1 Design Criteria, Alternatives, and Decisions

The design criteria, alternatives, and decisions for the project were divided into many parts based on the main software and hardware components of the gym reservation system. These main components are described below.

1.1 Sensor Systems

The sensor systems constitute the means by which data is gathered from gym machines as to whether or not they are currently in use.

1.1.1 Design Criteria

The design criteria for the sensors used to gather machine usage information include a few characteristics. One criterion is that the sensors are capable of gathering movement data (like an accelerometer, gyroscope, or magnetometer) because each sensor will be mounted on a moving part of the machine. This data must be acquired reliably, which reflects the design norm of trust. Another criterion is that the sensors be relatively small (less than 5 in^2) so that they do not impede the use of the machine. Long battery life, on the order of years, is also a criterion for the sensors because it becomes inconvenient to replace the sensors for a whole gym. Through the sensors being small and having a long battery life, the design norm of delightful harmony will be incorporated. Additionally, a criterion is the ability of the sensors to operate in a mesh network, since this will decrease the amount of communication the hub has to do and will allow for shorter data transmission to and from each sensor. Lastly, the price of the sensor network must not be too high so that a gym owner could feasibly purchase it. By keeping the price within a reasonable range, the design norm of caring will be incorporated by considering the customer's needs.

1.1.2 Design Alternatives

The sensors chosen for the project must satisfy the design criteria set forth above. In researching different types of sensors, it became apparent that usually only two out of three desired characteristics exist in most sensors. These three main characteristics are the ability to use the sensors in a mesh network, the low cost of the sensors, and the ability to run on battery power for long periods of time. The final choice of sensors possessed all three characteristics, and is one of the main reasons these sensors were chosen for implementation in the design.

Much research has been conducted recently regarding sensors and sensor nets, but there are not many commercial products that are available. Research has been conducted at the University of California-Berkeley [**?**, **?**] and the University of California-Los Angeles [**?**] regarding sensor systems and sensor networks. In addition, much research has been conducted at Stanford regarding the TinyOS operating system that runs on such sensor networks [**?**]. All of this research has demonstrated the ability to construct a sensor network that operates reliably and at very low power, but few companies have advanced this research into a commercial product.

Despite the lack of commercial products, a few places do sell sensor networks. One of them is MEMSIC, which sells wireless sensor networks for a variety of applications, including educational, industrial monitoring, research and development, and location tracking. However, the boards that serve as wireless sensor nodes are a little expensive and are larger than desired for the project. [**?**]

Linear Technology is another company that sells wireless sensor networks. It provides nodes, or motes, at a low price that operate at very low power. However, these motes lack the sensors desired for the project, so additional sensors would need to be bought. [**?**]

Developing or buying an Arduino board with the desired sensors is also an option, but the difficulty is found in finding an Arduino board that is capable of both mesh networking and sensor systems, is relatively cheap, and has long battery life. [**?**] There did not appear to be any public mesh networking libraries available for use that were built for Arduinos.

Bluetooth SensorBugs are also an option. They contain the desired sensors, are very small, and operate at low power. However, like many other sensors, they do not have the ability to form a mesh network, which is a key component of the project. [**?**]

Yet another option for wireless sensors is the CSRmesh development board. This board operates reliably within a wireless mesh network. However, it is expensive and does not include the sensors desired for the project (its main application is for lighting). [**?**]

Another option for sensors are TI SensorTags, which contain all of the desired sensors, operate reliably within a mesh network at low power, and are not too expensive. Because TI SensorTags satisfied all of the design criteria, they were chosen as the best design alternative. A more detailed description about this design alternative may be found in the section below. [**?**]

