University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Brew It Yourself

An Automated Single Vessel Home Brewery System

Group Number: 2016.019
Consultant: Douglas Harder

Kevin Nause (20413332)
Mathieu Tremblay (20420813)
Scott Wood (20379649)
Steve Jung (20411563)

Date: July 2, 2015

Contents

1 High-Level Description of Project **2**
1.1 Motivation . . . 2
1.2 Project Objective . . . 2
1.3 Block Diagram . . . 2
1.3.1 Description of System . . . 3
1.3.2 Designing and Not Designing Components . . . 4

2 Project Specifications **5**
2.1 Functional Specifications . . . 5
2.2 Non-Functional Specifications . . . 7

3 Detailed Design **8**
3.1 Mechanical Design . . . 8
3.1.1 Material . . . 8
3.1.2 Tank Structure . . . 8
3.2 Electrical System . . . 8
3.2.1 Heating and Cooling Controller . . . 8
3.2.2 Motor Controller . . . 8
3.2.3 Timing Actuators . . . 8
3.3 Computer System . . . 8
3.3.1 Embedded Computer . . . 9
3.3.2 Sensors and Micro-Controllers . . . 9
3.3.3 Web Server and Database . . . 12
3.4 Mobile Application . . . 13
3.4.1 Real Time Data . . . 14
3.4.2 Input Controls and Interaction . . . 16

4 Discussion and Project Timeline **17**
4.1 Evaluation of Final Design . . . 17
4.2 Use of Advanced Knowledge . . . 17
4.3 Creativity, Novelty, Elegance . . . 17
4.4 Student Hours . . . 17
4.5 Potential Safety Hazards . . . 17
4.6 Project Timeline . . . 17

Acronyms **18**

Beer Terminology **18**

Technical Terminology **18**

1 High-Level Description of Project

1.1 Motivation

The art of home brewing has been steadily gaining popularity over the past 35 years alongside the rise of craft breweries in North America, so much so that in 2010 there were over 2000 craft breweries in the United States, after starting with only 8 in 1980 [1]. The traditional method for homebrewing requires various components, constant monitoring and heavy maintenance. There should be a solution which reduces complexity, making it much more affordable and practical for home use. We hope to create a single vessel system that would make the home brewing process precise, automated and compact, all at a reasonable price.

1.2 Project Objective

The objective of this project is to combine homebrewing experience with engineering design, and construct a single vessel brewing system. By maintaining a strict control of key parameters, the brewing process is regulated using a combination of fluid mechanics, heat transfer, digital controls, power systems, embedded robotics and mobile development. The Robotics Operating System (ROS), allows for a design where sensors can be added to a modular setup and provide feedback. By receiving feedback from temperature readings, density measurements and pH monitoring, the brewing process can be accurately recorded, shared, and automated by the system.

1.3 Block Diagram

Figure 1 shows relevant links between the physical, mechanical design of the brewing system, as well as the computer systems and their underlying software.

