# RoastRight

The smart coffee roaster

A Senior Design Project by

Brian Webb | Evan Baytan
Patrick Sites | Sammy Roman

# Table of Contents

# 1.0 Executive Summary

RoastRight: The Smart Coffee Roaster is designed to give users the ability to quickly and easily make some superb coffee roasts in the comfort of their home. It combines the power of the Behmor 1600 Plus coffee roaster – The ability to achieve darker roasts and to roast up to a pound of coffee at a time – with the use of modern communications and multiple input, multiple output control systems. The RoastRight roaster not only senses temperature of the roasting chamber, it also senses how the weight of the beans changes over time and is capable of determining where the beans are in the roasting process. These control systems give users the ability to easily and accurately achieve the exact roast they are looking for, every time!

By combining these control systems with internet connectivity for control and monitoring with your iPhone or Apple Watch, the RoastRight coffee roaster brings coffee roasting into the modern era and provides new levels of customization and ease of use to an industry that has seen little innovation in the past two decades. The apps allow you to build and share custom roasting profiles with your friends so that other RoastRight users can easily replicate your results and try the same coffee that you had that very morning. Is your phone dead? Have no fear! RoastRight has a built in touch screen controller to allow you to use it with or without your phone.

Whether you source your beans from Brazil or Thailand; whether they have followed a dry process, wet process, or even a more exotic process such as a honey process; whether you prefer a light roast or a dark roast, the RoastRight coffee roaster can handle it. RoastRight can handle any variety of coffee bean and produce the exact roast that you were looking for, every time. Want to try the world's best cup of coffee? Look no further than RoastRight!

# 2.0 Project Description

As was said above, our project is designed to improve the ease with which people can access freshly roasted coffee. As Dr. Richie strongly suggested, we are retrofitting an existing, commercial coffee roaster in order to give it the ability to have significantly upgraded control systems and to provide it with Wi-Fi connectivity. This process not only involves designing every electrical system in the roaster, it also involves testing existing hardware in order to determine what specifications it has. While the idea of retrofitting has simplified mechanical aspects of the design for us, it also complicated electrical designs. As with anything in engineering, there are tradeoffs here.

## 2.1 Project Motivation & Goals

Many people don't realize that coffee is much like a cookie. There is a noticeable difference between a freshly baked cookie and one that is a month old. The same observation holds for coffee. Most coffee purchased has been roasted and stored for months at a time. Large amounts of the aromatics that give coffee it's rich flavor quickly evaporate and leave a comparatively flavorless, bitter mess behind. Various consumers don't realize the difference in taste and quality they are missing due to the added labor and work that is required to roast their own coffee. RoastRight will provide people a significantly better coffee experience than most coffee houses are able to provide with little effort.

Some may say that they don't have the time to worry about roasting their own coffee and would probably continue to buy roasted coffee off the shelves. The consumer is content with this tradeoff of losing quality in taste for added convenience. However, the solution we are providing will be familiar and simple for the user, in which they will second-guess buying pre-roasted coffee. The reason the coffee roaster will be familiar to the user is primarily due to the implementation of an iPhone and Apple Watch application that will be used to conveniently and easily control the roaster. A coffee roast would only require a few commands in the application to start and keep track of a roast in progress. We believe that the consumer will find this method to be convenient for them to enjoy the coffee that they deserve. Our goal is to provide a convenient and simple solution for new and avid coffee connoisseurs to roast fresh coffee.

## 2.2 Objectives

One of two primary objectives is to design a complete system that provides user's the ability to enjoy freshly roasted coffee with the convenience of controlling the roaster through an iOS/watchOS application. The use of these applications will aid in providing a user interface that is familiar and user friendly. The iOS application is intended to be the primary software interface with the watchOS application being an extension or secondary application interface.

Another primary objective is to design the control system that will be receiving commands from the API in order to wirelessly make adjustments to the roaster. The PCB and other various hardware components will be designed specifically for the purpose of controlling the roaster. The roaster will have an onboard touch screen interface for making changes to controls that will be directly interfacing with the MCU and various components.

We believe that most people are not enjoying coffee to its full quality and taste due to using coffee that has been sitting on a shelf for long periods of time. Our solution will allow more people to enjoy coffee with premium flavor and taste in the convenience of their homes.

# 2.3 Project Specifications

This section goes over the various software and hardware requirements and specifications for the project. The listed requirements and specifications for the various components of the project are very important in meeting the intended goal and use case for the project. These specifications are even more important if the product ended up being commercialized and further implemented after Senior Design II.

## 2.3.1 Chassis Specifications

Due to the choice of retrofitting an existing product, we had to determine a method of fitting our circuitry into the product. The existing product does not have a lot of room for circuitry as it was not designed for the use of microcontrollers, Wi-Fi, and a full color touchscreen display in mind. Since our product will have these capabilities, our circuit boards will be larger and more difficult to fit in place. We will also need a way to isolate our electronics from the heat of the roasting chamber.

While working on the weight sensing subsystem, it was determined that we could attach a metal box to the bottom of the roaster that incorporated both the load cells for the scale, and all of our added electronics. Wires could be routed through the existing chassis at allow for control of existing roaster systems. This design can be seen in Figure 2.3.1-1 below:

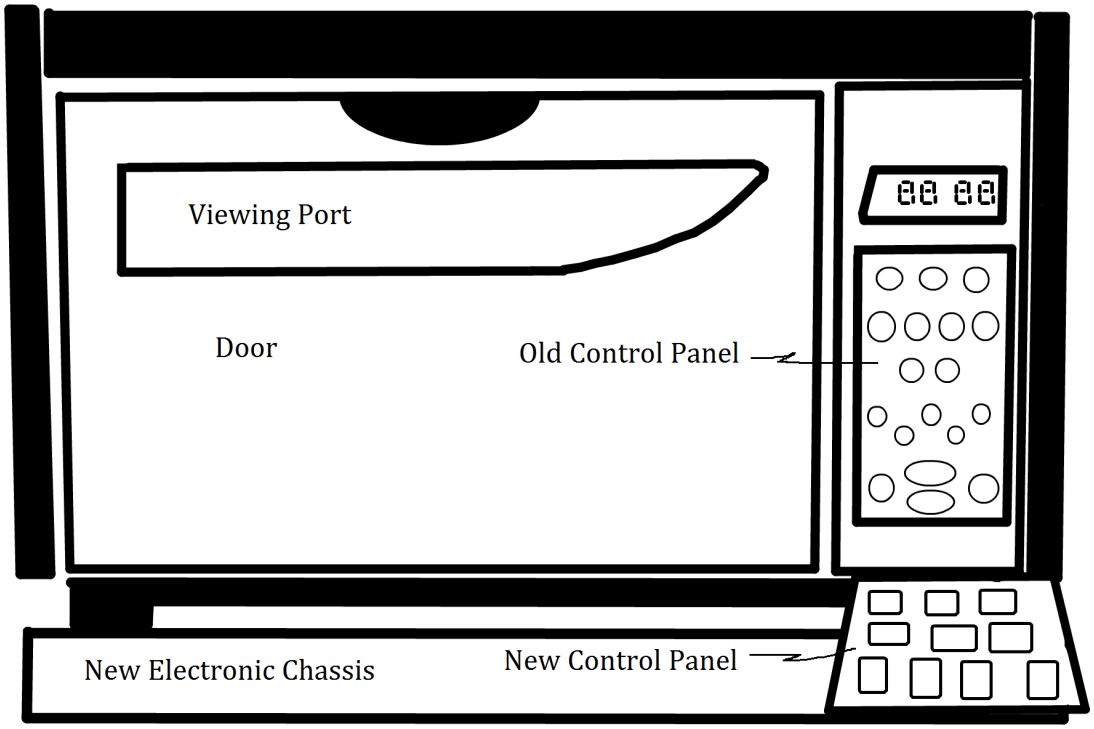



*Figure 2.3.1-1. Roaster chassis design.*

The load bearing tip of each load cell will be fastened to the feet of the existing roaster and the reverse side of each cell will be fastened to the chassis below it. The chassis shall be made of either aluminum sheet metal or cedar. It shall be at least 5cm tall and shall be limited in width and depth to at most as wide and deep as the profile of the existing roaster. There shall be vents on the reverse side to allow for air flow to assist in cooling of components. This design will allow us a larger amount of room to work with while providing large thermal separation from the heat produced in the roasting chamber. It will also provide a cleaner aesthetic than those which could be obtained through other chassis modification options.

## 2.3.2 Motor Specifications

The DC brushless and stepper motor will operate between 5 and 15 volts DC and be securely fastened in their proper positions. The motors will have sufficient torque to turn the fully loaded drum. The AC Induction motor will be positioned next to the inlet and outlet vents of the coffee roaster, and be used as blower motors. The startup torque will be small enough to ensure quick operation for the blowers to quickly and efficiently clear the chamber of smoke and steam and fill the chamber with the heated air in a timely fashion. This startup torque shall be no higher than 190% of the rated torque for the motor.

## 2.3.3 Controller Specifications

The controller will operate at +3.3 volts, up to a maximum of 4.17 volts. The analog and digital supply voltage will also remain at +3.3 volts to ensure the MCU will not experience failure. The temperature of the MCU will not exceed $95^{o}$ C. The MCU will be insulated from the power system and the heating elements to ensure this specification is met. The external capacitors will be 4.7μH with a tolerance of at most ±10%. Proper ESD safety techniques will be followed to ensure any discharge that occurs is between ±1000 volts.

## 2.3.4 Sensor Specifications

The temperature sensor will be accurate to within ±10% of its given operating range, and while within the range of temperatures experienced during the roasting process. The response time of the temperature sensor will need to be quick enough to allow for accurate readings and for the user to make changes. This response time will need to be less than 5 seconds. The weight sensor will need to be sensitive within ±2 ounces, to ensure that the roasting progress can be determined accurately and stopped before the beans are burned.

## 2.3.5 iOS & watchOS Application Specifications

The iOS Human Interface Guidelines provided by Apple will be closely followed for both design and user interaction purposes in the iOS application. The different interface and design elements will be structured similarly to the native iOS applications to maintain familiarity and usability to the users. The iOS application will be communicating over a Wi-Fi connection. The back-end server and API that is receiving HTTP requests will be directly communicating with the MCU. The iOS application is not directly interfacing with the coffee roaster. The iOS application is expected to include multiple features such as allowing the user to view different roast types, begin a roast, view a roast in progress and maintain a history of completed roasts.

Adjustments to temperature and time duration for a roast must be able to be modified within the iOS application. The communication between starting a roast and the coffee roaster reacting accordingly is expected to take no more than 30 to 60 seconds over a reliable Wi-Fi connection. Maintaining a short amount of time to successfully control the roaster is important in preventing the user from believing the system is not functioning properly due to lack of feedback. Figure 2.3.6-1 shows a mockup block diagram of the iOS application and its intended features.

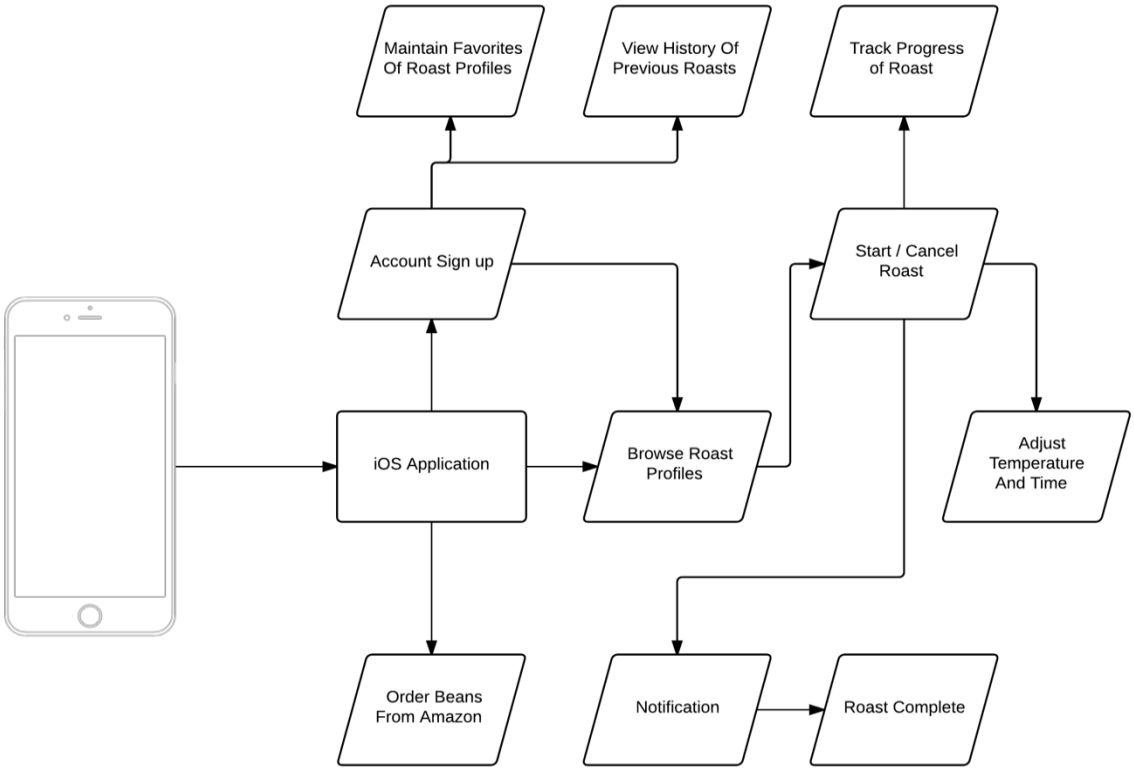



*Figure 2.3.5-1 iOS Application Block Diagram [1]*

The watchOS Human Interface Guidelines will also be closely followed for the watchOS applications design and various user interfaces. The watchOS application communicates with an iPhone device through a Bluetooth LE connection. The watchOS application is not necessary for the function of controlling the coffee roaster for the purpose of acting as an extension to the iOS application by offering features that are unique to the platform. Response times will be kept under 30 to 45 seconds when interfacing between the iOS application and the coffee roaster. The interface will be user friendly and familiar to the user due to the commonly used design elements being incorporated.

Figure 2.3.5-2 shows a mockup block diagram of the watchOS application, complication and glance. The glance and complication will provide added convenience to the user that is not possible on an iPhone device.

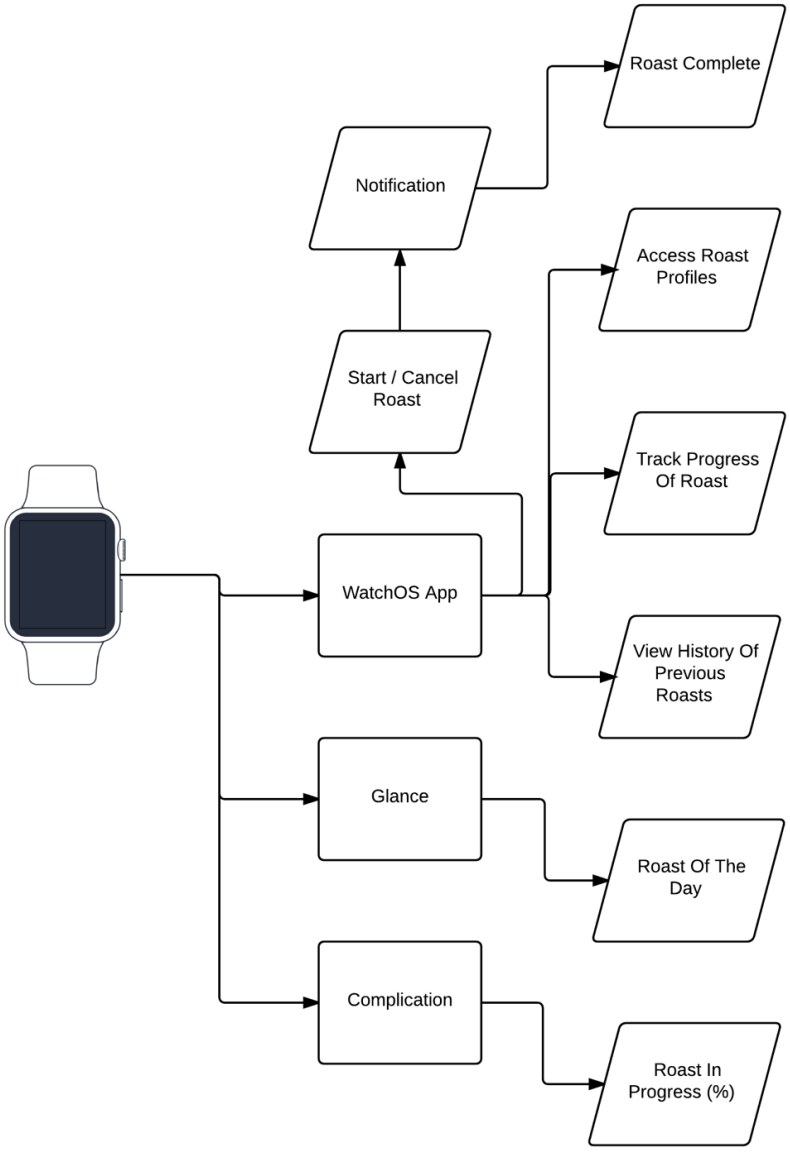



*Figure 2.3.5-2 watchOS Application Block Diagram [2]*

## 2.3.6 Microphone Specifications

Our microphone has a bidirectional, or “figure 8” pickup pattern. It has the following specifications, as outlined in table 2.3.6-1 below:

| Item | Value |
|---|---|
| Sensitivity | -54±3dB |
| Operating Frequency | 100Hz-10kHz |
| Signal to Noise Ratio | 56dBA |
| Operating Temperature | -20°C-70°C |
| Operating Voltage | 2v |
| Current Consumption | 0.5mA |
| Output Impedance | 2.2kΩ |

*Table 2.3.6-1. Microphone specifications [3]. Copyright CUI Inc. Used with permission.*

While either a flat or linear microphone response is considered optimal, that is rarely, if ever achieved. The microphone that we chose had the following response curve, as seen in figure 2.3.6-1 below:

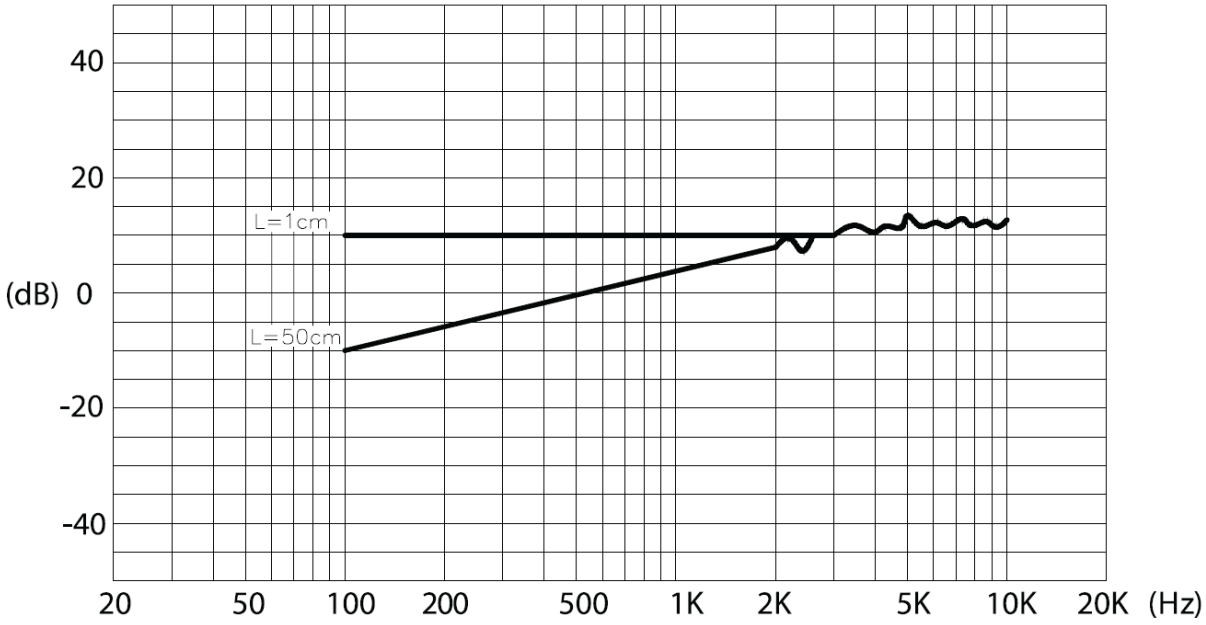



*Figure 2.3.6-1. Microphone Response Curve [3]. Copyright CUI Inc. Used with permission. L = distance from source to mic.*

While the performance of this microphone is not ideal, it comes close to approaching the ideal. This allows for easier programming of the threshold detector.

### 2.3.6.1 Analog Filter

Our filter needed to consist of a band pass filter with a very high roll off and with a very narrow pass band. This led us to use an inverted notch filter. This design provided us with a roll off greater than 100dB/Decade while using only a second order filter. The design also gave us a bandwidth of approximately 5 Hz. The design allows for a software controlled cutoff frequency while using an analog filter. The cutoff frequency of the filtered is controlled by sending a square wave from the microcontroller equal to fifty times the frequency that is to be filtered. This allows us to easily tune the filter after it has been manufactured and even allows for on the fly changes to accommodate different roasting profiles.

# 3.0 Realistic Design Constraints

This section goes over the various realistic design constraints that will be taken into account for completing the project. Numerous restrictions or constraints exist for commercial product development. Several constraints were chosen that more closely matched the intentions and end goal of the project. The constraints aid in making design choices that would be made in real world environments for product development and commercialization of projects.

## 3.1 Economic Constraints

When planning the project, several constraints will affect the overall outcome of both the design and implementation of the project, with one of them being economic constraints. The goal was to keep the cost for the project under or around a $1000 budget with everything including the printed circuit board (PCB) and the coffee roaster itself. The most expensive components of the project are going to be the coffee roaster and the numerous iterations of the PCB. The cost adds up fast with just those two major components alone. The coffee roaster was already purchased for around $400. Luckily, most of the components on the coffee roaster will be utilized. The PCB has not been purchased yet, but it is estimated to be around $450 (12 copies @ $37.50 each). This cost may drastically go down due to unnecessary precautions being taken for the PCB. It is estimated that it may be necessary to have numerous copies and iterations of the PCB from initial planning and research. With just the coffee roaster and PCB, it equates to about $850. The rest of the purchases will include sourcing coffee beans for testing the functionality of the coffee roaster, and various components that will be connected to the PCB for both controls and communication.

Fortunately, the development tools and software necessary to create the iOS and watchOS applications are open source to the public. There is no additional cost to build the applications or purchase development software such as Xcode. Also, the primary front end developer for the project owns both devices that the applications are intended to be installed on, an iPhone and Apple Watch. This

definitely helped with keeping the cost for the project down as both the developer tools and hardware devices are already taken care of outside of the budget. An Apple Watch alone would have been $349 for the cheapest version.

As it turns out, the hardware components necessary to control the coffee roaster and set up communication will need to be paid for out of pocket. This includes the MCU, Wi-Fi booster module, power management integrated controllers, microphone pre-amplifier and analog filter as some if not all components necessary. Our group was fortunate enough to sign up for the Texas Instruments Innovation Challenge which is a student competition sponsored by Mouser Electronics. By default, participants are granted with a $100 coupon that can be spent on the TI Store to purchase components, including analog integrated controllers and a processor. This should cover most if not the entirety of the components that were planned to be used from Texas Instruments. The Texas Instrument laboratory facility in UCF also provides students access to TI hardware and tools at no cost, which should help with any additional modular components needing to be acquired.

In order to properly test the major functionality of the coffee roaster and different varieties of roasts, un-roasted coffee beans will need to be obtained, preferably from multiple sources. Currently, our primary choices of roaster sources include Sweet Maria's, who are based in Oakland California, and Blessed Beans Coffee who are conveniently located in Longwood Florida. Blessed Beans Coffee will be used as a single source to test a majority of the coffee roasters functionality and complete the majority of the usability testing. The cost for un-roasted coffee beans will be significantly cheaper from Blessed Beans Coffee, which is why we are sourcing them for a majority of the initial testing. Sweet Maria's will be sourced for a variety of un-roasted coffee beans to test the different variations in functionality and roast types available from the coffee roaster. There will be a lesser quantity sourced from them due to an increase in cost for the wide variety of coffee beans and increased level in quality for taste.

## 3.2 Time Limitations

Several time limitations will be taken into consideration regarding how much time each team member is able to consistently allot and designate for the completion of the project in both Senior Design I and Senior Design II. One restriction being the choice for completing Senior Design I in the summer semester. The summer semester lasts about 11 to 12 weeks compared to a regular Fall or Spring semester having a duration of 16 weeks. The lost difference in time equates to about 4 to 5 weeks. The fact that senior design groups in the summer have about a month less of time will be a critical impact on upholding time management, and also planning milestones to stay on track for completing the project smoothly and efficiently. The team members were confident in creating a project that was not impacted in quality or function despite having less time to complete the design and implementation from start to finish.

The goal for Senior Design I was to get as far ahead of schedule for both the hardware and software design in order to lessen the work that will need to be completed in the must shorter summer semester for Senior Design II. The coffee roaster has been purchased and acquired in order to begin take apart and understand how the roaster functions and performs prior to being modified. Coffee beans have been purchased to test and measure the expected results from the coffee roaster and its various settings. Also, numerous components of the front-end and back-end software are in development. The design of the iOS application from both the planned features and initial user interface have been completed. Implementation of the iOS application is being developed and prototyped through regular testing on an iPhone device. The back-end server and API that will be interfacing with the front-end applications (iOS and watchOS) have been set up and are undergoing testing necessary to verify functionality for communication and control of the coffee roaster when it is time to be implemented.

Another potential restriction on time for Senior Design was the fact that all four team members work part-time jobs outside of their school schedules. A large majority of the work necessary to complete senior design will need to occur during the same days that each group member attends classes or are campus. In order to help retain focus and organization towards milestones and goals for the project, a dedicated meeting day was arranged. Thursday every week was devoted as a common meeting day in which ideally everyone on the team would meet for 1 to 2 hours. The expectations of the weekly meetings were to plan milestones, clarify any concerns or uncertainties regarding the project, and also maintain organization between the different tasks that each team member is currently working to complete. Having a set time for each team member to meet and discuss in person greatly increased productivity and communication amongst various concerns and plans for the project. Each team member had a full class schedule on top of work, which made it that much more crucial to be proactive and be mindful of time management in achieving tasks during Senior Design I.

## 3.3 Environmental Constraints

Since the coffee roaster is intended to be used in kitchens alongside other common household appliances, such as a coffee brewer or microwave; it was necessary for the project to meet and maintain certain environmental constraints. The coffee roaster that was acquired and is being modified for the project has patent pending built-in smoke suppression for safe use indoors. There are no issues with the coffee roaster producing excess smoke or pollution that may escape during the roasting process that may trigger a smoke alarm or raise concern to the user. The coffee roasters intended use case is similar to that of a conventional toaster oven due a similar heating element design and ability for settings to be controlled and modified for cooking. It is planned for the rotating chamfer inside the coffee roaster to be replaced with one that has smaller holes to prevent beans from falling through to the bottom of the roaster and potentially catching on fire and producing smoke due to the heat intensity and metal surface materials. If the project was eventually

commercialized and sold, it would be marketed as being a kitchen appliance that affects the environment similarly to that of a microwave or conventional toaster oven to simply explain it to the public.

The coffee roaster does not produce any air pollution or abnormal noise pollution that would be expected from a traditional coffee roaster that can be purchased by the public. The roaster itself is very quiet and the only noticeable noise is when the coffee beans are cracking open or when the roasting chamber is rotating. During a roast, dried skin from the coffee beans also known as chaff falls to the bottom of the coffee roaster and is the only sort of waste product that is a result of the entire process. The chaff itself cannot be reused by the user, but it can be recycled and used for different types of products, such as a bedding material for animals or as compost material for growing plants and vegetables in particular. The coffee roaster is powered from a traditional wall outlet and will not require any sort of fuel or resources that would produce air or water pollution as a byproduct of the roasting process. Overall, the entire project is environmentally friendly from the roasting process not producing any negative pollution and also the fact that the waste that is generated during a roast can be later recycled and reused for multiple purposes, including growing vegetables.

## 3.4 Social Constraints

Maintaining the goal for the coffee roaster to have the ability to be controlled wirelessly from a user's iPhone or Apple Watch created a social constraint for the project. Various home automation devices are able to be controlled through an iPhone due to the added convenience and seamless usability of the interfaces that can be implemented. The social basis and standard for people at the moment is that they always have their smartphone or device on them at all times. People are tied to their devices due to how many various tasks they can conveniently perform on a daily basis. The fact that there is a high chance that a user will always have some sort of iOS or watchOS device on them created a social constraint for the project by focusing on creating a cohesive experience and compelling user interface that the target audience would enjoy to continue using.

The user interface for the iOS and watchOS devices were modeled after popular applications on the App Store to have a familiar feel and usability that the user would enjoy using the first time they open up the respective applications. Different components of the user interface from the slide out menu to the home page were designed around what was most popular and seen common across applications that users will frequently use every day, from social networking applications to also video and music application.

## 3.5 Health & Safety Constraints

As previously mentioned, the coffee roaster functions very similar to that of a conventional toaster oven. There will not be any additional precautions necessary from the user that need to be followed with the modifications that are being implemented. The coffee roaster does not produce any air or water pollution that the user may be exposed to that would pose any health risks when being in proximity of a roast in progress or when consuming the coffee beans after they have been roasted and ground. The chaff waste product that will be found at the bottom of the coffee roaster is not harmful to the user if accidentally consumed or not disposed of properly. The coffee roaster does not produce any harmful radiation or dangerous fumes that would be imperceptible to the user. When the coffee beans are roasting, they will produce a unique and tolerable smell that does not pose any health risks when continuously inhaled during use.

The additional functionality to the coffee roaster for both hardware and software will also not require any additional safety measures or constraints than those that should be expected. A majority of the enclosure and internal surfaces of the coffee roaster are comprised of metal or thermal materials where the heat that is generated during the roasting process will need to be kept into consideration. The heat when a roast is in progress, and also when removing the coffee beans can be very dangerous to the user and it is recommended for the user to wait a safe amount of time for the coffee roaster and roasted beans to cool down before handling. It is recommended to use thermal resistant gloves when handling the roasting chamber and transferring the coffee beans. The coffee roaster is intended to be used in a kitchen around other appliances in which the same safety precautions apply. The chaff at the bottom of the coffee roaster will need to be regularly removed after each roast to prevent the possibility of them catching on fire due to the internal heat. Fortunately, the coffee roaster used already includes a convenient method to remove the chaff after each roast. The same safety measures practices should be applied to the coffee roaster in order to prevent any possibility of safety concerns that are due to misuse of the product.

Aside from the onboard interface on the coffee roaster, there will also be the interfaces provided by the iOS and watchOS applications. Both devices were not modified for the implementation of the product and thus meet all health and safety constraints set by Apple when designed and built. Both devices don’t pose any health and safety concerns when being used for controlling the roaster as intended. Both devices give off radiation that is not in an intensity high enough to cause harm to the user as expected by the different internal hardware components. The user can utilize both applications within proximity of a Wi-Fi or cellular connection. Both methods of data connection do not pose any health or safety risks to the user. From both the hardware and software modifications and implementation to the coffee roaster, there are minor health and safety risks that the user should be aware of. However, the health and safety risks are no different than the original design and function of the coffee roaster and also that of common

household appliances the user may have in their kitchen. The goal to not create any additional concern to the user regarding their health and safety was very important and was kept closely into consideration during the entirety of the development and build process for the project.

## 3.6 Manufacturability & Sustainability Constraints

The Behmor coffee roaster was purchased online can be easily sourced. The coffee roaster was not manufactured or custom built, in which any parts or components that fail will need to be sourced directly through the manufacturer Behmor. However, the roasting chamber was replaced and a new one can be purchased if it eventually suffers fatigue or damage from normal use. The printed circuit board (PCB) implemented was completely designed from the ground up, and the schematics can be used if the project is commercialized. Any components on the PCB will be outlined and a majority can be sourced from Texas Instruments (MCU, Wi-Fi Booster Module, QVGA display, etc.) if needing to be replaced or used for reproduction.

Aside from the hardware modifications added to the coffee roaster itself, an iOS device and or Apple Watch will be necessary in order to wirelessly interface and control it. Fortunately, a majority of the target audience own one or both of these devices based on the target audience for current home automation devices having iOS and Apple Watch support. It is not necessary to have an Apple Watch to have full functionality and control of the coffee roaster, but instead adds more convenience and usability for the user. Due to the fact that Behmor is currently developing a wirelessly controlled coffee roaster, it may be possible to contact Behmor and work directly with them towards the same product and niche market. It should be kept into consideration in not violating any terms and conditions that may be in place by Behmor if this product is commercialized. The ideal case outside of using the Behmor coffee roaster would be to custom build a coffee roaster with similar specifications in order to reproduce the PCB and reuse the schematics and designs.

The coffee roaster is powered by a traditional wall outlet that users would have access to in their kitchens. The only maintenance that would need to be performed is with the removal of the chaff that develops at the bottom of the roaster. Removal is necessary to prevent any possibility of fire or smoke that is produced if the chaff were to catch on fire. With the intended use for the roaster being in a kitchen environment, the sustainability should not be an issue in comparison to other common kitchen appliances that are frequently used and are infrequently replaced due to failures in components.

Overall, the coffee roaster implementation could be commercialized due to the minor modifications necessary to the coffee roaster itself. Both the iOS and watchOS applications could potentially be submitted to the App Store for monetary

value. The coffee roaster is very sustainable due to its simple design and intended use case being primarily for kitchen use. Little maintenance is required to continue using the coffee roaster with the waste it produces. Sustainability of the back-end server and front-end applications is only necessary when managing the services that communicate with both. Some services may be third party and it would be required to maintain a sustainable infrastructure for the communication systems and software that is used to interface between the applications and coffee roaster.

# 4.0 Related Standards

This section goes over the related standards that were considered and implemented for the project. The sections are divided into hardware and software standards to differentiate the two categories of standards from one another. More related standards may be utilized if any modifications to the design or features are changed in Senior Design II.

## 4.1 Hardware

It's important to work within standards in any type of design. The failure with the Mar's Climate orbiter comes to mind here. Unless a set of standards are followed, and every person on the team follows the same set of standards, disaster can occur. For this reason, we must have a clearly defined set of hardware standards. In doing so, we will avoid the problems encountered by NASA and Lockheed Martin, and greatly increase our chances of producing a successful product.

### 4.1.1 RS232

One standard option for wired connectivity that will interface the LCD and Wi-Fi module with the MCU is RS232. In 1962, the standard was initially introduced by the Electronic Industries Association (EIA) [23]. It was initially intended to be used for point to point communication between modems and terminals. Later versions and implementations of the standard were introduced for the inclusion of more features or solutions to any restraints of the original standard. Bits of information are transported over segmented time intervals or bursts maintained by a common clock. RS232 supports systems in both cases where the broadcast is synchronous or asynchronous with the clock signal. Control circuits are utilized to set up the communication between the terminal and the device it's broadcasting to.

Voltage levels are interpreted in order to represent the information that is transmitted and the various control signals that exist in the system. The logic for the circuit can be represented as being true or enabled on active low and false or disabled on active high. Voltage levels of +3V to +5V represent enabled control, while -15V to -3V represent disabled control [23]. Specialized circuits are necessary in deciphering the unordinary levels, since they're not a typical 0V to 1V for disabled and enabled respectively. Various connector types exist for the RS232

standard. Common convention is for terminal equipment to be male ended and device equipment to be female. For example, RS232 on the MCU side would be a male connection, while the Wi-Fi module would be a female connection to interconnect the two components. RS232 suggests to implement the D-subminiature 25 pin connector, which often results in having access to more pins than necessary [23]. 9 pin connectors are more commonly seen in current electronic devices and computers that support the standard.

Control circuit performance and function are largely dictated by the capacitance of the cables in use. The length of the cable is often used to determine the expected or ideal performance for the communication [23]. Typically, if the length of the physical connection is 50 feet or more, it will greatly affect the performance [23]. In order to signal the terminal that communication is trying to be established, the Ring Indicator is used to achieve this task. Often times the Ring Indicator will signal an interrupt to the operating system in order to indicate a context switch of some kind. Some signals that are commonly seen through the serial communication include Request To Send (RTS), Clear To Send (CTS) and Ready To Receive (RTR). These signals are implemented in the intentions of managing flow control of the communication traffic and data.

The RS232 standard is often used with Universal Asynchronous Receiver / Transmitter or more commonly known as UART for transmitting serial data to an MCU or computer. RS232 has later been replaced in more and more devices by the more commonly known standard, Universal Serial Bus (USB). Despite the various disadvantages of using RS232 over USB, various industrial and networking equipment continue to utilize the standard.

### 4.1.2 UL 1026

UL 1026 covers a wide range of aspects for designs of household electrical cooking appliances that operate on 250V or less. It has a very large scope of the overall design. Topics that it covers which include, but are not limited to:

- Enclosure and Frame
- Stable
- Corrosion Protection
- Internal Wiring
- Heating Elements
- Thermal Insulation
- Motors and Transformers
- Overcurrent protection
- Automatic Controls
- Grounding
- System Performance

Due to the high cost of obtaining this standard, we are unable to see what specific requirements there are for different aspects of the design. We are however able to follow practices used in the original roaster design. We are able to meet or exceed the wiring gauge used to connect to the heating elements using thermal wiring. We are able to use similar electrical connectors for high power parts, and ensure that there is proper thermal isolation between the roasting chamber and our components in order to meet potential thermal insulation requirements. As we go through building and testing our retrofitted unit, care will be taken to ensure that materials and workmanship are similar to what is found in the original roaster. While this obviously does not meet specification compliance, it is the closest that we are able to get with the information that is currently available to us.

