Team 16: GardeNet

Senior Design Report

John Connell,
Anthony Jin,
Charles Kingston,
and Kevin Kredit

Calvin College ENGR 339/340
Senior Design Project
5/11/2016

Abstract

Irrigation is a labor intensive task for gardens of all sizes. For gardens and farms too large for simple timing systems and too small for industrial solutions, managers must choose between expensive Wi-Fi based solutions and large amounts of menial work. Team 16 of Calvin College's senior design class of 2016, called Team GardeNet, has worked to fill this gap. The team has designed and implemented a cellular network-based solution that can control up to 64 watering zones through a website interface. Basic system components include the frontend website, a backend server to manage data storage and communications, an onsite gateway, and distributed nodes controlling flow rate sensors and valves. Current market offerings do not feature cellular network connectivity, and given the business survey and cost analysis, the team believes that if GardeNet comes in a cellular and Wi-Fi enabled version, it can compete in the industry. The team has successfully developed a functioning prototype over the past two semesters, and while they are not pursuing the project further, they look forward to installing the prototype at Caledonia Community Garden in May 2016.

Table of Contents

1 Introduction 1
1.1 Project Description 1
1.2 Requirements & Goals 1
2 Project Management 3
2.1 Team Organization 3
2.1.1 Team Roles 3
2.1.2 Other Important Players 4
2.2 Schedule 5
2.3 Budget 6
2.4 Method of Approach 7
2.4.1 Design Methodology 7
2.4.2 Team Connectivity 8
3 Requirements 9
3.1 Original Concept 9
3.2 Design Norms 9
3.3 Meetings with Customers and Final Concept 10
4 Task Specifications and Schedule 12
4.1 Work Breakdown Structure 12
5 System Architecture 14
5.1 Broad System Overview 14
5.2 Website 14
5.3 Server 15
5.4 Gateway 15
5.5 Node 15
6 Design Process 16
6.1 Design Criteria 16
6.1.1 General Design Criteria 16
6.1.2 Server Design Criteria 17
6.1.3 Onsite Communication Design Criteria 17
6.1.4 Gateway Design Criteria 17
6.1.5 Node Design Criteria 18
6.2 Design Alternatives 18
6.2.1 Server Design Alternatives 18
6.2.2 Onsite Communication Design Alternatives 19
6.2.3 Gateway Design Alternatives 21
6.2.4 Node Design Alternatives 21
6.3 Design Decisions 21
6.3.1 Server Decision 21
6.3.2 Onsite Communication Decision 22
6.3.3 Gateway Decision 23
6.3.4 Node Decision 24
7 Development 25
7.1 Website 25

7.2 Server 26
7.3 Gateway 29
7.4 Node 31
8 Integration, Test, and Debug 36
8.1 Integration, Test, and Debug Strategy 36
8.1.1 Website 36
8.1.2 Server 37
8.1.3 Gateway 37
8.1.4 Node 38
8.1.5 Total System 40
8.2 Results 40
9 Business Plan 43
9.1 Marketing Study 43
9.1.1 Competition 43
9.1.2 Market Survey 44
10 Cost Estimate 45
10.1 Development Costs 45
10.1.1 Design Costs 45
10.1.2 Test Costs 45
10.1.3 Administrative Costs 46
10.1.4 Total Development Costs 46
10.2 Production Costs 46
10.2.1 Fixed Costs 47
10.2.2 Variable Costs 49
10.2.3 Total Production Costs 51
10.3 Profitability 51
11 Conclusion 52
12 References 53
13 Acknowledgements 56
14 Appendix I: Requirements List 57
15 Appendix II: Work Breakdown Structure 61

Table of Figures

Figure 1: The V Model for Systems Engineering [1] 7
Figure 2: System Overview Diagram 14
Figure 3: A Page on the GardeNet Website 26
Figure 4: Server Architecture 29
Figure 5: The Gateway 30
Figure 6: The Node 32
Figure 7: The Self-Reset Circuit 34
Figure 8: A Detailed GardeNet Block Diagram 41
Figure 9: The Complete GardeNet System 42
Figure 10: Retail Price Comparison of Automated Watering Systems 44