1.1.3 Design Decisions

As seen in the previous section, each one of the design alternatives lacked one of the three main characteristics desired for the sensors with the exception of the TI SensorTags. As can be seen in **Figure 1** and **Figure 2**, the TI SensorTags possess the desired sensors in addition to many other sensors. These include a 9-axis motion sensor, a temperature sensor, a humidity sensor, an altimeter/pressure sensor, and an ambient light sensor. In addition, the TI SensorTags have the ability to communicate in a mesh network using Bluetooth, ZigBee, or 6LoWPAN. The battery lifetime for these sensors is about one year, and the data collection rate from the sensors can be modified as desired. In addition, the price per SensorTag is $29, which is much lower than some of the other design alternatives. Because the TI SensorTags satisfied the criteria of having the desired sensors, operating reliably in a mesh network at low power, and having a lower cost, they are chosen to be implemented in the design. Each of the other design alternatives lacks one of the desired design criteria, as demonstrated in the decision matrix in **Table 1**. [**?**]

Table 1: Sensor Systems Decision Matrix

Design Factors	Weight	MEMSIC	Linear Tech	Arduino	SensorBugs	CSR Mesh	SensorTag
Movement Sensor	9	8	0	8	9	0	10
Small Size	8	5	7	6	10	7	9
Long Battery Life	8	10	10	5	9	8	8
Mesh Networking	7	10	10	1	0	9	8
Low Price	6	6	7	8	7	4	7
Total		298	248	215	275	207	324