### 4.1.3 Bluetooth

One standard option for wireless connectivity between the coffee roaster and both the iOS and watchOS applications is Bluetooth Smart. In 2006, the standard was initially developed by Nokia under the name Wibree or more commonly known as Bluetooth Low Energy (BLE). The Bluetooth Special Interest Group which manages development of new standards and technologies later adopted the standard in 2010 and re-marketed it as Bluetooth Smart. Numerous advantages of using the standard include: low power consumption, low latency, low development costs, strong security and wide spread industry acceptance. Most devices within the Internet of Things (IoTs) utilize Bluetooth Smart due to the major advantages in efficiency, performance and development cost savings. Bluetooth Smart implementations operate on the same radio frequency range as Classic Bluetooth from 2.4 GHz to 2.4835 GHz. Bluetooth Smart makes use of Gaussian Frequency Shift Keying (GFSK) modulation for data transmission and direct sequence spread spectrum (DSSS) modulation for dealing with interference. Both modulation techniques exploit frequency hopping in which the signal is quickly switched or essentially spread across the different channels in order to minimize the time that a signal is disturbed.

Using the Bluetooth Smart standard would provide low power and high performance connectivity for the iOS and watchOS applications to interface with the coffee roaster, without largely impacting the devices battery life. Since the primary interface for the coffee roaster is through software, battery performance and power consumption should always be taken into account for the device that is running the application. The coffee roaster will be powered by a traditional 120V wall outlet, so there is not a concern about battery life but more so maintaining low power consumption compared to other wireless connectivity standards.

One drawback of using Bluetooth in general includes the proximity that needs to occur between the Bluetooth enabled device (iPhone or Apple Watch) and the antenna on the coffee roaster for a connection to be established and maintained. The user of either application would need to maintain a relatively close distance away from the coffee roaster in order for a successful connection to be set up.

Most interfaces that consumers use every day already utilize Bluetooth Smart including wireless speakers and vehicle infotainment systems. Since the standard is already widely accepted by developers and consumers, it should be considered as an option for the purpose of the project if it is potentially commercialized at a later time.

Another drawback or disadvantage of using Bluetooth technology for the purpose of the project is interference. Many common household devices run on the 2.4 GHz band, including microwaves, fluorescent lights and cordless phones. A result of numerous 2.4 GHz devices in proximity of each other is degradation in signal strength, increase in latency and interference between the devices connecting to the same frequency. Classic Bluetooth and Bluetooth Smart implement different frequency modulation schemes to prevent interference, or at the very least, the time interval of interference between the devices is so small that it is imperceptible to the user. Fortunately, there exists other wireless technologies that have more options to prevent or lessen the impact of interference when dealing with multiple devices connecting to the same signal.

Table 4.1.3 shows some but not all technical specifications comparing Classic Bluetooth to Bluetooth Smart. The technical specifications chosen were most related to the intended use and implementation for the project. Most notable improvements introduced with Bluetooth Smart include: Security, Latency and Power Consumption to name a few.

| Technical Specification | Classic Bluetooth | Bluetooth Smart |
|---|---|---|
| Radio Frequency | 2.4 GHz | 2.4 GHz |
| Range | 10 - 100 meters | 10 - 100 meters |
| Security | 56/128 bit | 128 bit AES |
| Latency (from idle) | > 100 ms | < 6 ms |
| Power Consumption | 1 W (reference) | 0.01 - 0.5 W (varies) |
| Current Consumption | < 30 mA (Peak) | < 15 mA (Peak) |

*Table 4.1.3-1: Classic Bluetooth vs. Bluetooth Smart (LE) [4]*

## 4.1.4 Wi-Fi

Another standard option available for wireless connectivity between the coffee roaster and both the iOS and watchOS applications is wireless fidelity, or more commonly known as Wi-Fi. The Wi-Fi Alliance is the organization that manages Wi-Fi certifications and verifies if a product meets the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards. Implementations of wireless local area networks (WLAN) that conform to IEEE 802.11 include medium access control (MAC) and physical layer specifications in order to maintain interoperability

and backwards compatibility across devices on the WLAN. The necessary steps to be considered Wi-Fi Certified include meeting IEEE 802.11, security and authentication standards. Products that pass certification testing by the Wi-Fi Alliance are deemed Wi-Fi Certified and are then authorized to use the trademark and logo.

Common versions of the IEEE 802.11 standard include 802.11a/b/g/n/ac. 802.11b and 802.11g both operate on the 2.4 GHz band. Devices using the 802.11b/g standard suffer from the possibility of interference between devices running on the same frequency. 802.11a operates on the 5 GHz band. Operating on the lesser used band drastically decreases interference and diminished network performance as a result of network traffic. However, the signal range is actually less than the previous 802.11b/g standards due to the increase in signal loss across physical mediums, such as walls in a building. 802.11n can operate on both 2.4 GHz or 5 GHz bands depending on the network topology and proximity between the devices in the network. Multiple Input Multiple Output (MIMO) technology was first introduced with 802.11n, which implements four antennas or spatial streams to dramatically increase data output across the network. 802.11n also includes the benefit of increasing the channel width by two fold from 20 MHz in previous 802.11 standards to 40 MHz.

The current standard being deployed to wireless devices is 802.11ac. Wider 80 MHz channels are introduced, with the option of 160 MHz to further increase bandwidth. Advances in MIMO technology were implemented in 802.11ac with the support of up to eight spatial streams and possibility of four clients receiving data from one transmitting client. A higher 256 quadrature amplitude modulation (QAM) scheme was introduced to increase the data transfer rate, but as a consequence, it may result in an increase in noise due to the higher accuracy needed for the neighboring symbols in the constellation to be accurately located. Beam-forming is a technique used in the new standard to have increased signal strength in the direction of the wirelessly connected device, instead of the signal traveling in all directions, regardless of where a device is located. The higher 5 GHz band that 802.11ac uses allows for higher data rate but lesser distance the signal can travel. Smaller frequencies travel farther distances, as a result, 802.11n devices have better range than 802.11ac since they can utilize both 2.4 GHz and 5 GHz bands.

Using Wi-Fi as the standard for wireless connectivity between the coffee roaster and the respective iOS and watchOS applications would provide numerous advantages in performance over Bluetooth Smart. Wi-Fi connectivity would not require the user of the application to be within proximity of the coffee roaster to connect and maintain a reliable connection. One goal or use case for the project is to allow the user to be able to control the roaster wirelessly in their bed when waking up in the morning. When a user is getting ready for the day, they would have the luxury of freshly roasted coffee beans ready to be ground for a cup of coffee. Bluetooth Smart would not allow this implementation in most cases where the user is not in close proximity to their kitchen or location of the coffee roaster.

The user could be virtually anywhere and have full control of the coffee roaster through the iOS and watchOS applications. The ability for the user to have the convenience of controlling the roaster wherever, whenever is a very important goal for the project.

Another benefit of using Wi-Fi over Bluetooth Smart is the possibility of decreasing the possibility of interference between the coffee roaster and nearby devices running and communicating on the same frequency band. The coffee roaster will likely be located in a user's kitchen, and could very well be in close proximity to a microwave, cordless phone and more devices that run on a 2.4 GHz band. Wi-Fi modules that are applicable for the implementation of the project exist in 802.11n and 802.11ac. Both options offer 5 GHz band connectivity and would lessen the interference that would be introduced by the numerous devices in proximity to where the coffee roaster is located in a user's kitchen. Numerous home automation products that consumers may have utilize Wi-Fi including Phillips Hue, Nest Thermostat and Belkin WeMo products. The reason they choose Wi-Fi over Bluetooth Smart connectivity is the convenience of controlling them anywhere using their respective applications on iOS and Android. A user can leave their home and not worry if they forget to turn off their lights or forget to turn off their air conditioning. Instead, they can turn them off without requiring them to be home or near the wirelessly connected device. Most consumers look for this type of feature when shopping for automated or connected devices, thus it would be beneficial to choose Wi-Fi connectivity over Bluetooth Smart if the project were to be commercialized.

Table 4.1.4 shows some but not all technical specifications comparing the different standards of 802.11 a/b/g/n/ac. Most notable improvements introduced for 802.11 include: Max Data Rate Per Channel Bandwidth and Approximate Indoor Range.

| Technical Specification | 802.11a | 802.11b | 802.11g | 802.11n | 802.11ac |
|---|---|---|---|---|---|
| Frequency Baseband (GHz) | 5 | 2.4 | 2.4 | 2.4/5 | 5 |
| Channel Bandwidth (MHz) | 20 | 22 | 20 | 20/40 | 20/40/80/160 |
| Max Data Rate Per Channel Bandwidth (MB/s) | 54 | 11 | 54 | 72.2/150 | 96.3/200/433.3/866.7 |
| MIMO (Spatial Streams) | X | X | X | 4 | 8 |
| Approximate Indoor Range (m) | 35 | 35 | 38 | 70 | 35 |

*Table 4.1.4-1: Comparison of 802.11a/b/g/n/ac standards [5]*

Based on the intended goal for the project and expected functionality for the iOS and watchOS applications, the choice was made to go with Wi-Fi connectivity over Bluetooth Smart for the coffee roaster. The ability for a user to utilize the applications without the possibility of severe interference, and the need to be in close proximity of the coffee roaster made it the clear choice. The convenience of a user successfully starting a new coffee roast regardless of their location was the intended goal of the project. Also, since most consumers have a few devices in

their kitchens that run on the 2.4 GHz band, there would be a high chance of interference affecting the performance of the coffee roaster. There is a high chance that the coffee roaster is located near a microwave and cordless phone in which it may interfere with the communication. Wi-Fi provides the best performance from multiple technical specifications that already make it the most utilized choice for existing home automation devices and solutions. The choice of implementing the same wireless connectivity solution that is widely accepted in the home automation market would increase the ability to penetrate the market if the project is commercialized.

## 4.2 Software

This section goes over the various software standards and guidelines that were closely followed and utilized for the project. The standards are specific to the front end application development, back end software development or embedded software development. The implementation of the various the standards that were utilized is briefly explained in the subsections.

### 4.2.1 SOAP/REST

A REST API relies on being stateless, client-server independence, and cacheable communications protocol. The primary idea behind rest is that rather than relying on complex mechanisms like COBRA and RPC to connect between machines, we would instead use simple HTTP calls between the machines. The six constraints of a RESTful API are: 1) Uniform Interface, 2) Statelessness, 3) Cacheable, 4) Client-server independence, 5) Layered System, and 6) Code on Demand—the last of which is optional.

The primary principles of a uniform interface being: resource based requests, manipulation of resources via representations and self-descriptive messages. The user would identify an individual resource based on the URI in the request. The server would send some formatted data such as in XML or JSON format. For example, if the user wanted to get a list of dark roasting profiles a request could look like “GET smartroaster.com/api/roasts/dark”. The response would depend on the implementation on the server side but assuming it would work like we assume it does, it would return data such as: {‘dark roasts’: {‘name’: ‘dark roast 1’, ‘profile’:’data’}, {‘name’: ‘dark roast 2’, ‘profile’:’data’}}. This tells us that there are two profiles that meet our criteria of being a dark roast those being ‘dark roast 1’ and ‘dark roast 2’. The client should have enough information in order to modify or delete a given resource given permissions, i.e. manipulation of resources via representations. Each message or response and request must include enough information to describe how they should be processed, such as, the MIME type.

Statelessness being one of the primary influences over the REST-style architecture imposes significant restrictions on how a service and their consumers may communicate. Key to an API being stateless is that the necessary state change should be sent via the URI, query parameters, body or headers.

For example, if the user wanted to change the temperature of their roaster to 500 degrees the URI might look like: 'POST smartroaster.com/api/roaster/temp=500'. Unlike session based communication which would maintain its state across numerous HTTP requests, rest would require that all information necessary to fulfill the request be sent. Meaning if you were attempting to receive some data that requires authentication you would likely need to pass a token on each request that required it.

An API response to the consumer may be labeled as either cacheable or non-cacheable. The idea being that if their components of previous responses that can either be partially or completely reused it would allow for eliminations of some interactions over the web thus leading to a reduction in the average latency during a session with multiple requests.

One of the primary constraints of a RESTful API being that the client and the server must be independent of each other. The separation allows for the software to become decoupled meaning that if a change occurs on one it won't affect the other given that the interface is not changed. This allows the developer to develop and/or replace each as independent bodies. The server would act as a listener for requests from the client. One a request is received the server would either reject or perform the request depending on the request and data sent. Either the requested data would be sent in a response or the request would be denied and the error message would be sent instead which would give clarity as to why the request was denied in the first place. A visual of that separation can be seen below in figure 4.2.1-1.

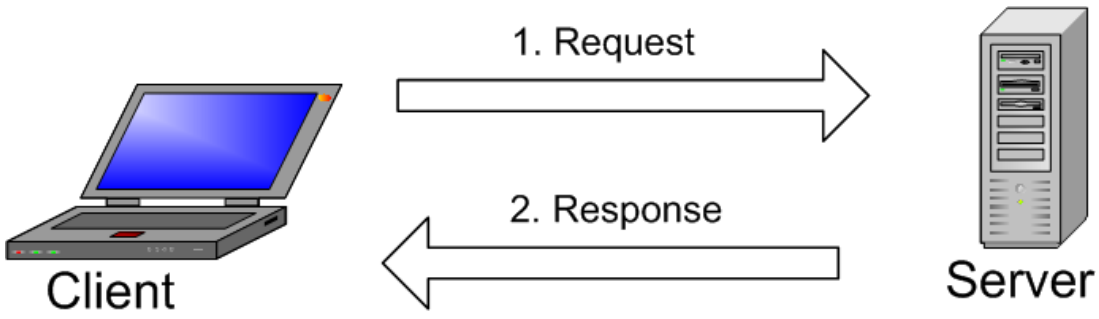



*Figure 4.2.1-1 Clear server client separation [6]*

One of the final constraints of a RESTful system are that it be a layered system. This means that on the consumer end they would not need to know what other services—middleware—needed when invoking some initial service within the application. This would allow for additional middleware layers to be added or to

remove others without changing the interface between the consumer and service. Given the layered approach we could take advantage of intermediary servers to improve our systems scalability by making use of load balancers and a shared cache.

The final constraint which again is optional allows for the server to provide logic for the consumer to run given some need. This would allow for features to be added dynamically without any formal upgrades on the client side. The large downside to this feature would be that the required execution environment for the consumer would introduce security vulnerabilities.

SOAP (Simple Object Access Protocol) was created in order to solve the problem of how can we standardize the communication between web services. The applications that are communicating can be on different operating systems, different technologies and even different languages.

Communication between web applications that use soap is done via soap messages. A soap message is simply an ordinary xml document that is very specifically formatted. First off, we have the envelope which allows the receiving application to recognize the message as being a SOAP message. The header element contains what you would expect which is all the header information relevant to the message. The body element contains all the information for the initial call and how to respond. If there were errors that information will be located in the fault element which also includes status information.

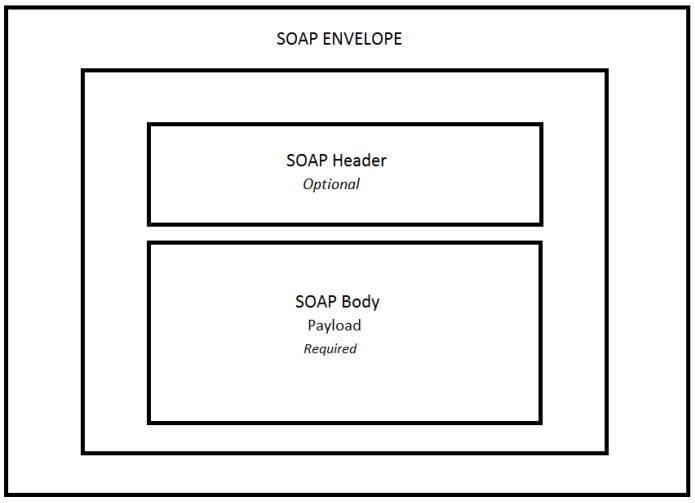



*Figure 4.2.2-2 SOAP Envelope*

### 4.2.2 ISO/IEC 9899:2011 (C11)

The current C programming language standard is ISO/IEC 9899:2011, or more commonly known as C11. C11 was introduced in 2011 when it became the successor to the C99 standard. The newest version of the standard introduced enhanced implementation for already existing features in standard compilers and a superior representation of memory when executing multiple threads in a program. The standard in itself specifies the overall representation and constraints imposed by the C language in general. Applying the current C language standard to the project will practice the importance and reason that standards exist in the industry for engineers and programmers to follow.

The microcontroller unit (MCU) that will be operating the coffee roaster from both the onboard touch screen interface and application interfaces will be programmed in the C language closely following the C11 standard. A common standard on syntax and representation of data will be kept into consideration and maintained constant across all of the embedded programming for the MCU. Following a set of common guidelines and programming standards across all of the code will make it easier for multiple people to understand what is occurring in the code, and allow for industry best practices to be applied and considered for the project.

### 4.2.3 Swift Standard Library

Both the iOS and watchOS applications that will be controlling the coffee roaster wirelessly will be programmed in the Swift programming language. Swift was initially introduced in 2014 at Apple's Worldwide Developers Conference (WWDC) as a successor to the Objective-C programming language for both desktop and mobile application development. Originally, the language was proprietary and required an Apple Developer membership to write applications for the platform. At WWDC in 2015, Swift was made open source for the public to learn, develop and load programs onto personal devices with no cost necessary. The language is marketed as being safer and more compact than Objective-C by providing extensive optimization and increases in performance that were only made possible by developing a new programming language.

The Swift standard library outlines the basic foundation and features available when writing in the Swift programming language. These resources are similar to the standard library that exists in the C programming language. Functionality available include the implementation of common data types (Int, Double, Float, String), standard data structures (Array, Set, Linked List, Enumeration), and useful built-in functions that may be frequently used and should not require the programmer to manually develop them. The Swift programming language supports several features that exist in well-known high level languages such as control flow, classes, error handling, inheritance and many others explained in "The Swift Language Guide" on the Apple Developer portal site. The Swift

Standard Library will be followed very closely for all aspects of development for both the iOS and watchOS applications since both platforms utilize the language.

### 4.2.4 iOS & watchOS Human Interface Guidelines

Various first party and third party applications developed for iOS and watchOS follow a set of standards and guidelines in order to create a consistent and user friendly interface across the entire application. Apple provides their own extensive list of iOS and watchOS Human Interface Guidelines that the public have access to view and keep into consideration by going to the Apple Developer portal site. The goal and intentions of the Human Interface Guidelines is for developers to understand the foundation of iOS and watchOS in its entirety, and to be critically aware of the reasons why different elements of user interface design and principles were chosen in particular. The numerous standards strongly recommend and encourage that the organization of an application and its various methods of user interaction be consistent and common all throughout. These standards are also important when an application is cross platform, such as iOS and watchOS or even iOS and Android. Developers that keep these guidelines closely into consideration tend to receive more positive feedback and regularly retain users of their application.

Effective implementation of common core design principles can make a drastic difference in the reaction to an application, and may very well influence a user in deciding whether to continue using an application or search for a better alternative to achieve their intended goal or task. The main reason the App Store contains multiple applications that essentially fulfill the same objective for a user is due to the different methods of execution developers choose to implement based on common guidelines and standards. The assortment of design principles and strategies explained in the iOS and watchOS Human Interface Guidelines include: aesthetic integrity, consistency, direct manipulation, feedback, metaphors and user control.

Aesthetic integrity is the idea of how the integration of an application and the design of an application directly follow its intended use. Games are a straightforward example of how the practice and implementation of this principle can result in a more cohesive experience for the user to enjoy and continue to use multiple times. Games tend to have more emphasis on user interface design elements, as well as making sure that the different methods of user input are relatively easy and simple to pick up and start enjoying for both new and returning users. Games that require little time investment to understand and play tend to become more popular amongst users, and are also more successful in maintaining users. Consistency has been mentioned previously, but ideally the principle focuses on the standards that users are familiar with, and harps on the experiences that are consistent across multiple interfaces within an application. Applications that have multiple layers of interaction should have consistent functionality and behavior that users can expect. If the behavior of an element that is used across multiple interfaces

fails to meet the user's expectations, it is improperly implementing the consistency and cohesive experience outlined in the iOS and watchOS design standards. A typical example of consistency being implemented is with the reuse of icons, buttons or user input methods that are seen across multiple layers of an application. Users can easily understand and guess the reaction of triggering different interface elements based on them consistently being utilized across an application.

Figures 4.2.4-1 and Figure 4.2.4-2 show examples of aesthetic integrity with the native Weather application on iOS. Clouds, color, brightness and icons are used to put more emphasis on the current weather conditions. Figures 4.2.4 - (3-5) show examples of how consistency is utilized in the native Settings application on iOS with the icons, labels and typography.

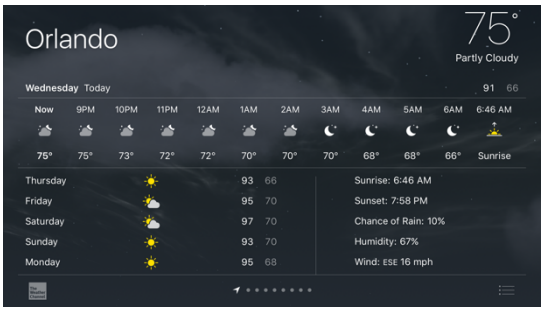



*Figure 4.2.4-1 Native Weather App*

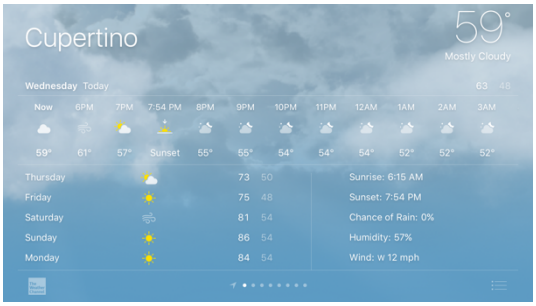



*Figure 4.2.4-2 Native Weather App*

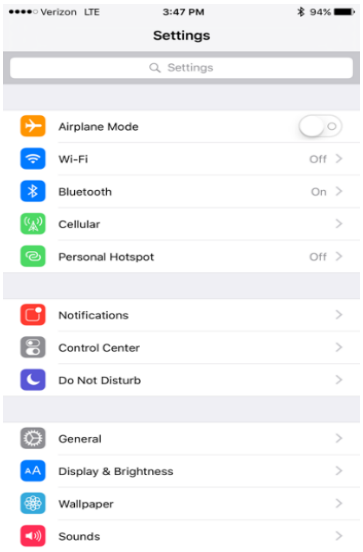



*Figure 4.2.4-3*
*Native Settings App*

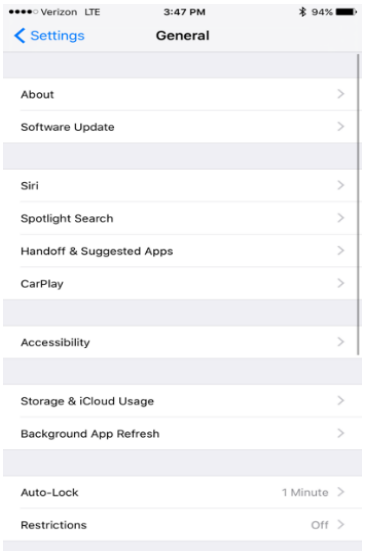



*Figure 4.2.4-4*
*Native Settings App*

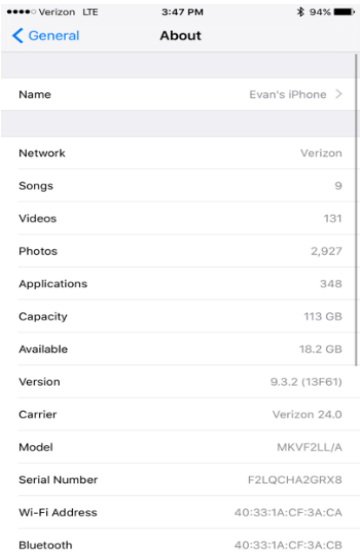



*Figure 4.2.4-5*
*Native Settings App*

Direct Manipulation describes the way that objects can be directly manipulated in order to make modifications or actions. Users tend to be more engaged or find it more natural to directly interact with an interface. Instead of using controls to achieve a task, developers may rely on direct manipulation methods to simplify interaction and create a more natural experience. Users may experience direct manipulation when moving their device or using gestures in order to make modifications to objects onscreen. For example, when a user is trying to zoom into a photo, it is standard to assume that it is possible to pinch the photo outwards in order to zoom in, compared to using a magnifier button or slider button to achieve the same goal. Another example is when a user is trying to view a video and they want to switch it from portrait to landscape mode to see it displayed larger. The user is immediately made aware of the impact they have in rotating their device in order to change a video from portrait to landscape mode, without hitting an onscreen button to achieve the same task otherwise.

Figure 4.2.4-5 and Figure 4.2.4-6 show examples of how direct manipulation is utilized in the native Photos application on iOS. When a user is viewing a video in portrait orientation, they can turn their device to landscape orientation or horizontally in order to view the video in a larger aspect ratio.


*Figure 4.2.4-6*
*Native Photos App*
*Portrait Orientation*

*Figure 4.2.4-7*
*Native Photos App*
*Landscape Orientation*

Feedback is the next design principle that has some relation to aspects of Direct Manipulation. Users specifically look out for some sort of indication to their actions in order to not only verify an application is functioning, but to also know that they are properly using an application as intended to make changes. Animations,

highlighting or sounds are some examples of feedback mechanisms that users look out for to verify their actions are taken into effect. For example, when a user receives a new message on their Apple Watch, they can choose to receive both haptic feedback and sound feedback to indicate a new message has been received. Sometimes the user does not want to allow for others to know when a new message is received by the sound that is played through the speaker. They have the option of choosing haptic feedback to be discreetly notified when a new message is received, without also consequently notifying others around them that are not concerned.

Metaphors are often used throughout iOS and watchOS to signify features or controls that are more familiar to a user to understand or use. The principle is often utilized when signifying common interfaces that the user would better understand otherwise. One common method where metaphors are widely used are switches within applications to toggle features or actions on or off. The Settings application on iOS and watchOS is a great example of where you will find multiple switches to toggle Wi-Fi, Bluetooth, Sound, Mail and other various features both within hardware and software. It's common for a user to associate the metaphor with a light switch having binary controls of either being enabled or disabled.

Figure 4.2.4-7 shows an example of how feedback is utilized in the native Calculator application. When a user selects one of the calculator buttons, it will be highlighted and will also make an audible noise to indicate feedback to the user. Figure 4.2.4-8 shows an example of how metaphors are utilized in the native Settings application. Buttons toggling certain functionality are often depicted as switches to easily inform the user of a basic enable/disable function.

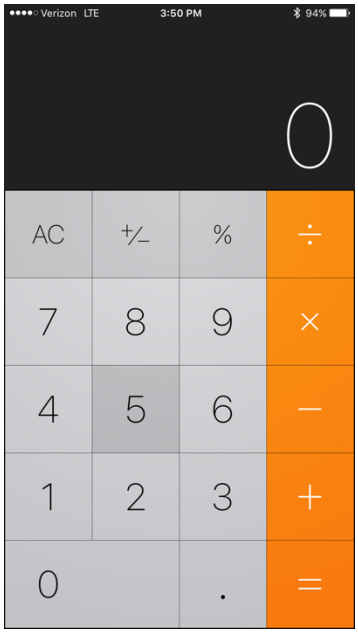



*Figure 4.2.4-8*
*Native Calculator App*

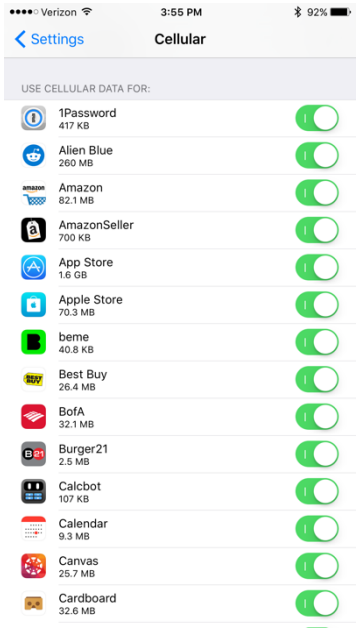



*Figure 4.2.4-9*
*Native Settings App*

User Control is the principle in which it may not be intended for the device to make all necessary decisions, but may prompt the user to choose an action to be performed or modify expected controls when applicable. It is not always best for the user to make unnecessary choices, such as automatically connecting to a familiar Wi-Fi connection or always prompting the user to connect. It is definitely necessary for the user to be prompted when confirming whether or not they want to erase all of the contents on their device. The intention of user control is to find the equilibrium between the user deciding actions and the device automatically doing so when intended.

# 5.0 Research

This section goes into the various examinations that were performed in finding any available options that are similar to the intentions of the project. The research performed impacted the implementation of certain functionality and features that may be widely accepted by the target market if the project was commercialized.

We performed several in-person and email interviews with various coffee professionals who, together, have nearly a century of combined professional experience dealing with roasting and distributing coffee. These interviews gave us a much clearer picture of not only the roasting process, but the industry as a whole. They helped shape our other research and helped us to ask ourselves better questions as we approached the rest of our research process.

## 5.1 Existing Projects & Products

This section goes over the existing products or options for wireless coffee roasters. The subsections compare both commercial and do it yourself options to show the differences and similarities in both implementation and functionality.

### 5.1.1 Commercial Options

There is a niche market for appliances that are now able to be controlled through a smartphone or wireless interface due to convenience and usability. People are very familiar with the interfaces on their smartphone and various home appliance companies are finding new and innovative ways of penetrating the market through users accessing and controlling their appliance through a smartphone application. Two successful and popular examples of this are the Phillips Hue and Nest Thermostat. Both mentioned are home automation devices that can be controlled through a smartphone application and are utilized in many homes across the globe. The success of these devices is largely due to how they are conveniently interfaced and controlled through a simple smartphone application that is both reliable and user friendly.

The market and demand for smart kitchen appliances is substantially less compared to the home automation market mentioned. It is not difficult to purchase a coffee maker or roaster online or at your local department store that can achieve the task of roasting and brewing fresh coffee beans. However, it is not as easy and as it turns out, it is impossible to purchase a coffee roaster that is controlled wirelessly through the same methods at this moment. Due to the unavailability of such a device, it provides one reason for deciding on this project due to it being a new market that is open to new ideas and opportunities. Home coffee roasting is increasingly rare compared to brewing and the project is aimed to close the gap for many individuals to enjoy.

Some popular coffee maker companies such as Keurig or Cuisinart have not decided to get into the smart coffee roaster market at this time. It is possible that more well-known coffee maker companies will decide to venture into the market once they see the response and demand from the smaller and lesser known companies. Two results were obtained for commercial smart coffee roasters that are about to release to the market. Smart roasters from Behmor and Ikawa are the two options found with accompanying websites explaining their features and capabilities. They are both very different in design, but have similar features and are controlled through an iPhone application.


*Figure 5.1.1-1*
*Behmor Connected Coffee Roaster [7]*

*Figure 5.1.1-2*
*Ikawa Coffee Roaster [8]*

## 5.1.1.1 Features

The Behmor is a drum style coffee roaster that is roughly the same size as a conventional toaster or convection oven. The size makes it adequate to be used on a kitchen countertop with minimal impact on space. Drum roasters are typically chosen over air roasters for the consistency of obtaining an even roast and also their ability to roast in larger ranges from light to dark roast profiles. The roaster has a door on the front that opens for easy access to the drum for the loading and removal of coffee beans. The drum is removable for the purpose of cleaning and

the removal of chaff as a result of the roasting process. The maintenance and clean-up of the roaster is nothing more involved than a conventional toaster. The low maintenance and up keep makes it great for consumer kitchen use. Quick start controls reside on the front of the roaster for minor adjustments and control of a roast in progress. The Behmor is able to roast 1 pound of coffee beans, which equates to roughly 80 typical cups of brewed coffee. The roaster has built in smoke suppression which makes it usable for kitchens. There is no worry of setting off a smoke detector due to any excess smoke from the roasting process.

Aside from the actual hardware features for the roaster, it is able to be controlled completely by the companion iPhone application. The application can perform various functions from starting and ending a roast, to viewing a roast in progress. Roast profiles and recipes can be added and viewed within the application to decide on how you want the coffee beans roasted based on time and temperature. Notes can be taken on completed roasts in order to keep track of the end product and provide real time feedback of quality and taste. The roaster appears to be designed very well for kitchen use with its features and design and it should allow consumers the ability to roast fresh coffee beans with convenience and quality. The website states that the roaster is coming soon with no estimated arrival date for online retailers or on store shelves.

The Ikawa is an air style coffee roaster that is roughly the same size as a conventional food processor or juicer. It has a much smaller base than the Behmor, but is noticeably taller and slimmer in design. The coffee beans are loaded from the top and there is a knob that twists to transfer the beans from the loading chamber to the roasting chamber. The roasting chamber spins while hot air enters to heat and roast the beans. There is a small container that the chaff feeds into during the roasting process. When the roast is complete, you replace the chaff container with an empty container for the roasted beans to drain into. The containers for the chaff and coffee beans are easily removable for retrieval and clean up. There is no other maintenance required to complete a roast and maintain the roaster. The Ikawa is able to roast 1 to 2 typical cups of brewed coffee. This is significantly less than the Behmor that is able to roast around 80 cups. There is no explanation for how the roaster deals with smoke suppression, but based on the sample video on their website, it appears that the roaster takes care of the smoke suppression as well.

Additionally, to the roasters compact and elegant hardware design, it is able to be controlled completely by the companion iPhone and Android applications. The application is used to follow the roast in progress by showing a live graphical representation of roast data including the roast profile, temperature and the current stage in the roasting process. When the roast is complete, the application shows a notification that roasting is done and it provides the ability to export and save the roasting data. User accounts can be made to keep track of unique user information for completed roasts. Roast profiles can be viewed, added and shared between other users. There is also a feature to mark when the 1st and 2nd cracks have

occurred during a roast. The Ikawa's application offers more features and ability to view live data compared to the Behmor's application. The website states that the roaster will be available for delivery after May 2016.

### 5.1.1.2 Cost

One reason that consumers choose to not roast their own coffee beans aside from the extra work involved is the cost of purchasing their own coffee roaster. Coffee roasters can easily range from costing less than $100 to over $1000 when comparing a few online and in department stores. Some of the primary deciding factors of the large difference in cost is due to the amount of features, quality of the roast and usability of the product itself. Consumers don't want to spend the extra premium if the benefits don't outweigh the cost to roast themselves at home. Unfortunately, the cost of the roaster largely determines the quality of the coffee roast itself.

Behmor does not list how much the roaster is going to cost on their website or any resources online. However, it can be assumed that it will cost more than their existing customizable drum coffee roaster that is being sold online. Behmor currently sells a customizable drum coffee roaster for $369. The roaster is designed very similarly to the soon to be released smart coffee roaster. The design includes a similar layout with a large door on the front to load and unload the coffee beans. The front panel has controls to increment the time for a roast, choose from 5 preset roast profiles, start and end a roast as well as specifying the weight of the coffee beans being loaded into the drum. The roaster is controlled completely through the onboard controls and does not have any wireless capabilities. Below is a side by side comparison of Behmor's currently sold customizable drum coffee roaster and the soon to be released "connected" customizable drum coffee roaster.

*Figure 5.1.1.2-1*
*Behmor Coffee Roaster [9]*


*Figure 5.1.1.2-2*
*Behmor Connected Coffee Roaster [7]*

Ikawa has their home coffee roaster available for pre-order for £600 which equates to roughly $860. International shipping is available for the coffee roaster if anyone is interested in purchasing from the United States. The technology, features of the smartphone application and elegant design appear to have played a large role in

the premium cost. The coffee roaster is not your typical black box with metal panels and onboard controls. The design is very simple with only a single switch for powering on the roaster and a button to start a roast. The choice of materials from a glossy white exterior to the glass panels make the design stand out compared to the conventional appliances on most kitchen countertops (microwave, toaster, conventional oven, etc.). The Behmor coffee roaster blends in with most existing appliances due to its familiar design and materials used.

As seen from comparing the options provided by Behmor and Ikawa, it will be very expensive for most consumers to enter the market of home connected coffee roasting. There is a large premium involved when adding the ability to control the device through wireless communications, but aside from that, the coffee roasters themselves are fairly expensive compared to most existing kitchen appliances that consumers already own. From a cost benefit perspective, most consumers would agree that it is not worth the extra premium and would prefer to just purchase coffee beans that are already roasted on store shelves or local roasters.

## 5.1.2 DIY Options

There are options when it comes to how one wants to roast coffee beans. Aside from a smart coffee roaster that saves your personal preferences, there are the options of using a popcorn popper or a metal grate with a camp stove. A popcorn popper coffee roaster operates by adding the beans to the popping chamber and having them roast in the same manner the popcorn is popped. Using a metal grate with a camp stove is a little more primitive, but it is less expensive. The least desirable option is to roast the beans using a frying pan. Although this is by far the cheapest and most readily accessible option for roasting coffee beans, it is horribly prone to user error, especially someone who is not experienced with the stages of the process or how to correct any mistakes along the way.

While all of these options may present certain benefits, however slight, they all require a great deal of knowledge of the roasting process. Many inexperienced users try to get started roasting by incorporating one of these methods, and becomes discouraged when they ruin the entire batch of beans. Even an experienced user can have trouble keeping a roast consistent or achieving the roast they want without a machine that is specifically designed to roast coffee beans. Aside from the very real possibility of completely ruining the batch of beans using one of these methods, these methods also make it incredibly hard to even achieve a proper or a specific roast.

### 5.1.2.1 Features

There aren't too many exceptional features to the popcorn roaster. It offers a similar amount of heat and surface area for the beans to roast, however, it lacks a proper agitator to stir the beans as they roast. Since popcorn is usually popped in smaller batches, there is no need for an agitator in the original design. When using

this method to roast coffee beans, efficient air flow and agitation are a necessity. Coffee beans are much more susceptible to burning if they are not constantly moved around and evenly exposed to the heat. With the lack of an agitator, the user must remain at the popcorn machine and consistently stir the beans at regular intervals. The popcorn machine also does not have temperature control, so there is only one possible roast you can achieve from the device.