Table of Tables

Table 1: Project Milestones 5
Table 2: Projected CCG Installation Expenses 6
Table 3: Actual CCG Installation Expenses 6
Table 4: Basic System Requirements 11
Table 5: Condensed Work Breakdown Structure with Time and Percent Completed 13
Table 6: Server Decision Matrix 22
Table 7: Protocol Decision Matrix 22
Table 8: Onsite Communication Decision Matrix 23
Table 9: Gateway Decision Matrix 24
Table 10: Node Decision Matrix 24
Table 11: Updated Server Decision Matrix 27
Table 12: Requirements Revisited 40
Table 13: Total Annual Fixed Costs of the GardeNet Venture 49
Table 14: GardeNet Component Costs and Prices 49
Table 15: Typical GardeNet System Costs and Prices 50
Table 16: Annual Production Costs of the Typical GardeNet System 50
Table 17: Total Annual Variable Costs of the GardeNet Venture 50
Table 18: Total Annual Costs of the GardeNet Venture 51
Table 19: Revenues, Costs, and Profit of GardeNet in the First Three Years of Operation 51
Table 20: Work Breakdown Structure with Hours Percent Completed 61

1 Introduction

Gardens need constant attention to be able to stay alive, blossom, and produce a good yield. The single most important factor in maintaining a garden is making sure that each plant gets the right amount of water to grow at its optimal level. Unfortunately, plans sometimes fall through and workers run out of time or, in the case of community gardens, volunteers do not show up, and the plants do not get the water they need. The GardeNet system (henceforth referred to simply as GardeNet) is designed to solve this problem by automating the watering process.

1.1 Project Description

GardeNet incorporates ease of use with reliable computing to make an elegant solution. The system provides a mobile-friendly website that integrates with a local server to easily send watering schedules and shut off requests to the controller without ever having to visit the garden. The website allows the user to change the watering schedule, view current weather conditions and forecasts, view historical data such as water usage and system performance, and edit settings such as sensitivity to weather and contact information for text and email alerts. Because GardeNet automates the watering process, gardeners can rest easy knowing that their garden is getting all the water it needs. One detail to note is that in order to keep costs low and flexibility high, GardeNet is not designed for a specific irrigation system, but is designed to automate preinstalled irrigation systems.

1.2 Requirements & Goals

At the beginning of the project, the primary objective of GardeNet was to develop a basic watering control system that the user controls from a website or onsite input. The team also has a customer for the project, Caledonia Community Garden (CCG). CCG's situation caused the team to add the following objectives: for the onsite controller to be connected to the server through a cellular connection, to have valves controlling water in many different zones at once, to have extreme reliability, and to remain cost-effective when making the system. Because GardeNet is designed for community

gardens, its operating cost must be kept low. The requirements are discussed in more detail in Section 3.3 and are comprehensively listed in Appendix I: Requirements List.

2 Project Management

2.1 Team Organization

The team consists of four members, all Engineering majors with concentrations in Electrical and Computer Engineering: John Connell, Anthony Jin, Charles Kingston, and Kevin Kredit. The team possesses a combined two Computer Science minors, two Mathematics minors, and one Business minor. Each team member is expected to graduate in May 2016.

2.1.1 Team Roles

2.1.1.1 John Connell: Scheduler and Web Developer

John took the role of managing the team's scheduling and work division. John has past experience with website development and chose to take on the role of developing and maintaining the team's Calvin website as well as the GardeNet website.

2.1.1.2 Anthony Jin: Chief Editor and Gateway Developer

Anthony took the role of being the final editor of all of the team's published documents. Anthony also assumed the role of developing the gateway's code and communications, most importantly managing the cellular modem. (This document refers to portions of GardeNet in terms of the four major system components: the website, the server, the gateway, and the node. For more on these, see Section 5, System Architecture.)

2.1.1.3 Charles Kingston: Supply Chain Manager and Server Developer

Charles took the role of managing the relationships with outside and industry contacts as well as communicating with the industry sources. Charles has also been assigned with the development of the GardeNet server.

2.1.1.4 Kevin Kredit: Project Manager, Budget Officer, and Node Developer

Kevin assumed the role of project manager where he set milestones and tracked the team's progress. Kevin also handled the team's budget and bill of materials. Lastly, Kevin was responsible of the development of the gateway hardware and the node.

2.1.2 Other Important Players

2.1.2.1 Mark Michmerhuizen: Team Advisor

Professor Mark Michmerhuizen, an Electrical and Computer Engineering professor at Calvin College, served as the project's faculty advisor. Professor Michmerhuizen's expertise includes more than two decades of experience in the industry to the project and master's degrees in Engineering and Business Administration.