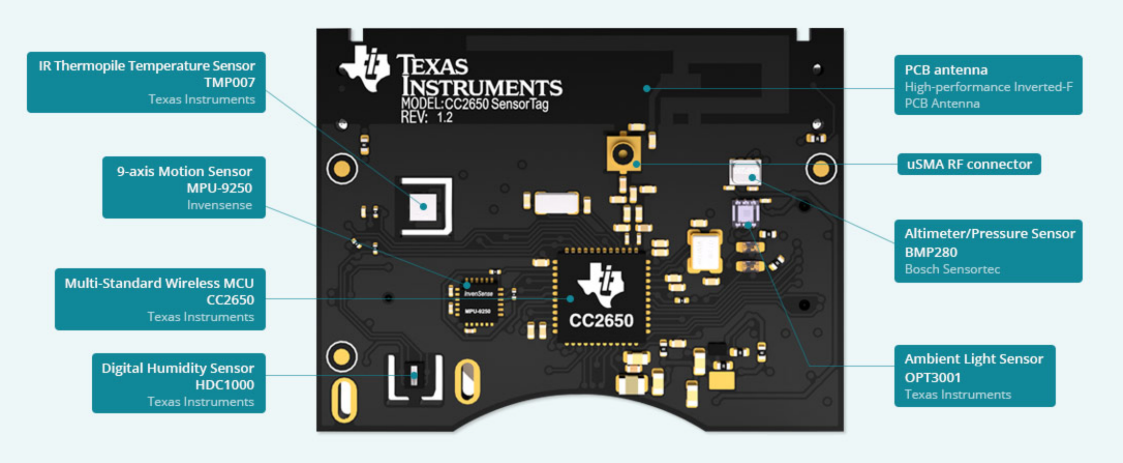

Figure 1: TI SensorTag Sensors and Microcontroller

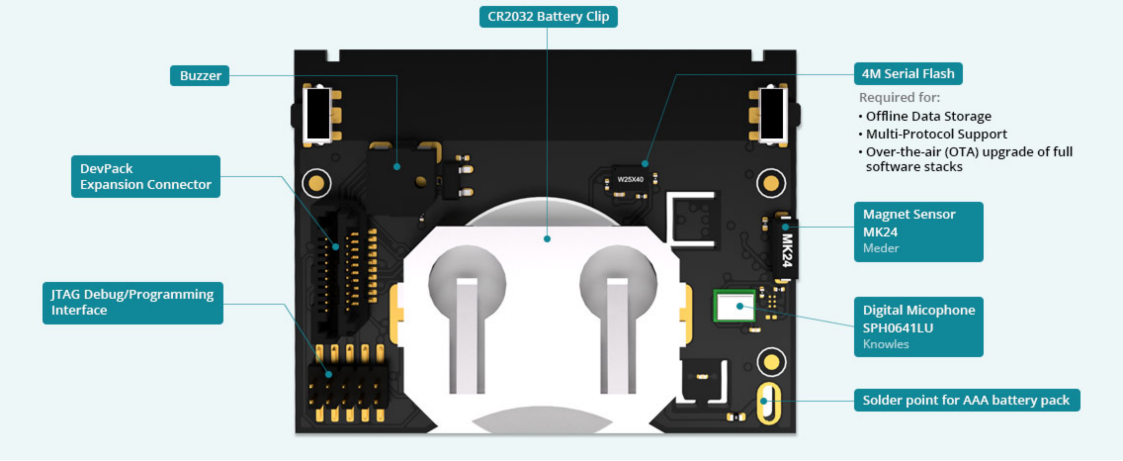

Figure 2: TI SensorTag Sensors and Connections

1.2 Display Interfaces

The display interface allows the user to see current reservation information for a machine.

1.2.1 Design Criteria

Multiple design criteria have been used to select the optimal display interface. Battery life is an important criterion as it reduces the time spent by gym managers replacing or recharging batteries.

The range of a display interface and its ability to connect in a mesh network are also design criteria as they affect how large of a gym in which the system can be installed.

Simplicity is a third criterion as smaller or simpler boards are less obtrusive, promoting delightful harmony. In order to make the transition from a prototype to a real product, boards with available and open source hardware design files are preferred.

Lastly, the display interface constitutes a large portion of the cost of the system. Any variation in the cost in a single unit is multiplied since many display interfaces are purchased for one system. Thus cost is an important design criterion.

1.2.2 Design Alternatives

As a result of these criteria, four devices have been chosen as alternatives for small low-cost boards that could fulfill the design requirements: a Raspberry Pi, an Arduino, a CC3200 Launchpad, and a TI SensorTag.

For the display component of the display interface, the applicable criteria are battery life and price. Due to the fact that the board needs to run on batteries, the display cannot require a significant amount of power. This negates most typical Liquid Crystal Display (LCD) options. The two low power options that were found are E-Ink (also known as E-Paper) and Memory LCD. E-Ink has the advantage of requiring no power when the image is not changing, which would be most of time in this project. Memory LCD, on the other hand, does require some power, but typically under 200 µW, which means a single AA battery would last over 200 days. The original target display size was 2" x 3"; however, the closest available size for either type of display is 2.7" (35mm x 59mm) with similar prices. Each of the four board alternatives can support at least one of these display options.

The Raspberry Pi (shown in **Figure 3**) is a credit card sized computer that runs a full Operating System (OS), such as Linux or Windows 10, with Universal Serial Bus (USB) ports and a High-Definition Multimedia Interface (HDMI) output. Most importantly, it has 40 General Purpose Input/Output (GPIO) pins that can be used to connect the display or other features like buttons and a wireless communication device.

The advantage of this board is that it has the most community developed libraries, including already built libraries for mesh networking and for controlling E-Ink displays, which would make the display interface very easy to build. Breakout boards for E-Ink displays are available for the Raspberry Pi and are well priced, which would make adding the display simpler. This board is also inexpensive; a $20 model would easily fit the requirements.

However, the Raspberry Pi has much more functionality than needed. Combined with fact that it is running an OS, it uses much more power than any of the other options, making it virtually impossible to run off battery. There are some batteries that might work for a few days, but they are expensive and rather large for this board. Along with the fact that it is not an open source board, the complexity of the Raspberry Pi would make it hard to use as an actual product. The price of a display interface with this board would likely be about $90.

The Arduino is an open-source prototyping board that runs a single program. It also has GPIO pins for controlling the display and connecting a wireless communication device.

Some basic Arduino boards are quite low priced, and there is also a big community behind Arduino boards. The fact that an Arduino runs only a single program can make it very power efficient, thus making operating it by battery over several months plausible, even using AA batteries. There is at least one reasonably priced E-Paper breakout board that eases the process of adding a display.

The only mesh network library found was for a radio frequency (RF) radio, though that may not be a disadvantage as it may be more power efficient than some other solution. Also, a lot of libraries use code that does not take power efficiency into account, which limits either the battery life or the libraries available. The price of a display interface with this board would likely be about $60.