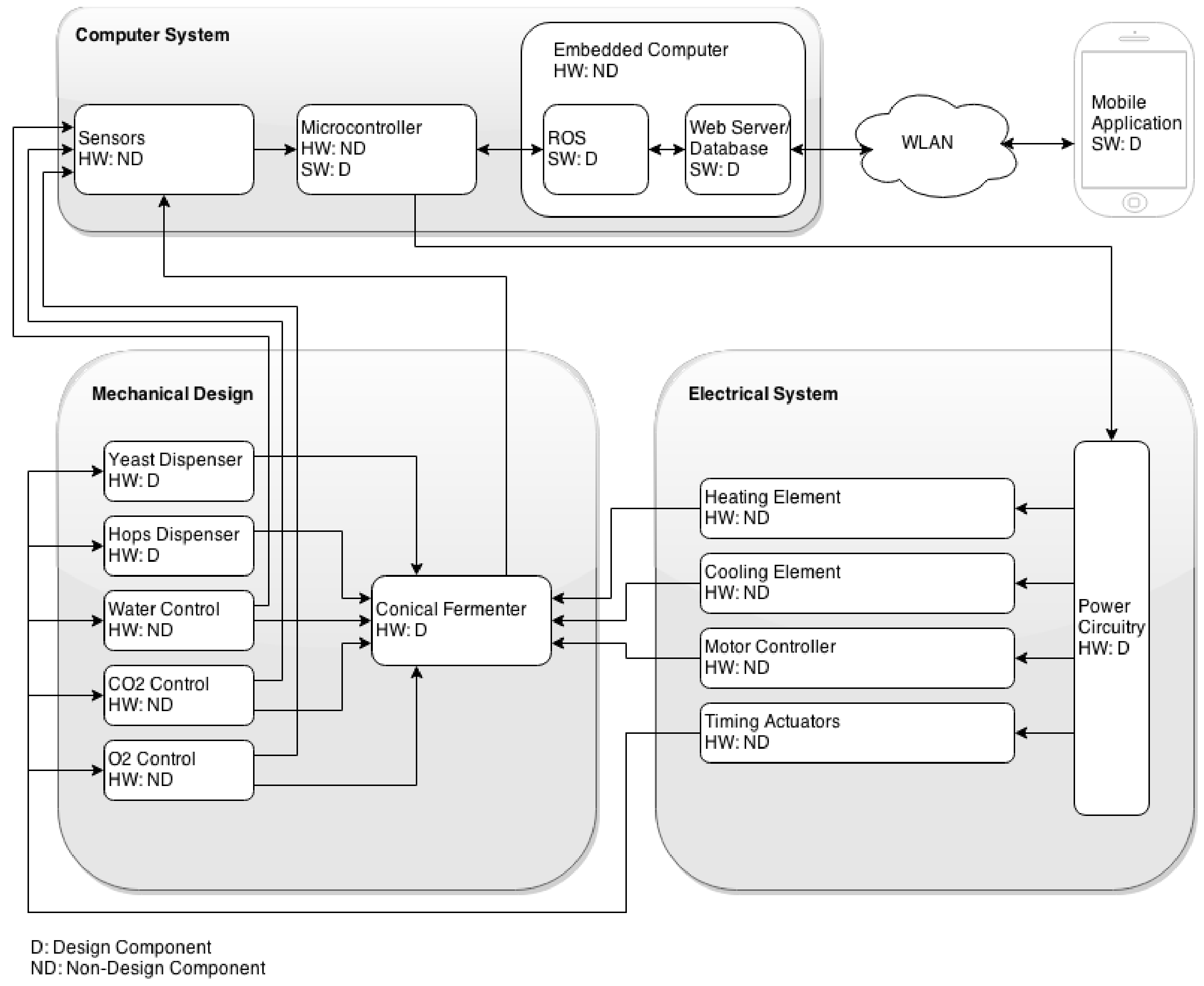

Figure 1: Block Diagram outlining the interactions between the computer, electrical, and mechanical systems

1.3.1 Description of System

The single vessel brewing system, as shown in Figure 1, contains various sensors which forward the environmental parameters to the microcontroller. The data from the various sensors accurately represents the state of the fermenter and feed into digital control loops running on the microcontroller. Data from sensor modules is obtained by a microcontroller via an I2C interface. This data is then sent to an embedded computer system over USB where it is logged in a database and used to control relevant subsystems. Control commands are sent from the embedded computer to the microcontroller, where the appropriate component can be communicated with in order to regulate the brewing environment. The embedded computer system is then able to send diagnostic information and push notifications to a mobile device through a WLAN connection.

1.3.2 Designing and Not Designing Components

The following is an outline of the components in the block diagram and the requirements associated for each Designing (D) and Non-Designing (ND) component.

- **Sensors**
  ND: Sensors being used in the vessel are not being designed. Instead, off-the-shelf sensors are being purchased and integrated into the system. Relevant sensor types for this system include, but arent limited to, pH sensors, volume sensors, flow meters, temperature probes, etc.

- **Microcontroller(s)** ND: The hardware of the microcontroller is not being designed since thats outside the scope of this project. The system instead uses an off-the-shelf microcontroller. D: The software running on the microcontroller is being designed. This software mainly consists of digital control loops for interfacing with the sensors and power electronic circuitry.

- **Conical Fermentation Vessel**
  D: The mechanical features and dimensions of the conical fermentation vessel are being designed to hold various sensors and other electrical and mechanical components.

- **Power Electronic Circuitry**
  D: The power electronic circuitry is being designed to power the pumps, motors, actuators, and heating/cooling mechanisms. The circuitry takes input from the digital controller running on the microcontroller and outputs the appropriate power to the end devices.

- **Heating and Cooling Elements**
  ND: The heating coils and refrigeration unit used to regulate temperature of the vessel are not being designed. Instead, off-the-shelf or salvaged and adapted components from existing appliances (e.g. hot water tank coils, home air conditioning heat pump) are being used and the power electronic circuitry is being designed to power these devices.

- **Motor Controller**
  ND: The motors, pumps, and mixing mechanisms used in our system are not being designed. Instead, off-the-shelf or salvaged and adapted components from existing appliances (e.g. blender mixing prop, aquarium pumps) are being used and the power electronic circuitry is being designed to power these devices.

- **Timing actuators**
  ND: The actuators used in our system are not being designed. Instead, off-the-shelf solenoids and servo motors are being used and the mechanical subsystems that they will actuate are being designed. The power electronic circuitry is also being deigned to be able to power the actuators.