2.1.2.2 Victor Norman and Chris Wieringa: Technical Advisors

Computer Science Professor Victor Norman and Computer Lab Administrator Chris Wieringa, both at Calvin College, have been helpful in advising the team through the networking portions of the project, encompassing radio networking, web hosting, and internet communication.

2.1.2.3 Kurt Dykema: Mentor

Kurt Dykema is the Technology Director at Twisthink in Holland, Michigan. Twisthink has a broad portfolio of projects, including a tracking and management system for swimming coaches. Kurt provided project management tips and a connection to a potential customer and valuable resource (Kyle VanEerden of Eighth Day Farm; see below).

2.1.2.4 Eric Walstra: Industrial Consultant

Eric Walstra is an engineering manager at Gentex Corporation in Zeeland. He served as the team's consultant in meetings held on November 10, 2015 and March 10, 2016. Eric stressed the importance of identifying areas of highest risk and priority and balancing time, budget, and scope; for this project, he identified GardeNet's cellular connectivity and radio network as both high risk and high priority.

2.1.2.5 David Benjamin and Kyle VanEerden: Potential Customers

David Benjamin and Kyle VanEerden, managers of Caledonia Community Garden (CCG) and Eighth Day Farm (EDF) respectively, have helped the team steer the project in a direction that would provide the most value. David and CCG in particular represent the team's primary customer, whom the

team intends to serve by installing the prototype GardeNet system. Over the course of the project, it became evident that the team would not be able to serve EDF, but Kyle remained a valuable resource nonetheless.

2.2 Schedule

At the beginning of the project, the team formed a work breakdown structure (WBS) to keep track of all project tasks. While a detailed list of the tasks is listed in Appendix II: Work Breakdown Structure, Table 1 lists the major milestones of the project.

Table 1: Project Milestones

Milestones	Description	Due Date
Develop Requirements	Define all technical and non-technical requirements of the design	November 27, 2015
Design	Design system architecture and top-level user interface	February 21, 2016
Test	Implement, test, and troubleshoot prototype	March 13, 2016
Refinement	Fix bugs and introduce additional features	April 24, 2016
Develop Documentation	Develop technical and user documentation	May 1, 2016
Onsite Installation	Install final system for client	May 19, 2016

John maintained the WBS and scheduling. During the fall semester, the team used an online Gantt chart tool called Instagantt to visualize the team's WBS and an online project management tool called Asana to manage the schedule and assignments. The team met every Monday and Wednesday for two hours to review the progress of the project and determine a list of action items for the same week. The team also established a monthly one-hour meeting with its mentor, Kurt of Twisthink. Each member of the team spent five to ten hours each week to work on various tasks for the Senior Design course and for the overall design of the GardeNet system.

During the spring semester, the team established regular meeting times three days a week but spent a total of six days a week working on the project. The team also increased the frequency of the meetings with Kurt to every other week. In total, each member spent 15 to 25 hours a week working on

the project. Because of the more frequent meeting times and more time spent working on the project together, the team found white-boarded tasks and schedules to be more practical than the online tools used the fall semester.

2.3 Budget

Kevin managed Team 16's budget. Table 2 lists the expected installation expenses for CCG at the beginning of the project.

Table 2: Projected CCG Installation Expenses

Item	Number of Units	Cost per Unit
Node	3	$ 46.47
Weather Station	1	$ 103.51
Gateway	1	$ 246.44
Cloud Server	1	$ 103.51
Website	1	$ -
Cloud Server	Monthly	$ -
Website	Monthly	$ -
Cellular Service	Monthly	$ 20.00
Total Installation Costs		$ 489.36

Actual expenditures for the CCG installation are listed in Table 3. While the team did go over budget by $20.05, it is confident that it made the best design decisions and uses of its funds possible.

Table 3: Actual CCG Installation Expenses

Item	Number of Units	Cost per Unit
Node	1	$ 228.52
Weather Station	0	$ 103.51
Gateway	1	$ 256.53
Server	1	$ 35.00
Website	1	$ -
Server	Monthly	$ -
Website	Monthly	$ -
Cellular Service	Monthly	$ 10.00
Total Installation Costs		$ 520.05

Besides the $500 budget provided by the Engineering Department, the team submitted a proposal for funding from the Eric DeGroot Engineering Fund (EDEF) in order to be able to serve Eighth Day Farm as well. Unfortunately, that did not happen, but the team is still excited about its partnership with CCG.