The CC3200 chip is a microcontroller with built-in Wi-Fi. The CC3200 Launchpad is a board designed to provide features for this chip such as GPIO pins, USB interface, and power.

Figure 3: The Raspberry PI, a Credit Card Sized Single Board Computer

The primary selling point of this chip is the built-in Wi-Fi which translates to good power efficiency, allowing the board to run very well on batteries. An E-Ink breakout board is available and has available design files. Along with the fact that the design files for the Launchpad are easily accessible, this board would be the easiest to transition from a prototype device to a commercial one. An additional advantage is that the built-in Wi-Fi is FCC approved, unlike many add-on Wi-Fi dongles, which removed a roadblock to commercialization.

However, no mesh network library could be found for the CC3200. The E-Ink breakout board for the CC3200 Launchpad is also quite expensive and large. The price of a display interface with this board would likely be about $80.

The TI SensorTag is a small board made for prototyping IoT devices and is shown in **Figure 1** and **Figure 2**. This board has a built-in communication device that supports ZigBee, a protocol designed for mesh networking, and is made to last extended periods of time on a coin cell battery. It also has a devpack for a "smart watch," which acts as a breakout board for memory LCD displays. The board has open hardware and software files which would facilitate the transition to a commercial product. Since the SensorTag was chosen as the sensor for the system, selecting it would result in only having to use one type of board for both the sensor and the display interface, thus streamlining the system. This might also eliminate the need for a separate sensor, which would reduce cost significantly.

However, the devpack comes with various accessories including a 1.3" memory LCD, while the only desired component is the connector for the memory LCD. The larger memory LCD display would need to be bought separately. The price of a display interface with this board would likely be about $70. This cost could be further reduced if the breakout board for the memory LCD display could be replicated so that the smartwatch devpack would not need to be bought for each display interface.

1.2.3 Design Decisions

While overall the Raspberry Pi is a good board, the difficulty and price of running it by battery and its complexity mean that it has not been chosen for the system. This would be a good solution if the project allowed for a power supply instead of batteries. The Arduino is a good option since it is the least costly and fits all the requirements fairly well, making it the second best choice. The CC3200 Launchpad would be the easiest choice in transitioning to an actual product, but it would be the most difficult to use in getting a prototype working. For this reason, along with the cost, this solution was not chosen for the project. While the price of the SensorTag may not be as low as the Arduino, the battery life and the built-in mesh

networking are huge advantages. The fact that this board can also be used for sensors and is open source make this the top candidate for the project, as shown by the decision matrix in **Table 2**.

Table 2: Decision Matrix for the Display Interface

Design Factors	Weight	Raspberry Pi	Arduino	CC3200 Launchpad	SensorTag
Battery Life	10	2	7	7	10
Network Capabilities	5	8	7	3	10
Simplicity	2	8	10	1	10
Open Hardware	3	8	8	5	9
Cost	10	11	17	12	14
Total		201	321	245	338

1.3 Hub

The Hub is the component through which the data from the display and sensor networks is passed to the server and from the server out to the displays. It is essentially the gateway for the system between the gym and the Server.

1.3.1 Design Criteria

The most important aspect of the Hub is its ability to communicate. Since it can be hidden and plugged in, form factor and battery power are not necessary considerations. This opens up options to using full small computers as long as they have the ability to communicate with the sensors and displays, along with the computing power necessary to serve as the client to the server. In order to minimize the cost of the system, the board needs to be cheap but still have the necessary computing power. For the communication to be as reliable as possible an Ethernet port is necessary on the Hub.

The design norms that can be applied to the Hub component are transparency and trust. Transparency is shown by being open and forward with the gym administrators as to what information the Hub is passing onto the server and how it uses the gym's existing Internet for that process. Trust is applied through the security of the software and the encryption of the data that is passed back and forth between the Hub and the Server. Ensuring that the user's data is secure is important in order for users to feel safe in continuing to use the system.