The metal grate drum roasting device is a bit more practical than the popcorn roaster. It is not automated like the popcorn machine, and you do have to continuously turn the drum to keep the air flow and heat exposure of the beans consistent. However, by using the camp stove, the heat that the beans experience is not consistent throughout the drum. Again, the temperature is a constant, so time becomes the major factor in the roasting process. There is no temperature sensor to know when the beans are at the temperature that they need. Instead, the user has to watch the beans as they change color and be able to identify when they are at the right stage of the roasting process to give them the type of roast they want. Though, even with doing this, there are a limited number of roasts that are available to the user.

Using a frying pan offers little to no features. The heating element on the stove can reach temperatures much higher than needed, or even desired, for the most extreme coffee roasts, such as a French roast. The heating element is also very hard to accurately calibrate and gauge what temperature it is at currently. There are also variations from stove to stove on the settings of temperature, so a certain process that may produce decent results on one user's stove may be completely obsolete on another user's stove, and render the entire batch of beans unusable. While the frying pan method does offer the most even distribution of heat out of these three options, again the user must consistently agitate and watch the beans as they roast in order to attempt to prevent burning the beans. When the coffee beans roast, they naturally produce smoke and steam. Being open to the air, like the metal grate roaster allows for these contaminates to be released into the surrounding environment, so proper ventilation and preparation is required to maintain safe air quality. With the popcorn roaster, there is at least some sort of chamber for the smoke and steam to be suppressed. That suppression can cause the smoke and steam to infuse a burnt and acrid flavor into the beans however, and even if the roast was successful the flavor may ruin the entire batch.

#### 5.1.2.2 Cost

Popcorn making machines range anywhere from $50.00 to hundreds of dollars, depending on the size and quality of the machine. At the lower levels, the machines will get be able to roast the coffee beans, but the quality will not be comparable to an actual coffee bean roaster. A metal grate and camp stove also range from $25.00 to $75.00 depending on the size of the grate and the size and burn time of the camp stove. Although this is also a cheaper option, the roasting operation itself is the most intensive, and requires a good amount of knowledge from the user to

be able to actually achieve the results they want. The frying pan method is by far the cheapest, but with the inconsistency from stove to stove, the material the frying pan is made of, the experience of the user, the exposure of the user to smoke and steam, and the consistent necessary monitoring of the process, makes it the least desirable option.

While cost is and should be a factor in any purchase, compromising the quality of the roast and even risking making the beans unusable eliminates these options for a user who wants a consistent and effective option for roasting their coffee beans.

## 5.2 Motors & Microcontrollers

This section introduces and compares many of the major types of microcontrollers and electric motors. The subsections explain the different characteristics of the devices, and compares the benefits towards our project based on those characteristics.

### 5.2.1 Electric Motors

There are three main types of electric motors, defined by the electrical properties that they take advantage of; these are: piezoelectric, magnetic, and electrostatic. Another classification is based on the power source they rely on: direct or alternating current. These motors contain a commutator which is used to reverse the input current on the armature, thus allowing the motor to continue to turn. We will be using a brushless DC brushless and Stepper motor in our design. One of the major benefits of brushless DC over brushed DC is the lack of friction.
Brushed DC motors rely on a metal or carbon filament moving over the fixed magnets on the stator to provide self-commutation. Although DC brushed motors are very reliable and the initial cost is lower, with repeated and vigorous use of the motor the brushes can wear down. In our application, the drum of the roaster will be experiencing high heat and be turning frequently with every roast, so we chose to use a brushless DC motor for its longevity and severely smaller need for maintenance.

Stepper motors operate by separating the rotation of the motor in to distinct equal steps. Stepper motors are unique in that when a voltage is applied to its terminals the motor only turns a predetermined amount, as opposed to a regular DC brushless motor where the rotation is constant as long as voltage is applied to its terminals. A stepper motor has sets of toothed electromagnets around the outside of a gear like rotor in the center of the device. Each of these magnets is separate from one another. The electromagnets are able to be turned on individually, causing the teeth of the rotor to be attracted to that electromagnet. When the teeth of the rotor connect with the electromagnet, the teeth of the next electromagnet is only slightly misaligned with the teeth of the rotor. The powered electromagnet is turned off and the next on is turned on, causing the rotor to rotate towards that electromagnet. An integer number of cycles are required to complete a full rotation

of the rotor. By increasing the number of cycles, the precision of each step can be increased. The switching of power to the electromagnets is usually provided by a square waveform from a microcontroller.

The rotor of a stepper motor is either a dedicated magnet or a regular piece of metal. With a dedicated magnet, the stepper motor relies on the attraction of the rotor to one magnet while being repelled by the others. Stepper motors that use a regular piece of metal operate based on the reluctance of the rotor to move to an electromagnet that if further away. The rotor will align with the electromagnet that is closest to its teeth. Some stepper motors use both synchronous and variable reluctance to lower the construction costs while also maintaining high power density.

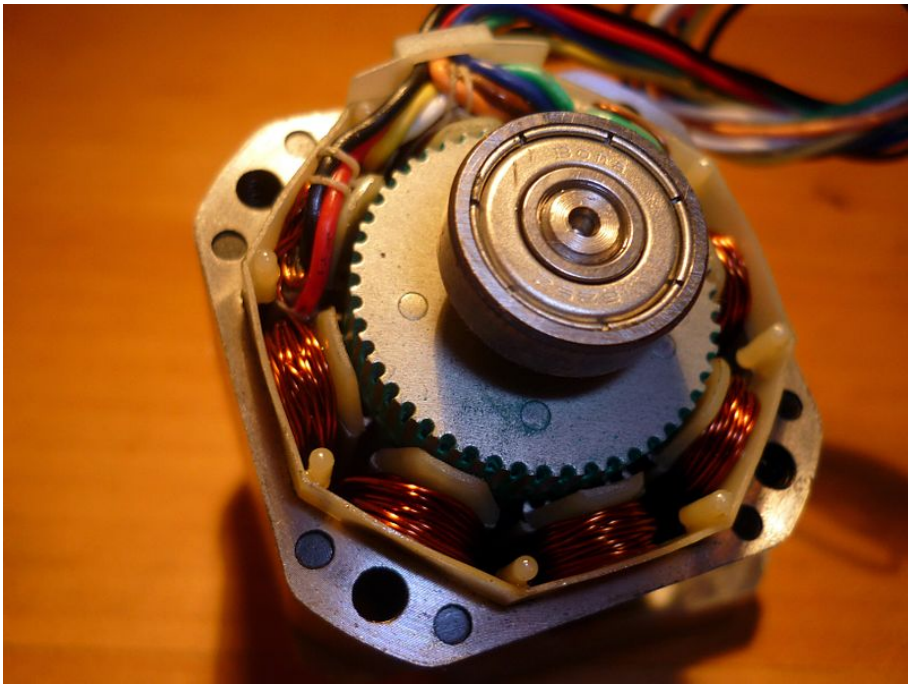


Figure 5.2.1-1: DC Stepper motor [10]

AC Induction motors operate by inducing and collapsing a magnetic field around a rotor. AC induction motors can operate without direct connections to the rotor and operate with low power loss through heat or torque disturbance. Most AC Induction motors operate using a 3 phase power supply. This allows for rotor to continue to rotate towards the powered induction coils wrapped around the stator. As each phase of the voltage hits the corresponding induction coil, the magnetic field draws the rotor towards it and cause it to rotate. This process continues to achieve the full rotation.

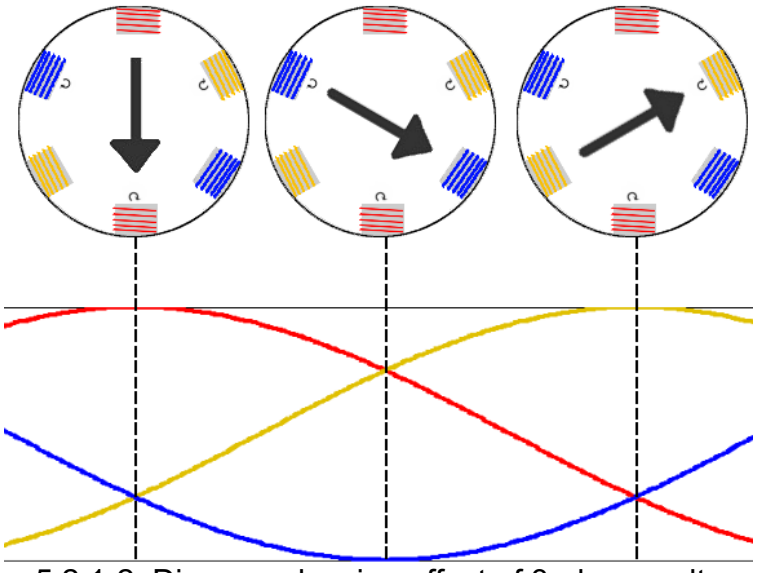


Figure 5.2.1-2: Diagram showing effect of 3-phase voltage [11]

AC Induction motors also boast high efficiency, primarily due to the lack of friction. Unlike DC motors, AC induction motors have a higher start-up toque, that is, the torque required to start the rotation of the motor and maintain rotation.

## 5.2.2 Microcontrollers

Microcontrollers are measured in terms bits, size of memory, and the instruction set used to communicate with the chip. Most MCs operate with either 8, 16, or 32-bit architecture, although some newer MCs are now able to run on 64-bits. 8-bit architecture is useful for arithmetic and logical operations, 16-bit architecture is quicker, and usually more precise. 32-bit architecture is useful for automated processes. Another metric is the type and size of memory on the MC. If the MC has its own memory it's considered embedded memory, otherwise it is called external memory. Modern MC's memory range from 32kB to 128kB, in factors of 32kB. There are 2 major types of instruction sets: complex instruction set computer (CISC) and reduced instruction set computer (RISC). CISC allows for the user to enter a single command to replace multiple commands. RISC does not allow for that, but it shortens the clock cycle per operation, usually allowing for quicker operating times.

There have also been highly successful combinations of certain specifications for MCs. One of the most notable is the 8051 series. The 8051 series is an 8-bit chip with a clock frequency of 12MHz, 128kB of RAM, and 4kB of ROM. Another popular model is a Peripheral Interface Controller (PIC). These chips, unlike embedded microcontrollers, are standalone chips that coordinate with outside hardware.

Initially the microcontroller that was going to be used was an Arduino Uno with an ATMEGA328P-PU. We started to consider the MSP430G2553, MSP430F5529,

and MSP432P401R. The ATMEGA is a very powerful chip that also requires a significant voltage to operate, whereas the MSP430 and MSP432 chips only require 1.62 to 3.7 volts, nearly half the suggested voltage for the Arduino chip. Its clock cycle is 16MHz, compared to the 25MHz of the MSP430F5529 and up to 48MHz with the MSP432P401R. The MSP432P401R also boasts 256kB of RAM and the MSP430F5529 offers 128kB of RAM, where the other two chips only offer 16kB and 32kB. With the amount of services in the device, communicating through Wi-Fi, a touchscreen, and monitoring the current state of the roasting process, the larger embedded memory is a big factor. The active power difference between the F5529 and the G2553 is only 74 µA/MHz. However, the active power required for the MSP432P401R is only 90µA at 128kHz.

The physical size of the chips is comparable on all fronts and a non-factor. Another key advantage of the MSP430 and MSP432 line of microcontrollers over the Arduino line is the access to Texas Instruments' (TI) support and product line. Considering the significant performance increase and the drastic difference in power consumption between the MSP432P401R and the MSP430F5529 and the MSP430G2553, we decided to eliminate an Arduino MCU as an option.

When we began to compare the three MSP microcontroller units, it readily became apparent that the MSP430G2553 would not be able to handle the number of peripheral device we needed in out design. The MSP430G2553 also had the lowest memory of the three chips, with only minor benefits in power consumption. If we were to incorporate the MSP430G2553 into our design, we needed to coordinate at least two them together to manage the other components. Due to the lack of SPI pins, lower memory, and the necessity for at least two chips, we eliminated this MCU from our design consideration.

At this point, we already had access to the MSP430F5529. This MCU appeared to be able to handle the size of the code and the number of peripheral devices we wanted to use. We began to incorporate a TI Wi-Fi booster to our design until we realized we were unable to use a booster and instead needed to use an independent Wi-Fi chip. With this change in our design plan, we began to construct our hardware designs and components, we realized that even this MCU would not be able to support the peripherals due to the lack of sufficient SPI pins. Due to this fact we switched to our final and current MCU choice, the MSP432P401R.

## 5.3 Sensors

This section covers the temperature, pressure, and audio sensors we will be incorporating in our design. The subsections go into the details for each of these categories of sensors, as well as how they will be applied, integrated, and utilized in the design.

### 5.3.1 Temperature

For our design we will be incorporating electronic temperature sensors, as opposed to a mechanical sensor. One of the major categories of electronic temperature sensors are thermistors. Thermistors operate on the principle that changing the temperature of a device inherently changes its electrical resistance. Different materials offer different applications ranging from high temperature, low temperature, cryogenic temperatures, high magnetic fields, and high radiation fields. Depending on the material used to manufacture the thermistor, such as Ruthenium Oxide for cryogenic range temperatures, they can be used in very specific environments. In a general range of temperatures, usually $0^{o}$ C to $100^{o}$ C, resistance changes tend to be linear. As the temperature of the device changes, the electrical resistance changes as well, and can be measured to determine the change in temperature. There are both positive and negative temperature coefficients associated with a material. For materials with a positive temperature coefficient, the resistance of the material increases as temperature increases. For a negative temperature coefficient, the resistance decreases as temperature increases. Depending on the material used, this change in resistance can be linear over a long range, or steep over a short range if the application call for very specific measurement.

Another way to measure temperature is by using thermocouples. Thermocouples utilize the thermoelectric effect, that is, a change in temperature causes a change in resistance, and therefore a voltage drop will occur. Two materials, usually with opposite Seebeck Coefficients, are connected at one end and placed into the area where the temperature needs to be measured. Using the difference in temperature from the end of the disconnected materials and the connected end in the area with the desired temperature, the change in voltage can be measured across the disconnected and the temperature can be calculated from that measurement.

Having a higher Seebeck coefficient means that more voltage can be produced per unit Kelvin change in temperature. A negative Seebeck coefficient means that a negative voltage will be produced by that material. By connecting two materials, one with a negative and one with a positive Seebeck Coefficient, the two voltage differences can be added together. The most efficient materials to use in a thermocouple are those with equal but opposite Seebeck coefficients. The amount of voltage supplied from a thermocouple, and indeed the sensitivity of the device as well, can be increased by connecting more of the individual devices together. Much like the cells of a battery, the more thermocouples that are connected, the larger the voltage difference.

The advantage to thermistors is their higher reliability, accuracy, and repeatability. Thermistors are also more practical in more sensible temperature ranges. However, thermocouples can be useful up to thousands of degrees Celsius. Thermocouples are also more compact. The time it takes a thermocouple is also much lower than a thermistor.

Another major temperature sensing device is a silicon bandgap temperature sensor. These are useful because they can be designed directly into and integrated circuit, and they are inexpensive to produce. One of the issues with silicon bandgap temperature sensors, however, is each sensor has to be independently calibrated to be accurate.

## 5.3.2 Crack Detection

In order to design a better control system, we needed some form of feedback on the roasting process. At different stages in the roasting process, the beans will make a cracking sound, commonly referred to as the first crack and the second crack. We intend to use this sound or vibration in order to obtain this feedback and adjust the roasting time and temperature accordingly. The options that we have considered are an accelerometer and a microphone.

An accelerometer is a device that detects acceleration and encodes it as an electrical signal. This can be used to detect the vibrations that are caused by each respective crack in the roasting process. This signal could then be put into an analog to digital converter on the microcontroller and processed in order to determine if a crack has indeed occurred. One accelerometer is required for each of the three axes. Multidimensional data may be required to filter out noise caused by both internal and external vibrations.

A microphone operates on a similar premise, but it acts by detecting vibrations in material or air which are induced on a diaphragm and then converted into an electrical signal. There are multiple pickup patterns for this transducer which are able to adjust the performance for this application. Omnidirectional picks up sounds from every direction with largely the same magnitude. Unidirectional has a “cone” that picks up in a single direction and largely rejects sounds from other directions.

### 5.3.2.1 Microphone

A microphone can detect a large variety of sounds. This unfortunately makes it highly susceptible to interference. In order to fix this problem, a filter must be used. After the signal has been filtered, a simple threshold detector can be implemented in software to detect if indeed a crack has occurred.

There are multiple parameters that affect microphone performance for our application. Notably, pickup pattern and pickup type. Pickup pattern determines how sensitive the microphone is to sounds from different directions. The principle pickup types are omnidirectional and bidirectional, or figure 8. Other patterns can be realized by aligning these pickups together and wiring them out of phase from one another. The pattern type can be modified by adjusting the ratio between these two signals. Figure 5.3.2.1-1, below illustrates the signal responses of these pickup patterns. Angle 0 indicates the front of the transducer.

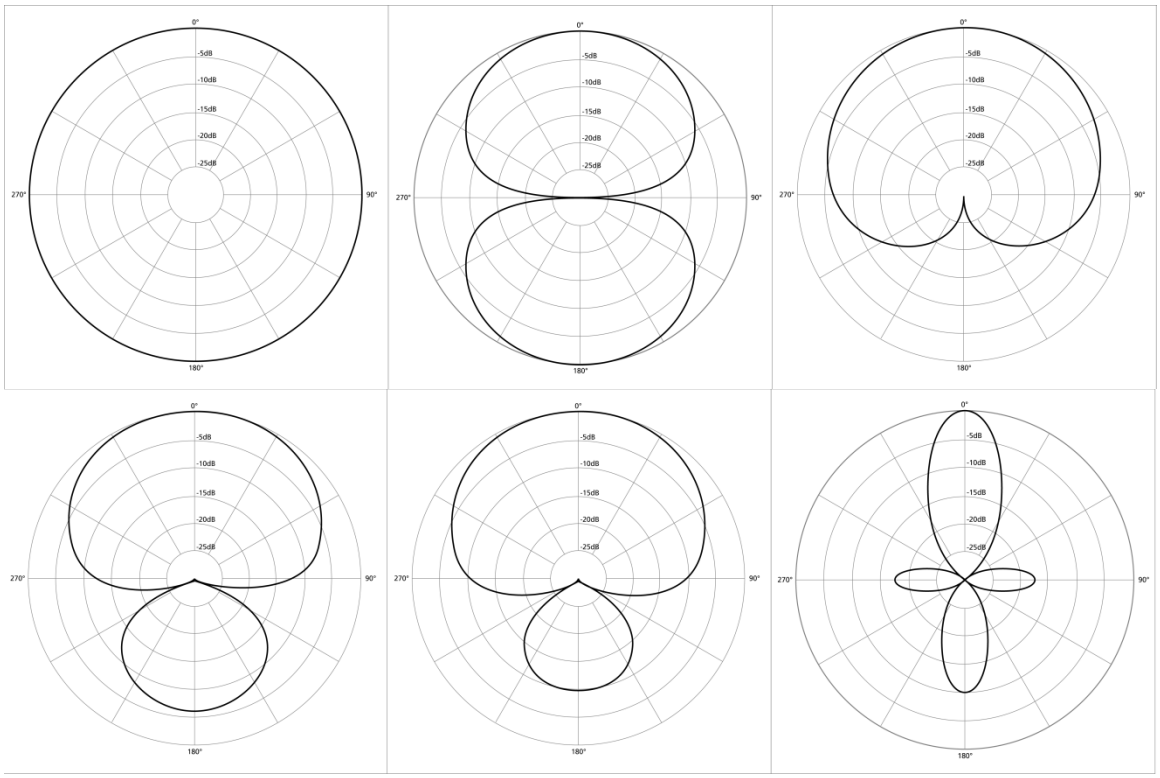



*Figure 5.3.2.1-1(a-f): a*. Omnidirectional, b. Bidirectional, c. Cardioid, d. Hypercardioid, e. Supercardioid, f. Shotgun. Produced by Wikipedia user Galak76, used under terms of Creative Commons license CC BY-SA 3.0 [11]

We needed a pickup pattern that would provide a strong input from the front of the transducer while rejecting signals from other directions. We initially considered a unidirectional, or cardioid pickup, as seen in figure 5.3.2.1-1c, but we wanted to reduce the strength of signals coming in from the sides. This made us examine super and hyper cardioid patterns, as seen in figures 5.3.2.1-1d and 5.3.2.1-1e, respectively. In examining those patterns, we realized that we could accept a signal that, with our microphone placement, came from the back of the microphone. If we attach the microphone to the bottom of the roaster, Signals from the back will only come from the surface that the roaster is set on. Under standard operating conditions, this should only produce a signal that is attenuated to the level that it should fall beneath the threshold that we set for our crack detection software. With this in mind, the standard unidirectional pickup pattern as seen in figure 5.3.2.1-1b seems like the best option to go with. While the “shotgun” pickup, as seen in figure 5.3.2.1-1f, is common for directional microphones, it could not be considered for our application due to the response lobes at $90^{o}$ and $270^{o}$, we were therefore able to narrow our pattern selection to a single choice.

For pickup type, we wanted a type that has an approximately linear signal response across the spectrum and that could easily work with the rest of our design. Options for these included dynamic, ribbon, condenser, and electret condenser microphones.

Dynamic microphones operate by moving an inductor attached to a diaphragm. The coil of the inductor is free to move in a constant magnetic field. This movement through the field induces current and thus produces an electrical signal. This construction yields a robust microphone that is able to withstand a large variety of operating conditions, such as environments with high temperatures, mechanical shocks, or moisture. This does make it attractive for many applications where audio must be transduced. Because the diaphragm must move a larger coil, each microphone element is designed to work best within a set range of frequencies and has a highly non-linear response.

Ribbon microphones operate in a similar manner to dynamic microphones. They have a long metal film suspended within a horizontal, static magnetic field. The motion of the metal through the magnetic field produces a current that yields an electrical signal. They naturally follow the figure 8 pattern, or bidirectional, pattern that we chose to go with, but they are very fragile. In order to work, they must have a very thin, long strip of foil. The foil is easily torn or warped. If moisture condenses on the metal diaphragm, it will alter the frequency response of the microphone. In general, they are closer to having a “flat” response than dynamic microphones, but the fragility and sensitivity to moisture that they exhibit makes them seem like more of a liability than asset in such a project as this, where food is being heated and cooled and produces a steam exhaust.

Condenser microphones follow a similar mode of operation to ribbon microphones. They have two metal plates with an air gap between them. These plates act as a parallel plate capacitor. Generally speaking, one plate is thicker than the other. The thinner plate acts as a diaphragm. A voltage is applied across the plates of this capacitor and when the diaphragm element is vibrated, it changes the capacitance of the capacitor and thus changes the electric field. This change in electric field causes an electrical current that transmits the electrical signal of the sound produced. Condenser microphones are generally more sensitive than dynamic microphones, but less sensitive than ribbon microphones. The diaphragm is able to be made much smaller than that of a ribbon microphone. It is also able to be partially sealed and still transduce sound. This avoids the moisture sensitivity of the ribbon microphone. It is capable of achieving a similar level of “flatness” as the ribbon microphone.

A subset of the condenser microphone is the electret condenser microphone. A traditional condenser microphone requires both positive and negative biasing in order to prevent negative clipping. An electret microphone has a sheet of insulator in it that contains charged particles. This sheet is formed in an electric field so that these ions will all be oriented in the same direction. Once the film is formed, it produces an intrinsic electric field. This gets rid of the issue of negative clipping and allows the microphone to operate with a single voltage applied. This type of microphone exhibits many of the properties of a traditional condenser microphone while simplifying the requisite support hardware. These microphones are often used in phones and other consumer devices that experience regular physical

shock excess moisture. They are able to reliably and consistently operate in such conditions with minimal distortions to the response. These characteristics make them ideal for our application.

#### 5.3.2.2 Accelerometer

Accelerometers transduce physical vibrations into electrical signals. They come in many varieties. An accelerometer may be less susceptible to external noise than other types of transducers because they do not readily transduce vibrations through the air. This makes them attractive for a wide variety of applications. At rest on the Earth, an accelerometer will read 9.81m/s$^2$ upwards. This is because they are made to be considered at rest in a free fall. Some types of accelerometers are made for high temperature uses, such as on active volcanoes. This does show that an accelerometer may be useable in our coffee roaster.

The coffee roaster that we are retrofitting has a rotating, mesh drum that holds the coffee beans. This drum is connected to an axle that is spun by a direct drive motor. This motor is damped from the rest of the chassis, presumably, to prevent noise from vibrations. This makes placement of an accelerometer tricky. It could be attached to the roasting drum or axle using a slip ring. This would allow it to rotate without pulling wires off of it. However, a problem arises due to the rotation itself. The rotation will constantly produce a slight acceleration outwards. At the same time, the at-rest acceleration will still be seen by it and will be in a constant state of change. The semi-regular spilling of beans over mixing ridges provides yet another challenge to the use of an accelerometer. These problems combine to make it a less than ideal solution to detect a crack.

### 5.3.3 Pressure & Weight

In order to have better control over how the beans are roasted, it is desired that their weight is measured. Multiple strategies and technologies were examined in order to accomplish this. We considered including a scale on top that would allow for easy measurement. This would be a simple solution to implement, but it would leave off some potentially valuable data – how the weight of the beans changes over time. Also, it would require more routing of power and potentially data wires through the body of the existing roaster we are retrofitting would require some form of additional machining. As this idea was explored, it was determined that it was not as simple as it first appeared nor did it offer advantages that other potential methods could provide.

The idea of measuring weight applied to the axels of the roasting drum was then considered. This would provide additional data that could potentially be useful in improving quality of roasts and may not require significant modifications to the chassis of the roaster. One concern with this method however, is heat; whatever measurement transducer is chosen, it will need to be able to not only withstand the high temperatures associated with the roasting chamber – sometimes up to 472°F

– but it will also need to be able to provide accurate data at said temperatures. As more thought was put into this, it was determined that no transducer could reasonably provide accurate measurements through such a range without using some form of correlation with data from a temperature sensor. This lacked the simplicity that was sought from a good design.

A further strategy that was considered, was to attach the pressure sensors to the bottom of the entire roaster and measure the difference between the roaster at rest with no beans added, and the roaster when it was filled with beans. This would afford us the opportunity of collecting changes to weight over time while avoiding problems of high temperatures. Unfortunately, it would require the ability to accurately measure much higher weights than was initially considered. At the same time, it brought about an idea for a casing for our overall design, as can be read about in section 2.3.1.

In our initial investigations into pressure sensing transducers, we came across load cells. As we looked into load cells at different electronics distributors, we discovered that they had highly prohibitive prices, often two orders of magnitude more than we were willing to pay for them. This led us to seek out other technologies, which is where we discovered force sensitive resistors from interlink electronics. These devices act similar to gels in computer keyboards and change the value of resistance that is seen for them based on applied pressure. After speaking with a sales representative from Interlink, it was determined that this technology would not work in our application. Force sensitive resistors are subject to deformations that can alter their properties when they have a significant force applied to them for a long time – like that of a coffee roaster resting atop them. Further, while they are able to sense the differences between applied forces, they are not capable of acquiring precise measurements of applied force. The representative from Interlink pointed us back to load cells.

We realized that we needed to either find a cheaper load cell for our application else we would need to remove this feature. We realized that there were digital kitchen scales that could be purchased for under $10. This led us to look into the transducers that were used for said scales. We discovered that there were load cells that could meet the requirements of this design that could be acquired far cheaper than the offerings from major electronic distributors. We discovered that the load cells from the electronics manufacturers were designed to be used in scientific instrumentation, such as in analytical balances in chemistry labs, and were far more accurate than what was required for our design. We went with the cheaper load cells for our design.

# 5.4 API

This section goes into detail about the back end server and API system that will communicate with the roaster and front end applications.

## 5.4.1 Software Frameworks

The following subsections describe the software frameworks that are being considered to be used for our RESTful API. But what is a software framework? A software framework is defined as a platform for developing software applications. A framework would include predefined classes and functions that allow for the process of software development to be streamlined. It's all about not having to reinvent the wheel every time development takes place.

### 5.4.1.1 NodeJS

In the development world, JavaScript is on the rise. This has led to some rather drastic changes in web development. Specifically given that nowadays we have the ability to use JavaScript to run on a server in addition to a browser. This is made possible through the framework, NodeJS.

What is NodeJS? It is a JavaScript framework that is open source and has a cross-platform runtime environment which is used to develop server side and networking applications. NodeJS shines in real-time web applications by employing push technology over websockets. Websockets are rather revolutionary in allowing for real-time, two-way connections between the client and server. This feature allows the client and server to exchange data freely.

The primary features of NodeJS include: 1) All APIs in the NodeJS library are asynchronous and event driven, 2) NodeJS is built on the VS JavaScript Engine, resulting in fast performance, and 3) NodeJS applications use a single thread model as far as the developer is concerned, which is highly scalable. Meaning, you do not want to use NodeJS for CPU intensive operations, otherwise you will negate the benefits of NodeJS altogether.

NodeJS being asynchronous means that rather than waiting for data to be received from an API call it would move on instead and simply go back when the data is available thus reducing down time and increasing performance. The big issue that can arise is that the developer can be caught in what is known as 'callback hell'. Which essentially means that one callback waits on data which passes on to another callback which passes on to another callback. Which leads to a waiting state rather than an executing state as you ideally would like. The solution would be to take advantage of the asynchronous modules within NodeJS which allow you to control the flow of a program.

When many developers are asked why NodeJS one of their first reasons is the performance which is due to the V8 JavaScript Engine. The engine compiles and executes the JavaScript code, handles memory allocation for objects, and garbage collects objects it no longer needs.

The traditional web application processing model would be a multi-threaded request-response model. Essentially a client would send a request to the server and a thread would be used to process said request performing the required operations and then sending back a response. The problem being that the number of threads are limited and as we get to a max concurrent request count the other requests would be pushed into a queue. There the request would wait for an available thread to process the request.

The most common framework for testing NodeJS applications is Mocha. Some notable Mocha features include:

- Browser support
- Async Support
- Test coverage reporting
- Asynctest timeout
- Reports test duration
- Highlights slow tests
- File watcher support
- Global variable leakage support
- Run tests that match a regular expression
- Can auto-exit in order to prevent 'hanging'
- Node debugger support
- Can use any assertion library
- Can detect multiple done() calls

Testing is a matter of making use of an assertion library such Chai which allow the developer to write lines such as, `expect(4+5).equal(9);`. The line of code would simply mean that we expect that 4+5 will equal 9. This would be used in our application by passing a function with some preset parameters and setting .equals() to a value that should be the output. If that assertion fails, then we know that a bug exists.

### 5.4.1.2 Java Spring Boot

Most professionals that made use of Java for their projects tended to stick with Spring or J2EE. One of the major issues and plus with those two platforms is the age of the platforms. Much of projects based on either tends to be riddle with deprecated functionality. But although they may have their problems they have remained the gold standard due to their reliability for large-scale, large-team enterprise Java development. But as you would expect, contenders for the top Java platform have risen, in our case we'll be covering Java Spring Boot.

Like I said above the spring boot framework is heavily opinionated library/framework that allows us to create executable Spring applications with a focus on convention over configuration. The annotation @EnableAutoConfiguration is where the magic is found. The annotation is tasked

with automatically loading all the beans that is required by the application. If in your Maven POM there were dependencies for JPA and a H2 driver, then Spring boot would configure your system to have a persistence unit based on H2.

The Spring Boot CLI gives the developer a few key features: 1) ability to compile and run groovy source code and 2) ability to package your application into a self-contained executable jar file. Groovy scripts are an alternative to java files which do not require the developer to create a main method, rather it is automatically created. This allows us to create a fully function restful interface with 5 lines of code. If we wanted to package our application into a self-contained executable jar file then we would make use of the command jar. Our final jar would have the complete compiled application and all its dependencies along with an embedded server by default a tomcat server.

Spring boot makes heavy use of annotations which were first introduced in Spring 3. The idea being that they would replace XML files in their entirety.

For testing a Spring Boot application, we would make use of JUnit. Junit is an open source framework that was designed specifically for testing Java applications. Some of the advantages of using Junit are:

- Simplicity of use
- Can choose between testing a single class or a group of classes
- You the developer are a lot more confident in the overall correctness of your code.
- Makes it clear if you don't test

Given that our API is restful we would need to make use of the package 'org.springframework.test.web.servlet.*' which will allow us to create mock API calls when needed. If we wanted to do a post request test for example our line of code could look like `mockMvc.perform(post("/api/roasts/").andExpect(list));` This request would verify that the post request matches the data found in .andExpect().

## 5.4.2 Hosting

The three big hitters as far as cloud computing are Amazon Web Services (AWS), Google Cloud Platform (GCP) and Azure. Primary categories in which these cloud giants are competing are in: computation, storage, networking, and pricing.

One of the big pulls to cloud computing is the ability to spin up virtual machines in which you are able to customize its size, power, memory and eve the number of virtual machines you would like running. Although you will likely not have a static number of virtual machines running due to the fact that you are able to make use of load balancing which either spins up or down VMs depending on overall load. Furthermore, auto managed database services are also a huge pull towards cloud computing. A huge variety of SQL and NoSQL databases being offered with each service so there is no limit to choice.

You can find the specifics of options among AWS, GCP, and Azure below in table 5.4.2-1.

| **Storage and Databases** | | | | | | |
|---|---|---|---|---|---|---|
| | Ephemeral | Block Storage | Object Storage | Relational DB | Archiving | NoSQL and Big Data |
| **AWS** | Yes | EBS | S3 | RDS | Glacier | DynamoDB, EMR, Kinesis, RedShift |
| **GCP** | Yes | Persistent disks | Google Cloud Storage | Google Cloud SQL | Nearline | CloudDatastore Big Query |
| **Azure** | Temporary Storage – D Drive | Page Blobs | Block Blobs and Files | Relational DBs | | Windows Azure Table, HDInsight |

*Table 5.4.2-1 Cloud Storage Data*

If the user were to want to create a group of isolated VMs for example he/she would make use of a cloud provider's VPC service. User would be able to define the topology of a network, create route tables and subnets, instantiate private IP address ranges and create gateways. The options that are offered at each of the cloud hosting providers being compared can be seen below in table 5.4.2-2.

| **Networking** | | | | | |
|---|---|---|---|---|---|
| | Virtual network | Public IP | Hybrid Cloud | DNS | Firewall/ACL |
| **AWS** | VPC | Yes | Yes | Route 53 | Yes |
| **GCP** | subnet | Yes | | | Yes |
| **Azure** | VNet | Yes | Yes | | Yes |

*Table 5.4.2-2 Networking Options*

A huge factor on whether or not a provider is chosen is their pricing model. Luckily most of the modern cloud providers give the option of either paying on a monthly basis or paying per use. The specific models per provider can be found below in table 5.4.2-3.

| | **Pricing** | **Models** |
|---|---|---|
| **AWS** | Per Hour | On demand |
| **GCP** | Per Minute (at least 10 minutes) | On demand |
| **Azure** | Per Minute (pre-paid or monthly) | On demand |

*Table 5.4.2-3 Pricing Model Comparison*

## 5.4.3 Security

When deciding how to go about securing the communication between all the devices we must keep in mind 1) the types of devices and 2) the performance requirements of our system.

JSON Web Tokens (JWTs) are a standard which define a self-contained method of transmitting data securely between two parties via a JSON object. The tokens are signed by either a secret—essentially a string in combination with the HMAC algorithm) or by making use of a public/private key pair.

JWT are ideal in two situations 1) authentication and 2) information exchange. In the case of authentication, the most common situation is that user would submit their token on each request in order to access resources and routes. An example sequence diagram outlining this can be seen below in figure 5.4.3-1. If JWT is being used for information exchange, we would use the public to encode our data before sending and the private key to decode it.

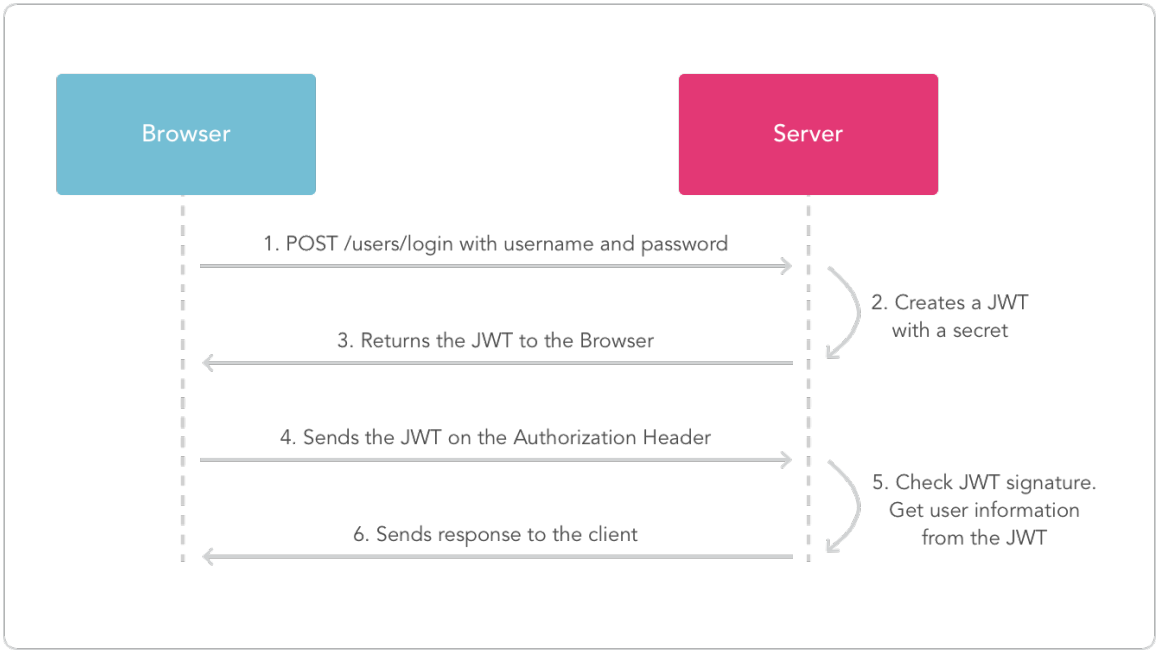



*Figure 5.4.3-1 JWT Sequence Diagram [12]*

## 5.4.4 Database

The data store for this project will store user and roast data for our application. The options for data stores are hugely varied whether we go with SQL or NoSQL. The following sub-sections outline the specifics of SQL and NoSQL data stores and the options available for each.