2.4 Method of Approach

This section covers the design methodology and team connectivity that was used throughout the project.

2.4.1 Design Methodology

The team decided that the design methodology that best fit the design norms and was the most efficient was the V-Model, as seen in Figure 1. The V-Model is a well-known systems design methodology that is used at most major companies.

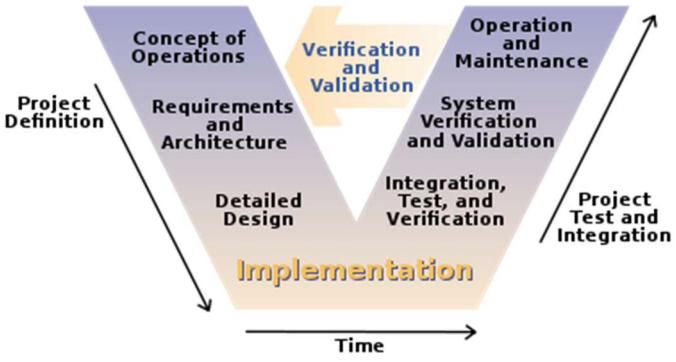

Figure 1: The V Model for Systems Engineering [1]

The V-Model starts with defining system level requirements that come from both potential customers and the team itself based on the end goal of the system. After the system level requirements were defined, the team developed component level requirements that spelled out exactly how each piece of the system should communicate and interface. Then, the team moved into detailed design requirements that spelled out specific functionality of every piece of the system. After the requirements were developed and reviewed, the team began implementing the system. Because the requirements were written to be specific and testable, as soon as initial implementation is complete, the team began testing first detailed

requirements, then component-level requirements, then system level requirements, along the way expanding and refining the design.

In practice, the team used more of an agile approach (get the basic framework running, then add features and test as you go) than a waterfall approach (plan the entire system in detail, then implement each component and test at the end), to put it in software development terms. Even so, thinking in terms of the V-model and developing system requirements proved very helpful for thinking about the system in the early stages. The team discovered truth in the old adage, “plans are useless, but planning is indispensable.”

2.4.2 Team Connectivity

The team decided that using the messaging app Slack was the easiest way to stay in contact. Slack offers group messaging features and integration with other tools, such as Microsoft’s OneDrive and Asana. The team decided to use a shared OneDrive folder to store all of the documents, research, and presentations. Lastly, the team made a GitHub repository that allows for simultaneous software development, file sharing, and version control. These three tools proved invaluable over the course of the project.

3 Requirements

The project requirements evolved from a combination of the team's original ideas, choices of design norms, and relationships with Twisthink, CCG, and EDF.

3.1 Original Concept

The team's original concept required control of up to four zones via wired connection to the controller and cellular connectivity to the internet; controls and system performance were to be accessed from a website, mobile app, or onsite LCD interface. This concept and set of basic requirements was from there informed by the team's design norms and significantly refocused by meetings with Twisthink, CCG, and EDF.

3.2 Design Norms

The team identified four design norms to guide the requirements making process—integrity, trust, humility and stewardship. First, integrity here refers to "delightful harmony"; the solution was to be complete and intuitive, living up to the requirements. That means that the team has to draft achievable requirements in the first place by choosing appropriate scope. Second, trust means that the system had to be reliable. In order for the customer to enjoy the full benefit of the system, they must be able to run it for weeks at a time with confidence that it will behave predictably. Third, humility means that the team was not to overestimate the system's capabilities, and the system was to be simple and usable by any person. Fourth, the design was to reflect good stewardship in terms of water, finances, and volunteers' time.

In general, the design norms complement each other and the project goals. However, in times that design norms conflicted with project goals, the design norms won out. For example, when the team would like to add a stretch feature but doing so would conflict with the system's integrity because the team would not be able to commit the time necessary to fully develop it, the team chose to focus instead on further refining existing features. When design norms conflicted with each other, the design norm of trust won out. Without trust that the system will deliver water as promised day in and day out, the system is

useless; therefore, the team at times sacrificed humility and financial stewardship by adding cost and complexity to the system in order to increase reliability.