- **Embedded Computer System**
  ND: The hardware and operating system of the embedded computer system is not a design objective as it is outside the scope of this project. Instead an off-the-shelf embedded computer is purchased and a Debian based Linux operating system is installed to satisfy the requirements for the development environment.
  D: The embedded computer system is configured with a Web Server, Database, and the Robotics

Operating System (ROS). Microcontrollers can be modularly added to the system via USB and automatically configured as ROS nodes through negotiations.

- **Mobile Application**
  D: The mobile application is designed such that the user can receive push notifications and see various sensor data during the brew process.

2 Project Specifications

This section outlines the functional and non-functional requirements of the project design.

2.1 Functional Specifications

Table 2.1 describes each functional requirement and highlights whether it is essential or not to the completion of the project.

Table 1: An overview of each functional specification of the project

Specification	**Classification**	**Description**
Completion of the Brewing Process	Essential	The device automatically completes the brewing process, consisting of these steps: • Mash • Sparge • Boil Wort • Dispense Hops • Aerate the Wort • Pitch the Yeast • Kegging and Dispensing in its entirety and in the correct order. For more information on the brewing process and the terms used, please see the Glossary.
Heating Unit	Essential	Mashing the grains, sparging the grist, and boiling the wort all require high water temperatures. The system is able to accurately heat the contents to a minimum of 110°C within 1°C of error.

Cooling Unit	Essential	Yeast pitching and fermentation happen immediately after boiling the wort, but require temperatures a specific temperature (typically 20°C). The system is able to rapidly cool the wort to a temperature within 1°C of the target temperature so that the yeast may be pitched safely.
Temperature Regulation	Essential	The system is able to control temperature for each step in the brewing process. An error of 1°C is the target specification.
Trub Removal	Non-Essential	Remove the undesired sediment and other byproducts at the bottom of the fermenter so that ageing may take place within the vessel. The user does not have to maintain the trub accumulated by the brewing process.
Application Notifications	Non-Essential	The system provides notifications to the user updating them on the current state or action being taken. Additionally errors or warnings can be sent, in cases of emergency such as clogs, low oxygen or carbon dioxide, or an improper environment.
Database	Essential	The system is able query the specific steps it needs to automate for a specific brew. Additionally, it can store various data involved in brewing including temperature, pH levels and density measurements to generate logs and reports.
Reproducibility	Non-Essential	The system is able to record recipes, ingredients and steps for various brews. The user can recreate the same type of brew if they select one of the recipes.

2.2 Non-Functional Specifications

Table 2 describes each non-functional requirement and highlights whether it is essential or not to the completion of the project.

Table 2: An overview of each non-functional specification of the project

Specification	**Classification**	**Description**
Temperature Accuracy	Essential	The system accurately regulates the temperature required for the brewing process within 1°C of the targeted temperature.
Volume Control	Essential	The end result must be greater than or equal to the target yield volume. Target yields typically consist of 15.5 gallons, 7.75 gallons, and 5.16 gallons. These are the standard sizes of a half barrel keg, quarter barrel keg, and a sixth barrel keg respectively.
Size	Essential	The dimensions are limited such that the vessel can fit within a standard 36 inch residential door frame.
Mobility	Essential	The system is able to be relocated by a single person using the aid of caster wheels and or a standard 18 inch wide utility dolly.
Sanitation	Essential	The system employs SUS 304 stainless steel to maintain food-grade sanitation conditions. The vessel is self cleaning since a sanitary environment is crucial for the brewing process.

3 Detailed Design

3.1 Mechanical Design

The mechanical design component of this project is comprised of two aspects: the structural design of the fermenter and the thermodynamics of the heating and cooling systems. There are many sources for pre-fabricated fermenters for home brewing, but the custom nature of this device required the construction of a purpose-built tank, shown in figure 2. The tank features a double walled construction to enclose the heating and cooling systems of the inner tank as well as provide thermal insulation.