### 5.4.4.1 SQL

SQL databases are data stores in which you are able to communicate with using the SQL (Structured Query Language) language. It is a standard language used for relational database management. With SQL we are able to perform operations such as updating, inserting, and deleting data. Some popular SQL data stores include:

- MySQL
- PostgreSQL
- SQLite

One of the SQL data stores that we are seriously considering is PostgreSQL. Some notable features include:

- Legendary reliability and stability
- Extensible
- Cross platform
- Designed for high volume environments

Given these features PostgreSQL has quickly become the industry standard for SQL based stores even though it has been out for decades.

#### 5.4.4.2 NoSQL

NoSQL data stores excel in speed and volume which is why as of late they have become so popular. One of the primary advantages of a NoSQL data store is its distributed nature meaning in order to scale the data store we would simply add machines to the cluster to meet the demand. The added bonus to their distributed nature is that NoSQL databases can pretty much be on all the time. Which is a huge advantage in modern business where it can be hugely costly to have a single moment of down time.

MongoDB is currently the most popular NoSQL data store available. This data store focuses on documents which highlight the application design via data modeling strategies. The primary advantages of MongoDB are:

- Schema-less document storing
- Automatic sharding
- Map/reduce processing

## 5.5 Mobile Application

When it comes to building smartphone applications and more specifically wirelessly connected devices, there is an important decision that has to be made regarding what devices and users to develop for. Determining the popularity and market share that a specific smartphone or operating system has will largely impact the penetration for the application to be widely used and maintain success. It can be argued that the two most popular smartphones are Apple's iPhone devices and Samsung's Galaxy devices. Knowing this information raises the decision to develop for either the iOS platform or Android OS platform for the application that will wirelessly control the coffee roaster.

For the purpose of the project and unfortunate time constraints of the summer semester, the decision was made to only develop for one of the two application platform choices. Both platforms will be compared in detail to showcase their options for development over one another and the features they uniquely provide. Later implementation may be added to accommodate two versions of the application to be available for both iOS and Android devices in order to increase market share and ability to be sold to more customers. As far as the devices the group members own, three out of the four group members own smartphones with two of the three owning iPhones, and the third owning an Android device. This situation did not necessarily impact the choice for the application platform that was chosen, but more so the research that was conducted regarding iOS versus Android application development. It was an added benefit to have multiple members test the applications in development on different types of devices,

whether they be iOS or Android devices. The performance and usability of the application will be compared across the different types of devices when available.

### 5.5.1 iOS

iOS is the operating system used on the iPhone and it is used across all iPhone models. There is little to no fragmentation between the different iPhone models and the operating system that is able to run on them. This fact increases the user base that an application would have on the market due to the ability for users with older and newer devices to utilize the same application and receive the same experience to some extent. To put this into perspective, the iPhone 4s that is currently five years old runs the same version of iOS that the current flagship iPhone 6s operates on. Not all consumers are on the latest and most up to date iPhone models and the ability to develop one application that can be used across a wide range of iPhone models make the iOS platform popular for application development and increases large acceptance into the application market.

- iOS 9 is the latest version and is compatible with these iPhone models [13]:
    - iPhone 6s (Released in 2016)
    - iPhone 6s Plus
    - iPhone 6
    - iPhone 6 Plus
    - iPhone SE
    - iPhone 5s
    - iPhone 5c
    - iPhone 5
    - iPhone 4s (Released in 2011)

When developing applications for iOS devices, the developer tools, software development kit (SDK) and first party resources are open source for the public to access. There is no cost to develop an iOS application and side load it onto a device for testing. Xcode is the desktop application used to write, develop and simulate applications on Apple computers. The application is a fully integrated development environment (IDE) that allows for code to be compiled and applications to be executed and running on devices through a software simulator or physically when connecting an iOS device. The ability to run applications directly on user devices make iOS convenient to use and greatly help during the development and quality analysis testing process.

One useful feature that's available when developing for iOS devices is TestFlight Beta Testing. When developing an application, it is always a good idea to allow multiple users to install and test the application before releasing it to the public and eventually the App Store for purchase. With TestFlight Beta Testing, users install a simple TestFlight application on their iOS device that would then provide them access to any beta apps that they have been invited to install and test. TestFlight

Beta Testing has the ability to allow developers to invite up to 2,000 unique beta testers simply through their email address.

Aside from iOS running on the iPhone, the same operating system runs on Apple's iPad as well. iPhone applications are able to be installed on iPads and they automatically scale in size due to the increased display real-estate. When writing applications in Xcode, developers are given the option to either write the application for iPhone, iPad or Universal between both hardware platforms. Since iPhone applications are able to be installed and run on iPads, this adds yet another benefit to iOS in its ability to access more users. Typically, iPad applications tend to include more functionality and features that can take advantage of their larger display. However, one assumption that can be made is that a user that does not own an iPhone may have an iPad. This would provide them the ability to install the same application and take advantage of the same features and benefits but with a larger display to use.

- iOS 9 is the latest version and is compatible with these iPad models [14]:
    - iPad Pro (12.9-inch) (Released in 2016)
    - iPad Pro (9.7-inch)
    - iPad Air 2
    - iPad Air
    - iPad 4th generation
    - iPad 3rd generation
    - iPad 2
    - iPad mini 4
    - iPad mini 3
    - iPad mini 2
    - iPad mini (Released in 2012)

The differences in functionality and amount of information displayed between an iPhone and iPad application can be easily distinguished side by side. The native Mail application on iOS is available on both iPhone and iPad. The Mail application iPad has more functionality on iPad with the ability to see the inbox and email on the same screen. The amount of information displayed to the user for an email differs substantially.

Some examples of wirelessly connected devices that support iOS devices include the Nest Thermostat and Phillips Hue. The Nest Thermostat and Phillips Hue are popular home automation devices that are highlighted for their ability to be controlled seamlessly through their accompanying applications. The Nest Thermostat provides users the ability to adjust the temperature in their homes wirelessly and also without the need of being near the device. The Phillips Hue provides users the ability to change the color of their light bulbs with the touch of a button to accommodate a holiday or mood. Both of these devices are great

examples of how a device that is user friendly and provides added convenience can be well received and popular to a large range of customers.

iOS provides great benefits from development and also market acceptance that the software has an impact on. Both reasons are valid in making a decision whether to choose iOS development over Android development for the purpose of the project. The decision was made to develop the smart coffee roaster application for the iOS platform with all features considered and research that was conducted to identify the best choice from development and commercial perspectives. The development tools, TestFlight Beta Testing and market share for iOS make it the best choice for the purpose of the project and experience that will be gained in the development process. The next section will explain in detail the features that the Android operating system provides from development and market share, as well as the reasons iOS was chosen in being the only application platform for the smart coffee roaster at this time.

### 5.5.2 Android

Android is the operating system used across Google, Motorola, Asus, HTC, Samsung, LG, Huawei, Sony, ONEPLUS and Lenovo devices. The mobile operating system is developed by Google and it is able to run across their own first party Nexus devices and also third party manufactured devices, such as the Samsung Galaxy. It may sound like an added benefit for the amount of devices and manufacturers that support the Android platform, however, this brings up the idea of fragmentation that Android devices are plagued with. Fragmentation is brought up with the fact that many manufacturers such as Samsung, Motorola and HTC have developed varieties or custom versions of Android to run on their devices, and as a result, some applications or features may not be supported across all of the devices when a new version of the Android operating system is released. Aside from features and ability to install the same operating systems, a lot of devices may also not have the option of installing the newest version of an Android operating system or the release may be delayed for certain devices.

When developing applications for Android devices, similarly to iOS, the developer tools, software development kit (SDK) and first party resources are open source for the public to access. There is no cost to develop an Android application and side load it onto a device for testing. Android Studio is the desktop application used to write, develop and simulate applications on Windows, Mac OS X and also Linux. This is greatly different when it is only capable to develop an iOS application on a computer running Mac OS X. This provides developers more options to jump into Android development without requiring specialized hardware to even install the software development kit (SDK) and integrated development environment (IDE). Similarly, Android Studio provides developers the option of running their application on an emulator or by physically connecting their devices for testing.

One feature that Apple provides its developers is TestFlight Beta Testing. As explained in the iOS section, TestFlight Beta Testing allows for developers to invite beta testers to install and run applications that are currently in development. Beta testers are invited through email and can easily access the applications that are able to be installed using the TestFlight application on iOS. Unfortunately, Android does not provide a service similar to TestFlight that would allow for widespread quality analysis testing of applications in development. However, there exists numerous third party services that allow for Android beta testing, including options from TestFairy and Appaloosa just to name a few. The option of first party beta testing provided from Apple make it more preferred and convenient over the third party options available for the Android platform.

Similarly, to how the same version of iOS can be installed on both iPhones and iPads, Android has the capability of being installed on tablet devices as well. The tablet devices are offered from most of the same companies that offer their smartphone counterparts, such as Samsung and Google. The same issue of fragmentation arises for tablet devices as well when it comes down to the ability of installing the newest version of Android and their availability to the same features and applications.

## 5.5.3 Comparison Summary

As can be seen in Figure 5.5.3-1, there is substantial fragmentation regarding the Android version that is installed on consumer's devices. The percentages in the figure were collected from the Android Developer About page and the website states the data was collected during a 7-day period ending on March 7th, 2016. It can be seen that over 50% of Android devices are running version 4.4 KitKat or 4.1 - 4.3 Jelly Bean. Kitkat and Jelly Bean were released in 2012 and 2013 respectively, meaning that over 50% of Android devices are running 3 to 4-year-old operating systems.

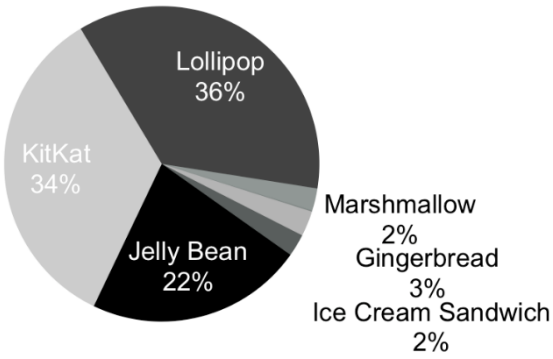



*Figure 5.5.3-1*
*Android Usage [15]*

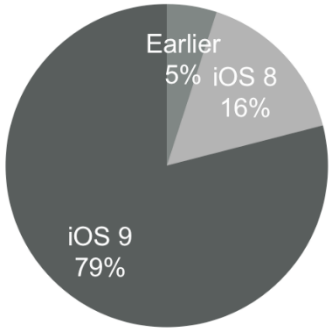



*Figure 5.5.3-2*
*iOS Usage [16]*

The fragmentation in the Android operating system makes it very difficult for developers to penetrate the application market. Some features and capabilities are specific to a version of the operating system and most users may not have the same experience due to them not running the latest Android version available. The latest version of Android is 6.0 Marshmallow and only 2% of devices have it installed. Developers need to be mindful of this fact and some may decide to ignore it and develop for specific devices or Android versions all together.

As can be seen in Figure 5.5.3-2, a large majority of users are running the latest version of Apple's mobile operating system, iOS 9. The percentages in the figure were collected from the Apple Developer App Store page and the website states that the data was collected by the App Store on March 7th, 2016. It can be seen that 80% of iOS devices are running iOS 9, while only 2% of Android users are running the latest version, 6.0 Marshmallow. iOS developers have more peace of mind and guarantee that a majority of their customers are able to install their application and take advantage of the same software features that may be specific to the version of iOS they are developing for. This fact makes iOS development more popular for new developers to start writing applications and building connected devices since there is a larger market share of devices running the latest operating system.

The choice of choosing iOS over Android for the purpose of the project weighed heavily on this fact and the ability for the project and application to be used by more users due to features that may be specific to the latest version of iOS. One example of how running iOS 9 can affect user experience is the inclusion of the Metal API. Metal allows for more efficient CPU and GPU performance that can largely affect user experience and also activity usage. Metal is only available in iOS 9 and can largely impact the experience a user has with older and newer iOS devices due to differences in specifications and performance of the hardware. Another feature specific to iOS 9 is the inclusion of Low Power mode. This allows for the battery life to be extended up to an hour by limiting notifications, fetching of data, background downloads and other features that a user can live without until they are able to charge their devices battery. These features can only be taken advantage of when running the latest version of iOS and can be important when making choices during the application development process.

# 5.6 Wearable Application

This section goes over the wearable application market and the various options for wearable devices that are currently sold. The subsections focus primarily on the software differences between the numerous wearable device options that are popular to consumers. The comparisons made impacted the choice of including a wearable application and the platform chosen.

### 5.6.1 Wearable Technology

Aside from consumers owning smartphones, tablets and computers, a new category in the Internet of Things is gaining traction, Wearable Technology. Wearables are accessories that consumers wear particularly on their wrist to provide them specific functions as features, such as activity tracking, keeping track of time and date or notifications. Some popular wearables include the Apple Watch, Android Wear devices and Fitbit devices. More specific wearables that can be seen worn more frequently are smart watches in particular. Smart watches provide quick access to view information, ability to use diverse, compact applications and more depending on the device. The Apple Watch and Android Wear devices both run specific operating systems that are similar to their smartphone counterparts, but have less functionality and features due to the small hardware footprint.

More companies are beginning to enter the smart watch application market with tailored applications that provide quick access to functions that their accompanying iPhone applications are able to achieve. The convenience of having a smart watch on a user's wrist make them popular for controlling connected devices and appliances. For example, the Phillips Hue and Nest Thermostat provide smart watch applications that allow for quick controls and adjustments to be made without having to pull out your smartphone or having it on you for that matter. A user can conveniently adjust the color of their Phillips Hue light bulbs in their entertainment center to match a certain theme all through the convenience of the smart watch on their wrist.

Since the decision was made to develop an iOS application for the smart coffee roaster, there are plans to include an Apple Watch application to provide quick controls and access to data from the coffee roaster. Developing an application for the Apple Watch requires an iPhone application to communicate with. An Apple Watch application is essentially an extension of the iPhone application and it would have similar features and functionality that would best utilize the smaller screen and unique hardware interfaces. The Apple Watch application would essentially have less features and options than the accompanying iOS application, but it would provide quicker access to the most used controls or features that can be interfaced more conveniently on the wrist.

Examples of such features and benefits specific to the coffee roaster include the ability to view a roast in progress with an indication of the current stage in the roasting process, or access to quick controls to start and end a roast without requiring access to the iOS application. Interfaces and features that provide convenience or enhanced functionality to the user explain the purpose of why most Apple Watch applications exist and the reason why developers are looking to extend their iOS applications with the Apple Watch. The next section goes into more detail about the operating system that runs on the Apple Watch and how developers are looking to develop on the platform.

### 5.6.2 watchOS

The operating system that runs on the Apple Watch is watchOS, with the newest version being watchOS 2. There are multiple versions or what Apple likes to call collections of the Apple Watch available. The choices range due to the choice of materials used for the enclosure, display glass and watch bands. The actual internal hardware of the Apple Watch is the same across all of their collection's which create uniformity across all of its users. Customers all receive the same features and user experience regardless of how much money they decide to spend. Developers have the guarantee that if they develop an application for the Apple Watch, it can be used by every user that owns one and not have to worry about fragmentation between the hardware and software. A user's iPhone needs to be running iOS 9 in order to interface with an Apple Watch. Luckily, over 75% of iOS devices already have iOS 9 installed, which ideally gives developers the ability to impact a large market share as compared to other smart watch operating systems that will be discussed in a later section.

watchOS applications are developed using Xcode, which is the same desktop integrated development environment (IDE) used for building iOS applications. WatchKit and ClockKit are open source software development kits (SDKs) available for creating applications and watch faces on the Apple Watch that take full advantage of the hardware and software. There is no cost of entry to start learning and developing for the watchOS platform. Xcode provides the same ability to side load Apple Watch applications that are in development for testing and quality analysis through simulators or connected Apple Watch devices. In order to develop an Apple Watch application, it is required to have a dedicated iOS application to interface with. It is currently not possible to develop an Apple Watch application without also building an iOS application that it communicates and receives data from. This is due to the fact that it is required to own an iPhone in order to use an Apple Watch. The Apple Watch is an extension of the iPhone and it is typically used for quickly interfacing between the iOS application with the convenience of having a device on the wrist for easy access and control.

Aside from applications that are able to run natively on the Apple Watch and are extensions of their accompanying iOS applications, there are several other software interfaces that developers can take advantage of on the Apple Watch. One interface being the clock face that displays by default. The clock face is able to be customized with Complications that can display up to date information and interface directly with installed Apple Watch applications. For example, instead of having to launch the Weather app every time you want to see the current temperature, the clock face has a built-in complication for the temperature that updates throughout the day. Third party applications now also have access to Complications and this was made possible with the introduction of watchOS 2.

One example of a third party application that takes advantage of the Apple Watch is BMW's i Remote application. BMW offers two electric vehicles that allow for the

viewing of data and multiple functions to be controlled through the Apple Watch application and the clock face Complication. The i Remote application allows for the addition of a Complication that shows the current battery charge of the vehicle on the clock face. The application itself provides the ability to flash the headlights, start the air conditioning system or even track the location of the vehicle. All of these features are conveniently accessed through the Apple Watch on the wrist and add more benefits to owning devices that are able to be interfaced with it.

Another interface that developers have access to on the Apple Watch are Glances. Instead of opening the application or viewing the Complication, Glances can be quickly accessed by swiping up from the clock face. They provide quick access to important information that is viewed most frequently and updates automatically in the background. One example of Glances on the Apple Watch is the Activity Glance. The Activity Glance provides quick access to an overview of the user's progress towards their activity goal. If the user wants to see more information about a specific goal, they can tap on the Glance and it will launch the Activity application that includes more data about the user's activity goals.

Figure's 5.6.2-(1-3) show examples of the Activity Complication, Glance and Application itself from left to right respectively. The amount of information displayed for the Activity application between the three interfaces increases from left to right. The three screenshots were taken on a personal Apple Watch with current Activity data.

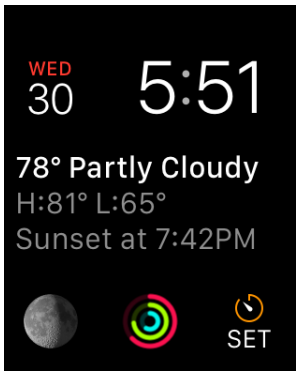



*Figure 5.6.2-1*
*Activity Complication*

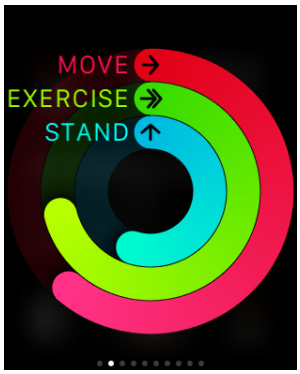



*Figure 5.6.2-2*
*Activity Glance*


*Figure 5.6.2-3*
*Activity Application*

## 5.6.3 Android Wear

The operating system that runs on Android watches is Android Wear, with the latest version being v1.4. Android Wear watches are available from a wide variety of technology brands including: LG, Motorola, Huawei, ASUS and Sony. Even some familiar wristwatch brands are jumping into the market including Android Wear watches from: Casio, Fossil and TAG Heuer. Most of the offerings available from the familiar wristwatch brands don't include as much Android Wear functionality as compared to the aforementioned technology manufacturers. The

fact that not all Android Wear watches provide the same functionality introduces fragmentation in the development process and end user experience. One example of hardware fragmentation is that some manufacturers have either round or square designs for the watch itself. Thus, applications should be developed to support both shapes and display content in a similar manner. Not all options available for Android Wear hardware will allow the ability to install an Android Wear application with the full functionality that was intended by the developer. This fragmentation created a reason to not develop the project application using the Android Wear platform. One requirement for interfacing an Android device with an Android Wear watch is that it needs to be running Android 4.3 or later. Version 4.3 Jelly Bean is only installed on less than a quarter of Android devices, which makes the market fairly small for new developers to have any incentive or reason of developing for little market share.

Android Wear applications are developed using Android Studio, which is the same desktop integrated development environment (IDE) used for building Android applications for phones and tablets. Similarly, to iOS, Android's software development kits (SDKs) are open source and available for creating applications and custom watch faces on Android Wear watches with no cost of entry. There is no disadvantage or penalty for experimenting with either wearable operating system platforms (watchOS or Android Wear) since they're both free to use to the public. This provides developers the ease of testing both options and deciding first hand during the application development process what platform to choose. Comparably to watchOS, it is required to have a stand-alone Android application on a phone or tablet device in order to develop an Android Wear application. The Android Wear application communicates and interfaces directly with the Android application on the phone or tablet device. Thus, a user is not able to utilize the features of an Android Wear watch without also owning an Android phone or tablet device. This should not come as a surprise to most consumers that already own a smartphone and are looking into wearable technology options. An Android Wear watch acts as an extension of the user's phone or tablet to quickly interface between applications in order to view or send data conveniently on their wrist. One neat feature of Android Wear watches is that they can be paired with an iPhone device. It is not possible to pair an Apple Watch with an Android device. However, the functionality and features are limited when using an Android Wear watch with an iPhone due to the differences in operating systems and execution of features offered by each.

Aside from applications that are able to run natively on Android Wear watches, there are several other software interfaces that developers have access to. Equivalently to watchOS, the watch face that displays by default can be completely customized. The clock face can be customized with complications that display data communicated from the companion app on the user's Android device. For example, instead of having to launch the Calendar application every time you want to see your next event for the day, the watch face has a built-in complication for the calendar that actively updates throughout the day without the need of opening

the Calendar application. Unlike watchOS, Android developers also have the added ability to create custom watch faces that include third-party complications and services to interact with. Currently, watchOS does not provide the ability to develop a custom watch face, but instead allow the customization of Apple's own developed watch faces with predetermined controls on what can be modified.

Another interface that developers can customize on Android Wear watches are cards. When an application notifies the user of new information, a card will appear on the display. Instead of requiring the user to open the corresponding application that received new information, cards provide convenient access to this data on the watch face. Developers can customize how the cards are displayed to the user, what kind of information streams into them, and how the user can interact with the information they are viewing. One example of how cards can be utilized on Android Wear watches is Google's mail service Gmail. When a user receives new mail, a Gmail card will appear on the watch face for the user to interact with. A preview of the message contents will be displayed and the user can touch the card in order to view it in its entirety. When the user wants to reply to a message, they can swipe ← on the message until they see the Reply button. The user can then touch the Reply button and dictate their response that will automatically send to the recipient. Another option when swiping ← on a message is the ability to Archive the message. All of these actions are accessible through cards and do not require the user to open the Gmail application. User experience on smart watches needs to be fast and allow the user to view and respond to data promptly. Cards in Android Wear is no exception to this expectation of user experience and human technology interaction.

The figures below show examples of the different interfaces for cards from the Gmail application. The three screenshots for the figures were taken using a personal Moto 360 Android Wear watch. Figure 5.6.3-1 shows how an incoming card appears on the watch face. Figure 5.6.3-2 shows the message preview when clicking on the card. Figure 5.6.3-3 shows the Reply button that is accessible when swiping ← on the message.

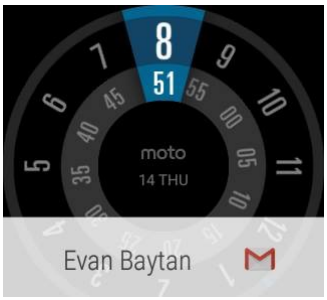



*Figure 5.6.3-1*
*Incoming Message Card*

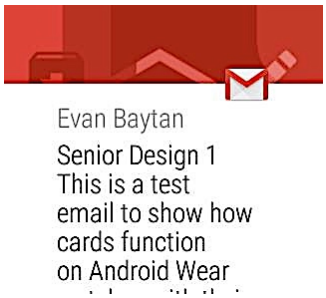



*Figure 5.6.3-2*
*Message Preview*

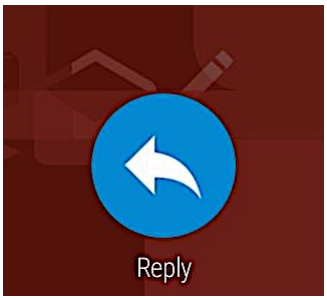



*Figure 5.6.3-3*
*Swipe to Reply*

### 5.6.4 Comparison Summary

watchOS and Android Wear were both analyzed to show the advantages and disadvantages of developing for either platform. Both platforms provide open source development tools and resources to decide which may be better to utilize and implement for applications that will be used on both phones, tablets and wearables. When deciding which application platform to use for the project, a couple of reasons stood out that created a distinction between the correct choice. The intentions of the applications were to deliver a quality application that will allow for a seamless experience that could end up being commercialized for more users to utilize if successfully executed. watchOS ended up being the choice for the wearable application that will provide controls of the coffee roaster and access to user data aside from the iOS application. The reasons for choosing watchOS over Android Wear were simple and will be explained in more detail.

iOS was chosen to be the primary application platform that is going to be used to interface with the coffee roaster. Notably, Android Wear watches have the unique ability of being paired with either an Android device or an iPhone device. However, most of the features are limited or don't exist when configuring an Android Wear watch with an iPhone, compared to an intended Android device. The lack of full functionality if choosing to use Android Wear over watchOS made it easy to decide since we didn't want to limit the features or experience on the watch application due to simply the platform chosen. Another reason for going with watchOS was that it would be more accessible to the application market. iOS 9 is required to pair an Apple Watch, and over 75% of iOS devices are currently running the operating system. Development of an Android Wear application would not have as many potential users due to the fragmentation that plagues Android devices. The goals of both the phone and watch application are to provide familiar functionality and a user friendly experience across both platforms. iOS and watchOS were needed to allow the most amount of users the ability to download and install the applications with their fully intended functionality and features.

## 5.7 Microcontroller Software

Everywhere you look you surrounded by hundreds potentially thousands of electronic devices all of which have their own complex microprocessors running within. Each of the microprocessors functions are guided by the programming that has been embedded within each them. The following subsections will outline two of our options when programming a MCU.

### 5.7.1 Embedded C

Currently one of the most popular languages to program embedded hardware is embedded C. Embedded C is a variation of C which focuses on using less resources and wide capability to a variety of architectures. The fact that less

resources overall are being used is important due to in embedded environments you are dealing with limited memory and processing power.

## 5.7.2 Real Time OS

When considering whether or not to make use of a real time operating system there are a few things that would be project requirements that would lead an industry professional to use one. Those being if the product requires: a TCP/IP stack (complex networking), a complex graphical user interface or a file system. In addition, real time operating systems excel in creating predictable execution patterns which is essential given that system that make use of real time operating systems tend to have system needs that must be fulfilled quickly. Real time operating systems are able to achieve this by allowing the user to assign a priority for a given thread for execution.

### 5.7.2.1 TI-RTOS

TI-RTOS is a Real Time Operating System developed by Texas Instruments. It supports TI microcontrollers and Wi-Fi modules. It is integrally linked to Code Composer Studio. It comes with pre-build drivers for:

- TCP/IP
- FAT
- Wi-Fi
- SPI
- GPIO
- UART
- PWM

Among many others. This set of available drivers makes TI-RTOS an attractive option for embedded development on TI platforms.

### 5.7.2.2 FreeRTOS

FreeRTOS is a free operating system meant for embedded systems. Also being one of the most popular embedded system operating systems with 113,000 downloads in 2014 alone which is understandable given that they support 35 architectures.

Some technical highlights of the operating system are:

- Pre-emptive scheduling option
- Co-operative scheduling option
- 6k to 12k ROM footprint
- Configurable and scalable
- Chip/compiler agnostic
- Easy to use API
- message passing
- Round robin with time slicing
- Mutexes with priority inheritance
- Recursive mutexes
- Binary and counting semaphores

# 6.0 Project Hardware Schematic

This section introduces the designs and schematics of the different subsystems incorporated in this project. The subsections cover the initial design phase and the designs and schematics of the DC power apparatus, including the current limiter, switching power and linear power. They also cover high power AC, temperature sensors, pressure sensors, microphones, PCB design, and smoke suppression.

## 6.1 Initial Designs

Our initial design for our coffee roaster involved building an entire apparatus, including an agitator, heating chamber, and housing unit. We were also going to incorporate an exhaust system to handle the steam and the smoke that is produced when roasting the coffee beans at such high temperatures. For the heating chamber, we considered using a powerful Plexiglas material that would be able to withstand the intense heat. A material that could endure that stress is expensive and difficult to work with. It is also incredibly dense and therefore relatively heavy for its size. Another challenge with that type of material is affixing the different sensors, agitator, motor, and lid to the chamber.

In order to ensure that the coffee beans were roasted evenly, we needed to incorporate an agitator as well. An auger design was the most promising, but with the high temperatures it would experience, it would need to have been either machined from metal or purchased from a vendor. Due to certain constraints such as time and experience in materials CAD software, our two options were to find and purchase one that fit without heating chamber or to acquire someone who could do the modeling for us and employ a machine shop to create the piece. If one was purchased, the rest of our design would have to incorporate the measurements if they differed, and we would need to find a motor that could handle the weight of the coffee beans while under extreme stress from the ambient heat.

Most motors operate well under a range of conditions. These conditions, such as temperature and torque supply, are generally in an average range for what the motor should experience. When the environment becomes extreme, specialized devices can be necessary, and they are not inexpensive. This also applies for the weight sensor, temperature sensor, audio sensor, agitator speed sensor, and the blower fan speed sensor. Most of the other hardware, such as the DSP chip, WiFi chip, MCU, low bandwidth bandpass filter, and low noise signal amplifier could be isolated from these extreme conditions through insulating material and positioning in the roaster. However, the sensors and motors would have to be exposed to that environment, and in turn would have to able to operate reliably, consistently, and for a long time to avoid having to replace any elements. Finding and acquiring these specialized components was not only difficult but expensive.

Next we needed to design a housing unit for the roasting chamber, printed circuit boards, power management system, insulation system, and the LCD display. In order to develop an acceptable housing unit, the materials would have to also be able to handle the temperature. On top of that, they would need to be designed in a CAD system and ordered and built. Again, with lack of experience in CAD software and the time constraints we have, this seemed unfeasible. With the already mounting challenges to designing and building the coffee roaster, less time would be able to be spent on actually designing the schematics, printed circuit boards, and developing the web application, iOS application, watchOS application, and the MCU interface. Below is the initial block diagram for the schematic and printed circuit board we would build to achieve the design we wanted.

After considering the challenges that would come along with trying to design and build our device from the ground up, we came to the conclusion that we were deviating too far from the goal of the project. Instead of spending much needed time on designing, fabricating, and assembling the physical system, we decided we should spend more time with the hardware and software sides.

## 6.2 Power Systems Design

This design will incorporate a relatively complicated power system. It will use DC to run the microcontroller, sensors, Wi-Fi, smoke suppression/exhaust system, and the analog filter. AC will be used to power the heating elements and the motor. Because we intend to have fine control over drum speed and temperature, we will need to incorporate some type of control mechanism in the power system to handle this. We will first look at the DC design and follow that with the AC systems.

### 6.2.1 DC

We will begin with a step down transformer that will give us approximately 6.3VAC across two outputs. We chose to use a center tap transformer that will enable us to have both positive and negative biases. Full bridge rectifiers follow this transformer stage which yields an approximate 5.4VDC. These rectifiers will

provide power to various regulators and limiters in the DC power subsystem. We will use both linear and switching regulators for different subsystems. Switching regulators are much more efficient but produce a lot of electrical noise. This makes them good for logic, but less than ideal for analog signal processing. Because of this drawback, we will use linear regulators that provide both positive and negative bias to power the analog filter that will be used for crack detection and the load cells and their associated instrumentation amplifiers that will be used for weight measurements.

### 6.2.1.1 Current Limiter

Current can be very destructive. Some user-configurable circuits, such as audio amplifiers, can be shorted and, in a very short period, draw enough current to burn out electronics in the amplifier. Electric motors can also draw too much current when they are subjected to excessive torques and damage whatever system they are a part of, whether that be batteries, microcontrollers, or any other devices. Because of this, we need a good way of controlling the amount of current provided to certain circuit elements. This can be accomplished either through the use of fuses or active current limiters.

Traditionally, fuses have been used to limit current. Fuses are able to effectively stop current flow when there is a sudden peak, but they burn out and must be replaced. They also have a relatively slow response time. This response may be unacceptable when dealing with microelectronics. Because we are sometimes dealing with gate insulators that are only a few atoms thick, we need response times in the nanoseconds, not milliseconds. By using active electronics, it is possible to achieve response times within this order of magnitude, even when using BJTs. We will make use of current limiters between each regulator and the subsystem which they power in order to ensure said subsystems are not damaged through an over current event such as a short or a lower than expected load.

### 6.2.1.2 Linear Power for Analog Filter

In order to prevent electrical noise from adding interference to our analog signal, a linear regulator will be used. This regulator will need to provide approximately 50mA at 1.35v. It will need to be able to operate on a 4.6v-6.2v input. Since there is up to a 15% variance on line power. The regulators we are using here have built-in current limiters that will help to simplify our overall design.

The crack detection circuitry uses both positive and negative bias. As such, it requires two power rails to reach the requisite output voltage. We make use of a center tap in the transformer to achieve this voltage difference. We chose to go with two Texas Instruments Linear Regulators to accomplish this task. Each regulator has accompanying resistors and capacitors in order to tune the voltage level and ensure a combination of both low transients and stability. The components chosen are listed in table 6.2.1.2-1 below:

| Item | Value |
|---|---|
| C8 = C9 | 2.2µF, 16V Tanatlum |
| R4 | 240Ω |
| R5 | 720Ω |
| R6 | 1,400Ω |
| R7 | 470Ω |
| P2 | TI LM337LZ |
| P3 | TI TL317CLPR |

*Table 6.2.1.2-1. Linear Regulator Component Values*

## 6.2.1.3 Switching Power for MCU/DSP

We will use a switching regulator to power our logic circuits. This linear regulator will provide 3.6v of regulated output from up to a 6v input. We have chosen to go with a TI regulator that incorporates a built-in inductor in order to simplify our design process and get rid of the magnetic analysis that would otherwise be required. This regulator will require input and output capacitors as well as three resistors in order to obtain the desired output. The switching function will operate at 1.5MHz; therefore, ceramic capacitors are preferred due to their lower ESR at high frequencies. The regulator that we have chosen is rated to provide up to seven watts of power. It also has built in current limiters in order to protect itself from over current events that may be caused from shorts or other issues that may arise. This is more than double the power that our circuit will draw, which provides us with a significant power margin, thus increasing both the life and efficiency of the regulator. The components accomplish this design are listed in table 6.2.3-1 below:

| Item | Value |
|---|---|
| C1 = C5 | 1000µF, 35v Aluminum-Electrolytic |
| C2 = C6 | 100µF, 10v ceramic |
| C3 | 330µF, 10v tantalum-polymer |
| C4 | 470pF |
| R1 | 412Ω |
| R2 | 174kΩ |
| R3 | 1kΩ |
| P1 | TI LMZ30602 0.8v-3.6v DC/DC Converter. |

*Table 6.2.1.3-1. Switching Regulator Component Values.*

The completed DC power system design can be seen in Figure 6.2.1.3-1 below:

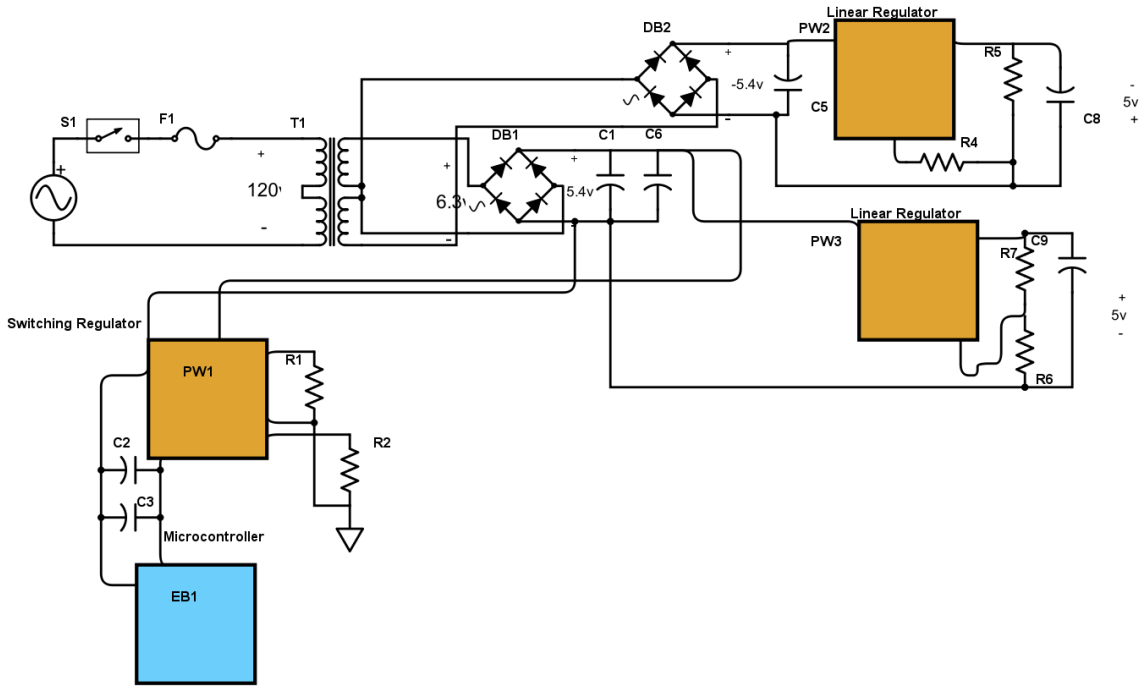



*Figure 6.2.1.3-1. DC Power Supply.*

## 6.2.1.4 Analog-Digital Ground Plane

Because our system contains both analog and digital components, we must make use of both analog and digital groundings. While one may at first naively assume that everything can be connected to the same ground, closer examination of the types of loads connected to each ground reveals a different story.