3.3 Meetings with Customers and Final Concept

Meetings with the team's customers CCG and EDF gave the team a better perspective as to which elements of the original concept actually added value. From these meetings the team made two significant design shifts and added some stretch goals. First they pointed out that developing an LCD interface would add little value for the amount of effort it would take to develop. Intuitive and useful LCD interfaces are notoriously difficult to develop and would rarely be used if the internet connection is reliable. In lieu of an LCD, the new requirement was to have a simple two button interface paired with an LED to indicate the current state. Second, they showed that only four zones are not enough control for moderately sized gardens and that requiring wired connections severely limits system usefulness. Therefore, the team updated the requirements to allow an arbitrarily large number of zones which can be controlled wirelessly via radio communication with the controller.

Customer meetings also resulted in modification of and additions to the list of stretch goals. The team initially set stretch goals to make the system work completely off the grid via solar power and to pair the system with a "suggested setup" irrigation system. Twisthink pointed out that doing any work in the realm of irrigation systems (as opposed to control of preexisting irrigation systems) would require significantly more research outside the team's area of expertise. On the other hand, EDF's needs showed that the ability to control devices such as air conditioners and lights would add value. Therefore, the team no longer sought to do any work with irrigation installations themselves, but set the stretch goal of controlling alternative devices. A summary of the basic requirements is shown in Table 4, and is shown with column indicating success or failure in Section 8.2; for a full listing of system requirements, refer to Appendix I: Requirements List.

Table 4: Basic System Requirements

Category	**Initial Requirement**	**Design Norm Influence**	**Final Requirement**
Power Requirements	120 VAC input power	Must be reliable, insensitive to power surges, energy efficient	Design system to be AC and DC compatible; stretch goal to provide solar power
Control Type	Valves	Must be reliable	Valves; stretch goal to control AC units, lights
Control Zones	Up to four	Must be achievable, but also must be enough to be useful	Arbitrarily large number
Actuator-Controller Communication	Wired	Must be reliable, able to reach full extents of property	Wireless
Controller-Web Communication	3G cellular network connectivity	Must be reliable, affordable, able to use without Wi-Fi	3G cellular network connectivity
User Interface	Website, mobile app, onsite LCD interface	Must be simple, intuitive, complete	Website, mobile-friendly website, onsite buttons and LEDs

4 Task Specifications and Schedule

4.1 Work Breakdown Structure

Table 5 below shows the WBS with each milestone of the project and an estimate of how long it would take to finish. The percent completion comes from the time-weighted average of the expanded work breakdown structure. Progress made by the end of the fall and the spring semesters are shown in two separate columns. A detailed version of the work breakdown structure is listed in Appendix II: Work Breakdown Structure. Appendix II lists the time and percent completed as of the end of the spring semester. The fall semester's time and percent completed are listed below for reference but are not listed in Appendix II.

Table 5 also shows the completion dates of each event in the second column. Dates were only listed for portions of the WBS that were 100% completed. For portions uncompleted as of the time of this report, "Pending Completion" is written in lieu of an actual date. The requirements for Engineering 340 were significantly less time-consuming comparing to the fall semester as reflected in the WBS. The total hours spent on the project come out to 855 hours to date with an extra 8 hours to be spent after this report in order to finish documentation for CCG and onsite installation. At the time of this report, the project is 99.1% complete.

Table 5: Condensed Work Breakdown Structure with Time and Percent Completed

Number	**Title**	**Description**	**Fall Semester: Time and %Complete**	**Spring Semester: Time and %Complete**
1.0	Develop Requirements	All of the research needed for the system.	20 hours, 100% (11/4/2015)	20 hours, 100% (11/4/2015)
2.0	Design	Designing and Completing the System	240 hours, 4.6%	480 hours, 100% (4/15/2015)
3.0	Test	Testing Functionality	30 hours, 0%	30 hours, 100% (5/6/2015)
4.0	Refinement	Refine based on testing	50 hours, 0%	30 hours, 100% (5/7/2015)
5.0	Develop Documentation	Create documentation for ease of use	10 hours, 0%	10 hours, 50% (Pending Completion)
6.0	Onsite Installation	Install to current customers	3 hours, 0%	3 hours, 0% (Pending Completion)
7.0	339 Requirements	Class work for Engineering 339	218 hours, 100% (12/12/2015)	218 hours, 100% (12/12/2015)
8.0	340 Requirements	Class work for Engineering 340	72 hours, 0%	72 hours, 100% (5/12/2015)

One issue arose at the beginning of the spring semester when the team entered the design phase. In lieu of designing the GardeNet server, the team had originally planned to use the platform provided by Exosite, a cloud service provider, as their server. After spending about 20 hours exploring Exosite's utilities, the team decided that commercial-grade cloud servers such as Exosite would not fulfill the requirements of GardeNet for reasons to be discussed in detail in Section 7.2. There were two implications of this design change: first, the team had to discard all progress they had made on Exosite and start designing the GardeNet server; second, two team members switched roles in order to maximize efficiency. Anthony was initially designated to develop the server on Exosite and Charles to develop the gateway. Developing the GardeNet server from scratch would require sufficient knowledge in networking, which Anthony did not possess; in the meantime, Charles had taken several computer science courses and was enrolled in a networking course during the schedule change. Consequently, Anthony took over the work Charles had done, and Charles started developing the GardeNet server.