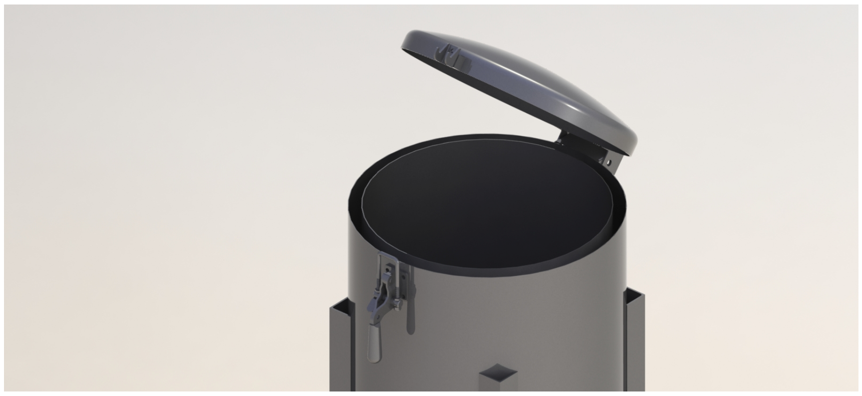

Figure 2: Illustration of the double wall design for the fermenter

3.1.1 Material

To meet the requirements of food grade equipment, the components of the fermenter that will come into contact with the brewing ingredients are to be constructed from stainless steel. While the Canadian Food Inspection Agency (CFIA) does not outline specific details for the materials to be used in food processing, it states that, Food contact surfaces of equipment and utensils are smooth, non-corrosive, non-absorbent, non-toxic, free from pitting, cracks or crevices, and able to withstand repeated cleaning and sanitation. [2]

3.1.2 Tank Structure

3.2 Electrical System

3.2.1 Heating and Cooling Controller

3.2.2 Motor Controller

3.2.3 Timing Actuators

3.3 Computer System

The computer systems consist of three main components, the embedded central computer, the micro-controller and sensor pairs, as well as a Web Server and Database pair to allow for storage and communications. The detailed description of each subcomponent and the methods of interaction between them is discussed in this section.

3.3.1 Embedded Computer

The primary function of the embedded computer system is to aggregate data in close to real-time from the multiple sensors used in the design. Analysis of this data is crucial for determining the current progress in the mash, sparge, and boiling of the wort. For specific analysis of sensor data please see subsection 3.3.2.

For ease of repeatability, availability, and cost the Raspberry Pi was selected. At the time of design and creation of this report, this is the most recent hardware revision of the Raspberry Pi. The relevant hardware specifications for the Raspberry Pi is a quad-core Acorn Reduced Instruction Set Computing (RISC) Machine (ARM) processor, 1GB of Random Access Memory (RAM), and a Bluetooth dongle for local wireless communications (rated to 50m). Since the outer shell of the fermenter is aluminum, the casing for the Raspberry Pi can act as a heat sink, allowing it to be safely overclocked to 1GHz, 500MHz, and 500MHz for the ARM processor frequency, sdram frequency, and L2 cache (core) frequency respectively. The configuration file below allows for a substantial performance boost with negligible thermal impact.

```
arm_freq=1000
sdram_freq=500
core_freq=500
over_voltage=2
arm_freq_min=400
sdram_freq_min=250
core_freq_min=250
```

For a list of stock specifications and configurations for the Raspberry Pi, please see Appendix **??**. The operating system chosen was the ARM build of Ubuntu 14.04 Long Term Support (LTS), codename Trusty Tahr, as well as Robot Operating System (ROS) Jade Turtle for sensor coordination and communication. Ubuntu 14.04 was chosen over the default Raspbian operating system for increased security due to the use of SELinux policies. Since the Raspberry Pi will be acting as a LAMP server as well for mobile interaction and remote notifications, enforcing read only access over the configured port of choice is a necessity. For an in depth overview of the mobile application please see subsection 3.4. Additionally, the LTS version of Ubuntu was chosen for a maximum of five years support, as well as increased compatibility and reliability for repository packages specific to the trusty distribution.

3.3.2 Sensors and Micro-Controllers

Every sensor in the system is paired with a micro-controller to enable a modular design and a plug and play like support. By adding more sensors to the system, more information can be provided as feedback to enable better logging and regulation of the brewing environment. The only mandatory sensors for the system are a main temperature sensor for the wort and flow meters on the liquid inputs and outputs of the system. By introducing additional sensors, such as pH and density sensors, certain yield thresholds of interest in the sparge and fermentation procedure can be met rather than approximated. Additionally, this will allow for a tier based configuration for the automated vessel to reduce cost and complexity if desired. As the end goal for this project is to have a community influenced input, having a customizable sensor configuration with open hardware and software pairs is encouraged. For maximum affordability the Arduino compatible Teensy-LC microcontroller is used to poll the sensors and report data, however any Arduino compatible microcontroller may be used. Technical specifications for the Teensy-LC microcontroller can be seen in Appendix **??** The bridge between the ROS instance on the Raspberry Pi and the Arduino compatible microcontroller is achieved through the use of the ROSserial package. Example code for obtaining data from a TMP102 temperature sensor over Inter-Integrated Circuit (I2C) at address 0x91 [3] can be seen below [4].

```
#include <Wire.h>
#include <ros.h>
#include <std_msgs/Float32.h>

//Set up the ros node and publisher
std_msgs::Float32 temp_msg;
ros::Publisher pub_temp("temperature", &temp_msg);
ros::NodeHandle nh;

int sensorAddress = 0x91 >> 1; // From datasheet
long publisher_timer;

void setup()
{
  Wire.begin();      // join i2c bus (address optional for master)
  nh.initNode();
  nh.advertise(pub_temp);
}

void loop()
{
  if (millis() > publisher_timer) {
  // step 1: request reading from sensor
    Wire.requestFrom(sensorAddress,2);
    delay(10);
    if (2 <= Wire.available()) // if two bytes were received
    {
      byte msb;
      byte lsb;
      int temperature;

      msb = Wire.read(); // receive high byte (full degrees)
      lsb = Wire.read(); // receive low byte (fraction degrees)
      temperature = ((msb) << 4); // MSB
      temperature |= (lsb >> 4); // LSB

      temp_msg.data = temperature*0.0625;
      pub_temp.publish(&temp_msg);
    }
  publisher_timer = millis() + 1000; //publish once a second
  }
  nh.spinOnce();
}
```