As we know, wires are not in fact ideal. They do have resistance and when current travels through them, it causes a voltage drop. Resistors in turn are not ideal either. They have a slight inductance and a slight capacitance associated with them. Thus, wires can act as resistors, inductors, and capacitors. When we run many parallel wires on a printed circuit board, this capacitance is increased all the more. We cannot have capacitance between wires on the board. With digital signals, there is a high frequency switching that occurs. This switching can cause voltage spikes in inductors. We know that resistance can be found via 6.2.1.4-1, below, where ρ in the resistivity of the material, L is the length, and A is the cross sectional surface area of the item in which we are calculating resistance for.

$$R = \frac{\rho * L}{A} \quad (-1) \qquad Figure\ 6.2.1.4-1$$

Thus by increasing the area of the ground, we decrease the resistance associated with it. This can also help to reduce the non-idealities associated with resistance. This leads us to the standard practice of putting ground on a plane, rather than individual traces – ground carries the combined currents of every component in the circuit and thus will have the highest overall current. We can reduce the inductive voltage spikes across ground circuits due to switching by increasing the area of the ground traces to full planes.

But why do we separate analog and digital ground to separate planes? As we said, digital signals have high rates of switching which yields high amounts of inductive voltage across the ground plane. This inductive voltage can be seen in the analog circuitry as noise and cause interference in the analog circuitry. This interference cannot be filtered in a simple manner, since the digital frequency is subject to change during operation. This leads us to choose the option of physically separating the grounds of both signal types in order to decrease noise and increase overall reliability of the entire system we are working with.

Because these are both ground, they must ultimately be connected to each other to ensure that both systems within our design are operating at the same voltage, and to allow for analog signals to be converted into digital and vice-versa. Another naïve assumption that can be made here, is that these planes can be connected wherever the engineer sees fit. That is simply not the case. When several connections are made between these planes, it produces loops of current. As we know, loops of current produce inductance, which can further exacerbate the problem. Not only that, but these loops provide more places in which there may be capacitive coupling between traces on the board. For these reasons, we generally choose a single point to connect these ground planes.

However, a problem can occasionally arise when using this method. If there is a semi-large voltage difference between the analog and digital planes, it can lead to overvoltage in any mixed-signal ICs. In our project, that would be the microcontroller and the analog filter. A voltage near 0.3v can lead to component damage [17]. To prevent this, it is customary to add back-to-back Schottky diodes on the connection point [17].

### 6.2.2 High Voltage AC

Some components of the roaster make use of AC power. These include the main heating elements, smoke suppression heating elements, and the blower motor. Controlling the power delivered to these components requires a different subsystem than that which was used for the DC systems. Power to the blower motors and the smoke suppression elements are more straight forward, they only require capability of turning them on or off. Because we are looking for better control of temperature, the main heating elements will require more robust power systems.

The heating elements were measured to each have a resistance of 4.8Ω for a total of 9.6Ω across them. At 120VAC, this results in a max power consumption of 1,500W, they pull 12.5 amps of current at full power. Our design requires that we adjust power in a near continuous range from zero watts, up to the maximum of 1,500 watts. In order to achieve this, several methods were examined. The most obvious choices were the use of TRIACs for phase control and pulse width modulation using MOSFET devices. A further method that we investigated was the use of pulse width modulation with Insulated Gate Bipolar Transistors.

Both TRIACs and pulse width modulated MOSFET systems must stay on for a full period, without the ability to turn them off at any point within the period. Neither one of them is capable of linearly ramping from zero power to full power. This is less than ideal for our system so another power control system was sought out. This system was pulse width modulation utilizing insulated gate bipolar transistors. This system was capable of fully controlling power output in a near linear fashion from zero watts to 1,500 watts but it had some drawbacks. It is not capable of having a reverse current, which meant that the AC power had to be rectified before use. Further, it was incapable of using logic level voltages, an average transistor has a threshold voltage greater that ten volts. This required the use of further voltage control circuitry in order for the microcontroller to manage heating element power. If these hurdles could be overcome, it would allow near linear control of power and thus simplify the design of the temperature control system.

A solution was found that allowed us to efficiently make use of a design utilizing an insulated gate bipolar transistor. We made use of a low voltage drop rectifying bridge and a voltage divider that was controlled using an optoelectric isolator to provide a reasonable gate voltage to the transistor while providing another layer of protection to our microcontroller from the high voltages seen by the heating elements. This solution is detailed in figure 6.2.2-1 below:

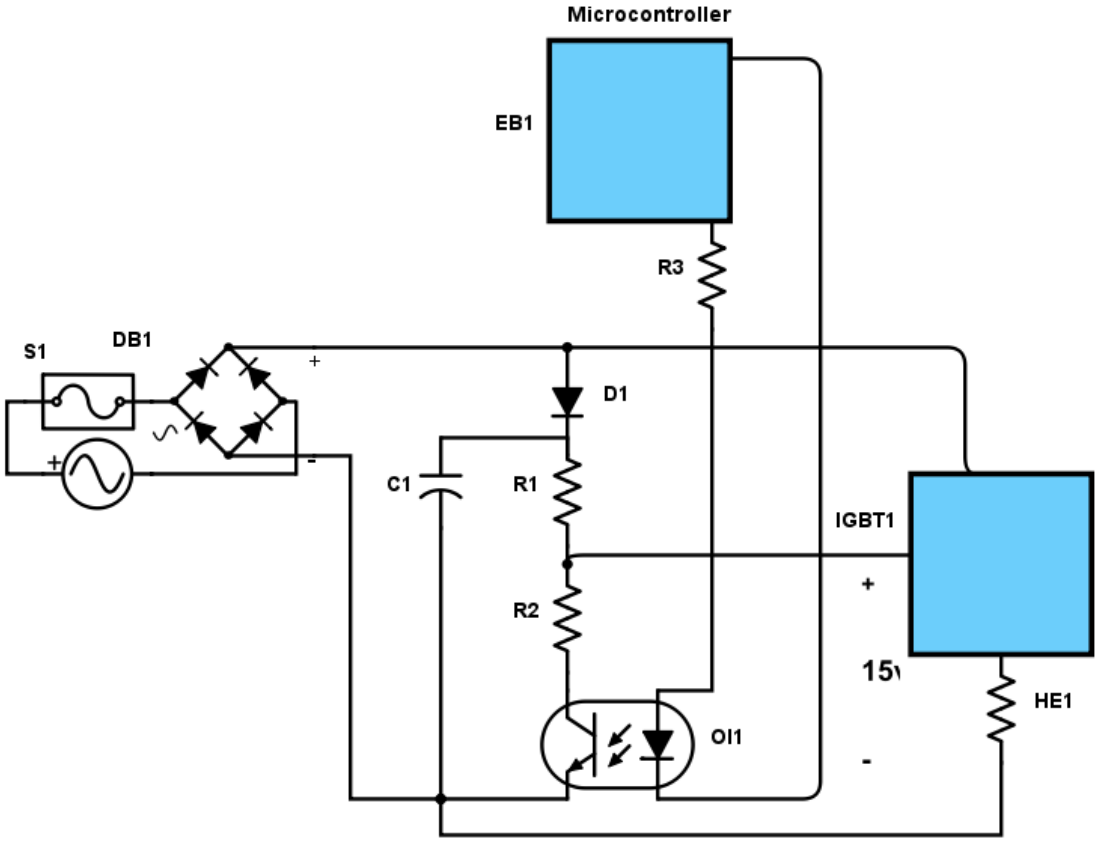



*Figure 6.2.2-1. Primary heating element control circuit.*

The capacitor helps the voltage across the resistors to stay approximately DC. The diode prevents the capacitor from discharging back into the rest of the circuit. Resistor R3 is used to protect the internal diode of the optoelectric isolator from over current events. Component values are listed in table -1 below:

| Item | Value |
|---|---|
| S1 | 300V, 20A Rated Fuse Holder |
| S1 | 125VAC, 15A Fuse |
| DB1 | MCC GBJL2010 1kV, 20A Rectifier Bridge |
| D1 | 200V, 1A Schottky Diode |
| R1 | 154kΩ, 1% 1/4W 1206 Resistor |
| R2 | 15kΩ, 1% 1/4W 1206 Resistor |
| R3 | 100Ω, 1% 1/4W 1206 Resistor |
| IGBT1 | 440V, 20A, 125W IGBT |
| OI1 | 60mA, 6v Input; 100mA, 0.3W output; 3350Vrms Isolation |
| HE1 | 9.6Ω, 1,500W Quartz heating element |

*Table 6.2.2-1. Heating element power element values.*

The upper heating element also runs on AC power. It has a max power consumption of 300 watts. It operates at 120VAC and thus has a resistance of 48Ω. Because we are not interested in have strong control over this device, we are able to use a slightly simplified power system. We will tap into the rectified voltage in the above circuit in order to allow us to use a TRIAC which is controlled in a similar manner to the insulated gate bipolar junction used above. Because we are able to use able to use AC power, we can use TRIACs which can only switch on or off at the end of each cycle. This provides an overall cheaper design, which is also slightly more efficient due to the avoidance of voltage losses across the diodes in the rectifier bridge.

Both the inlet and outlet blower motors also run on AC. Because these have inductive loads, they may not be controlled with TRIACs. The inductive kick back can force the TRIAC back on and thus prevent it from ever turning off. For these applications, solid state relays or solid state contactors must be used. These devices are able to handle the resulting kick back from stopping an inductive load and thus are well suited for use with motors. While these are significantly more expensive than some other semiconductor devices, they appear to be the only viable method of reliably controlling an AC inductive load without significantly more complicated designs that introduce many more possible points of failure. With these items in mind, we updated our schematic to power to all AC systems, as can be seen in figure 6.2.2-2 below:

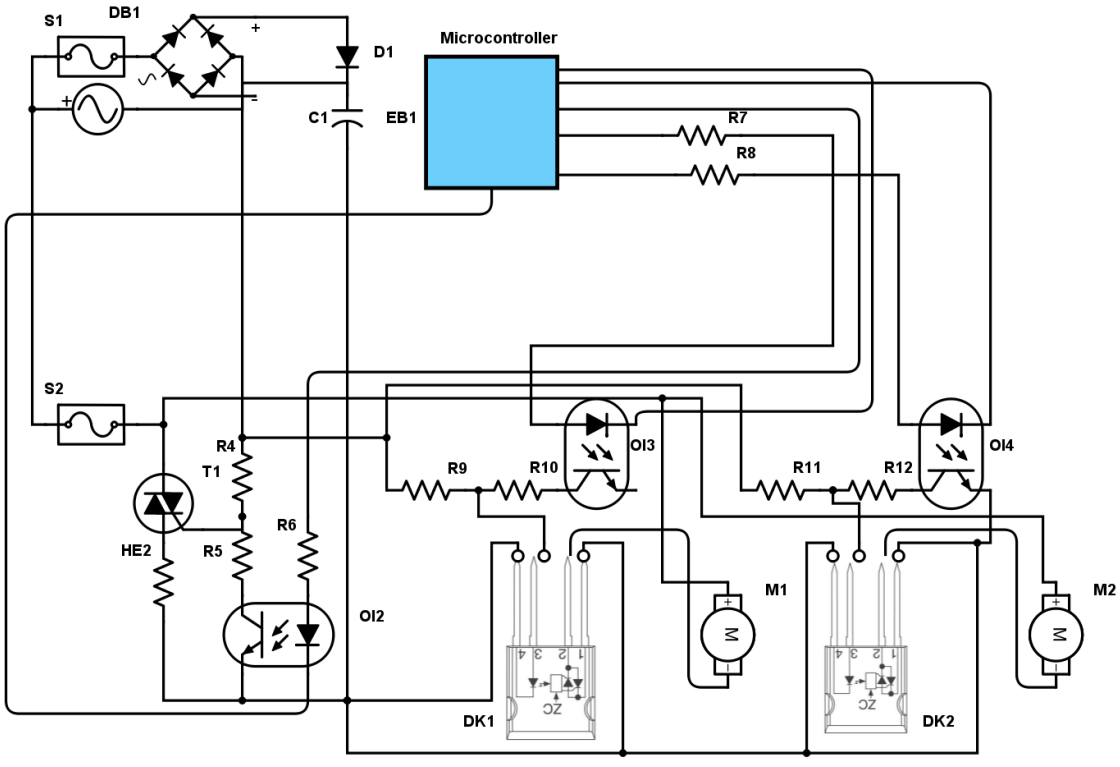



*Figure 6.2.2-2. AC power control system.*

Table 6.2.2-2 contains values for the additional components in Figure 6.2.2-2:

| Item | Value |
|---|---|
| OI2 = OI3 = OI4 | 60mA, 6v Input; 100mA, 0.3W output; 3350Vrms Isolation |
| T1 | 600v, 4A Triac |
| R5 | 30Ω |
| R4 = R9 = R11 | 154kΩ |
| R6=R7=R8 | 100Ω |
| DK1=DK2 | 240Vrms, 20A Solid State Relay |
| M1 | In-situ exhaust blower motor |
| M2 | In-situ inlet blower motor |
| HE2 | 48Ω, 300W Heating element |
| S2 | 300V, 20A Rated Fuse Holder |
| S2 | 125VAC, 5A Fuse |

*Table 6.2.2-2. Component Values for figure 6.2.2-2.*

This gives us full control over every major component in the existing roaster, allowing our microcontroller to run major functionality of the roaster and be able to handle the basics of a full roast.

# 6.3 Analog Filter

An analog filter was chosen in lieu of a digital filter because it would provide us with better opportunity to work in a mixed signal environment. The analog filter that we chose provides some very impressive capabilities. It is capable of changing cutoff frequencies on the fly based on software, while still being implemented through hardware.

## 6.3.1 Active

Active filters are sometimes more expensive than passive filters. The use of operational amplifiers as buffers can make them easier to work with and make it easier to obtain higher order filter designs. The active electronics allow you to both boost and attenuate a signal. Further, it is possible to realize designs of every filter type using them. Different design topologies have vastly different sensitivities to component tolerances, therefore deciding on a topology to use becomes quite an important decision. The Current Generalized Imittance (CGIC) Structure was chosen. This design was first made by Dr. Wasfy Mikhael, a professor at UCF. It is often referred to as “the best biquad” due to its low sensitivity to component tolerances and its ability to have many filter outputs from a single circuit. Whereas some biquads have a squared sensitivity to tolerance, these have a square root sensitivity, making them far easier to implement in real-world designs than traditional designs.

Further consideration made us realize the need for some form of adaptable signal processor. Because the first crack and the second crack emit different spectra of sound, we needed a filter design that is capable of changing based on where it is in the roasting process. In order to achieve this, we had to consider either going back to a digital signal processor, or figure out a way of implementing an adaptive analog signal processor. For the aforementioned reasons, we wished to go with the latter. This led us to investigate the use of digitally controlled switched capacitance resistors. We could send a clock signal to our filter and obtain a different cutoff frequency at different points in the roasting process. This would enable us to have that adaptive nature that our system needed while still affording us the opportunity to use an analog signal processor.

## 6.3.2 Passive

Passive filters can be cheaper than active filters. It is possible to achieve many filter types with them and it is possible to implement higher order filters with these filters. High order filters can however be difficult to design. They also only offer the option of attenuating a signal and are prone to errors due to component tolerance levels. Passive filters can be used and are technically able to filter selected frequencies, but they are not optimal and pose several challenges in obtaining a robust design for our roaster.

## 6.3.3 Schematic

We found that by taking our original signal and subtracting a notch filtered version of said signal, we could obtain a very high order, very narrow bandpass filter. At this point, we began looking at commercially available filtering integrated circuits that made use of switched capacitance resistors in order to implement our design. We needed to take into account the power systems that were available to us in other parts of our design. This led us to a Linear Technologies product that makes use of switched capacitance resistors while having a far lower voltage requirement than competing products from other firms. With this, we were able to make use of the design depicted in Figure 6.3.3-1 and Table 6.3.3-1.

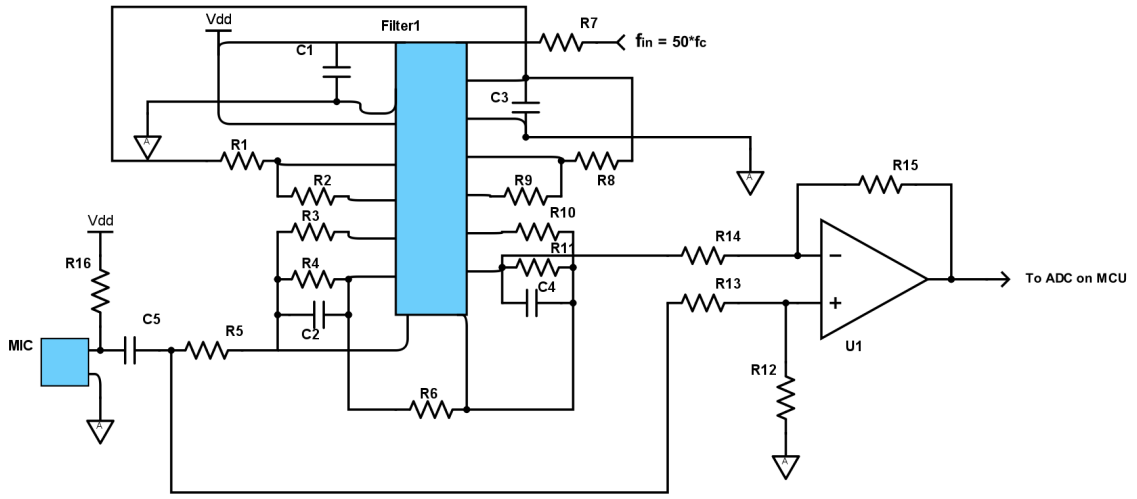



*Figure 6.3.3-1. Crack detection circuit diagram.*

Table 6.3.3-1 below lists component values associated with Figure 6.3.3-1:

| Item | Value |
|---|---|
| R1 | 9.88kΩ, 0.1% tolerance |
| R2 = R9 | 4.99kΩ, 0.1% tolerance |
| R3 | 61.9kΩ |
| R4 | 10kΩ |
| R5 | 18.7kΩ |
| R6 | 40.2kΩ |
| R7 | 200Ω |
| R8 | 10kΩ, 0.1% tolerance |
| R10 | 464kΩ |
| R11 | 75kΩ |
| R12 = R13 = R14 = R15 | 200Ω |
| R16 | 2.2kΩ |
| C1 | 0.1µF |
| C2 | 300pF |
| C3 | 1µF |
| C4 | 30pF |
| C5 | 100µF |
| MIC | CMR-5054TB-A |
| Filter1 | LTC1067-50 |
| U1 | TL971IDR |

*Table 6.3.3-1. Filter component values.*

This leaves us with a relatively robust design capable of being modified in software to allow for changes to the sounds produced by different beans and able to change on the fly in order to adapt to different steps in the roasting process.

# 6.4 Pressure Sensor

Load cells can be represented in electrical schematics as Wheatstone Bridges. They operate by having resistors that slightly change value, based on applied pressure. These resistors are in a larger network that is electrically equivalent to a Wheatstone bridge. By measuring the voltage drop across this bridge, we are able to determine the load applied. The details for our selected load cells are listed in table 6.4-1 below:

| Item | Value |
|---|---|
| Load Cell Type | Strain Gauge |
| Capacity | 5kg |
| Mounting Holes | M5 (Screw Size) |
| Precision | 0.05% |
| Rated Output | 1.0±0.15 mv/V |
| Non-Linearity | 0.05% FS |
| Hysteresis | 0.05% FS |
| Non-Repeatability | 0.05% FS |
| Creep (per 30 minutes) | 0.1% FS |
| Temperature Effect on Zero (per 10°C) | 0.05% FS |
| Temperature Effect on Span (per 10°C) | 0.05% FS |
| Zero Balance | ±1.5% FS |
| Input Impedance | 1130±10 Ohm |
| Output Impedance | 1000±10 Ohm |
| Insulation Resistance (Under 50VDC) | ≥5000 MOhm |
| Excitation Voltage | 5 VDC |
| Operating Temperature Range | -20 to ~+55°C |

*Table 6.4-1. Load Cell Specifications, provided by Phidgets [18]*

The use of the load cell does bring in some additional considerations. The load must be applied to the end of the cell for accuracy, also, the signals produced by it are to0 small to detect with just the analog to digital converter found in our microcontroller, it must first be connected to an instrumentation amplifier in order to bring the voltage up to a reasonable level. We went with a two channel TI instrumentation amplifier. Our design will call for a total of 4 load cells, one on each foot of the roaster, so this allowed us to use two integrated circuits, each with two resistors in order to bring our signal up to a reasonable level. Our instrumentation amp is set to have a gain of 100. The design of our weight sensor system can be seen in figure 6.5-1, below:

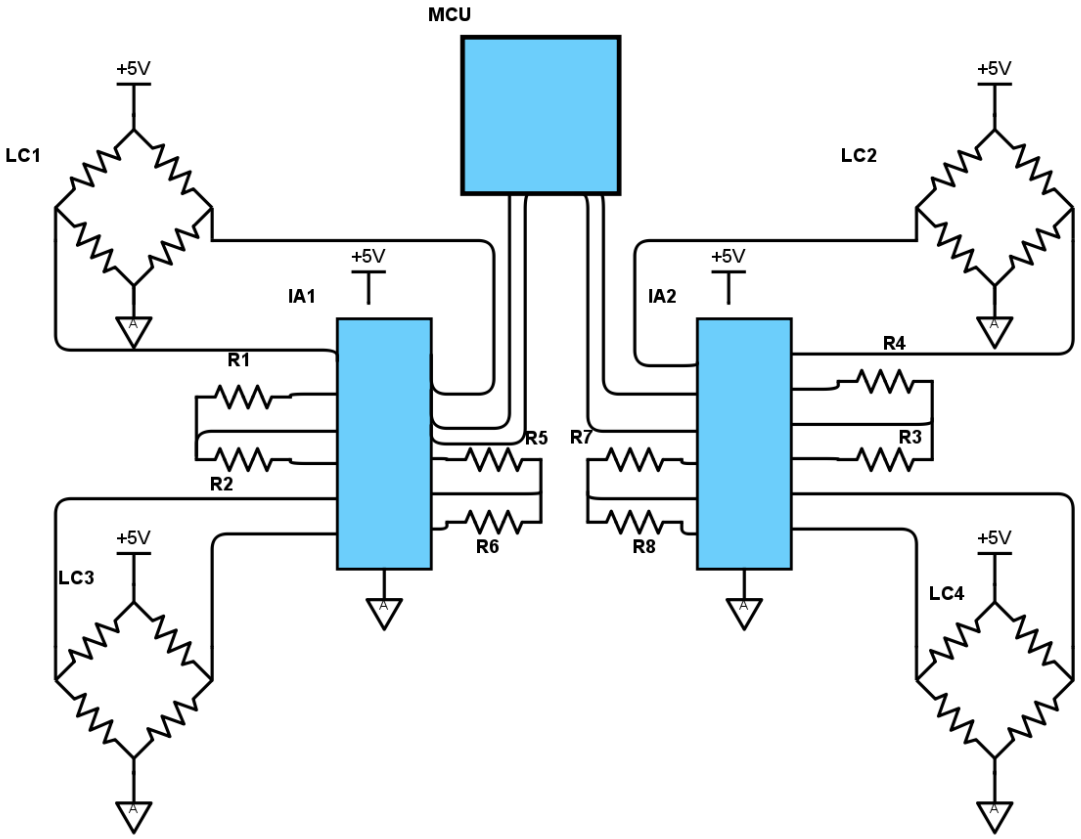



*Figure 6.4-1. Weight sensing subsystem.*

Component values for this subsystem can be found in table -2 below:

| | |
|---|---|
| LC1 = LC2 = LC3 = LC4 | YZC-131 5Kg Load Cell |
| IA1 = IA2 | TI INA2332AIPWR |
| R1 = R4 = R5 = R7 | 10kΩ |
| R2 = R3 = R6 = R8 | 190kΩ |

*Table 6.4-2. Load cell subsystem values.*

# 6.5 Controller Systems Design

Our MCU unit will be controlling the CC3200MOD Wi-Fi module and the BOOSTXL-K350QVG-1S1 LCD display. We decided to use a supply voltage of +3.3 volts. Although the MSP432P401R can handle up to +4.17 volts, the CC3200MOD Wi-Fi module's maximum input voltage is +3.8 volts and the BOOSTXL-K350QVG-S1 is +3.3 volts. The MSP432P401R will communicate with the Wi-Fi module through the CLK signal and shared with the LCD Display, coordinating the timing between the three chips. The inputs from the user on the LCD display will be sent through the MSP432P401R.

Below are the connections between the MCU and the Wi-Fi module, as well as the connections between the MCU and the LCD Display.

| MCU Pin # | MCU (MSP432P401R) | Wi-Fi (CC3200MOD) | Wi-Fi Pin # |
|---|---|---|---|
| **Pin 17** | CLK | HOST_SPI_CLK | Pin 5 |
| **Pin 18** | SOMI | HOST_SPI_DOUT | Pin 6 |
| **Pin 19** | SOMO | HOST_SPI_DIN | Pin 7 |
| **Pin 25** | P1051TA3.1 | HOST_SPI_nCS | Pin 8 |
| | Power Source | VBAT_DCDC_ANA | Pin 36 |
| | Power Source | VBAT_DCDC_PA | Pin 37 |
| | Power Source | VBAT_DCDC_DIG_IO | Pin 40 |
| **Pin 34** | RXD | UART1_TX | Pin 46 |
| **Pin 35** | TXD | UART1_RX | Pin 47 |
| | | **LCD (BOOSTXL-K350QVG-S1)** | **LCD Pin #** |
| | Power Source | 3.3 Volts | Pin 1 |
| | Power Source | 5 Volts | Pin 21 |
| | Power Source | Ground | Pin 22 |
| **Pin 17** | CLK | LCD_SCL | Pin 7 |
| **Pin 18** | SOMI | LCD_CS | Pin 13 |
| **Pin 19** | SIMO | MOSI | Pin 15 |
| **Pin 26** | | LCD_BS | Pin 8 |
| **Pin 27** | | LCD_BL | Pin 40 |
| **Pin 28** | | RESET | Pin 32 |
| **Pin 29** | | TOUCH XN | Pin 31 |
| **Pin 30** | | TOUCH YN | Pin 11 |
| **Pin 56** | A13 | TOUCH XP | Pin 24 |
| **Pin 57** | A12 | TOUCH YP | Pin 23 |

Figure 6.5-1: I/O pin connections between the MCU, Wi-Fi module, and LCD

As you may have noticed for both the Wi-Fi module and the touch screen LCD, their SPI pins are being shared, specifically pins 17, 18 and 19 on the MCU. The question will arise as how do we differentiate between the signals. We will be making use of a Slave Select pin for each slave module. So essentially if we wanted to make use of the SPI lines for the LCD display we would send set the nHIB pin on the Wi-Fi module to LOW. At which point we would be able to begin interfacing with the LCD touchscreen. This style of configuration is known as Master/Slave.

### 6.5.1 Thermal Control System

To control temperature, we must start with a temperature sensor. While there are many temperature sensing options available, from thermocouples to integrated circuits, and even infrared sensing photodiodes, we are using a simple metal wire that is part of the existing roaster that we are retrofitting. The impedance of this wire increases as temperature increases, allowing us to easily sense temperatures on an approximately linear scale. For more accurate measurements, the sensor will need to be calibrated by using known temperatures, such as boiling water at sea level and water with a lot of ice in it, and using that information to determine the resistance at specific, known values. Once the thermometer is calibrated, we are able to accurately measure the temperature in the roasting chamber and tune our control system to match it.

Next, we should determine the transfer function for our system. This can be done by applying a unit step input and measuring the output. The step response of the system is its transfer function. Unfortunately, when dealing with high voltage, high current, and high temperatures, this turns into a bit of a safety concern Since none of us are thermal engineers and none of us have access to good thermal measurement instruments, this seems like a dangerous, impractical process to venture into. Without knowing the best practices for doing thermal testing nor with access to a proper thermal test lab, we will need to search for alternate means of developing our control system. Fortunately, there is an alternative process of developing a control system when the transfer function is unknown: PID tuning.

We examined multiple methods of controlling temperatures in the roasting chamber. We started by considering a hardware based PID controller. This could be implemented using a single, 4 channel op-amp and some resistors, capacitors, and potentiometers. This would allow us to have a weighted sum of separate, components of the PID algorithm. The product can be made as a single input to the adder. The integrator can be realized with an op-amp that is configured as a low pass filter. The derivative can be realized with an op-amp that is configured as a high pass filter. The outputs of these signals are then fed into our summer. We are able to use potentiometers for each signal in order to set the gain of each component in the PID and thus tune our control circuit to match our system. Since each op-amp has a buffered input, they can be used without loading effects, thus changes to our gain are isolated from the feedback loop in our circuit. Because we do not have a transfer function for the temperature inside the roasting chamber, this seems the most appropriate method of designing our control system.

After asking speaking with engineers who have worked with thermal controls, such as UCF professor Dr. Shady Elashhab, it was found that the integral portion in the PID algorithm is often not used. We do not care so much about how things have happened in the past, our primary focus is on the current temperature and how quickly it is either rising or falling, therefore we are able to properly control the system using just product and derivative aspects of the PID algorithm. This

simplification would allow us to implement a PD controller using a two channel op-amp, rather than the four channel that we initially believed was needed.
With further thought, it was determined that we a more practical controller could potentially be designed in software. We decided that it was best to avoid working with circuit elements in a system that is operating at high temperatures using high currents at high voltage levels. The risks of burns and electrocution were too high. If we digitized the signal from the thermometer, we could use a PD algorithm in software and work to adjust the system in software while it was running and avoid said risks. This would also simplify the PCB design of our project, which had already leapt to higher complexity than many senior design projects have.

Further details about designs for hardware that switches actual AC components on and off may be found in section 6.2.2, AC Power Control.

## 6.5.2 DC Motor Control

Some aspects of our design require control of DC components. These include the brushless motor that causes the roasting drum to rotate as well as the DC stepped motors for the side panel fan and the exhaust cooling fan. Since these are all inductive loads, they will require a path for back current to flow once the fan is switched off. This will be done using a diode in parallel to the control electronics. Since we are using DC power here, we can use MOSFETs without worry. This will allow for a less expensive design using components that are more familiar to us. The motors can be controlled using pulse width modulation in conjunction with the aforementioned MOSFETs in order to achieve variable speed. This is especially for use with the brushless motor that turns the roasting drum. Since we do not need to control the direction of the motors nor need to have a hard stop, we do not need to use an H-Bridge and therefore can use discrete MOS components. The design for this subsystem can be seen in figure 6.5.2-1 below:

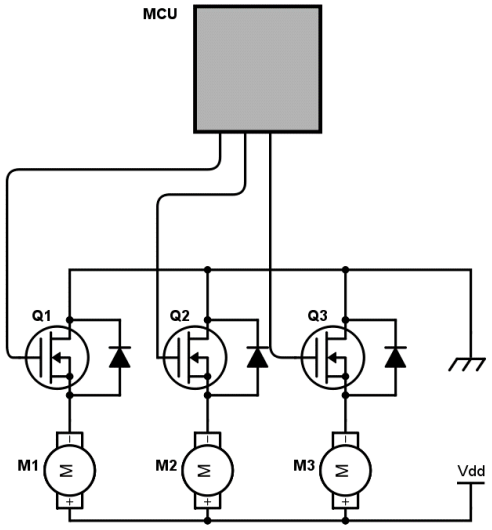



Figure 6.5.2-1. DC Motor Control System

All transistors in this design are Panasonic MTM862270LBF NMOS. They are rated at Vds = 20v and for 2.2A of current to pass through them. They have internal diodes to allow for inductive current to flow once the transistors are off. They have a low enough threshold voltage that they can be directly controlled by logic level signals from the microcontroller.

# 6.6 PCB Design

This section covers the schematic editor used for the design and planning phase. Also, the revisions and changes that have occurred throughout these phases will be discussed.

## 6.6.1 Schematic Editor Selection

There are many different Schematic capturing tools that each offer their own advantages. Eagle CAD is one of the most widely used schematic editors. They have a vast array of online and in-person resources for help, tutorials, and examples. Being one of the major schematic editors, Eagle CAD also has a large pool of independent users who offer their own expertise and projects to help other users. The Eagle CAD community has both user and professional forums to help others with their problems and questions. This is an invaluable resource when starting to use a major new piece of vital software. Eagle CAD offers export/import options from LT Spice. Eagle also has a connection with Element14 allowing for a large compilation of standard libraries, and again exporting/importing parts directly from the Element14 website.

The libraries provided by Eagle cover a wide variety of components and packaging types. More significant, however, is the community of Eagle users who build their own libraries and share with others. By doing this, nearly every component that one would want to utilize in their circuitry can be found in these libraries. If a component is still not in a library, there are other Eagle users out there who are proficient with building the associated symbols and blueprints and are more than eager to help.

Another consideration was CircuitMaker. CircuitMaker is a bit more intuitive in its user interface, but does not offer the access to major resources that Eagle CAD can offer. While CircuitMaker is completely free, as compared to only the Freeware version of Eagle, it is less compatible with other services that go in tandem with a schematic editor. CircuitLab is a browser based schematic editor that is incredibly easy to use. It is not entirely free though, a lot of the upper level components and options are not available in the free version. Additionally, importing and exporting the circuit layout into a schematic editor that does offer PCB connectivity, layout, and design is difficult. To avoid this unnecessary step, CircuitLab was not considered.

Eagle CAD was the best fit for what was needed in the project. It allows for importing and exporting diagrams from LTSpice and other software, a vast base of libraries from all different manufacturers, and powerful and easy to use PCB design options that are automatically synchronized with the schematic that is being built. Additionally, being able to find necessary components from Element14, Sparkfun, and TI, and directly generating the product list was incredibly helpful.

### 6.6.2 Revision

After we had selected the MSP430F5529 as the microcontroller unit we wanted to use, we began work on the schematic. However, as we selected more components that were needed to implement our design, we realized that the MSP430F5529 did not have the required number of serial peripheral interface (SPI) pins that we needed to operate and communicate with the BOOSTXL-K350QVG-S1display and the CC3200MOD Wi-Fi module. After investigating a few other options we switched to the MSP432P401R microcontroller because it offered enough SPI pins to manage the peripheral elements.

Initially we wanted to incorporate Bluetooth into the circuitry as well. Bluetooth offers an added convenience of connecting to the device, as well offering low power consumption through Bluetooth Low Energy (Bluetooth LE). However, Bluetooth can be difficult to get working correctly on a PCB and is not as reliable as Wi-Fi. Although the added option of being able to connect to the device through Bluetooth is nice, it is redundant when there is already Wi-Fi in the device. The CC3200MOD Wi-Fi chip is easier to implement and offers practically the same connectivity and relatively similar power consumption as Bluetooth LE, we decided to remove Bluetooth LE from our board. By utilizing the CC3200MOD Wi-Fi chip exclusively, we also cut down on the number of SPI pins that we needed to use from the MSP432P401R, and, slightly, the amount of power our control system will need to draw.

When choosing the LCD Display we wanted to use, we had to account for the interactions we needed to allow the user to make changes to the roasting process, as well as the number of SPI pins that were required to operate the display. The BOOSTXL-K350QVG-S1 offered the ability to utilize a differing amount of pins depending on what we needed the device to accomplish. It also operated at +3.3 volts which was in the range of the CC3200MOD Wi-Fi chip and the MSP432P401R operating conditions, and one +5 volt input. By having all three major components operating almost exclusively at the same voltage, we were able to cut down on the number of necessary outputs from our power system. The BOOSTXL-K350QVG-S1 also has a majority of the components incorporated in the device itself, and relies on pin headers to communicate with the MSP432402R. By keeping the other necessary elements contained in the device itself, using the BOOSTXL-K350QVG-S1 helped to cut down on the complexity of the circuitry while still being able to deliver the operations that we needed to keep the user interface robust.

## 6.7 Smoke Suppression

The Behmor roaster that we are retrofitting has a built in smoke suppression system that is capable of causing some reductions in the level of smoke that is produced by burning off the smoke. It has an upper heating element that covers the exhaust post and provides a significant amount of heat while there is a large air flow through the exhaust port. Thus burns most complex hydrocarbons and produces water vapor and carbon dioxide gas. While this approach is effective at eliminating large amounts of the smoke that is produced, it does not entirely prevent it. Many users have complained about it setting off smoke alarms if they do a few roasts in a short amount of time. Some users also have medical conditions, such as asthma, that can make them significantly more sensitive to the smoke that is usually produced during normal operation. For these reasons, we sought to build a different smoke suppression system that will be more effective.

We considered a few options. Catalytic converters are effective at converting large masses of harmful chemicals into more acceptable chemicals. These have some short comings. They can be very expensive and they are designed to convert very specific chemicals. While they are effective at converting carbon monoxide and nitrates from vehicle emissions, it is unknown if they are capable of converting exhaust produced by coffee roasters. Another option that was considered, was the use of filters. These are capable of preventing a large variety of chemicals from flowing through them, largely based on the size of the molecules that are flowing. There were concerns that filters may not work properly due to the large temperatures of the air will flow through it and the sheer volume of said air. This led us to investigate a third option: condensing the smoke. This would be accomplished by causing the smoke to flow through a water reservoir that will cool it and trap chemicals in the water. We ultimately chose to go with this option.

The roaster has an existing exhaust duct as part of its smoke suppression system. This duct can be utilized to collect exhaust. This exhaust can be routed through a dryer duct and into an external reservoir. This system can be implemented as an alternative to the existing smoke suppression system – when the external suppression unit is attached, the heating element associated with the internal smoke suppression system will not activate, when the external unit is attached, the internal system will be re-enabled. This provides flexibility where people who have the space to spare can get the benefit of the external system, but those who do not are still able to use the internal system.

Smoke is directed through the duct into the external reservoir. This reservoir is made from a vehicle radiator. As such, it should be capable of dissipating large amounts of heat without the aid of active cooling measures due to its large surface area. The smoke will enter through the lower inlet of the radiator and will exit through the upper outlet. This will allow the smoke to bubble through the full volume of water and bring the highest heat dissipation possible, which will in turn provide the largest amount of condensing of the smoke. A flapper may be needed

in the inlet to cause the smoke to enter the reservoir in bursts. This will prevent the hot gasses from forming a channel through the water and ensure that a higher surface area of the smoke itself is in contact with water. There will be a water collection cap on the outlet to prevent water from spilling when bubbles exit it. While many chemicals will be condensed in the water, carbon dioxide and other gaseous compounds will still exit as a gas, thus care must be taken to prevent water from splashing out. To accommodate switching between smoke suppression modes, and to allow for replacement of reservoir water, a quick release clip will be used to attach the reservoir to the rest of the system and thus allow for easy maintenance.

