-tech-market-to-double-by-2021/#394b52e9d8f3

Linear Technology. (n.d.). *Nanopower Energy Harvesting Power Supply.* Retrieved from http://www.analog.com/media/en/technical-documentation/data-sheets/35881fc.pdf

Lo, K. (2012). *Download Speeds: What do 2G, 3G and 4G actually mean for you?* Retrieved from https://kenstechtips.com/index.php/download-speeds-2g-3g-and-4g-actual-meaning#How_do_the_download_speeds_compare_on_2G_3G_and_4G

MatterHackers. (2018). *Taulman Black PCTPE - 1.75mm (0.45kg).* Retrieved from https://www.matterhackers.com/store/l/taulman-black-pctpe-filament-175mm/sk/MF1SV5MP?rcode=GAT9HR&gclid=Cj0KCQjw0dHdBRDEARIsAHjZYYDA6VmY7QY3KDw9Un3oK_78S63tapgvDIWmhVeBvXaZuuQeOh6J078aAq30EALw_wcB

MongoDB. (2018). *Advantages Of NoSQL.* Retrieved from https://www.mongodb.com/scale/advantages-of-nosql

MQTT. (2018). *MQTT Homepage.* Retrieved from https://mqtt.org/

Nicholas, S. (2012). *Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android.* Retrieved from http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https

Nordic Semiconductor. (2018). *Online Power Profiler.* Retrieved from https://devzone.nordicsemi.com/power/

PUI Audio, Inc. (2018). *BUZZER ELEMENT PIEZO 1.2KHZ 50MM.* Digi-Key. Retrieved from https://www.digikey.com/product-detail/en/APS4812B-LW100-R/668-1190-ND/1738483

PUI Audio, Inc. (2018). *PIEZO CERAMIC BENDER 30V 1300HZ.* Digi-Key. Retrieved from https://www.digikey.com/product-detail/en/AB4113B-LW100-R/668-1409-ND/4147333

Quora. (2018). *What is the distribution of men's shoe sizes?* Retrieved from https://www.quora.com/What-is-the-distribution-of-mens-shoe-sizes

React. (2018). *React Homepage.* Retrieved from https://reactjs.org/

Schae, J. (2018). *A Real-World Comparison of Front-End Frameworks with Benchmarks (2018 update).* Medium. Retrieved from https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-2018-update-e5760fb4a962

Select USA. (2018). *SOFTWARE AND INFORMATION TECHNOLOGY SPOTLIGHT.* Retrieved from https://www.selectusa.gov/software-and-information-technology-services-industry-united-states

Semiconductor, N. (n.d.). *nRF52832 Product Specification v1.3.* Nordic Semiconductor. Retrieved from https://www.mouser.com/datasheet/2/297/nRF52832_PS_v1.3-1117956.pdf

Shareef, A., Goh, W. L., Gao, Y., & Narasimalu, S. (n.d.). *Synchronous Electric Charge Extraction for Low.* IEEE. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7829690&tag=1

Silicone Solutions. (2018). *SS-8112 Fast Cure Liquid Silicone Rubber - Per Pound - GE RTV 8112 Offset.* Retrieved from http://siliconesolutions.com/ss-8112.html?_vsrefdom=adwords&gclid=Cj0KCQjw0dHdBRDEARIsAHjZYYB4GZ9MGousYpa0Q5rTiDGB3IViZbp3X2PF8dFdV0rzuUL9G7smgIMaAhqfEALw_wcB

Song, H., Huang, X., Jiang, X., & Wang, J. (n.d.). *The synchronized switch harvesting circuit on.* IEEE. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5768727

Song, J., Zhao, G., Li, B., & Wang, J. (2017). *Design optimization of PVDF-based piezoelectric energy harvesters.* Heliyon. Retrieved from https://www.sciencedirect.com/science/article/pii/S240584401731263X#fig0005

Sparkfun. (2018). *SparkFun ESP32 Thing.* Retrieved from https://www.sparkfun.com/products/13907

Sparkfun. (2018). *SparkFun nRF52832 Breakout.* Retrieved from https://www.sparkfun.com/products/13990

Square. (2018). *OkHttp.* Retrieved from https://square.github.io/okhttp/

Statista. (2016). *Percentage of the global population that used a mobile app or fitness tracking device to track their health as of 2016, by age.* Statista. Retrieved from https://www.statista.com/statistics/742448/global-fitness-tracking-and-technology-by-age/

Statista. (2018). *Percentage of the global population that used a mobile app or fitness tracking device to track their health as of 2016, by age.* Retrieved from https://www.statista.com/statistics/742448/global-fitness-tracking-and-technology-by-age/

Statista. (2018). *Wearables.* Statista. Retrieved from https://www.statista.com/outlook/319/109/wearables/united-states

Systems, E. (2016). *ESP32 Datasheet.* Retrieved from https://cdn.sparkfun.com/datasheets/IoT/esp32_datasheet_en.pdf

TE Connectivity Measurement Specialties. (2018). *PIEZO FILM SHEET.* Digi-Key. Retrieved from https://www.digikey.com/product-detail/en/te-connectivity-measurement-specialties/1-1004347-0/223-1320-ND/5277280

TE Connectivity Measurement Specialties. (2018). *SENS PIEZO PLYCBL.* Digi-Key. Retrieved from https://www.digikey.com/product-detail/en/te-connectivity-measurement-specialties/1005801-1/223-1223-ND/4862192

Texas Instruments. (2014). *CC253x System-on-Chip Solution for 2.4-GHz User's Guide.* Retrieved from https://www.ti.com/lit/ug/swru191f/swru191f.pdf?HQS=TI-null-null-mousermode-df-pf-null-wwe&DCM=yes

Verizon. (2013). *4G LTE speeds vs. your home network.* Retrieved from https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/

VueJS. (2018). *Getting Started.* Retrieved from https://012.vuejs.org/guide/

Zhao, J., & You, Z. (2014). *A Shoe-Embedded Piezoelectric Energy Harvester for Wearable Sensors.* NCBI. doi:10.3390/s140712497