Additionally, controllers can be linked to the system via Arduino compatible micro-controllers and receive inputs based on sensor data gathered via ROS publications. To improve the resolution or accuracy of data and controls, the polling interval of each of the micro-controllers can be calibrated appropriately. Since most sensors will remain idle for a majority of the brewing process, their sampling rates can be scaled using a simple algorithm as suggested in the code sample below.

```
#include <std_msgs/Float32.h>

int sample_rate = MIN_SAMPLE_RATE;
int sample_count = 0;

void scale_rate(std_msgs::Float32 prev, std_msgs::Float32 curr)
{
    if(prev == curr)
    {
        sample_count++;
    }
    else
   {
        sample_count--;
   }

   if(sample_count == SCALE_THRESHOLD)
   {
         sample_rate = increase_rate();
      sample_count = SCALE_THRESHOLD >> 1;
   }
   else if(sample_count == 0)
   {
      sample_rate = decrease_rate();
      sample_count = SCALE_THRESHOLD >> 1;
   }
}
```

The two fundamental controllers that require communications for the operation of the system are the water controller and the heating and cooling controllers. Figure 3 demonstrates a state diagram for the water controller and Figure 4 demonstrate the state diagrams for toggling the heating and cooling states of the temperature controller. The variables volume, temp, target and offset represent the current volume, current temperature, desired resultant, and amount of allowed deviation from target respectively.

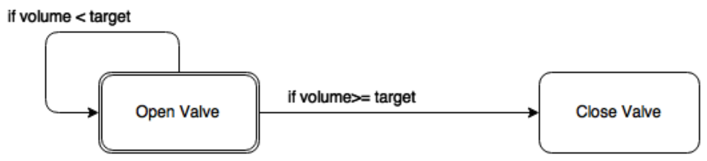

Figure 3: State diagram for the water controller component

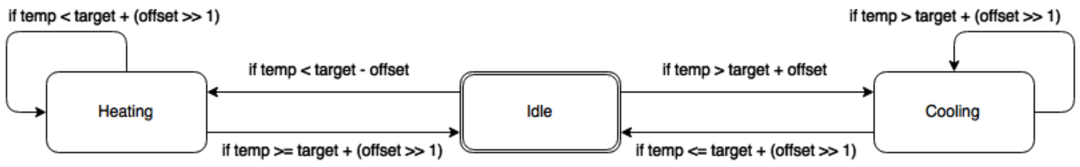
if temp < target + (offset >> 1)
if temp > target + (offset >> 1)
if temp < target - offset
if temp > target + offset
Heating
Idle
Cooling
if temp >= target + (offset >> 1)
if temp <= target + (offset >> 1)

Figure 4: State diagram for the temperature controller component

3.3.3 Web Server and Database

A Lamp Apache MySQL PHP (LAMP) server is the current solution for delivering information from the embedded computer to the mobile application. All communications and parameters are to be communicated using a JavaScript Object Notation (JSON) format. The database will consist of relational connections between recipes, instructions, ingredients, sensors, and logged data as demonstrated in Figure 5.

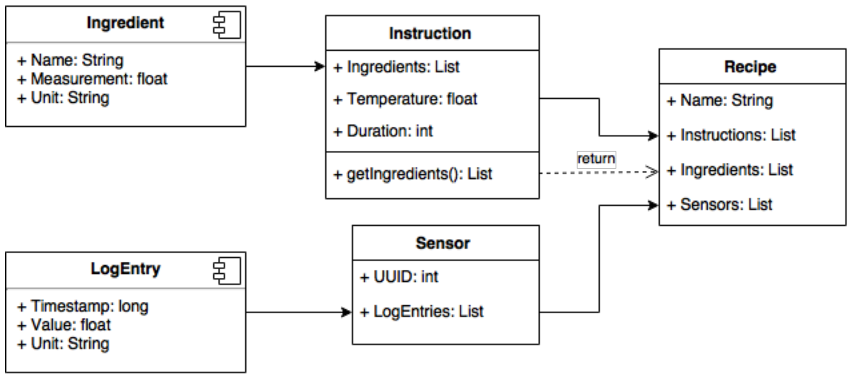

Figure 5: A UML diagram showing the relation between tables and their entries in the logging database

An important component of the web server is the server to client communication. The communication will be done through using the Google Cloud Messaging (GCM). This service allows a server to send messages to clients on various platforms [5]. There are 2 types of GCM server protocols that can be used to implement the server. Table 5 provides an overview of software design decisions and their respective weighted benefits.