# 7.0 Project Software

The following subsections will describe the software designs and their respective function in the grand scheme of the project.

## 7.1 API

The API was coded in NodeJS using the ExpressJS framework. We decided that we would use NodeJS rather Spring Boot primarily due to the developer of the API being more comfortable with the framework. The API serves our project by dealing with authentication and storing and serving requested data via HTTP requests. The API will be RESTful rather than based on SOAP primarily due to that a rest API works better with multiple types of clients and is not in need of constant feedback on the current state of the client. The data store that will be used will be MongoDB given two reasons 1) there will be little to no relational queries needed and 2) the developer is very familiar with MongoDB. Our API's stack is outlined in figure 7.1-1.


*Figure 7.1-1 Stack [19]*

### 7.1.1 Routes

/api/user

The following routes are a REST wrapper for the user service. Provides methods for account creation, authentication, update and deletion.
For instance, if we wanted to create a user with a username of “RoastMaster5000”, a password of “I’mthebest”, email “RoastMaster@aol.com” and roaster unique identifier of “1337Roaster”. The payload along with the response would look like the example below in table in table 7.1.1-1. In order to sign in to the application in the future the user will make use of the username and password selected. The email provided would be used if the user is needed to be notified of some bit of information. The roasterUID input is a unique identifier used to differentiate between roasters of all users, meaning it must be unique.

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/user/signup |
| **Description** | Creates a new user given the user does not already exist. |
| **Payload** | POST /api/user/signup<br>{<br>“username”:” RoastMaster5000”,<br>“password“:”I’mthebest”,<br>“email”:”RoastMaster@aol.com”,<br>“roasterUID”:”1337Roaster”<br>} |
| **Response** | If the user has been created:<br>200 – application/json<br>{<br>“success”: true,<br>“msg”:”Successfully created user!”<br>}<br><br>If the username submitted already exits:<br>200 – application/json<br>{<br>“success”:false,<br>“msg”:”Username already exists”<br>} |

*Table 7.1.1-1 – User Signup*

Once the user has created an account, the user would submit his username and password via the post request shown below in table 7.1.1-2. The method below is an HTTP POST request which requires the user to submit his/her username and password. If successful, the user would receive a JWT token which is used for authentication. This token—simply a string specific to the user—must be attached to the header. Every time the user would need some time from the API that required authentication the API would look for and validate that token then would fetch the requested data.

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/user/authenticate |
| **Description** | Used to login a user a.k.a. authenticate a user |
| **Payload** | If we were logging in a user with the username "RoastMaster5000" and a password of "I'mthebest". The payload would look like so:<br><br>POST /api/user/authenticate<br>{<br>"username":"RoastMaster5000",<br>"password":"I'mthebest"<br>} |
| **Response** | If we are successfully authenticated:<br>200 – application/json<br>{<br>"success":true,<br><br>"token":"JWTeyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NmY4NzV"<br>}<br>Note: the token must be stored locally and appended to the header on request of data from the server.<br><br>If the user is not found, e.g. does not exist:<br>200 – application/json<br>{<br>"success":false,<br>"msg":"Authentication failed. User not found."<br>}<br>If the password provided is incorrect:<br>200 – application/json<br>{<br>"success":false,<br>"msg":"Authentication failed. Wrong password."<br>} |

*Table 7.1.1-2 – User Authentication*

If the user intends to update his username, email, password, or roaster unique id; he/she would make use of the below HTTP UPDATE request. First off ensure that JWT token header is attached. If the user would like to update their password, he/she must pass their current password along with the new password they would like to use from now on. Otherwise for the other values that the user would like to update simply pass the new value that he/she would like to use. For specifics of the request view the table below, table 7.1.1-3.

| **TYPE** | **UPDATE** |
|---|---|
| **URI** | /api/user/ |
| **Description** | Used to update user information |
| **Payload** | If we were updating some user information for a user with the username “RoastMaster5000”. If we wanted to update the user’s password and email. For all requests we must add the authorization header which is thejwttoken that the user received during authentication.<br>Header:<br>authorization = ”JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NmY4NzV”<br>The payload would look like so:<br>POST /api/user/<br>{<br>“currentPassword“:”I’mthebest”,<br>“newPassword”:”1337roaster”,<br>“email”:”RoastMasteroftheUniverse@netscape.com”<br>}<br>Note: Provide the values that you would like to update. If you want some value to stay the same just don’t pass it. The only time you must send the current version is when you change the password. |
| **Response** | If successful:<br>200 – application/json<br>{<br>“success”:true<br>}<br>If the password provided is incorrect:<br>200 – application/json<br>{<br>“success”:false,<br>“msg”:”Authentication failed. Wrong password.”<br>} |

*Table 7.1.1-3 Update User Information*

If the user wanted to delete their account, they would make use of the route below. It is an HTTP DELETE request. The user merely needs to pass the token in the header. The API deletes all the user data including their roasts that have been created and shared. Keep in mind that this request is permanent, meaning simply once the user data is deleted it cannot be retrieved or undone.

| **TYPE** | **DELETE** |
|---|---|
| **URI** | /api/user/ |
| **Description** | Used to update user information |
| **Payload** | If we want to delete a user, they must be logged in and thus have their token on their header. Beyond the token on the header no other data is needed.<br><br>Header:<br>authorization = ”JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NmY4NzV”<br><br>The payload would look like so:<br><br>POST /api/user/<br>{} |
| **Response** | If successful:<br>200 – application/json<br>{<br>“success”:true<br>}<br><br>If the token is not valid:<br>200 – application/json<br>{<br>“success”:false,<br>“msg”:”Invalid token.”<br>} |

*Table 7.1.1-4 Delete User Account*

If the user would like to retrieve an array of their favorite roasts objects they would make use of the following HTTP GET request. The user must simply pass their JWT token in the header which will allow the API to identify the user and their favorites list. If the user's favorites list is found the response would like similar to the response below in table 7.1.1-5 except the roasts would be specific to the user.

| **TYPE** | **GET** |
|---|---|
| **URI** | /api/user/favorites |
| **Description** | Returns an object array of user's favorite coffee roasts. |
| **Payload** | GET /api/user/favorites |
| **Response** | Header:<br>authorization = "JWT<br>eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NmY"<br>If the user has been created:<br>200 – application/json<br>{<br>"roasts":<br>{<br>{<br>"roastid":32<br>"Name":"FrenchRoast",<br>"Roast Type": "Dark",<br>"BeanType":"Peruvian"<br>"Rating":43,<br>"CreaterID":23,<br>"RoastingData":"(0,340,45);(200,400,50);(423,450,-)"<br>},<br>{<br>"roastid":42<br>"Name":"American",<br>"Roast Type": "Medium",<br>"BeanType":"Venezuelan"<br>"Rating":43,<br>"CreaterID":23,<br>"RoastingData":"(0,340,45);(200,400,50);(423,450,-)"<br>}<br>},<br>"success": true<br>}<br>The favorites list does not exist<br>200 – application/json<br>{<br>"success":false<br>} |

*Table 7.1.1-5 User's Favorites List*

If the user would like to add a new roast to their favorites list, they would make use of the following HTTP POST request. The client would simply provide the roastid of the roast that they would like to add to their list along with ensuring that their JWT token is attached to the header. Ideally this route would be used asynchronously meaning it would happen inline within a page and no redirection would be necessary. Simply update an icon to signify that the roast is in their favorites list. As reference

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/user/favorites |
| **Description** | Allows the user to add a new roast to their favorites list |
| **Payload** | Pass the roast id of the roast that the user would like to add to the array of their favorite roasts.<br>Header:<br>authorization = ”JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1Nm”<br>POST /api/user/favorites<br>{<br>“roastObjects”:<br>{<br>{<br>“roastid”:32<br>“Name”:”FrenchRoast”,<br>“Roast Type”: “Dark”,<br>“BeanType”:”Peruvian”<br>“Rating”:43,<br>“CreaterID”:23,<br>“RoastingData”:”(0,340,45);(200,400,50);(423,450,-)”<br>},<br>*so on*<br>}<br><br>} |
| **Response** | If the user has been created:<br>200 – application/json<br>{<br>“success”: true,<br>“msg”:”Successfully added new roast to favorites!”<br>}<br>If API was not able to add roast as a favorite:<br>200 – application/json<br>{<br>“success”:false<br>} |

*Table 7.1.1-6 Add new roast to favorites*

/api/roast

The following routes are a rest wrapper for the roasts service. Providing methods to receive specific lists of roasts and update and delete roast profiles.
If the user would like an object array of all the roasting profiles in our database, the user would make use following HTTP GET request. For this route there is no requirement for the user to be authenticated meaning it does not require a token to be attached to the header of the request. This route would be used in the application when the user is attempting to browse through the available user created roasts. The specifics of the route can be found in the table below, table 7.1.1-7.

| **TYPE** | **GET** |
|---|---|
| **URI** | /api/roast |
| **Description** | Returns a list of all roasting profiles ordered byroastID |
| **Payload** | GET /api/roast |
| **Response** | If we are successful:<br>200 – application/json<br>{<br>“success”:true,<br>“roasts”:<br>{<br>“roastid”:32<br>“Name”:”FrenchRoast”,<br>“Roast Type”: “Dark”,<br>“BeanType”:”Peruvian”<br>“Rating”:43,<br>“CreaterID”:23,<br>“RoastingData”:”(0,340,45);(200,400,50);(423,450,-)”<br>},<br>{<br>“roastid”:42<br>“Name”:”American”,<br>“Roast Type”: “Medium”,<br>“BeanType”:”Venezuelan”<br>“Rating”:43,<br>“CreaterID”:23,<br>“RoastingData”:”(0,340,45);(200,400,50);(423,450,-);<br>(600,0,0)”<br>},<br>*And so on*<br>} |

*Table 7.1.1-7 Get a list of all roast profiles*

Roasting data is essentially a string that consists of an array of tuples. For example, (0,340,45) means at 0 seconds set temperature to 340 degrees and air rate to 45%. Each time we change a value we insert a new tuple with the time which it is being done the new temp and air rate. If either the temp or air rate is not being changed simply insert a '-'. The tuples are divided by a semicolon. Roasting ends with a tuple consisting of the end time and temperature and air rate set at 0.

If the user would like to return a specific roast object from the database, the user would make use of the following HTTP GET request. The user would merely pass the roast object id in the URI as shown below in table

<table>
<tr><td>TYPE</td><td>GET</td></tr>
<tr><td>URI</td><td>/api/roast/{roastid}</td></tr>
<tr><td>Description</td><td>Returns a single roast object with a specific roast id.</td></tr>
<tr><td>Payload/ parameters</td><td>GET /api/roast/32<br><table><tr><th>Parameter</th><th>Value</th><th>Description</th></tr><tr><td>type</td><td>string</td><td>Defines the type of roast. Acceptable values are Light, Medium, or Dark.</td></tr></table></td></tr>
<tr><td>Response</td><td>If we are successful:<br>200 – application/json<br>{<br>“success”:true,<br>“roast”:<br>{<br>“roastid”:32<br>“Name”:”FrenchRoast”,<br>“Roast Type”: “Dark”,<br>“BeanType”:”Peruvian”<br>“Rating”:43,<br>“CreaterID”:23,<br>“RoastingData”:”(0,340,45);(200,400,55);(423,0,0)”<br>}<br>}<br><br>Note: Roasting data is explained above in the method GETapi/roast/</td></tr>
</table>

*Table 7.1.1-8 Get a specific roast object*

If the user would like to create a new roast the client would make use of the following POST HTTP request. The request requires the name, roast type, bean type as well as the array of tuples which represent the time, temperature, and fan rate data. The roast name must be unique, e.g. must not already exist in our data base otherwise the client will receive an error in response.

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/roast/ |
| **Description** | Saves a custom roast profile given the parameters passed. |
| **Payload/ parameters** | Header:<br>authorization = ”JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NmY4NzV”<br><br>POST /api/roast/<br>{<br>“roast”:<br>{<br>“roastid”:32<br>“Name”:”FrenchRoast”,<br>“Roast Type”: “Dark”,<br>“BeanType”:”Peruvian”<br>“Rating”:43,<br>“CreaterID”:23,<br>“RoastingData”:”(0,340,45);(200,400,50);(423,450,-)”<br>}<br>} |
| **Response** | If we are successful:<br>200 – application/json<br>{<br>“success”:true,<br>}<br><br>If we are not successful:<br>200 – application/json<br>{<br>“success”:false,<br>}<br><br>Note: Roasting data is explained above in the method GETapi/roast/ |

*Table 7.1.1-9 Create a new custom roast*

If the user would like to up vote a user's roast profile the client would make use of the following POST HTTP request. The roast object id would be passed in the URI as shown below in table 7.1.1-9.

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/roast/{id}/rateup |
| **Description** | Increments the roast rating of a roasting profile with a roast id of id |
| **Payload/ parameters** | POST /api/roast/{id}/rateup<br>{} |
| **Response** | If we are successful:<br>200 – application/json<br>{<br>"success":true<br>}<br>If we are not successful:<br>200 – application/json<br>{<br>"success":false<br>} |

*Table 7.1.1-10 Rate a roast up*

If the user would like to undo their up vote the client would make use of the following HTTP POST request. As the HTTP request above the client must pass the roast object id in the URI. This specifics of the request can be seen in the table below, table 7.1.1-11.

| **TYPE** | **POST** |
|---|---|
| **URI** | /api/roast/{id}/ratedown |
| **Description** | Decrements the roast rating of a roasting profile with a roast id of id |
| **Payload/ parameters** | POST /api/roast/{id}/ratedown |
| **Response** | If we are successful:<br>200 – application/json<br>{<br>"success":true,<br>}<br>If we are not successful:<br>200 – application/json<br>{"success":false} |

*Table 7.1.1-11 Rate a roast down*

If the client would like to return a list of all roasts that are of a specific type they would make use of the following HTTP GET request. The type that is requested must be added to the uri of the request such as '/api/roast/light/' in order to get a list of roasts that are of type light. If the request is successful the response will be an object array of roasts as what has been seen before. Otherwise they will receive the value success as false in their response payload.

| **TYPE** | **GET** |
|---|---|
| **URI** | /api/roast/{type} |
| **Description** | Displays all roast of a specific type. Such as: light, medium and dark. |
| **Payload/ parameters** | GET /api/roast/ |
| **Response** | If we are successful:<br>200 – application/json<br>{<br>"roast":<br>{<br>"roastid":32<br>"Name":"FrenchRoast",<br>"Roast Type": "Dark",<br>"BeanType":"Peruvian"<br>"Rating":43,<br>"CreaterID":23,<br>"RoastingData":"(0,340,45);(200,400,50);(423,450,-)"<br>},<br>{<br>"roastid":21<br>"Name":"American",<br>"Roast Type": "Light",<br>"BeanType":"Peruvian"<br>"Rating":27,<br>"CreaterID":23,<br>"RoastingData":"(0,340,45);(200,400,50);(423,450,-)"<br>}<br>}<br>"success":true,<br>}<br>If we are not successful:<br>200 – application/json<br>{<br>"success":false<br>} |

*Table 7.1.1-12 Rate a roast down*

The following HTTP GET request would be used if the client wanted a sorted list or roasts. The URI used would be 'api/roast/{type}/{order}. The possible values for type where explained above. The possible values of order are +name or –name, +rating or –rating, +roaster or –roaster. That essentially means if you would like your list to be in ascending order of the roast names you would pass '+name' into the URI and '-name' if you wanted it to be in descending order. As you can probably guess the same holds true if you wanted to list the roast in ascending or descending order based on the roaster and rating. The functionality has been added simply to enhance the user experience and to simply make it easier for them to find the exact roast they are looking for in that specific case. For the specifics look at table 7.1.1-13 below.

| TYPE | GET |
|---|---|
| URI | /api/roast/{type}/{order} |
| Description | Returns a list of roasts in the requested order. |
| Payload/ parameters | GET /api/roast/{type}/name+ |
| Response | If we are successful: 200 – application/json<br>{<br>"roast":<br>{<br>"roastid":32,<br>"Name":"FrenchRoast",<br>"Roast Type": "Dark",<br>"BeanType":"Peruvian"<br>"Rating":43, "CreaterID":23,<br>"RoastingData":<br>"(0,340,45);(200,400,50);(423,450,-)"<br>},<br>{<br>"roastid":21,<br>"Name":"American",<br>"Roast Type": "Light",<br>"BeanType":"Peruvian"<br>"Rating":27, "CreaterID":23,<br><br>"RoastingData":"(0,340,45);(200,400,50);(423,450,-)"<br>"success":true,<br>}<br>If we are not successful: 200 – application/json<br>{ "success":false } |

*Table 7.1.1-13 Ordered Lists*

## 7.1.2 Hosting

After some discussion we have decided to go with AWS for two reasons: 1) We have access to most of the AWS services for free for a year and 2) in our experiences most companies that deal with cloud computing tend to use AWS followed by Azure.

For our hosting we will be making use of AWS' EC2 service which is essentially a VPS in the cloud with a public IP. The fact that it has a public IP is essential to allowing whatever client that needs to interface with our API to connect to it via an internet connection.

Our AWS EC2 instance is Ubuntu 14.04.4 server. Although since our API is being designed in a Windows environment there can be dependency issues since an application may work in one environment but that does not mean it will work in another. In order to get around this, we are making use of Docker which is an open platform meant for distributed applications. Docker essentially acts a container storing all dependencies in a neat bow. So running the application in a different environment is a matter of starting up Docker and loading up the container of our application.

## 7.1.3 Security

For security / authentication we are using JSON web tokens also known as JWT. The idea is rather than create a session on successful login we would send the user a string called 'authentication' aka the token. This can be seen in my sequence UML diagram below, figure 7.1.3-1.

This token would be sent whenever the user needed data that is dependent on knowing the identity of the user. For example, if we wanted a list of the user's favorite roasts he would make use of a route like 'GET smartroaster.com/api/user/favorites' which would return a JSON object array of roasts—this route has not yet been implemented or specified in any way.

Once the user/client has their respective token which they would have stored in local storage and placed the token on their header, they can begin making use of routes that require authentication. An example can be found below in figure 7.1.3-1. If the user wanted to make use of the route /memberinfo our method would first, ensure that the token exists. Then we would make use of secret—just a string used as a key such as 'secret' for encryption and decryption—to decrypt the token and make use of the username stored there. If the user is found, we would return the requested data otherwise we would return an error in the form of a JSON object which relays that request as failed along with what failed.

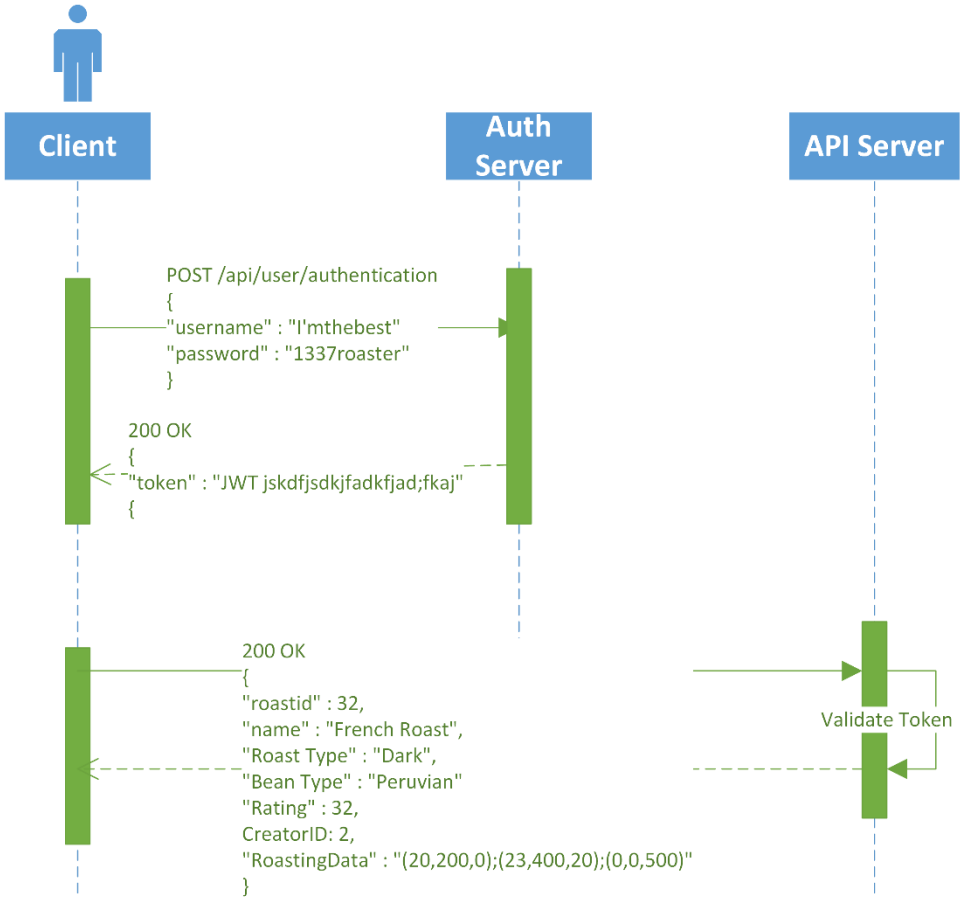



*Figure 7.1.3-1 Token Encoding*

# 7.2 Front End Design

This section goes over both the iOS and watchOS applications with their intendent features and functionality. A mockup of the user interfaces is depicted in the following subsections to show how a possible implementation of the applications.

## 7.2.1 iOS Application

The primary interface with the coffee roaster is intended to be through the front-end iOS application. The iOS application can be installed on both iPhone or iPad devices for increased application support. The iOS application will have multiple interfaces and pages displaying various information to the user.

One component of the iOS application is depicted in Figure 7.2.1-1 below. The figure shows a mockup home page design for when a user first opens the application on their iOS device. The menu bar will consist of a slide out menu button on the left and the name of the application in the center. Beneath the menu bar will include an actively sliding menu of buttons that is currently depicted as

placeholders with the grey boxes and diagonal lines through them. The sliding menu will consist of popular coffee roasts, or any relevant information regarding application updates as examples once the design is finalized. There will be three large buttons taking up most of the screen with each being labeled as the three general types of coffee roasts: Light, Medium and Dark. The bottom of the home page has a placeholder for relevant shortcuts or controls, such as a home page shortcut that can be accessed throughout any location of the application, or quick access to controls for a roast that is in progress.

A secondary component of the iOS application is depicted in Figure 7.2.1-2 below. The figure shows a mockup slide out menu design for when a user clicks on the slide out menu icon located on the left side of the menu bar. The slide out menu will provide access to various features, such as sign up for a new user of the application, history of previously completed roasts, favorites list of commonly used roast profiles and access to settings for the application. The slide out menu will be accessible from every page of the application, similar to the design of popular applications on the App Store. The slide out menu can also be accessed by sliding from the left edge of the screen to the center on the user's device. Access to more features or controls are intended to be added to the slide out menu.

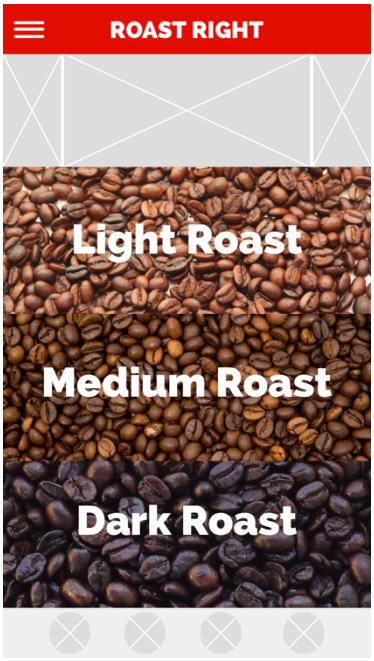



*Figure 7.2.1-1*
*iOS App Home Page [20]*

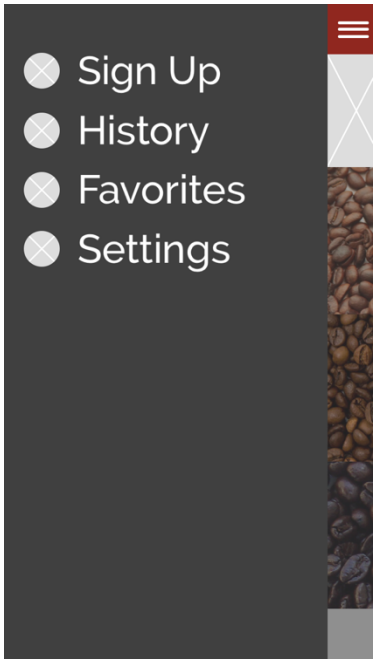



*Figure 7.2.1-2*
*iOS App Slide Out Menu [20]*

When a user clicks on either of the three buttons on the home pages labeled as: Light Roast, Medium Roast or Dark Roast, they will be taken to the page depicted in Figure 7.2.1-3. The figure shows a mockup design for the roast type page. The menu bar will include a back button that will take the user back to the home page if they would like to select a different roast type. In the center of the menu bar will be the name of the application. The roast type may be substituted instead of the

name of the application being in the center of the menu bar. Depending on the roast type selected, a picture with the name of the roast type will be displayed to the user in order to indicate the information that they are viewing. Various textual information will be included on the

A description of the roast and its expected flavor or taste will be included to provide the user a brief explanation on the roast type. Different examples of common coffee drinks that utilize coffee beans of that roast type will be described to better compare or familiarize the user. An estimated time to complete the respective roast type will be displayed to inform the user of how much time is required to complete the respective roast type in general. A video of the team completing a light roast and showing the roasted coffee beans is planned to be included on the page. A start roast button located at the bottom of the roast type page will allow the user to set up a new roast with default settings for a light roast. The bottom of the home page has a placeholder for relevant shortcuts or controls, such as a home page shortcut that can be accessed throughout any location of the application, or quick access to controls for a roast that is in progress.

The expected application workflow can be seen in Figure 7.2.1-1 and Figure 7.2.1-3 below showing the outcome of clicking on the light roast button. A very similar interface and design will be implemented for the following Medium and Dark Roast buttons accordingly.

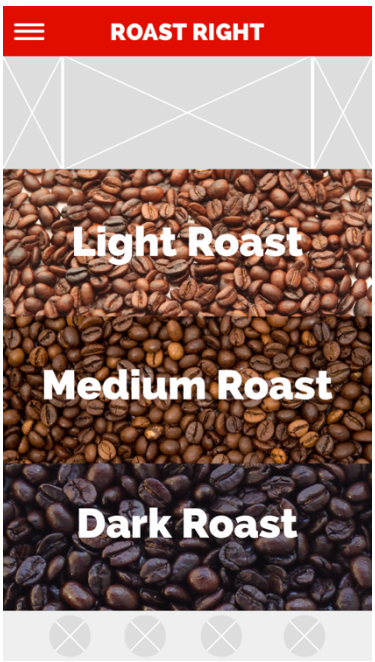



*Figure 7.2.1-1 Home Page [20]*

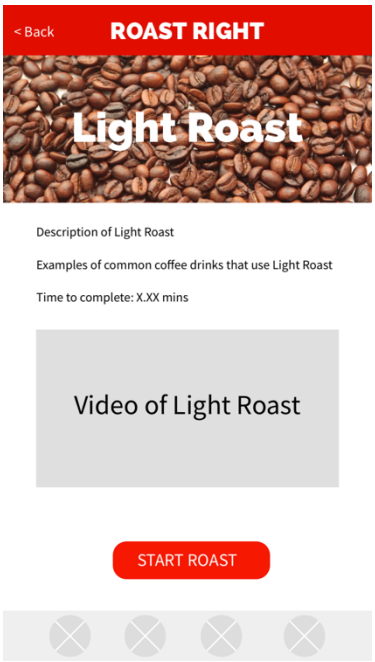



*Figure 7.2.1-3 Light Roast Type Page [20]*

Once a user finds the roast type they want to use and click on the start roast button, they will be brought to the roast in progress page depicted in Figure 7.2.1-4. The same menu bar layout will be utilized to include a back button if the user wants to

browse while a roast is in progress. The center of the menu bar will either include the name of the application or name of the roast type that was started. The center of the page will include a live roast graph that will show the changes in temperature over time during a roast. Depending on the roast type chosen, the roast graph can greatly differ.

Beneath the roast graph there will be a control panel for temperature and time that can be adjusted. Roast progress will be depicted as a loading bar to show the current stage of the roast process. There will be an end roast button beneath the control panel to prematurely end a roast and disable the coffee roaster if necessary. The bottom of the home page has a placeholder for relevant shortcuts or controls, such as a home page shortcut that can be accessed throughout any location of the application, or quick access to controls for a roast that is in progress. Numerous design aspects of the iOS application will be modeled or inspired from popular iOS applications on the App Store. Some applications that inspired various aspects of the user interface or features include: Twitch.TV, DEVOUR and Spotify just to name a few. The user interface and interface elements are intended to be simple and user friendly while providing concise data to the user. These values can be depicted throughout the multiple levels of the application and the different interface elements chosen to be implemented.

The expected application workflow can be seen in Figure 7.2.1-3 and Figure 7.2.1-4 below showing the outcome of clicking on the start roast button and the respective roast in progress page as a result.

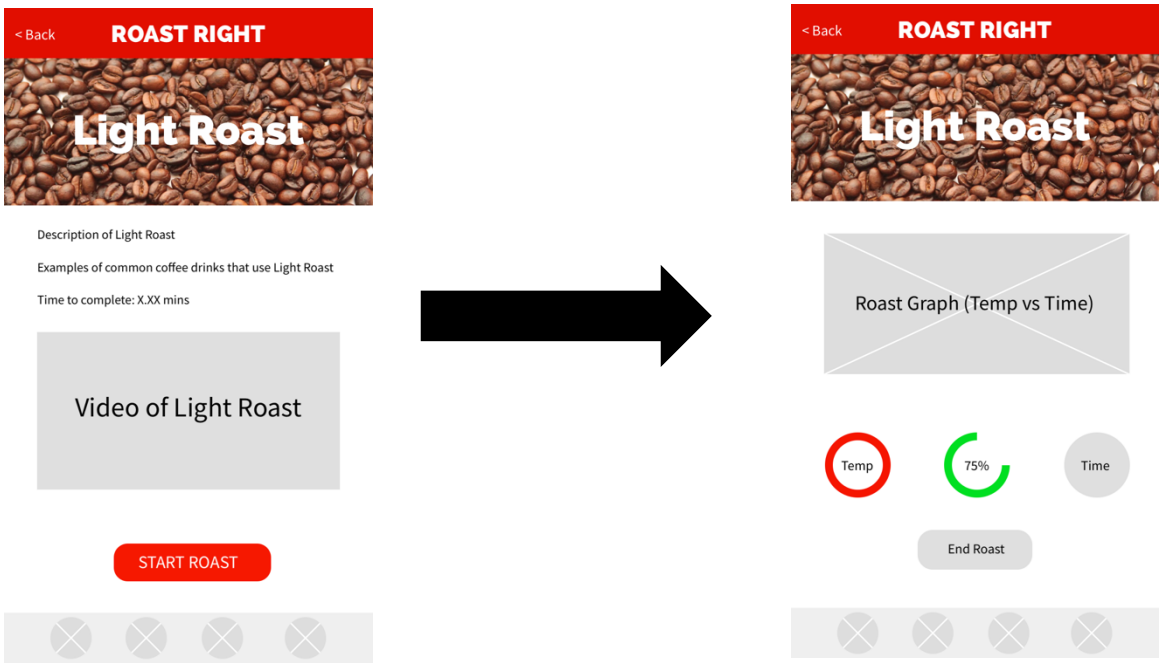



*Figure 7.2.1-3*
*Light Roast Type Page [20]*

*Figure 7.2.1-4*
*Roast in Progress Page [20]*

3D Touch is one interface element new to iOS 9 and only available on the newest iteration of the iPhone. 3D Touch allows the user to apply pressure to their display and have access to controls otherwise not available. One common implementation of 3D Touch is when a user applies pressure to the application icon on their device. Doing so will provide the user quick access to functionality within the respective application, without having to open it. Figure 7.2.1-5 and Figure 7.2.1-6 show examples of 3D Touch when a user applies pressure to the Settings or Safari application icons. With the Settings application, the user is provided with shortcuts to access Battery, Wi-Fi and Bluetooth preferences within the Settings application. Figure 7.2.1-7 shows a mockup interface for implementing 3D Touch for the project. If a user applies pressure to the application icon, shortcuts appear for a current roast in progress, roast types, favorites and settings. These are just examples of possible implementations of 3D Touch that may change during development.

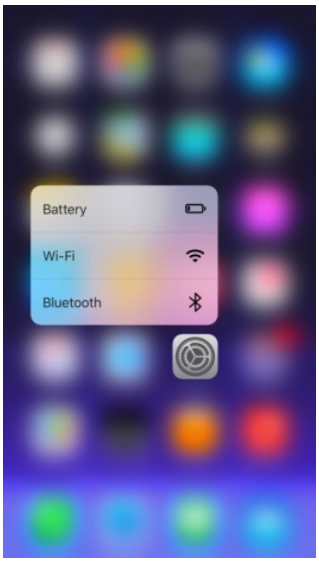



*Figure 7.2.1-5*
*3D Touch Settings App*

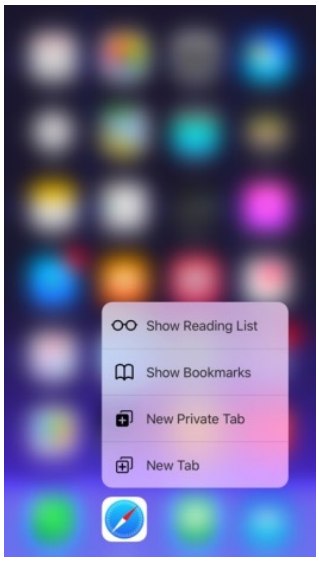



*Figure 7.2.1-6*
*3D Touch Safari App*

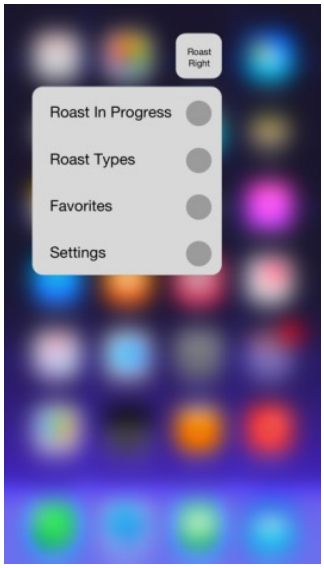



*Figure 7.2.1-7*
*3D Touch Project App*

## 7.2.2 watchOS Application

The secondary interface with the coffee roaster is through the front-end watchOS application. The watchOS application will have multiple levels of interaction or interfaces displaying various types of information to the user. The intention of the watchOS application is to provide added convenience and usability for wirelessly controlling the coffee roaster with the fact that a user's Apple Watch is on their wrist. The advantage of having quick access to common controls through the watchOS application will provide another interface to the user that is familiar and easy to use. Numerous popular iOS applications also provide companion watchOS applications for added application support and differences in user interaction between the devices. Developers have another market and target audience to

consider when maintaining continued support of their iOS applications. Some users may purchase or use an application just for the fact that both an iOS and watchOS application is available from the developers.

Some examples of how the watchOS application may be utilized for the coffee roaster are depicted in Figure 7.2.2-2 and Figure 7.2.2-3 below. Figure 7.2.2-1 in particular depicts an early mockup of the watchOS application logo placed on the home screen of an Apple Watch. Figure 7.2.2-2 shows the home page when a user first opens the application on their Apple Watch. The menu bar shows the name of the application and the current time. The home page consists of four buttons: Types, RFR, RIP and Settings. The types button takes the user to a condensed view of the different roast types with a corresponding description and examples of coffee drinks of that respective roast type. RFR stands for Ready for Roast and if the user selects the RFR button, they will be able to select a roast type and start a roast. RIP stands for Roast in Progress and it is planned for the button to have a live progress of the current roast. If the user selects the RIP button, they will be able to view a condensed roast graph showing the changes in temperature over the time duration for the roast. Underneath the roast graph will be a progress bar indicating the current stage of the roasting process.

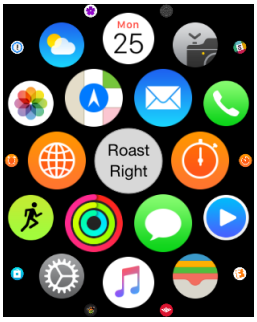



*Figure 7.2.2-1 Apple Watch Home Screen*

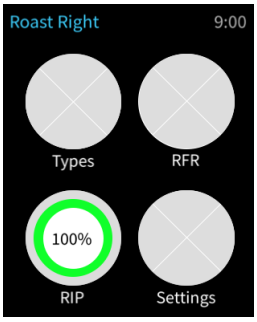



*Figure 7.2.2-2 Home Page*

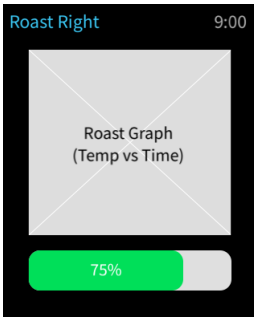



*Figure 7.2.2-3 Roast in Progress Page*

These mockups are the initial design implementations for the watchOS application and it is expected to change once the iOS application is finalized. The iOS application is the primary interface for the coffee roaster, and depending on the features available in the iOS application, it will modify the features available for the watchOS application. The intention for the watchOS application was for it to be an extension and not an exact replica of the iOS application for both design and features. The differences in interfaces and user input will be considered for implementing an experience that is unique and user friendly on the watchOS application.

Aside from the watchOS application itself, there are various other interfaces that can be explored and implemented on the Apple Watch. One of them being complications on the clock face. One purpose of using complications with the

project is by showing the current progress of a roast on the clock face. A circular loading icon can be implemented to easily show the user how much time is needed to complete the roast. The complication on the clock face can also act as a shortcut to open the watchOS application. Another unique interface only available on the Apple Watch are Glances. Glances are a quick way of accessing or viewing live information regarding an application without having to open the respective application. One use case would be for viewing a popular coffee roast of the day by simply swiping up from the home screen on the Apple Watch.