5 System Architecture

5.1 Broad System Overview

GardeNet is designed to be modular and simple. The user is able to control the whole system from their phone from any location using the website. The user's input is then submitted and processed in the server. The server communicates with the gateway over cellular connection, and the gateway stores the schedule and controls the nodes. At least once per day, the gateway collects data from the nodes and sends it to the server, where it is stored and displayed on the website. The basic architecture and lines of communication are shown in Figure 2.

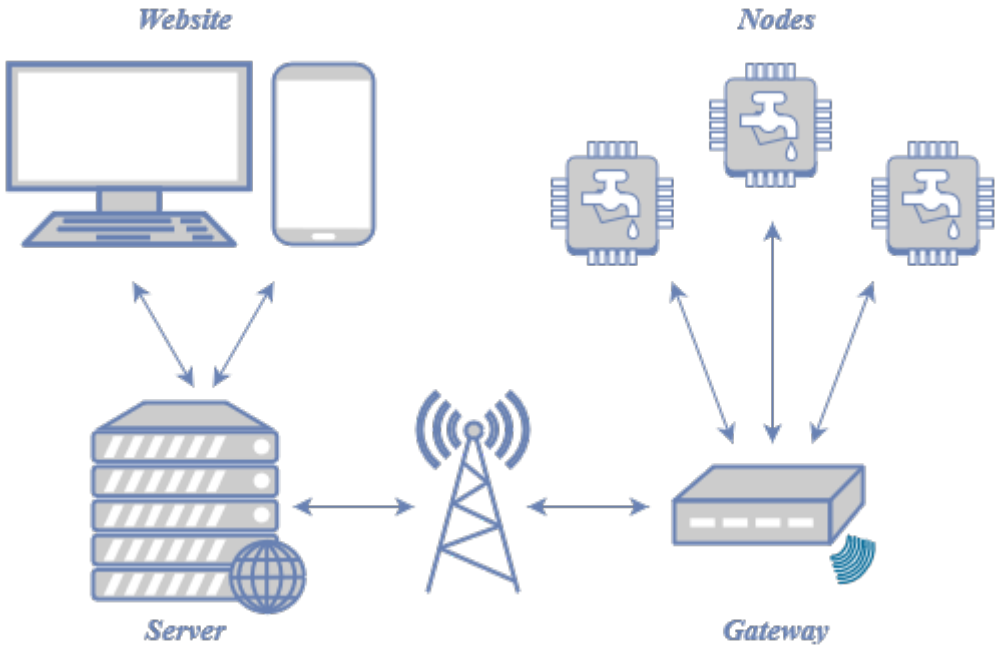

Figure 2: System Overview Diagram

5.2 Website

The website serves as the user interface for the entire system. Its functions include allowing the user to view and modify the schedule, add and remove zones, view historical data, set alert notification settings, and set account settings.

5.3 Server

The server acts as a hub for the system's data. Its functions include storing the data submitted from the website, modifying schedules according to weather reports, communicating with the gateway, storing historical data, and sending email alerts.

5.4 Gateway

The gateway serves as the onsite controller of the nodes and provides the cellular internet connection to the server. Duties of the gateway include keeping track of the time, commanding the nodes in accordance with the schedule, monitoring the status of the garden as a whole, and communicating with the server.

5.5 Node

Nodes act as the actual valve and flow rate meter controllers. Each garden can have many nodes. The node's duties include controlling up to four valves; monitoring up to one flow rate meter; monitoring its own status with respect to leaks, blockages, input voltage, and valve states; and communicating with the gateway.

6 Design Process

This section discusses the team's design criteria, alternatives, and decisions during the planning stage of the project. Section 7, Development, discusses the changes made during the implementation stage and the final design of the prototype.