Table 3: GCM Protocols

Protocol	Messaging	Upstream/Downstream
XMPP	Asynchronous	Upstream and downstream
HTTP	Synchronous	Downstream only

The Web Server will implement a Extensible Messaging and Presence Protocol (XMPP) protocol since it requires client to server (downstream) communication as well as server to client (upstream) communication. Since HTTP cannot offer that, it is not a viable option.

There are two types of messages that can be sent using GCM, notification and data messages. The differences between the notification and data messages is that GCM automatically displays the notification messages on the client's behalf. Where as the data messages has to be parsed by the client in order to be used. Another difference is while the notification type messages has fixed key value pairs, data type messages can have custom ones. Both message types are in JSON format, which is shown below.

```
{
    "to" : "XYZ",
    "data" : {
    "degrees" : "85",
    "component" : "Water"
    }
}
```

The "to" value holds the unique application id and the data block holds the key value pairs that are encapsulated in the message. For message passing, this application uses the data message type. This is because notification messages does not offer custom key-value pairs that is required for transferring information from server to client. Although additional processing is done in the client side, data messages can be flexible to handle the various state changes that the system may undergo. In conclusion, the GCM component of the web server uses the XMPP protocol that sends data type messages to the client.

3.4 Mobile Application

The mobile application subsystem provides an interface to the user and consists of three main components. The first is a service that provides real time data of the current brew to the user, the second component is an interface that provides data analytics recorded by the sensor, as well as providing an interface that allows the user to input controls. The following section describes how the appropriate platform for the application was chosen, as well as describing the three components. Platforms considered to develop the mobile application includes the Native Android platform, the Native Apple iPhone Operating System (iOS) platform and the HyperText Markup Language 5 (HTML5) web platform. Table [criterion table] below shows the criterion used to choose the appropriate platform. GCM support ranks the platforms based on the amount of support available for client development. This is the most important criteria since the Web Server mainly communicates with the clients using the GCM service, making it essential that the client is able to process the message. The implementation criteria ranks the platforms based on the amount of time required to develop, members previous knowledge about developing for the platform and amount of developer support that is available. This criteria is important because there is a limited amount of time for development, and the application must be done on time. Accessibility ranks the platforms based on how many smart-phones can use the mobile application. The more smart-phones that the platform can support, the higher its rank. The last criteria, ranks the platforms on the equipment and software tools required to develop the application on the platform. The platforms will be ranked based on the amount of costs to develop the application.

Table 4: Criterion table

Criteria:	GCM Support	Implementation	Accessibility	Resources
Weight:	40%	25%	10%	15%

Each development platform is compared based on previous knowledge. A score from 1 to 10 is given to each criteria for the 3 platforms. A weighted sum is calculated to determine the appropriate platform to develop the application. GCM Support: Both Android and iOS platforms have GCM client Application Program Interface (API)s developed by Google, and there are detailed steps in how to create a GCM client [5]. HTML5 based apps do not have GCM client APIs, but there are 3rd party support available for development, giving the platform a score of 5. Implementation: The members have the most experience with Android platform, and have little to no experience with the others. Also, both Google and Apple have released various Software Development Kit (SDK)s and documentation for developing Android and iOS applications respectively. The members do not have any experience in developing for HTML5 web applications, nor is there are any SDKs or documentation. However, all 3 platforms have various support online, from forums, blogs and repositories all having sample code and logic that can be very helpful in development. Considering this, a final score of 8 for Android, a score of 6 for iOS and a score of 4 for HTML5 is given. Accessibility: The most accessible application to develop would be the HTLM5 application. This is because all mobile devices can use HTML5 applications through a web browser. However, both Android and iOS have their own application distribution (Google PlayStore and Apple app store). 2014 market share of mobile devices shows that 80.7% of smartphones run the Android OS while 15.7% of smartphones run iOS [6]. Therefore a score of 10 for HTML5, a score of 8 for android and a score of 1.6 for iOS is given based on how many percentage of smartphone users can be reached. Resources: Developing the application for Android can be done using the Android Studio (Integrated Development Environment) IDE and can be done on various computers [7]. There are no specific IDEs to develop for the HTML5 platform, but web applications can be developed using any text editor. However, developing iOS applications involve getting additional resources since they can only be developed on Apple computers. Therefore a score of 8, 6 and 4 is given to Android, iOS and HTML5 respectively.