### 7.2.3 Roaster Interface

Aside from the front end applications, the roaster will also have an onboard touch screen interface. The display will be directly connected to the MCU and PCB to receive commands and user input.

## 7.3 Embedded Software Design

Embedded software design is one of the more difficult aspects for this project. It involves a lot of low level code, an aspect in programming that few engineers or software developers are comfortable with. As such, it is important to have a well-documented plan in order to ensure that our embedded software not only works, but performs well with every other aspect of the roaster. This involves both access to hardware, as well as client-server communications. The following sub sections outline the specifics on how the software for our PCB was implemented and designed overall. This includes both the microcontroller and the Wi-Fi module.

### 7.3.1 Features

One of the core features of our project is the ability to live control the smart coffee roaster. Specifically, the ability to control the rate the fan spins and the temperature in which the beans roast. An example of our potential interface can be found below in figure 7.3.1-1.

We have to boxes which show the current temperature and fan rate along with arrows to sides of the boxes. If the user would like to raise the current temperature it would be a matter of pressing the arrow to the left of the current temperature box. Below the temperature and fan controls you will see a checkbox along with the question ‘First Crack Occurred?’ which will initially be unchecked. In the design of our system we have included the ability for detection of the bean crack via some sound processing done at the hardware level. When a bean crack is detected the checkbox will be checked and the user would then be able to proceed from there. In the world of coffee roasting being able to recognize the crack of a coffee bean is essential to the roasting process.

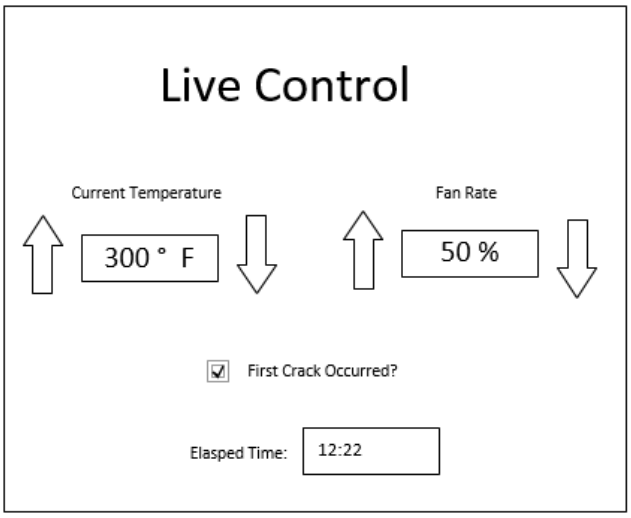



*Figure 7.3.1-1 Live Control Graphical Interface*

Beyond the ability to control the roast process live we have created an iOS application where users would be able to create custom profiles that others may favorite and use with their own roasters. An example of the graphical interface is shown below in figure 7.3.1-2. Once we have selected the roast that we would like to use we are presented the view below which outlines the specific information to the roast. Such as, the roast name, bean type, who created the roast and its rating. In addition, the user is able to always know when the roast will be done so they can plan accordingly. The eta value also takes into account the time necessary for the cooling process which always occurs once the roast has finished.

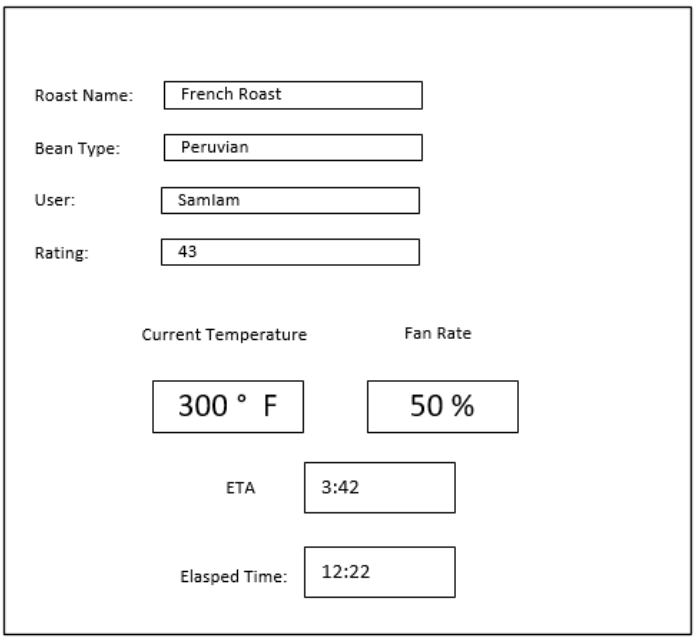



*Figure 7.3.1-2 Load Roast Profile*

One of the key features again of our project is the ability to load user created profiles directly to the roaster. The interface that will allow a user to browse the through the user created profiles can be found below in figure 7.3.1-3 specifically the left image. The list would give the title and the user who created, along with the rating of the roast. As I'm sure it has been noticed you can see to the right of each roast profile is a check mark with the text 'Added' next to it or a plus sign with the text 'Add' next to it. Those that have check mark tell you that they have been added to your favorites list otherwise you are given the ability to add it to your list. Assuming that the only roasts that have been added on to your favorites list was on that page an example favorites page can be seen on the right image of figure 7.3.1-3.

In order pull the list of all roasts the user would make use of the POST request with the URI of '/api/roast/'. This route does not require authentication. The response payload will be JSON based and will simply be an array of roast objects. When attempting to display the user's favorites list the client, i.e. the roaster will have to make use of the URI '/api/user/favorites'/ using an HTTP GET request which simply returns an array of roast profile objects.

Back to the browse view, as you can will notice the user is given the ability to see how many pages total there are for the roasting profiles and the ability to move on to the next page. As the user progresses through the list they will gain the ability to go back in the list i.e. to the left. This allows us to make the embedded programming aspect of our project to be simplified quite a bit by not having to write code to detect swipes.

As I stated above, in order to allow the user to make more educated decisions on which roasts to try we have implemented a rating system based around stars. Essentially if a user likes a roast they would simply press the star and the rating will increment both locally and via an asynchronous post request to the API to notify that roast's rating needs to be incremented.

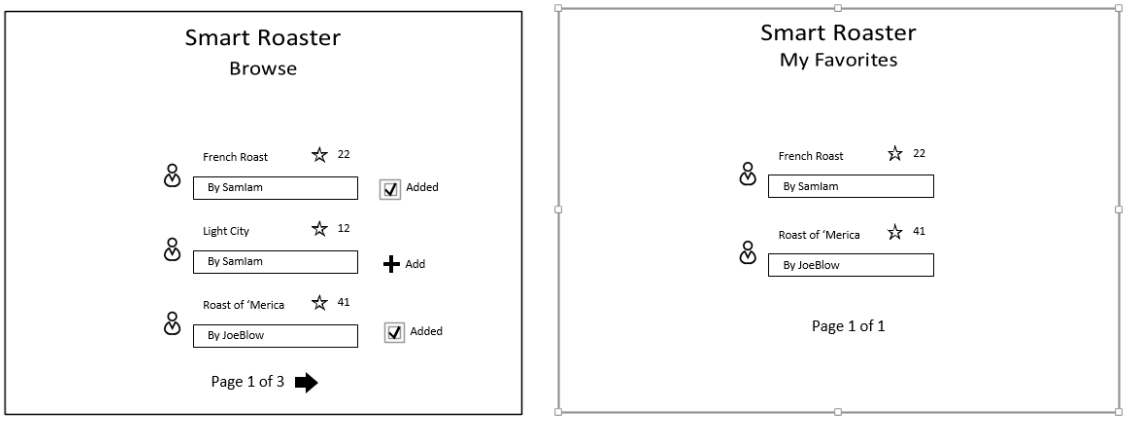



*Figure 7.3.1-3 Roast Lists*

Given that our roaster will have a community focus we will implement a browsing system within the roaster. Essentially we would provide the option for the user to display a list of roasts by type, such as: light, medium, or dark roasts. It will look similar to figure 7.3.1-4 below.

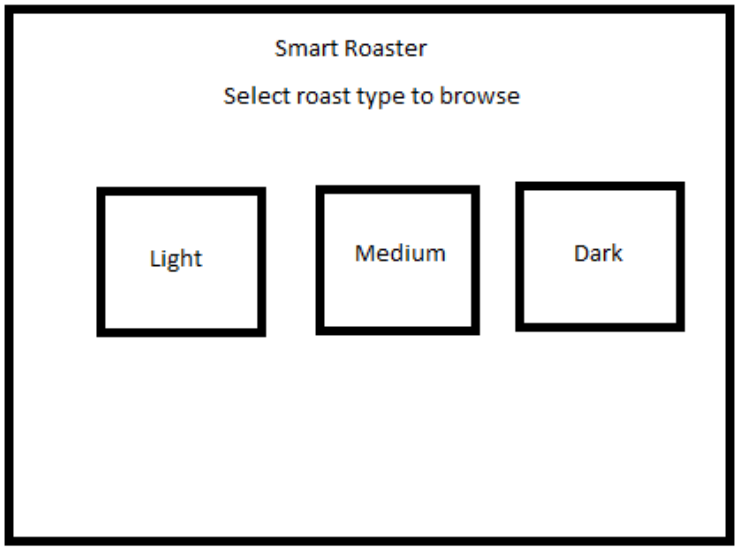



*Figure 7.3.1-4 Choose list to browse*

Once the user selects the type of roast that they would like to view roasts from the user will be redirected to a view similar to figure 7.3.1-5. The roaster make use of the uri '/api/roast/{type}' where type is the type of roast as expected. Valid values include light, medium and dark. As long as roasts for that type exist which they should the user will be redirected to a list of all the roasts that match that type or roast.

If the user would like to sort further, such as sorting by the roast's rating they would make use of the button such as the one presented below in figure 7.3-5. The sorting would be done at the server level given that the lists are not that long and thus it would not be process intensive and end up taking very little time to compile. This would be done by making use of the URI 'api/roast/{type}/{order}. The possible values of order are +name or –name, +rating or –rating, +roaster or –roaster. That essentially means if you would like your list to be in ascending order of the roast names you would pass '+name' into the URI and '-name' if you wanted it to be in descending order. As you can probably guess the same holds true if you wanted to list the roast in ascending or descending order based on the roaster and rating. The functionality has been added simply to enhance the user experience and to simply make it easier for them to find the exact roast they are looking for in that specific case.

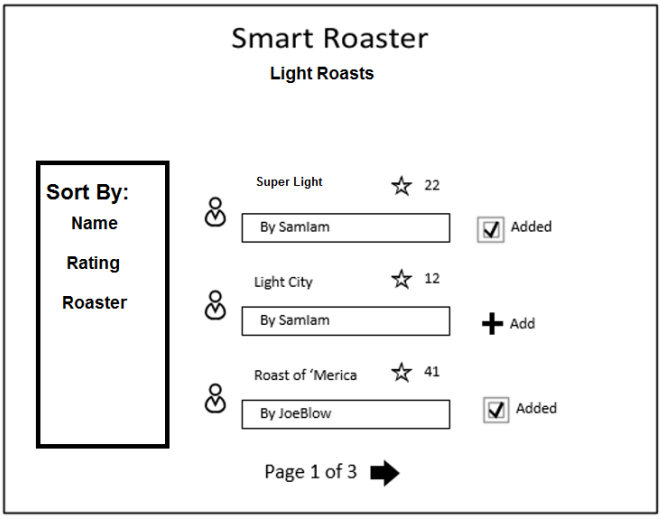



*Figure 7.3.1-5*

## 7.3.2 Communication

When talking about communication we must make sure to place it in context. In our case communication of the MCU and its modules specifically the MCU and the Wi-Fi module and LCD touch screen display. Both the modules will be making use of both SPI which leads to an initial conflict given that we only have one SPI CLK, one SPI MISO, and one SPI MOSI line. In order to solve this problem, we will have to make use of the master/slave model of SPI communication. Essentially when we need to make use of the other the digital connection of the original one will be digitally severed and our new device will then be able to communicate with the MCU. An example of the wiring configuration can be found below in figure 7.3.2-1.

An example of this would be if the user clicked on the button on the LCD touch screen that would then redirect the user to the page where their favorite roasts are located. We would first read the input received by the touch screen and begin disabling the LCD data line and waking up the Wi-Fi module out of hibernate. We would then make the necessary HTTP request to retrieve the requested data and store it on the MCU. We would then proceed to place the Wi-Fi module back into hibernate mode and enable the LCD's SPI lines. We would then render the user's favorites list using the data that is now stored on the MCU's main memory.

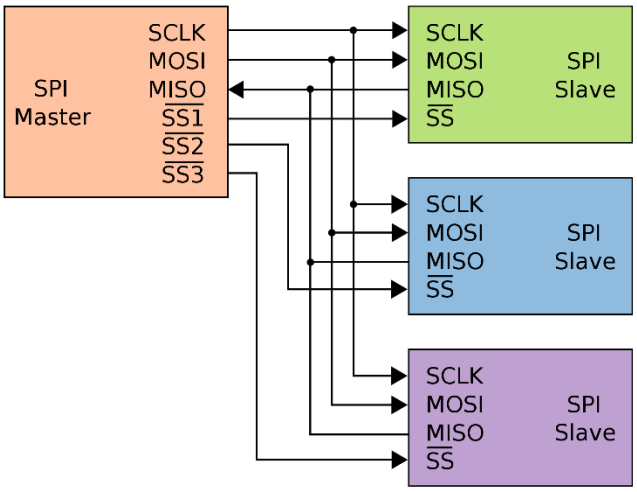



*Figure 7.3.2-1 SPI Master/Slave Wiring Example [22]*

This style of configuration allows for multiple SPI slaves to coexist and allow the engineer to make the most out of the hardware. If we wanted to add another module that makes use of SPI we would do the same simply connect the device in parallel along with the other devices.

# 8.0 Project Summary

The RoastRight coffee roaster encapsulates some novel ideas that commercial roasters have yet to adopt. By adding additional sensors, we are able to achieve a higher level of automation than any other roaster has accomplished. The use of roast profile sharing adds an important social dynamic that is largely lacking from other roasting products. This function will make roasting a social experience, rather than just a utility to achieve better coffee. This functionality will help to bring coffee roasting to the mainstream market. In doing so, it will bring the cost of coffee roasters down and thus help open the world to a better coffee than they have traditionally been able to experience.

While there are many challenges in achieving both these design and product goals, there has been sufficient planning throughout the design phase to make these goals seem more realistic. As you are already aware after reading through this document, many design aspects have been considered. It is our belief that the best design has been achieved. Since we are retrofitting an existing commercially available roaster, this cannot be considered a final product, but rather a proof of concept on what is possible.

## 8.1 Circuitry Design

We will utilize Eagle CAD to do our schematic and PCB design. The MSP432P401R was not included with the master TI library file for Eagle, and after

searching online for user made libraries it was not found. The MCU had to be built using the specifications and dimensions given in the datasheet. The MSP432P401R boasts a 100 pin connection with a 14x14mm casing and a total committed area of 16x16mm. The pitch of the pin is 0.5mm and a pin width of 0.22mm.

The small size of the MCU we chose made it difficult to get an accurate package design. The base grid measurements had to be changed from the default settings to allow for the millimeter scale needed. After ensuring that the package size, pin width, and pin pitch we all accurate to the data on the datasheet, the package was compiled. The accompanying symbol was then generated, modeled after the shape of the physical package of the MSP432P401R to make reading the schematic and PCB setup a bit easier. The package and the symbol were then connected to the created “MSP432P401R” device and the appropriate pins were connected to the package from the schematic symbol. After checking that the pins and the symbols were correctly connected, completed MCU components was saved in a custom library. Below is the generated package for the MSP432P401R.

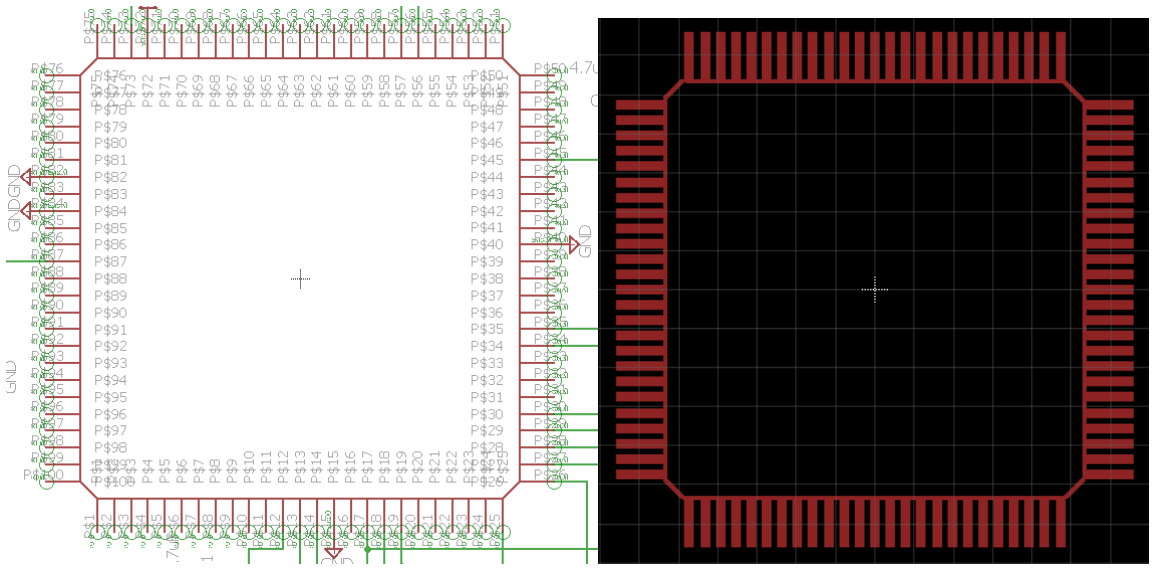


Figure 8.1-1: Symbol and Package for MSP432P401R

The CC3200MOD had to be built in the same manner as the MSP432P401R. The dimensions were pulled from the datasheet and the package was created. Again, the base measurements had to be edited to fit the unusual shape and varying pin pitch. The package was confirmed to match the specifications given on the datasheet, the symbol was generated to match the shape of the package. The pins were connected from the schematic symbol and the package and verified. Below are the symbol and package for the CC3200MOD.

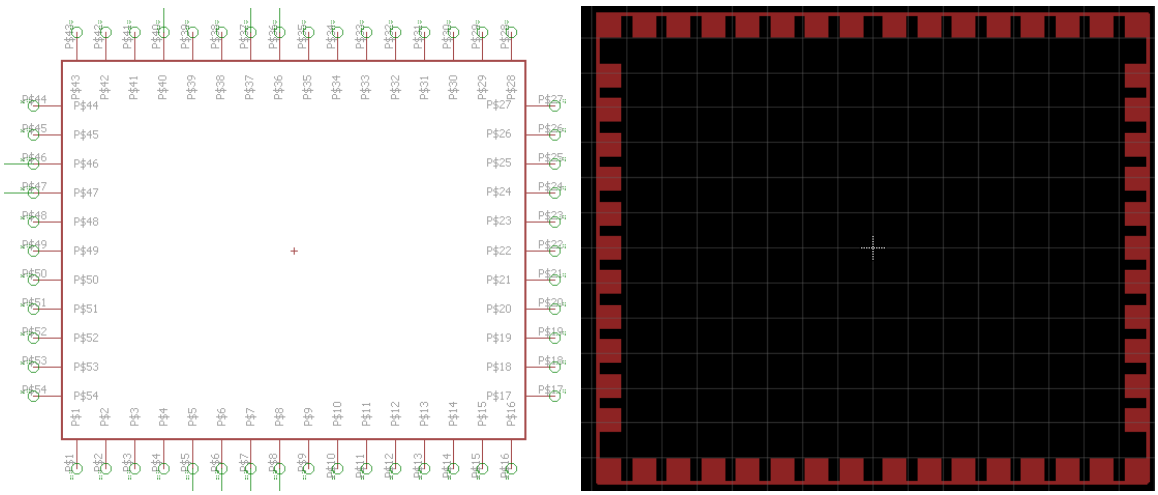


*Figure 8.1-2: CC3200MOD Wi-Fi chip realized in Eagle*

The setup of the LCD Display allows for us to connect the display to the PCB through the use of header pins. This enables us to freely move the LCD to the proper location when we insert the completed circuitry into the housing unit. The pin headers were added to the schematic to coincide with those given from the BOOSTXL-K350QVG-S1 datasheet. After manipulating the orientation of the headers, they were connected to the appropriate pins on the MCU. Below is the setup of the female pin headers for the LCD Display.

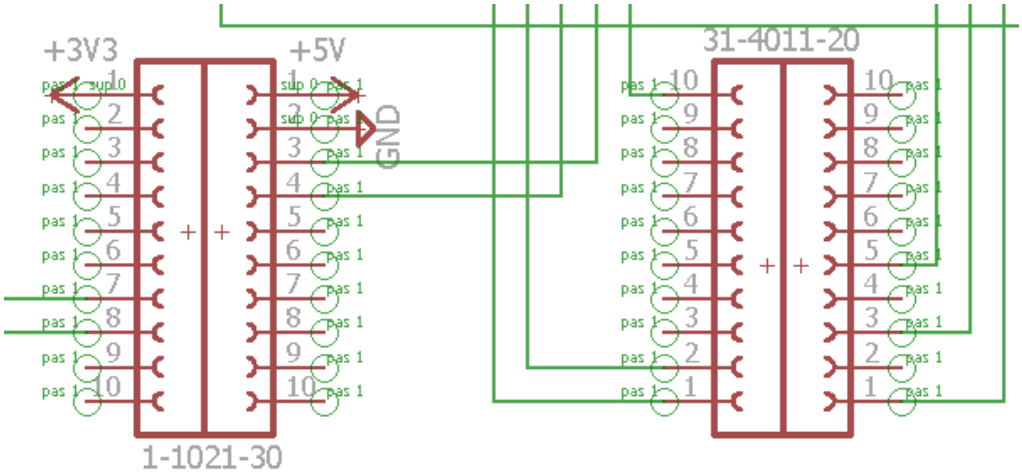



*Figure 8.1-3: Associated pin headers for the BOOSTXL-K350QVG-S1*

The data sheet for the MCU recommends powering the analog $V_{cc}$ pins and the digital $V_{cc}$ pins with the same power supply to avoid and discrepancies that may occur if there were two different supply voltages. A 4.7µH capacitor, as per the datasheet, was added in parallel with the $AV_{cc}$, $DV_{cc}$, and $V_{core}$ pins to allow for DC-DC operations. A 4.7µF capacitor was connected between the DC-to-DC switching converter output $V_{sw}$ and $V_{core}$.

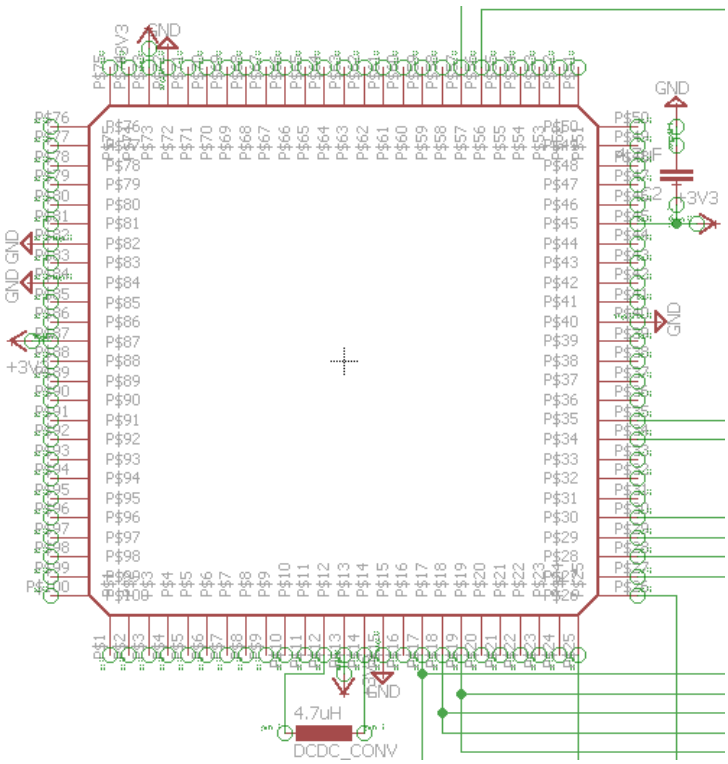



*Figure 8.1-4: External components for the MSP432P401R*

# 9.0 Project Testing

As with any Engineering project, testing is an important aspect of the development process. Because there are so many related parts working together with the RoastRight, that is even more important here. We must test hardware, embedded software, and client-server communications. With all of these varied elements, testing becomes both more important, but also more difficult. We must have a firm plan in place to ensure that our project operates as expected in many varying situations.

## 9.1 Hardware

This section covers the various hardware components of that project that need to be tested and verified for functionality and reliability. This is an exceptionally important process. We need to ensure that our project does the things that it was designed to, and do them safely, effectively, and consistently. Our hardware testing procedures will encapsulate these objectives.

### 9.1.1 Power System

It is important to test power systems before they are connected to a load in order to ensure that they operate in the manner which you expect them to. With, say a

switching power supply, you do not design it with proper damping, you can experience large overshoots that can be damaging to your electronics. There are also safety concerns with power systems. If you short the power, will it potentially cause a fire, or does it have protection against such faults? Does that protection operate in the manner that you expect? These things must be tested in order to ensure safe, reliable operation for the product as a whole.

In order to adequately test our power systems, we will need to come up with criteria that they must meet. For the switching power supply, we expect it to operate at 3.6 volts with a maximum overshoot of 5% for a max voltage of 3.78v. The max ripple voltage should be 36mVpp. This should all occur while providing up to 1.95 amps of current. It will be connected to the DC load tester in the university's senior design lab and tested under various loading conditions with an oscilloscope. We will ensure that over current protection operates as it is supposed to and that voltage does indeed ramp down when current exceeds its limit.

The linear regulators will be designed to provide positive and negative 5 volts. They shall have a max ripple of ±5% which will yield a maximum ripple of 50mVpp. They shall each provide a max current of 150mA. These too will be tested using the aforementioned lab equipment to ensure that they meet or exceed the required specifications. We will ensure that the current limiters in them act as expected and that no damaging overcurrent events arise.

Finally, we will test every part of the power system at load in order to ensure that there are no ill effects seen by our step down transformer or other parts of the overall power system design. We will also test that our fuse acts to prevent damage or injury by shorting it. This will be done using proper safety precautions to avoid injury. A team member will be at the emergency power cutoff, another member will have a fire extinguisher ready, and electrical/heat insulating gloves will be worn. This test will be performed with components outside of any housing so as not to damage other materials in case the fuse does not act as expected.

### 9.1.2 Sensors

The temperature sensor will be tested initially at room temperature, and then cycled through different stages of the roasting cycle. We will also need a reference to test any error that may come with the higher temperature roasts The temperature sensors will be tested both on the temperature rising and falling cycles to ensure they remain accurate, and that the response time is adequate enough to make changes throughout the roast. After testing in the given temperature range for the roast, we will also test them to ±5% of the maximum and minimum temperature they will experience to ensure that they maintain their consistency even through non-ideal conditions.

The weight sensor must also be calibrated to ensure that it will tare correctly when applied to the coffee roaster. After it has been determined that the sensor can ignore the weight of the roaster, we will have to test its ability to zero correctly. In order to do this, we will add varying amounts of coffee beans to the drum and verify that what it is measuring is the correct weight of the beans in the drum. Next, we will need to test how the sensor reacts to the roasting cycle. As the beans lose their water weight, the sensor should accurately show the change. A certain portion of the beans weight is water, and, depending of the amount of beans that are initially put in to the roaster, the temperature sensor needs to be able to measure the change and determine the difference throughout the roasting process. An accurate tare is imperative as it will determine when the coffee roaster will end the roasting cycle. If the tare and zero operations are unsuccessful, the roaster will not turn off at the appropriate times, causing the beans to either be burnt or under roasted.

The microphone will need to be tested to ensure it can send signals to the MCU throughout the roasting process. In tandem, the narrow bandgap filter that is applied to the input from the microphone will need to be tested as well. To test the microphone, we will measure its output through a range of frequencies that it will encounter as the beans roast. Once we have confirmed that microphone responds correctly to the range of frequencies, we will connect it to the narrow bandpass filter. Next, we will run the same test on the microphone again. This time, we will be measuring the output from the narrow bandpass filter to ensure that it has the proper cutoff frequencies to filter out the unwanted frequencies. In order to keep the testing uniform, the frequencies we use to test will be at the same amplitude. After testing the microphone with uniform amplitude signals, we will vary the amplitude to simulate the environment of the coffee roaster. We will ensure that the microphone's sensitivity is high enough to pick up on the cracking of the beans, but not too sensitive to pick up any other ambient sounds that may share the same frequency range.

### 9.1.3 PCB

Major issues that arises with PCBs are the lack of traces or a connection where you did not design one, and a short occurs. These issues cause the PCB to not operate in the manner it was designed, if operate at all. These errors can be catastrophic to the rest of the elements and components on the PCB. A short or an open in your PCB can redirect current from your power system to highly sensitive components and destroy them. In order to test these components, we will need a digital multi-meter, an oscilloscope, and safety goggles in case of a component, such as a capacitor, exploding.

The soldering for each component and element needs to be inspected to ensure a good connection. We will incorporate a digital multi-meter for the dead testing. We will measure the resistors to ensure the resistance is within its tolerance, and within what our circuitry needs to operate. The PCB will be tested over each

individual element, as well as over the pins and pin headers of the major components.

After the dead testing is complete, and the power system has been fully vetted, we will connect the power system to the board and then, where appropriate, use either an oscilloscope or a digital multi-meter to ensure the components are operational to the specifications of our design. If they do not perform or behave as expected, the traces will need to be inspected to ensure connectivity, consistency, and accuracy with our design.

### 9.1.4 Safety

It is vital that while testing the power system, PCB, and sensors that proper safety precautions are followed. When testing the power system, there is a risk of fire or explosion of the elements. One team member will be at the emergency shutoff switch and another team member will have a fire extinguisher. When testing the sensors, team members will ensure that they are not interfering with the signals being generated by the sensors, whether that is from ambient noise, weight, or proximity to the sensor itself. While testing the PCB, team members will ensure that they are properly grounded to avoid electrostatic discharge onto the components and elements. Team members will also ensure proper grounding for the PCB and a safe and clean environment on and around the PCB. The PCB will be cleaned with pressurized air before connecting power or testing the components.

Before working with any circuitry, we will ensure that the power is disconnected from the roaster. Further, we will ensure that any high voltage power storing devices are discharged by using a high voltage, insulated wiring to ground them. This will ensure that even if power is stored in a device, it cannot be transferred to a human and thus further reduce the risk of electrocution. We will also make use of an electric field testing instrument before touching the chassis once the unit is plugged in. This will protect against injury in case there is a voltage applied to the chassis itself.

Before working with any circuitry, we will ensure that the power is disconnected from the roaster. Further, we will ensure that any high voltage power storing devices are discharged by using a high voltage, insulated wiring to ground them. This will ensure that even if power is stored in a device, it cannot be transferred to a human and thus further reduce the risk of electrocution. We will also make use of an electric field testing instrument before touching the chassis once the unit is plugged in. This will protect against injury in case there is a voltage applied to the chassis.

## 9.2 Software

The following subsections outline how the project software was/will be tested. Both the front end applications and back end server API will be communicating. Postman will be used to verify API request functionality. Xcode Simulator and team member's devices will be used to test the front end applications.

### 9.2.1 iOS & watchOS Application

Successful communication between the coffee roaster and the iOS and watchOS applications is crucial for verifying operation of the system. The end goal of the project was for any iOS user to have control of the coffee roaster wirelessly over a Wi-Fi connection. Users that also own Apple Watches will be able to have access to unique controls and features of the coffee roaster. Functionality of the iOS application will be tested thoroughly on multiple iPhone devices to verify compatibility across different screen sizes and hardware. TestFlight Beta Testing provided by Apple will be utilized to provide users access to the various iterations of the application during development to validate intended features and components of both the iOS and watchOS applications. Users will be able to download the TestFlight application from the App Store in order to be invited to the numerous beta versions of the iOS and watchOS applications.

The different aspects of testing for the iOS application include:

- New user Sign Up
- Home Page
    - Browse Popular Roasts
    - Application Update Notes
    - Browse Roast Profiles (Light/Medium/Dark)
- Roast Type Page
    - Start/End Roast
    - View Progress of Roast
        - Roast Graph & Loading Bar
    - Modify Temperature and Time
- Notification
    - Roast Complete
- Settings
    - View History of Completed Roasts
    - View/Modify List of Favorites
    - View/Modify Default & Custom Roast Profiles
    - Order coffee beans from Amazon.com

The different aspects of testing for the watchOS application include:

- Home Page
    - Browse Roast Profiles (Light/Medium/Dark)
    - Start/End Roast
    - View Progress of Roast
    - Access Settings
- Roast in Progress Page
    - Roast Graph & Loading Bar
    - Force Touch to End Roast

## 9.2.2 API

The API for our project acts as the middle man of communication between our iOS application and the smart coffee roaster. Meaning if it does not function our project itself will not work. In order to test the API, we made use of Postman to verify routes and Mocha to verify functionality.

### 9.2.2.1 Security

The security of our API is primarily based around JWT. In order to determine the integrity of our authentication method that we will be using I first verified that when the API receives an accurate username and password that we are given JWT token. I first created a user with username ‘testing’ and password ‘testing’. I then attempted to sign in with the user and received the following confirmation shown in figure 9.2.2.1-1. Once the client receives the JWT token the client would store it in local storage and attach it to the header for future requests.

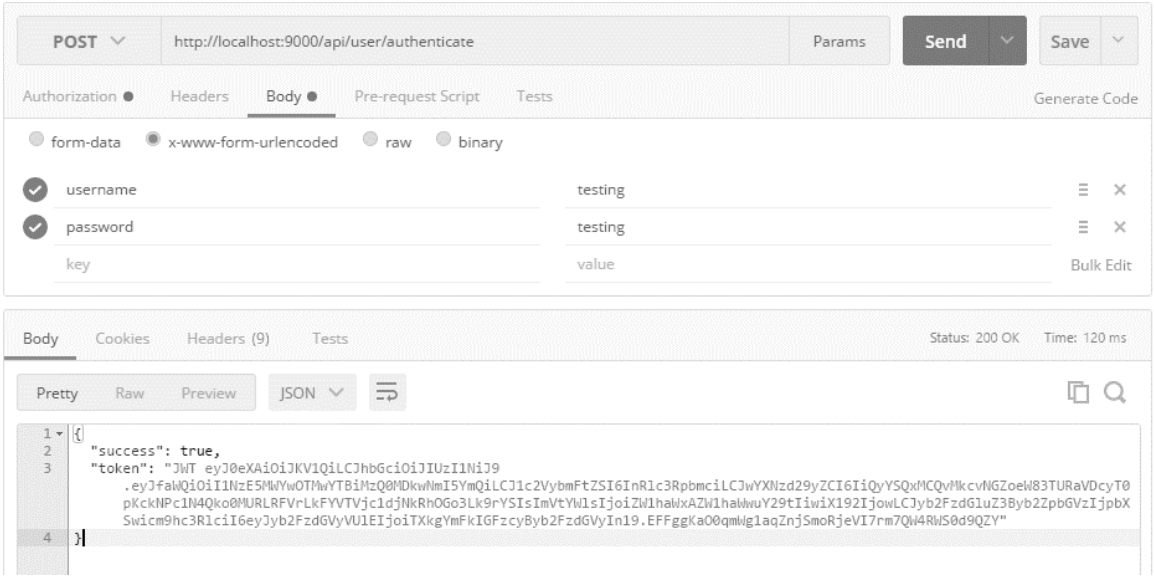



*Figure 9.2.2.1-1 Successfully logged in*

Then we inputted an incorrect username and password. Specifically, we inputted username ‘testing’ and password ‘randompassword’ and ended up with the following response payload shown in figure 9.2.2.1-2.

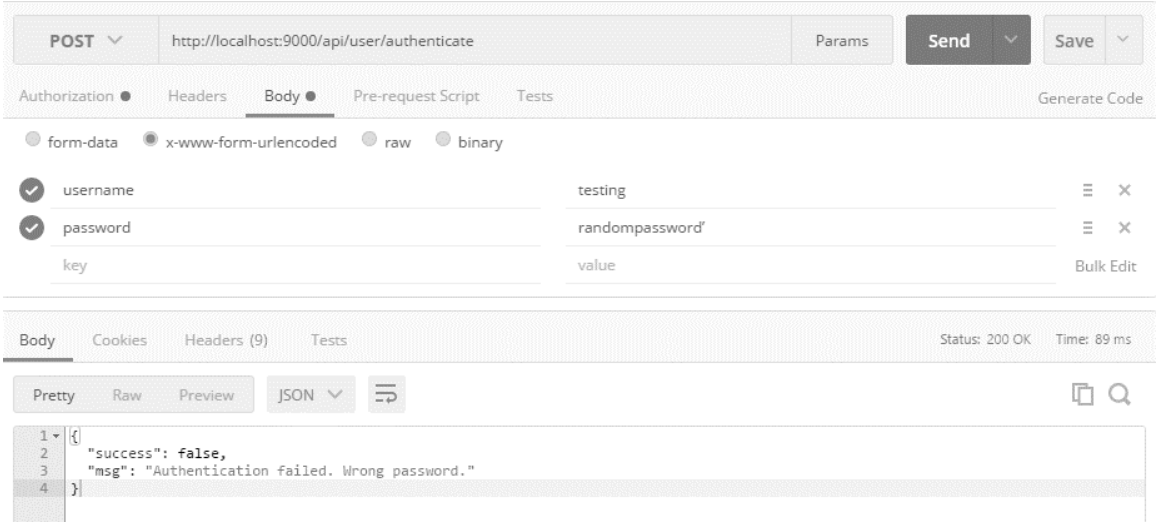



*Figure 9.2.2.1-2 Wrong Password Received*

Now that we have our JWT token we can know access data that requires authentication. For example, if we wanted to access a special member area of our application the request and response payload would look like below in figure 9.2.2-3. Keep in mind that the API received the JWT from the client via the header.