1.3.2 Design Alternatives

The primary options for the hub were different models of the Raspberry Pi, the Intel Edison board, or possibly some other single board computer. There is an increasing number of single board computers available on the market, and they are getting cheaper and faster. In considering these various boards, one large difference is the size of the community using the boards and the number of devices available that can easily interface with them. Most of the boards would have worked well for this application, as the necessary network communication can be connected via a USB port and the software needed to collect the ZigBee data can run on Linux. Due to the fact that this component can be plugged into permanent power, this opens up the possibility of any computer device running Linux and having a USB port as being a viable option. The reasons for narrowing this field down to simply the Pis and the Edison board is due to their large support, stability, and their low price point.

The Edison board has the advantage of having lower power consumption and the support of a large company. However, it does not have an Ethernet port and is approximately double the price of a Pi, as can be seen in **Table 3**. The Edison board is also intended as more of a sensor board than a small computer, though it could work for this project with the right breakout boards. [**?**]

There are multiple models of the Raspberry Pi boards, each one with improved specs over the previous model, generally in the form of more Random Access Memory (RAM), a faster Central Processing Unit (CPU) and Graphics Processing Unit (GPU), and additional ports. The main difference with the most recent Pi, as compared to previous ones, is its ability to run Windows 10. This is not currently a requirement of this project, though. However, if the sensor's software is made for only Windows, it could be a potentially useful aspect of choosing the newest Pi, the Raspberry Pi 2 Model B. The older versions of the Raspberry Pi should be adequate for this project, though. As long as there is no need for Windows 10, then the computational power differences should not make an impact on this project. For the amount of computing power that Pis provide they are the cheapest single board computer currently available on the market. This is due to their success and large scale production. An additional advantage of this large scale is that Pis have a stable Linux Operating System (OS) developed specifically for them, which should help minimize any bugs in the system and increase the stability. [**?**]

The other aspect of the hub to be considered besides the single board computer is how it will communicate with the sensor and display network. The only connectivity built into the Raspberry Pi is an Ethernet port. This means that whatever wireless technology is used will need to be added through either the GPIO pins or a USB dongle. There are options for Bluetooth dongles [**?**], Wi-Fi dongles [**?**], and ZigBee modules [**?**] that a Pi can use. All of these options enable the Pis to connect to whichever network ends up being used. This is one area where the Edison board has an advantage, as it has Wi-Fi and Bluetooth connectivity built into the main chip and is primarily designed as an IoT board [**?**]. However, since reliability is an important factor, the inclusion of an Ethernet port on the Pis is more important than the Edison Board's capability with Wi-Fi and Bluetooth. This is especially true since the network technology may be something besides Wi-Fi or Bluetooth, which would require an additional dongle to any chosen board, such as ZigBee or 6LoWPAN.

1.3.3 Design Decisions

The final board that was chosen was the Raspberry Pi 2 Model B. This board was chosen because for its price it had the most options. Primarily, it was chosen to accommodate for potential future design changes, such as running Windows 10 on it. This board is available for $35, has more than the needed requirements, and can use the ZigBee module via the GPIO pins as opposed to the USB through a breakout board, saving some money. However, the board that is currently being used as a prototype is a Raspberry Pi Model B+. This is because Calvin already had several older Pi models available for use and the small advantage that the Pi 2 would offer is not necessary unless for some future design change Windows 10 becomes necessary. This change from the initial design decision saved some money in the budget and enables the team, if necessary, to purchase an additional sensor or display later in the project. There is also enough money in the budget if the purchase of a different single board computer becomes necessary.

1.4 Server

The server will handle all connections requesting information about the current status of the machines, along with making and viewing reservations for the equipment. It will also handle sending historical data for administrators and personal historical data for individual users. It takes all the data from the server and stores it, creating these services for the client.