Table 5: Decision Matrix

	GCM Support	Implementation	Accessibility	Resources	Total
Native Android	10	8	8	8	8
Native iOS	10	6	1.5	4	6.25
HTML WebApp	5	4	10	6	4.9

The decision matrix shows that the best option to develop the mobile application is through the Android platform. The application implements the GCM client side of the system to communicate with the GCM Application Server running on the embedded system. The following subsections will focus on the component design for the Android application.

3.4.1 Real Time Data

This component of the application provides data from the brew system to the user's device. There are main two methods that the application provides data to the user. The first method is to display the current state of the brew in system through push notifications. Whenever a change is detected in the vessel such as temperature change, process change or any other state changes there might be, there would be a notification on the client's device to let the user know. The second method of the component is to provide interface that provides data statistics based on sensor data recorded by previous brews. This section will focus on the design for both methods. Figure 6 demonstrates the design flow for the push notification service.

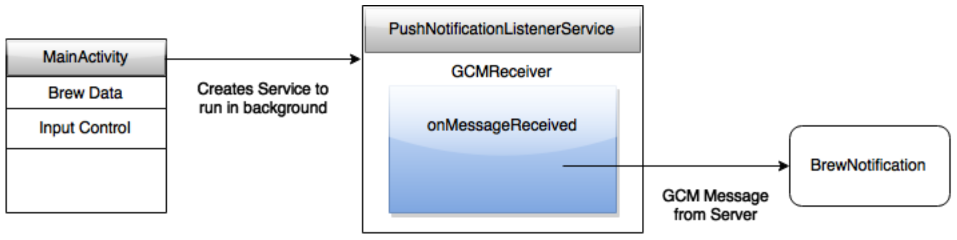

Figure 6: Flow Design for Notification Service

MainActivity: This activity starts on application launch (when user taps on the app icon). The activity will start the PushNotificationListenerService to run in the background. PushNotificationListenerService: The Listener uses the GCMReceiver class to listen for and to receive the data messages from the server. Whenever a JSON message arrives to the client side, the onMessageReceived function is called to parse the JSON message. onMessageReceived: This function parses the JSON message received by the Listener and creates a type of BrewNotification to display to the user. BrewNotification: This object will encapsulate notification created for the end user. Types of BrewNotification are based on priority: max, medium and low. Max priority notification is given to messages that relay critical information about the brew system (overheat, pressure build up, etc). Medium priority notifications are created for important messages (brew changing state, brew completed). Low priority notifications are given to messages that contain messages which the user does not necessarily have to know (boil temperature reached). To summarize, once the server sends a message to the client application, the Listener service will process the message through the onMessageReceived function. Once the function parses which type of message it is, it displays the appropriate BrewNotification to the user. The next part of the component displays previous data statistics recorded by the sensors on previous brews. The data displayed includes temperature readings, volume, sugar concentration and other sensor measurement. This gives the user the ability to look at past brews and possibly make adjustments. Figure 7 illustrates the application flow for the data statistics.

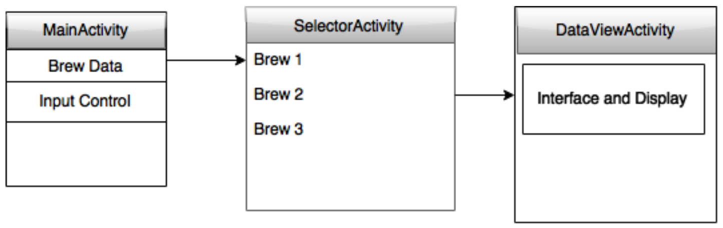

Figure 7: Flow Design for Notification Service

MainActivity: The main activity (same from the previous component) allows the user to launch the activity to select previous brews. SelectorActivity: This activity contains a selectable list of all previously done brews. By selecting one item launches the DataViewActivity based on the brew. DataViewActivity: Displays various data (temperature, pressure, yield percentage) of the brew selected by the user. Providing this information allows the user to view and experiment with previous brews to enhance the quality of future brews. In conclusion, the real time data component of the application is designed to keep the user notified about the brew system, whether that would be about the current state of the brew process or about previous brew data. This enhances user experience such that the user does not have to constantly check the brew process to receive information since the system informs the user.

3.4.2 Input Controls and Interaction

The second component of the application is to allow the user to control the brew system with the mobile application. The goals of this component is to allow the user to start the brew process through the application and to provide control over various conditions within the brew process such as the maximum temperature, the target sugar concentration and others that would affect the brew's quality. Figure 8 below shows the design flow for this component.