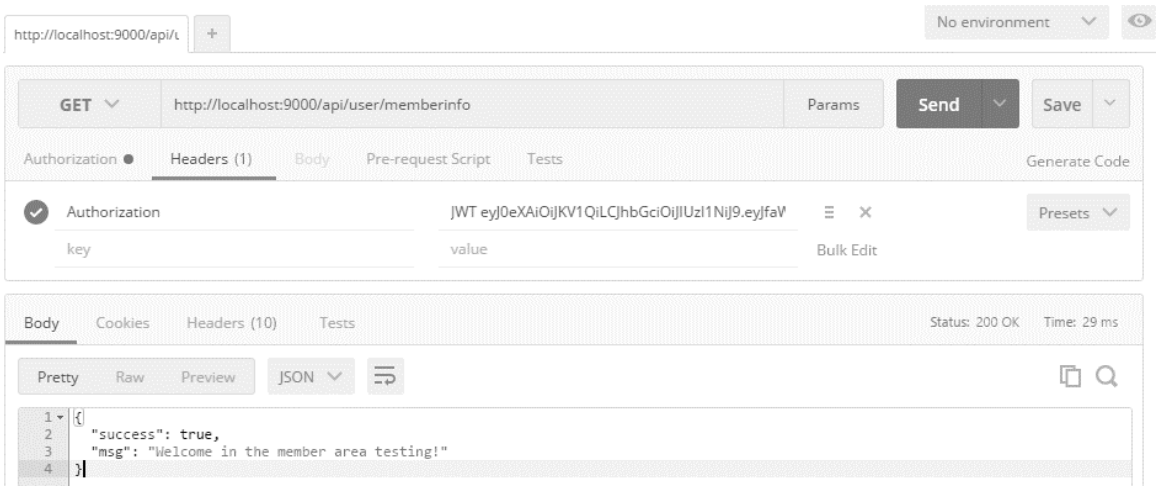



*Figure 9.2.2.1-3 Access to member route granted*

## 9.2.2.2 Functionality

Once I’ve tested some general functionality manually e.g. through the Postman application, we began to make use of the Mocha testing framework. Given Nodejs’ asynchronous nature use of Mocha is ideal given its asynchronous support. First

off, we ensured the functionality of the user functions that publish, pull and modify user data.

# 10.0 Billing

Our project is largely self-funded, with minor funding coming from some external sponsors in the form of materials or supplies. These sponsors include the following:

Sweet Maria's – 8LBS of unroasted coffee of several varieties. This allows us to determine how our roaster performs on a wide range of coffee sources. ARV: $48.49

Blessed Beans Coffee – Approximately 25LBS of unroasted coffee from a single source. This will allow us to get major functionality of the roaster complete and provide a large amount of coffee for testing purposes. ARV: $105.24

Texas Instruments – $100 to spend in the TI e-store, provided through the TI Innovation Challenge. Will provide for the microcontroller, wifi controller, power management ICs, microphone pre-amp, and analog filter for microphone signal. ARV: $100

All expenses beyond these will be split among team members with the following cost distribution:

- Evan Baytan – 20%
- Samuel Roman – 20%
- Patrick Sites – 20%
- Brian Webb – 40%

This cost distribution was chosen because the team agreed to allow Brian to keep the roaster when the project was complete. Thus far, the team has spent the following in non-sponsored funds:

1. Behmor 1600 Plus Coffee Roaster – $395.89
2. Crack Detection – $ 18.14
3. DC Power System – $25.57
4. AC Power Control System – $75.53
5. DC Power Control System – $2.19
6. PCB Manufacturing – $250
7. PCB Placement – $25
8. Miscellaneous Expenses – $15.00
9. Equipment – $45
10. Total – $852.32

These costs include all fees, including shipping, handling, and taxes.

Each team member therefore has a cost of $170.46, with the exception of Brian, he has a share of $340.93. Costs of some aspects may be minimized by utilizing free samples, though this is not a guaranteed reduction. Therefore, we will plan on paying the full amount that was calculated here.

# 11.0 Milestones

As with any large project, a list of project milestones is important to have. It helps ensure that the project progresses in a timely manner. Milestones for our project are listed in table 11-1 below. Due to the large width of the table, it is presented in a portrait, rather than a traditional landscape view.

We were able to achieve most of the milestones we set for ourselves at the beginning of the project. On the hardware side, the AC and DC power systems were designed. The weight sensing, crack detection, and temperature sensing subsystems were completed. The main PCB components, including the BOOSTXL-K350QVG-S1 LCD Display, the CC3200MOD Wi-Fi chip, and the MSP432P401R microcontroller unit, were finalized and added to the Eagle schematic. A few changes in the MCU choice, as well as the MCU, LCD Display, and the Wi-Fi chip not being located in the Eagle libraries, set back the development of the schematic. Now that all of our subsystems are designed, and the majority of the low level development is finished, each subsystem can be added to the main schematic in Eagle, and the power system can begin to be realized as well. Unfortunately we were not able to reach a working prototype at this point due to these setbacks.

On the software side the website with login routes and browsing capabilities has been completed. The UI skeleton for the iOS application has been completed as well. The final iOS application is in development, along with the accompanying watchOS application. The embedded software required for the MCU to communicate with the other peripheral has begun development.

Moving forward, the main PCB with all the added peripherals will be realized in Eagle and the PCB will be ordered in triplicate to account for any errors. The power system will also be built in Eagle and ordered as soon as it is finished. The website will continue to be tested and debugged, to ensure that as more communication avenues are added it will continue to operate as expected. The watchOS application will be paired with the iOS application to ensure continuity. Once the hardware is procured and assembled, it will be integrated and tested as per our specifications. In the meantime, the chassis and housing unit will be built and modified to maintain as little downtime as possible.

| Senior Design I Weekly Milestones | | | Legend: |
|---|---|---|---|
| Spring 2016 - Summer 2016 | | | Completed |
| | | | Behind |
| | | | In progress |
| | | | |
| | | | |
| **Week** | **Docs** | **Hardware** | **Software** |
| 4 | Draft ToC | | |
| 5 | Final ToC | MCU Chosen, Eagle/Software | Login routes (api) |
| 6 | | Research what we need | |
| 7 | Break Sections | | Browsing, Roast Profiles |
| 8 | ToC Due | Obtain Coffee Roaster | iOS UI Skeleton, Onboard Controls routing |
| 9 | 6 pages/person | | |
| 10 | 6 pages/person | Complete components (Wi-Fi, BT, etc) | Research Embedded Software |
| 11 | 6 pages/person | Design AC Power Control System, DC Motor Control System | Begin iOS App Development |
| 12 | Final Draft | Design Weight Sensing Subsystem, Crack Detection circuit | Begin Embedded Software Development |
| 13 | Final Complete | Design Primary Heating Element Cirucit, Complete PCB | |
| 14 | Revisions | | |
| 15 | Revisions | Eagle schematic | |
| 16 | Final Doc | Working prototype | iOS App, Start watchOS App |

# A.1 Data Sheets

http://www.linear.com/docs/3269

http://www.cui.com/product/resource/digikeypdf/cmr-5054tb-a.pdf

http://www.ti.com/lit/ds/symlink/tl971.pdf

http://www.semicon.panasonic.co.jp/ds4/MTM86227_E.pdf

http://www.onsemi.com/pub_link/Collateral/NGD8201N-D.PDF

http://optoelectronics.liteon.com/upload/download/DS-70-99-0012/S_110_4N35%20%204N37%20(%20M,%20S,%20S-TA1%20)%20(Rev.pdf

http://www.ixysic.com/home/pdfs.nsf/www/CPC1998.pdf/$file/CPC1998.pdf

http://www.onsemi.com/pub_link/Collateral/MAC4DHM-D.PDF

http://www.ti.com/lit/ds/symlink/lmz30602.pdf

http://www.ti.com/lit/ds/symlink/tl317.pdf

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf

http://www.ti.com/lit/ds/symlink/cc3200mod.pdf

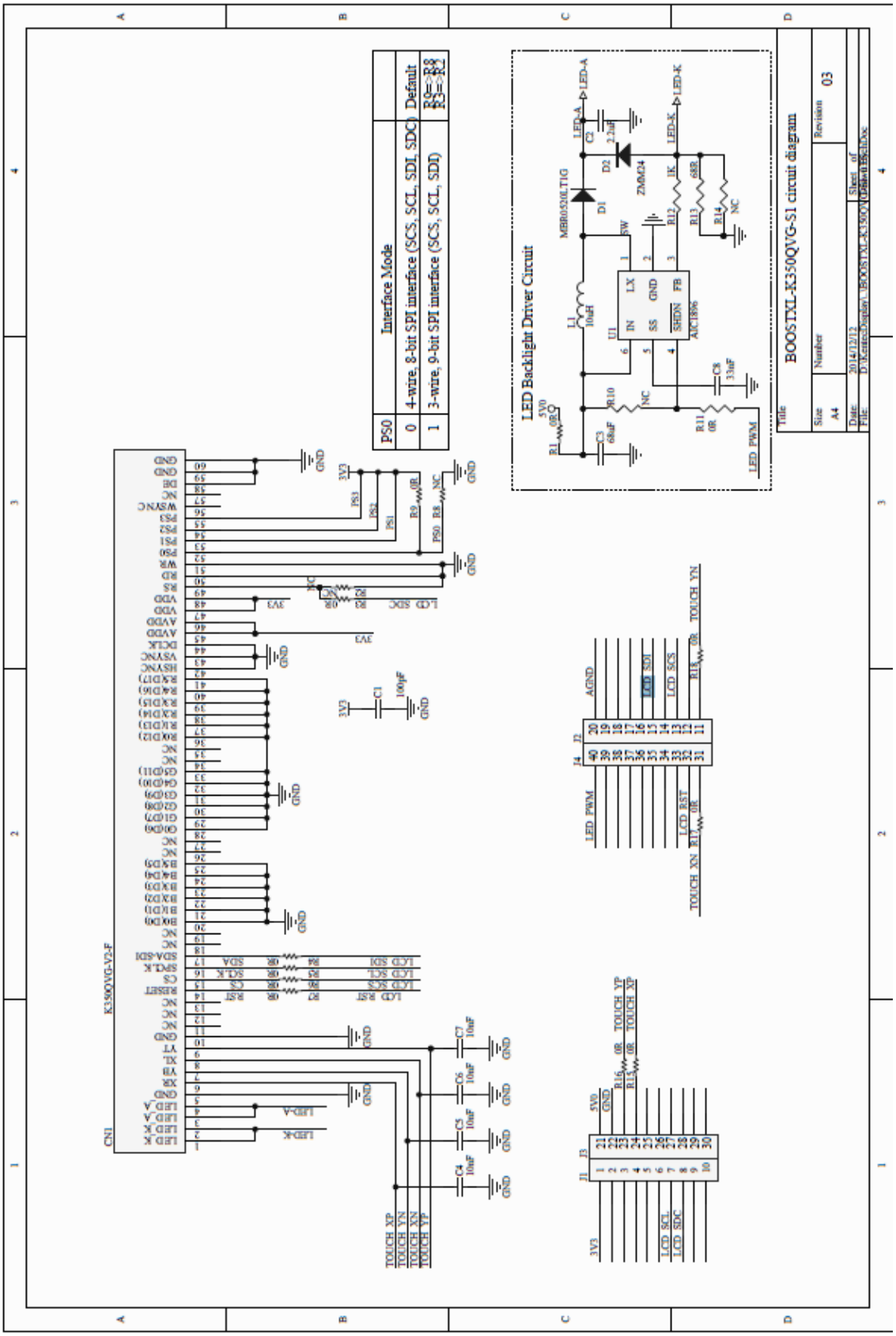



Figure A.1-1: BOOSTXL-K350QVG-S1 LCD Display

# A.2 Copyright Permissions

Figure 2.3.6-1

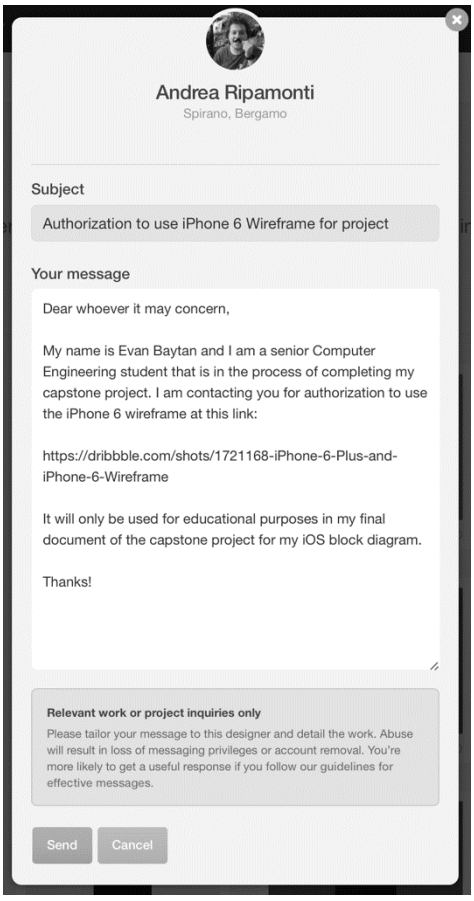



Figure 4.4.4.1-1

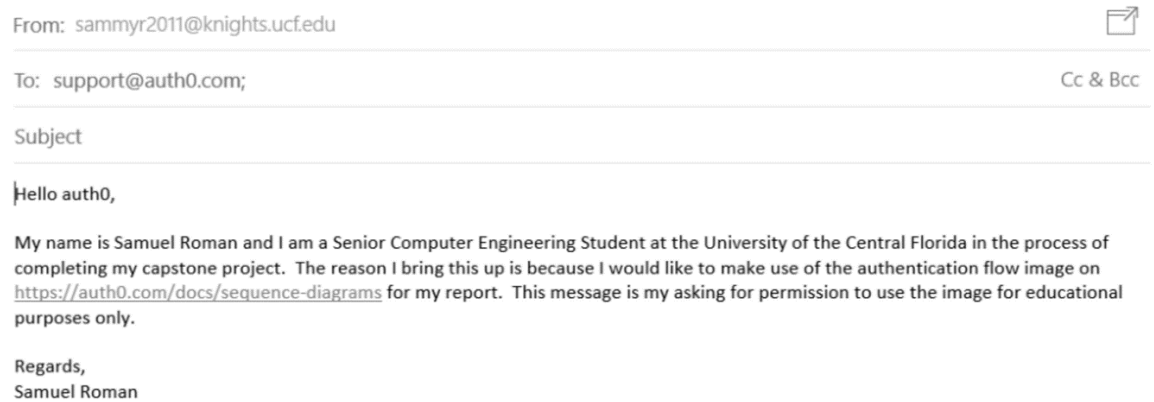
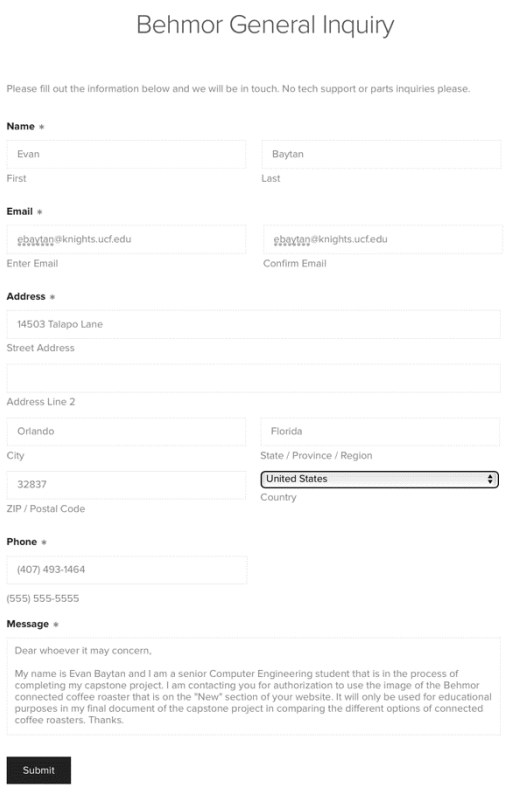


Figure 5.1.1.1-1



Behmor General Inquiry

Please fill out the information below and we will be in touch. No tech support or parts inquiries please.

**Name** *

Evan — First

Baytan — Last

**Email** *

ebaytan@knights.ucf.edu — Enter Email

ebaytan@knights.ucf.edu — Confirm Email

**Address** *

14503 Talapo Lane — Street Address

Address Line 2

Orlando — City

Florida — State / Province / Region

32837 — ZIP / Postal Code

United States — Country

**Phone** *

(407) 493-1464

(555) 555-5555

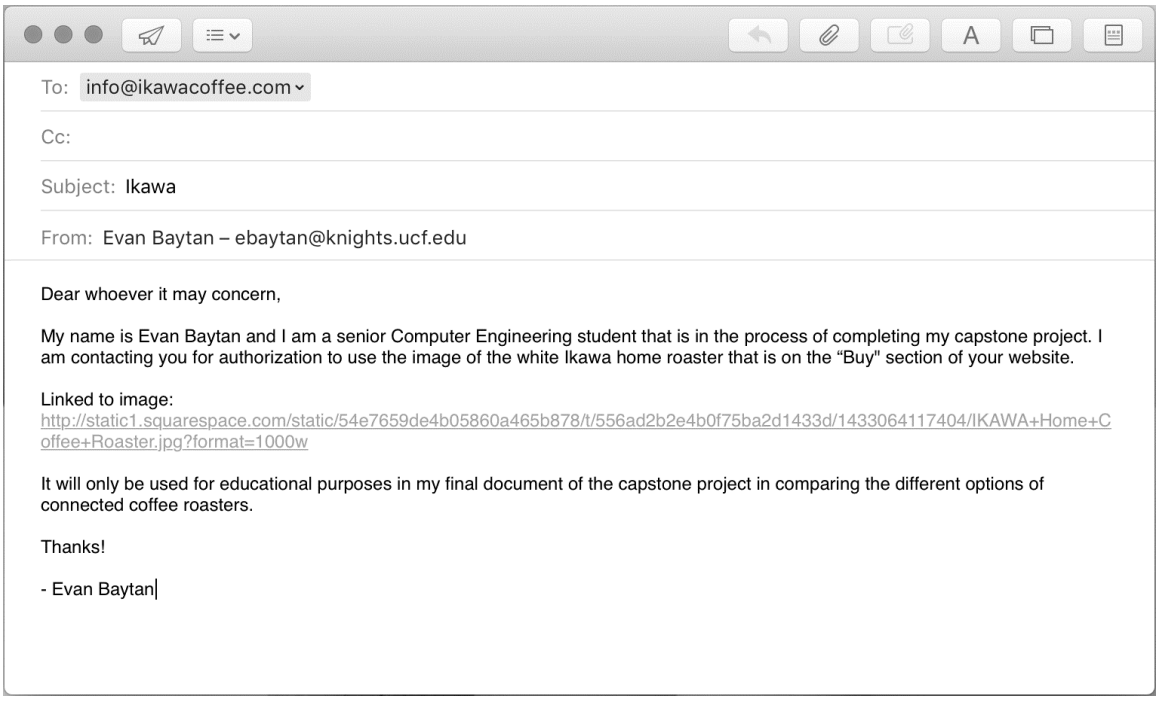
**Message** *

Dear whoever it may concern,

My name is Evan Baytan and I am a senior Computer Engineering student that is in the process of completing my capstone project. I am contacting you for authorization to use the image of the Behmor connected coffee roaster that is on the "New" section of your website. It will only be used for educational purposes in my final document of the capstone project in comparing the different options of connected coffee roasters. Thanks.

Submit

Figure 5.1.1.1-2



To: info@ikawacoffee.com

Cc:

Subject: Ikawa

From: Evan Baytan – ebaytan@knights.ucf.edu

Dear whoever it may concern,

My name is Evan Baytan and I am a senior Computer Engineering student that is in the process of completing my capstone project. I am contacting you for authorization to use the image of the white Ikawa home roaster that is on the “Buy" section of your website.

Linked to image:
http://static1.squarespace.com/static/54e7659de4b05860a465b878/t/556ad2b2e4b0f75ba2d1433d/1433064117404/IKAWA+Home+Coffee+Roaster.jpg?format=1000w

It will only be used for educational purposes in my final document of the capstone project in comparing the different options of connected coffee roasters.

Thanks!

- Evan Baytan

Figure 5.1.1.3-1

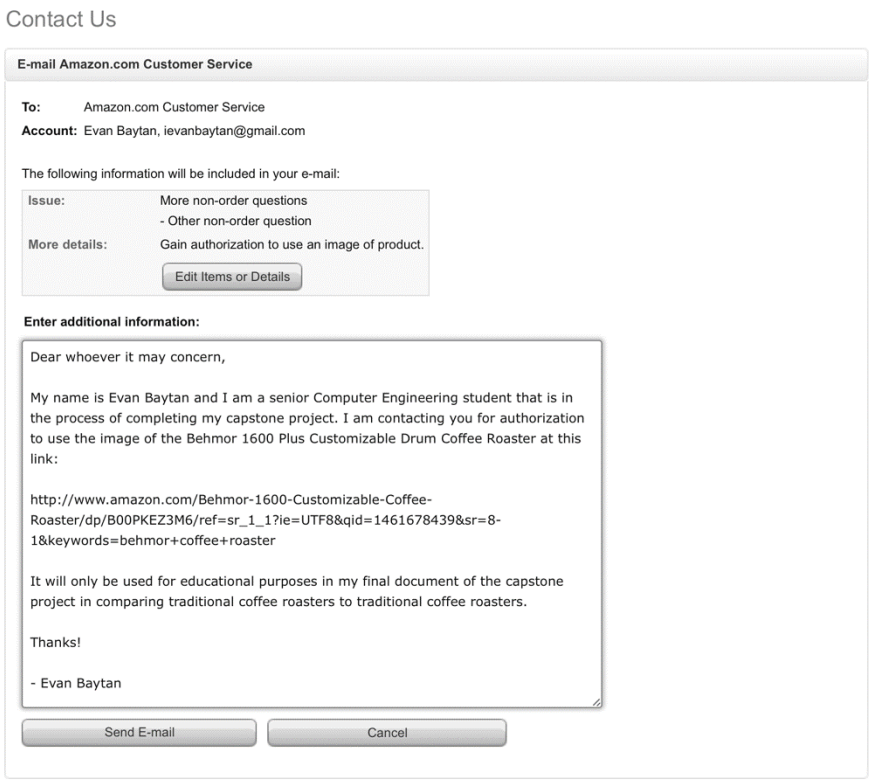



Figure 5.1.1.3-2

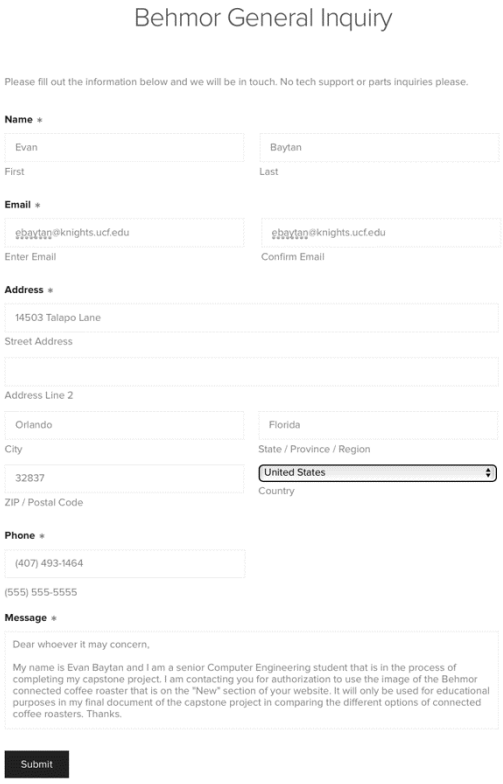
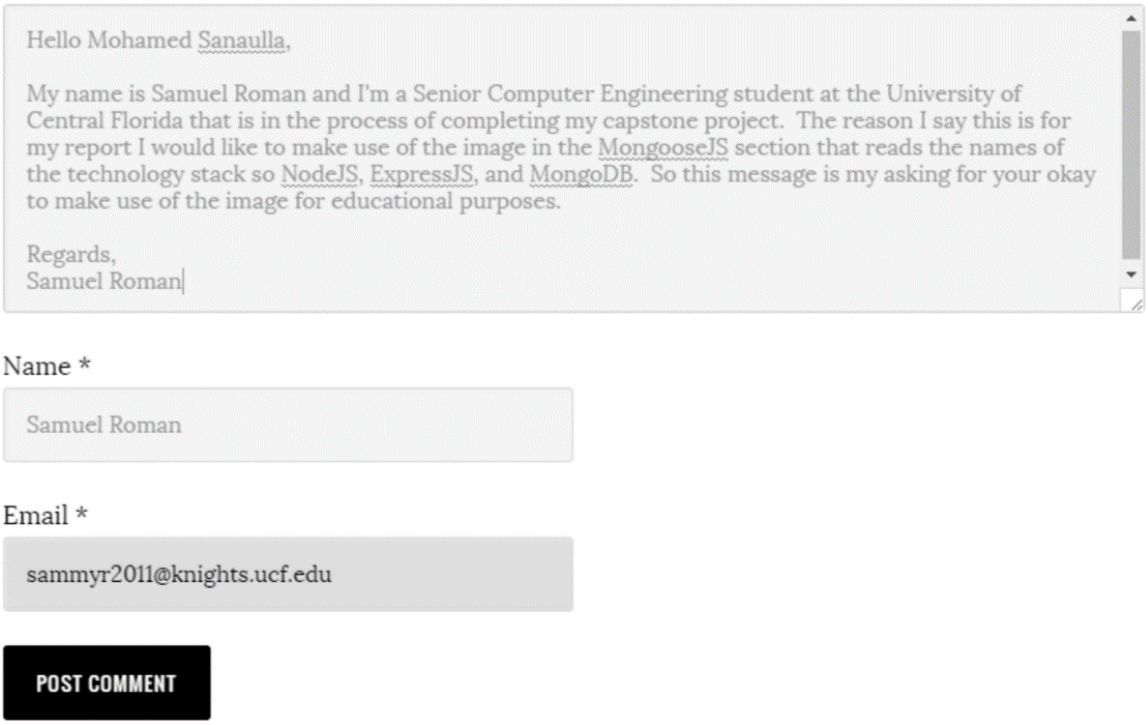


Figure 7.1-1




Figures 7.2.1-1, 7.2.1-2, 7.2.1-3, 7.2.1-4

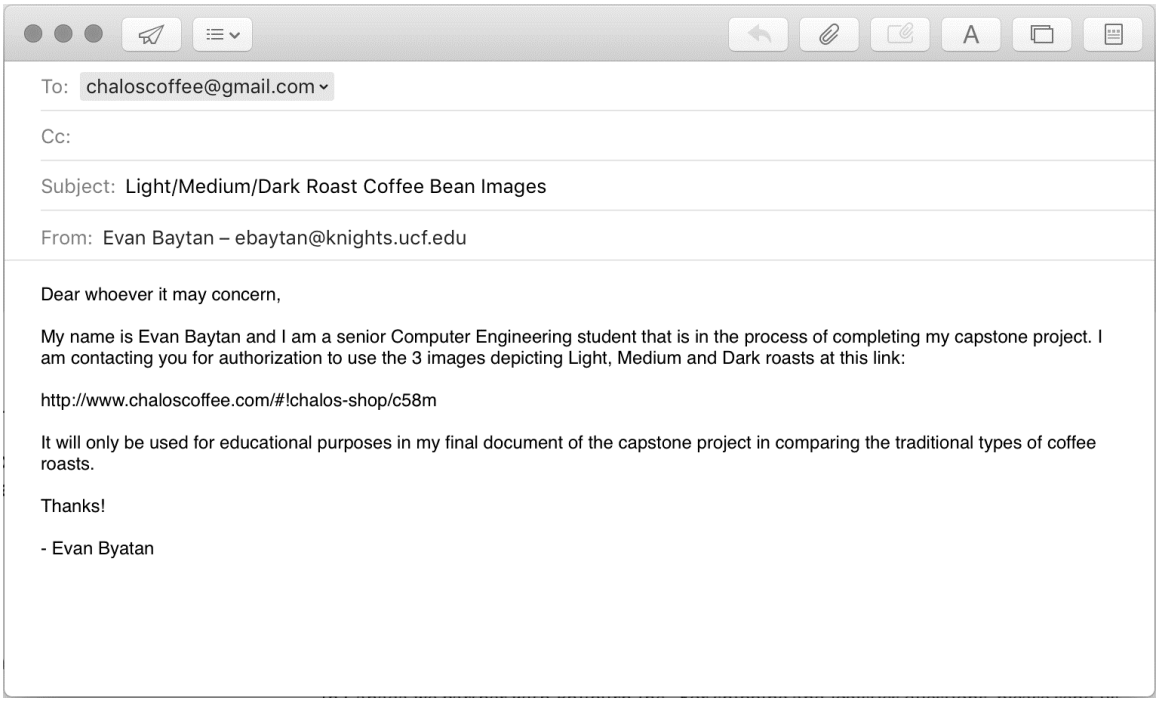


Table 6.4-1:

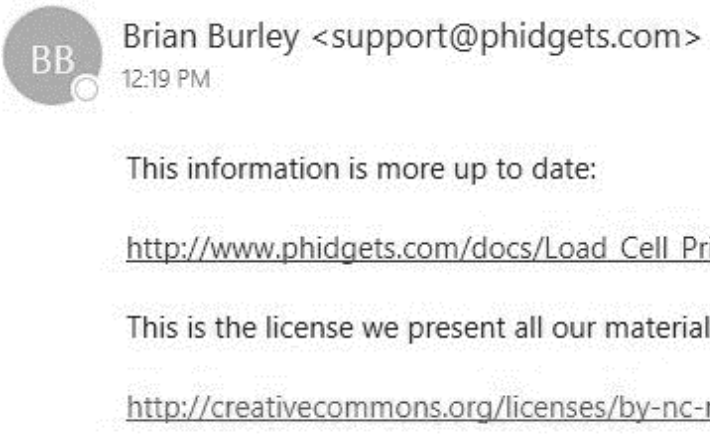
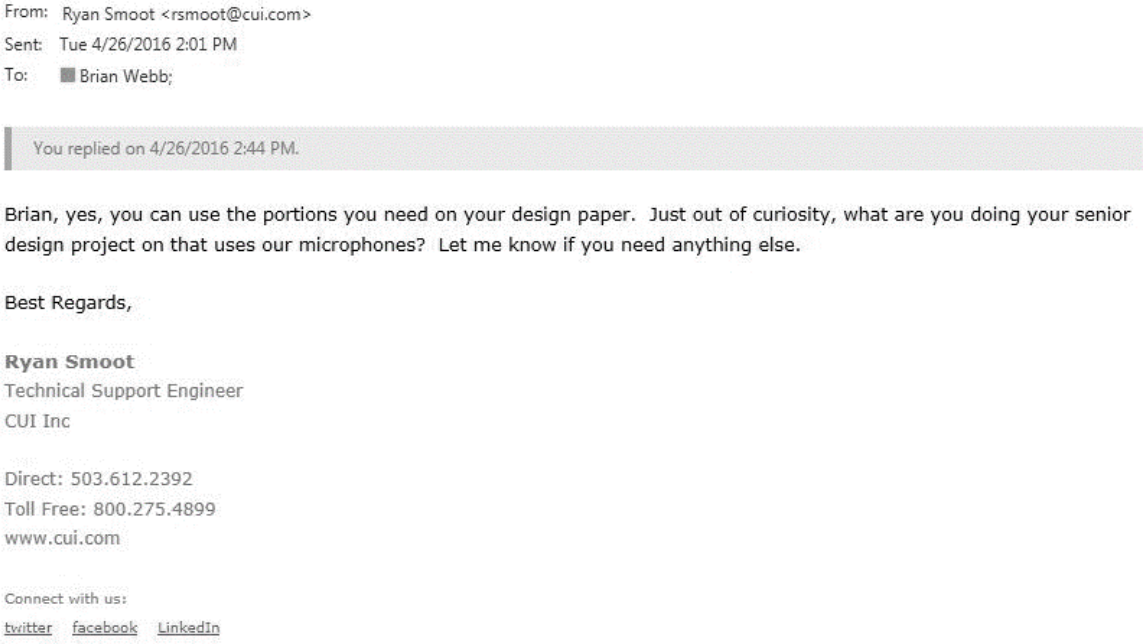



Figure 2.3.6-1



# A.3 References


[1] iPhone Wireframe

http://www.cssauthor.com/wp-content/uploads/2014/01/iPhone-outline-Wireframe-Mockup-PSD.jpgh

[2] Apple Watch Wireframe

https://dribbble.com/shots/2256219-Apple-Watch-Wireframe-for-SketchApp-FREEBIE

[3] CUI Microphone Datasheet with Specifications & Response Curve

http://www.cui.com/product/resource/digikeypdf/cmr-5054tb-a.pdf

[4] Classic Bluetooth vs. Bluetooth Smart (LE) Specifications

https://en.wikipedia.org/wiki/Bluetooth_low_energyh

[5] 802.11a/b/g/n/ac Specifications

https://en.wikipedia.org/wiki/IEEE_802.11 - cite_note-10h

[6] Clear Server Client

http://docs.daisy-pipeline.googlecode.com/hg-history/b5810183400e86db83330a655955b45c9be62842/2012/2012-02-workshop/presentations/pipeline2-rest-api.html#slide-2

[7] Behmor Connected Coffee Roaster

http://behmor.com/wp-content/themes/behmor/assets/img/roaster-sm.pngh

[8] Ikawa Coffee Roaster

http://static1.squarespace.com/static/54e7659de4b05860a465b878/t/556ad2b2e4b0f75ba2d1433d/1433064117404/IKAWA+Home+Coffee+Roaster.jpg?format=500wh

[9] Behmor Coffee Roaster

http://ecx.images-amazon.com/images/I/61CrYUCDXaL._SL1000_.jpg

[10] DC Stepper Motor, no changes were made

https://en.wikipedia.org/wiki/Stepper_motor#/media/File:Stepper_motor.jpg

[11] AC Induction Motor, Rotating Field

https://en.wikipedia.org/wiki/Induction_motor#/media/File:Rotatingfield.png

[12] Microphone Pickup Patterns

https://en.wikipedia.org/wiki/Microphone#Microphone_polar_patterns

[13] JWT Sequence Diagram

http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html

[14] iOS 9 iPhone Compatibility

http://www.apple.com/ios/whats-new/

[15] iOS 9 iPad Compatibility

http://www.apple.com/ios/whats-new/

[16] Android Usage

http://developer.android.com/about/dashboards/index.html

[17] iOS Usage

https://developer.apple.com/support/app-store/

[18] TI Whitepaper on Analog-Digital Ground Plane

http://www.ti.com/lit/an/slyt499/slyt499.pdf

[19] Load Cell Datasheet with Specifications & Related Information

[20] Stack

http://codek-tv.blogspot.com/2016/02/node-js-mongodb-node-js-mongodb.html#!

[21] Screenshots of Light/Medium/Dark Roasts

http://www.chaloscoffee.com/#!chalos-shop/c58m

[22] SPI Three Slaves

https://commons.wikimedia.org/wiki/File:SPI_three_slaves.svg

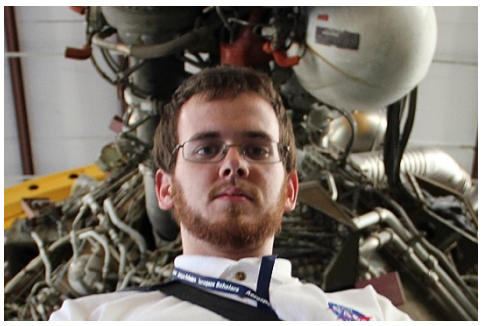
[23] RS232

https://en.wikipedia.org/wiki/RS-232 - Development_tools

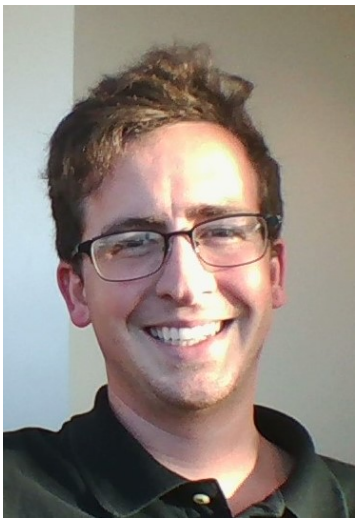
## A.4 Biographies

- Evan Baytan
- Graduating in Summer 2016
- Majoring in Computer Engineering
- Hobbies:
    - Cycling
    - Video Games
    - Learning new programming languages
- Plans After Graduation:
    - Apple Hardware Reliability Engineering (Interviews in progress)
- Contributions to Project:
    - Developed iOS & watchOS Applications
    - Developed portion of Embedded Software

- Samuel Roman
- Graduating in Summer 2016
- Majoring in Computer Engineering
- Hobbies:
    - Video Games
    - Learning new technologies
    - Building software applications
- Plans After Graduation:
    - Aiming to become a Full Stack Software Engineering focusing in cloud computing specifically distributed computing.
- Contribution:
    - Designed the RESTful API
    - Collaborated on the Embedded Software Programming



- Brian Webb
- Graduating in Summer 2016
- Hobbies:
  - Playing Bass, Guitar, and/or Drums
  - Learning about and observing space flight
  - Baking
  - Photography
- Plans After Graduation:
  - Brian looks to pursue his Master's Degree while working in a professional environment. He has interviewed with US Airforce civilian engineering and would love to work in defense or aerospace.
- Contribution:
  - Took a partial PM role, came up with project vision and conducted many interviews with industry professionals
  - Designed all electrical subsystems



- Patrick Sites
- Graduating in Summer 2016
- Majoring in Electrical Engineering
- Hobbies:
  - Diving
  - Biking
  - Mycology
- Plans After Graduation:
  - Planning on going into Satellite Simulations with NASA
- Contributions to Project:
  - Developed MCU and main system schematics
  - Collaborated on hardware and sensing systems