Table 3: Decision Matrix for Hub Hardware

Price		$40	$95	$35
Design Factors	Weight	Pi 2 Model B	Edison Board	Pi 1 Model B+
Network Capabilities	10	5	5	5
Ethernet	8	10	1	10
Range	8	5	5	5
Costs	5	8	1	9
Reliability	10	10	10	9
Total		310	203	305

1.4.1 Design Criteria

The server's primary requirement was that it must be reliable. Both the gym administrators and the gym's clients must be able to trust that when attempting to ascertain information about the machines, whether historical or current information, that the information can be sent to them. The server must be capable of modern networking protocols and functions, including HTTP/HTTPS requests and creating and maintaining TCP network connections. The HTTP requests requirement is to enable the server to send information about the state of the machines over the internet to clients of the system. The TCP network connections are to ensure a reliable connection between the hub and the server for consistent and reliable information sharing. The server must also be powerful enough to encrypt data and send data over encrypted connections to make sure that all client information is protected.

1.4.2 Design Alternatives

There are in essence two main players in the server world: Windows and Linux. Both of these options have the capabilities to meet the requirements stated above in the Design Criteria section, however Linux has several advantages. The first is that it is free, which will reduce the cost of our system. The second is that it is often more reliable.

1.4.3 Design Decisions

It was chosen to use Ubuntu 14.04 as the server. It provides the reliability required to provide the users with a trustworthy experience. It also should be low maintenance once installed, as security and software updates can be handled automatically using built-in Linux functionality. Ubuntu 14.04 is the Long Term Supported (LTS) version of Linux currently and will be supported for several more years. It also has the modern capabilities to use Python and other libraries to create a secure and consistent connection with the Hub. Also because the hub is running Linux, it makes sense to continue to use the same operating system for ease of upkeep. It also aligns with the team's goal of transparency because Linux is an Open Source solution.

1.5 SensorTag Network Type

In order to pass information between SensorTags, a type of network protocol must be used. Each network protocol has various strengths and weaknesses, and some network types are better suited for the requirements and application of this project.

1.5.1 Design Criteria

As listed in the requirements section, each sensor must have network capabilities and have the ability to form a mesh network where the sensors can communicate with and pass data to one another. This communication must be reliable and timely in order to give the user real-time data about machine usage. In addition, the network type used must be capable of communicating over long distances so that information from a SensorTag at the edge of the gym is able to reach the hub, either directly or through a series of other sensors.

1.5.2 Design Alternatives

The design alternatives for the type of network used with the SensorTags was dependent on the kinds of networks that were compatible with the SensorTags and could be flashed onto them. One alternative was Bluetooth, which comes installed on the SensorTags and is easy to use. However, the Bluetooth option could primarily only be used with a mobile application, and Bluetooth does not support mesh networking. Another alternative was ZigBee, which does support mesh networking and can communicate over longer distances. However, very little to no documentation about using ZigBee with SensorTags is provided, and it was not obvious as to how to get ZigBee working properly in this context. The last alternative was 6LoWPAN, which also supports mesh networking and can communicate over longer distances than ZigBee. 6LoWPAN is relatively new, but enough documentation and open source code is provided for it to be implemented on the SensorTags. This open source code is Contiki, an operating system written for small sensor nodes such as a SensorTag.

1.5.3 Design Decisions

Contiki is an open source real time operating system that can run on the SensorTags and utilizes 6LoWPAN wireless networking. Its website describes Contiki as "The Open Source OS for the Internet of Things". The team decided to use Contiki primarily because it was the only way found to use 6LoWPAN networking with the SensorTags and it provides several examples and tutorials on how to use it and get started with it. Contiki is publicly available through a git repository and there are pre-built .bin files that can be flashed to the SensorTags using the debugger for several different examples.

Contiki also offers an easy way to get a development environment up and running with something they call instant Contiki. This is a download from their site which is an Ubuntu image made to run in VMWare and already has all the tools and libraries needed to work with Contiki. There are many more tools for Contiki than what we used, some of which are meant to simulate a larger scale implementation of a sensor network and others made to just help in tweaking the code. The only thing that seemed to be missing from the Contiki tool kit is a good debugger, but this is likely because they'd have to make a different debugger for each board Contiki is available for, which is quite a few. This only impacted the project in that in order to test changes to the Contiki codes, it needed to be flashed to a SensorTag and tested directly.