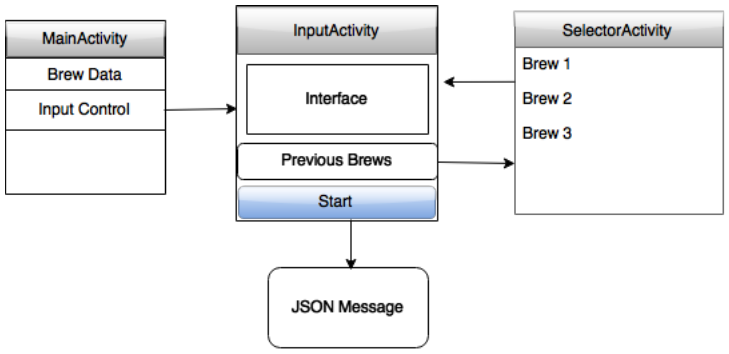

Figure 8: Flow Design for Notification Service

MainActivity: Option to select the input control activity, the main activity first checks if the mobile device is on the same local network as the vessel. If the device is not on the same network, the MainActivity will not allow the user to select the "Input Controls" option. This is because the mobile application is designed to allow users to perform write applications only when connected on the same network as the web server, otherwise the application allows read-only operations. Although this will limit the application's usability, it ensures a high level of security on the brew system, where it does not allow others to exploit the application over the internet to cause the main vessel to malfunction. InputActivity: The InputActivity allows the user to select a previous brew process to modify or create a brand new brew process. The user can change various states such as max temperature for boils, number of hours and other conditions that might affect the brew. The user is also able to start the brew from the interface, by tapping the START button snapped to the bottom of the interface. SelectorActivity: If the user selects a previous brew, the activity retrieves the relevant data from the server, and populate the interface of the InputActivity. If the user taps the START button in the InputActivity interface, the application creates a JSON file with all the data mapped to a key value. The client will send the JSON file to the server and wait for a response. An error response from the server will have another key explaining the details of the error. The errors are based on lack of ingredients in the vessel or a malfunction detected by the system. A success response from the server means that the system started the brew.

4 Discussion and Project Timeline

4.1 Evaluation of Final Design

4.2 Use of Advanced Knowledge

4.3 Creativity, Novelty, Elegance

4.4 Student Hours

4.5 Potential Safety Hazards

4.6 Project Timeline

Acronyms

ARM	Acorn RISC Machine.
I2C	Inter-Integrated Circuit.
JSON	JavaScript Object Notation.
LAMP	Lamp Apache MySQL PHP.
LTS	Long Term Support.
RAM	Random Access Memory.
RISC	Reduced Instruction Set Computing.
ROS	Robot Operating System.

Beer Terminology

Grist	The combination of milled grains to be used in a particular brew. Also sometimes applied to hops [8]..
Mash	(Verb) To release malt sugars by soaking the grains in water. (Noun) The resultant mixture [8]..
Pitch	To add yeast..
Sparge	To spray grist with hot water in order to remove soluble sugars (maltose). This takes place at the end of the mash [8]..
Trub	The layer of sediment that appears at the bottom of the fermentation vessel upon the completion of fermentation..
Wort	The solution of grain sugars strained from the mash tun [8]..

Technical Terminology

Raspberry Pi 2 Model B	The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015..

References

[1] A. Tepedelen, "History of homebrewing," 2013. `http://allaboutbeer.com/article/power-to-the-people/`. [Accessed 2015-05-19].

[2] C. F. I. Agency, "General principles of food hygiene, composition and labelling," 2014. `http://www.inspection.gc.ca/food/non-federally-registered/safe-food-production/general-principles/eng/1352919343654/1352920880237?chap=3`. [Accessed 2015-07-01].

[3] T. Instruments, "Tmp102," 2015. `http://www.ti.com/product/tmp102`. [Accessed 2015-07-01].

[4] ROS, "Measuring temperature," 2014. `http://wiki.ros.org/rosserial_arduino/Tutorials/Measuring%20Temperature`. [Accessed 2015-07-01].

[5] Google, "Use google cloud messaging (gcm)," 2015. `https://support.google.com/googleplay/android-developer/answer/2663268?hl=en`. [Accessed 2015-07-01].

[6] G. Inc, "Gartner says smartphone sales surpassed one billion units in 2014," 2015. `http://www.gartner.com/newsroom/id/2996817`. [Accessed 2015-07-01].

[7] Android, "Android studio," 2015. `http://developer.android.com/sdk/index.html`. [Accessed 2015-07-01].

[8] BeerAdvocate, "Beer & brewing terminology," 2015. `http://www.beeradvocate.com/beer/101/terms/`. [Accessed 2015-05-30].