6.1 Design Criteria

This section describes the criteria that the team used when evaluating the design alternatives. While many criteria apply to the system in general, each component also has its own special considerations. Therefore, this section begins with general criteria and then splits up into sections outlining specific criteria for each component.

6.1.1 General Design Criteria

There are some criteria that apply to all portions of the system. First is reliability, which corresponds directly to the design norm of trust—if the system is not reliable, it is worth nothing. Second is cost. Comparable systems already for sale offer competitive prices, and the target customers of urban and community gardens do not have a lot of money to spend. Third is system scalability. Because the target garden size ranges from small home installations to large gardens of several acres, the system must scale elegantly to large sizes. This corresponds to the design norm of integrity—the system shall be a joy to use for installations of all types and sizes. Another important component in a system such as this is security. This would be a general design criterion for the system before moving to production, though beyond password logins, it is outside the scope of this project.

One assumption that the team has held from the beginning is that the prototype would be designed using available development platforms such as an Arduino development board. While a marketable product would be designed from the chip-level up, the team has designed the system from the board-level up due to time and budget constraints.

6.1.2 Server Design Criteria

In addition to reliability, cost, and scalability, the criteria used to evaluate the alternative server options were ease of use and connectivity. Ease of use here means ease with which the team can start developing on the platform due to the short time period given for design. Connectivity was weighted next because it is important that the server be able to communicate with all the potential individual installations and with other websites such as weather.com and the frontend user interface.

6.1.3 Onsite Communication Design Criteria

While not itself a system component, the onsite communication means between the gateway and the nodes was a topic that required significant consideration.

In addition to reliability, cost, and scalability, the criteria used to evaluate the onsite communication options were ease of use and power consumption. Ease of use in this context refers both to the previous sense as ease of development and to making the system easy for the customer to install and manipulate. Because it is something that customers will interact with firsthand during setup, it must be simple to configure. This goes with the design norm of humility. Power consumption is important because onsite communication needs to use as little power as possible in order to increase battery life. In the case that the product is adapted to run on solar power, the power savings will translate to smaller, cheaper solar panels. Note that cost is a particularly important factor here, because the cost of the communication module is multiplied for each node; a dollar saved here is worth $10 saved elsewhere.

6.1.4 Gateway Design Criteria

The additional criteria that the team used to evaluate different gateway alternatives were connectivity, number of configurable pins, and power consumption. The connectivity based evaluation was the highest priority to the team because the controller serves primarily as a distributor of information between the server and the nodes. The number of configurable pins was another high priority when choosing this component both because they are needed them for the connectivity means and because extra

available pins are always a good thing. Lastly, the gateway should be designed to consume as little power as possible to save money if plugging it in or to save solar costs if connecting it to a solar panel.

6.1.5 Node Design Criteria

The additional design criteria used for evaluating the node component options were number of configurable pins and power consumption. Just like the gateway, the valve controllers must have enough pins to be able to communicate with the chosen onsite communication method. Just like the onsite communication component, power consumption is important for battery life or solar panel size, and cost is especially important because it is multiplied by the number of nodes.

6.2 Design Alternatives

The design alternatives offered in the following sections correspond to the sections analyzed in the design criteria section, with the addition in the cloud server section of a discussion on possible protocols to use to communicate between the gateway and the server.

6.2.1 Server Design Alternatives

6.2.1.1 Host Site Design Alternatives

There are three major cloud servers that provide online platforms for Internet of Thinkgs (IoT) projects: Exosite, Aeris, and Telit. While all three host sites offer comparable reliability, scalability, and ease of use, they differ in connectivity and cost.

The biggest advantage of Exosite is its connectivity. Exosite's platform supports a variety of development boards as well as the Constrained Application Protocol (CoAP) and Hypertext Transfer Protocol (HTTP) protocols. In addition, although other features are limited, a free portal account on Exosite has unlimited data storage and can create a dashboard open to view to the public.

While providing similar functionalities as Exosite and an easy-to-use user interface, Aeris does not offer free data. In order to exploit the cloud's computational power, a specialized SIM card is also required to establish a connection node on each device using 3G or LTE cellular network.

Similar to Aeris, Telit uses specialized modems in place of SIM cards. Telit does not offer a free data plan.

6.2.1.2 Protocol Design Alternatives

The communication between two entities is impossible without a system of rules to transmit information—a communication protocol. While various protocols are available for IoT design, three of them are particularly suitable for data collection and web transfer between a cloud server and a device: Constrained Application Protocol (CoAP), Hypertext Transfer Protocol (HTTP), and Message Queue Telemetry Transport (MQTT).