1.6 Sensor Data Acquisition and Communication

In order to reliably acquire data from various sensors in real time, a few methods existed within the Contiki operating system. Each option had its benefits and downsides, and many different methods and approaches were taken before the final method was selected.

1.6.1 Design Criteria

As outlined in the requirements section, the sensors need to output data in a way that preserves battery life on the order of months or years. In addition, sensor data must be reliable, accurate, and have a clear way to be passed from one sensor to another.

1.6.2 Design Alternatives

Each design alternative was based off of an example for the CC2650 SensorTag provided by the Contiki operating system. The simplest design alternative, which was implemented first, was outputting sensor data over UART (Universal Asynchronous Receiver/Transmitter). The example was modified to output a stream of data where each set of data contained the machine ID number and gyroscope and accelerometer data for each of the coordinate axes. With this example, each SensorTag had to be connected to a Raspberry Pi, but the success of this design alternative showed proof of concept in that a working system could be designed from the bottom to the top (sensors to the user interface).

Many other design alternatives were tried with varying degrees of success. These included modifying the various examples in the Contiki operating system and flashing them onto the SensorTags for testing. Some methods implemented UDP or TCP over 6LoWPAN, others implemented other versions of IPv6, and still others consisted of modifying examples written for other devices, but each example used 6LoWPAN mesh networking to pass data between the SensorTags and the ultimate destination, the Raspberry Pi. Multiple methods were attempted in gathering data from the SensorTags, including a USB dongle that could be configured in multiple ways, such as a packet sniffer or a slip radio.

The last design alternative that was tried was developing and modifying the code for the CC26xx web demo, provided by the Contiki operating system. This web demo supported multiple ways of transferring sensor data between devices and an ultimate destination using the 6LoWPAN mesh network, including network UART, a 6lbr client, a CoAP server, HTTPD, and an IBM Quickstart/MQTT Client. A SensorTag was used as an antenna to receive 6LoWPAN packets for the Raspberry Pi, and this example provided basic functionality in terms of gathering sensor data from each SensorTag.

1.6.3 Design Decisions

The web demo example provided by the Contiki operating system was the final design chosen to be used for the project. A few configuration changes were made to the example to better suit the application of the demo in this specific project, such as turning off certain sensors and modifying the network address prefix. The demo passes reliable sensor data and device ID information from various SensorTags to the Raspberry Pi in real time using mesh networking, which fulfills all of the requirements and design criteria for this part of the project. Once collected, a short Python script implements some signal processing to determine whether or not each machine is in use, and this information is then passed on to the server.

1.7 Display Software and Communication

The SensorTag displays can be utilized by a few different RTOS's or networking methods. The watch devpack display is used with the CC2650 SensorTags from TI.

1.7.1 Design Criteria

The display software and communication needs to be reliable, secure, and quick to update. It needs to also display the information in a way that is easily seen and understood by the user.

1.7.2 Design Alternatives

The display software and communication has two different options. It could work with Contiki using 6LoWPAN, which would be preferable as it would then share a network with the other SensorTags that aren't acting as displays. However, with this path there are no display drivers currently available with Contiki so they would have to be completely re written from TI-RTOS to work on Contiki.

The second option is to use the existing Bluetooth phone app available from TI. With this option the SensorTags would have the default TI-RTOS running on them, and just need an over the air (OTA) update from the app in order for the display to work with them. Once updated there is a textbox in the app, and whatever is entered there is then displayed on the SensorTag. This wouldn't use a mesh network, but would require much less work to get functioning with our database to automatically read the reservation status of the machines.

1.7.3 Design Decisions

Several weeks were spent trying to re write the device drivers to work on Contiki. However, after that time it was decided that the team would switch and add to the Bluetooth app for the demo, as some of the files that the driver's interface with for TI-RTOS weren't available or found for Contiki, primarily one to interface with the device pins (where the display attaches) and one providing power to peripheral devices (needed to power the display). The functionality that needs to be added to the android app is to automatically check the database or website for the reservation status of a specific machine. Using the Bluetooth app for this is primarily for the senior design night demo, for a production scale system we would have a custom built and designed board and display, utilizing a mesh network technology.

1.8 Hub to Server Communication

The server must communicate with the hub to determine the status of the machines and communicate reservations to the displays

1.8.1 Design Criteria

The communication between the hub and the server must be reliable, and fast enough to transfer all the required data.

1.8.2 Design Alternatives

One alternative is for the hub and the server to communicate via TCP sockets. The hub and the server would both setup sockets listening for updates. This has the advantage of being fast, reliable, and relatively easy to implement.

A second alternative is for the hub to use http requests to send updates through the server's http server that delivers the website, and also use those requests to get updates from the server. This has the advantage of

being simple, and streamlining the database access methods on the server. This option is slower than the TCP socket method.

1.8.3 Design Decisions

The http request method was chosen because the database was reporting errors when updated from multiple TCP socket transmissions, which could impede reliability. Because the http requests go through the http server, thus reducing the threads accessing the database to one, the http request method does not present this problem. Any issues that may arise due to the slower speed of the HTTP method can be negated if the data transferred is aggregated before being sent, thus reducing the number of packets sent.

1.9 Database

The database must be able to scale with both large amounts of traffic and large records. The large amount of traffic scalability is required so that when the website has large amounts of visitors, they do not experience long wait times when attempting to view information. In terms of the large records scalability, the database needs to be able to handle queries on large amounts of data quickly so that when managers want to look at historical usage of machines over extended periods of time, they are able to without having to wait inordinate amounts of time between requests.

1.9.1 Design Criteria

The database must be well structured, easy to add features to, and well abstracted.

1.9.2 Design Alternatives

An alternative to the database would have been in memory storage or file storage for historical data. In memory storage, while fast, is not reliable and if the system were to go down or restart, all the data would be lost. While file storage is easy to set up, it is very slow to read through large files and parse them. This would not allow the database to be scalable as required.

1.9.3 Design Decisions

The team decided to use the SQLAlchemy framework in Python for a variety of reasons. The first reason was that our model was already well defined in Python, so to interact with the database in native Python is a large advantage. Also, SQLAlchemy provides an abstraction layer above the database, which allows us to easily switch between different database implementations. This allows us to test many different scenarios, as well as have a production version of the database that can be changed later if necessary.

1.10 Website

The website serves as an interfaces with the end users, including gym users wanting to check what machines are available or reserve machines, and gym administrators wanting to edit the system or view machine statistics. The website contains a front end, which is what the users interact with, and a back end that interacts with the database, provides the dynamic data that needs to be displayed and generates the web pages.

1.10.1 Design Criteria

From a design standpoint, we wanted the website back end to easily be able to interact with both our database and our model. These are both written in Python, so it would follow to use a website that could take advantage of Python.

The website front end needs to look good for the end user so that it will not detract from their experience using our system. We are also assuming that most gym users will be using the website on their phones, so mobile friendliness is also essential.

1.10.2 Design Alternatives

A first alternative for our back end is Flask, which is a micro webdevelopment framework for Python. A major alternative to Flask is Django. While Django is appealing, it has more setup and configuration options that the team was not familiar with how to set up.

There are many front end packages intended to make beautiful and mobile friendly websites. The first alternative was using the same one used for our team website, called future imperfect. The second alternative considered is the bootstrap framework, a popular front-end framework which advertises itself as "the world's most popular HTML, CSS and JS framework for making responsive, mobile first projects on the web."

1.10.3 Design Decisions

Flask was chosen as our back end framework because it was simple to start, easy to use, and provided very good documentation. Additionally, it interacted very well with our Python database, written in SQLAlchemy, and when interacting with the native Python objects that represent machines, reservations, etc.

The bootstrap framework was chosen for our front end because it was more general and allowed more features, whereas the future imperfect package is designed for making website centered around articles and posts.