CoAP is a specialized web transfer protocol for use with constrained nodes and networks in IoT [2]. While specialized in obtaining data from devices such as sensors, CoAP includes key concepts of the Web such as URLs and Internet media type. More importantly, it has very low overhead and simplicity for constrained environment [3].

HTTP functions as a request-response protocol in the client-server computing model [4] and is intended for bandwidth-constrained applications. Although the protocol is very similar to CoAP from a developer's point of view, it aims at obtaining data from a Web API (Application Program Interface) [2].

MQTT is an extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks [5]. This protocol is especially suitable for mobile applications where bandwidth and battery power are at a premium.

6.2.2 Onsite Communication Design Alternatives

There were several ways to establish onsite communication within the system. One alternative was to physically wire up every system component. Even though this approach would resolve the connectivity issue and reduce the cost of data transfer, it puts a lot of constraints on the scalability of the design. While wiring up three valves to the controller within 100 feet is not a big concern, it becomes extremely cumbersome as the number of valves or the size of the garden increases. Therefore, a wireless and low-cost communication means was a better fit with Team 16's goal of scalability.

This led to the possibility of a personal area wireless network. Since community gardens and farms such as CCG and EDF are in open fields, where Wi-Fi does not extend, Wi-Fi was not an ideal candidate. This led to the possibilities of MiFi, Bluetooth, or a digital radio network. MiFi is a brand name describing a wireless router that acts as a mobile Wi-Fi hotspot. Although a MiFi device can be connected to a cellular network, it can only provide internet access for up to ten devices [6]. On the other hand, a Bluetooth network would not solve the scalability problem because of its limited range and number of nodes [7].

An ideal digital radio network will have scalability, low cost, and long range. Team 16 identified three RF transceivers as potential candidates: ZigBee, LoRaWAN, and nRF24L01+-based custom networks.

Based upon IEEE 802.15.4 standard, ZigBee is intended to be simpler and less expensive than other wireless personal area networks such as Bluetooth or Wi-Fi. Although its stand-alone range is limited, this protocol can support industrial-size mesh networks and is very reliable due to its low power consumption and cost [8]. The disadvantages of ZigBee are higher component costs.

LoRaWAN is a Low Power Wide Area Network (LPWAN) and aims at key requirements of IoT such as bi-directional communication, mobility, and localization services. As Richard Viel, the Chief Operating Officer of Bouygues Telecom points out, the LoRaWAN technology is ideal to target battery operated sensors and low power applications as a complement to machine-to-machine (M2M) cellular. LoRaWAN is a promising candidate of M2M communication. Its topology effectively mitigates the latency between the end-devices and the central network server by making the gateway transparent [9]. However, it also poses major challenges such as sophistication and a steep learning curve to developers.

Lastly, nRF24L01+ is a highly integrated, ultra-low power RF transceiver and has similar features as ZigBee and LoRaWAN. However, the major difference among these three wireless communication schemes is the level of complexity. While ZigBee modules requires large host boards and many pins, the nRF24L01+ needs fewer pins from the host board as the chip uses simple serial peripheral interface (SPI) communication, requiring just three pins for communication and eight pins total [10].

6.2.3 Gateway Design Alternatives

The team also needed to select a development board which will serve as the gateway and main controller of their watering system as well as development boards for each individual valve. Development boards suitable for IoT design are available from many manufacturers. To name a few, Team 16 looked at products of Arduino, Microchip, and MultiTech. In their design, the team needs a development with enough general purpose input and output (GPIO) pins, processing power, and external device drivers to control the valves and sensors as well as communicate with the cloud server.

6.2.4 Node Design Alternatives

Lastly the team needed to select a development board which would control the valves and flow rate meters in the field. Like the main board, products from Arduino, Microchip, and MultiTech give competitive prices for small boards that can be customized to control the valves but each board needs enough pins to successfully turn on and off the valves.

6.3 Design Decisions

This section describes components that the team chose, and how each component best suits the design criteria.

6.3.1 Server Decision

6.3.1.1 Host Site Decision

The team decided Exosite best fit the design criteria. Exosite's connectivity is ranked highest because it provides the team with a vast number of devices that will be able to connect to its database and portal. Exosite also provides the team with a very high degree of scalability allowing for an arbitrary amount of devices to be connected to it. While using Exosite's full capabilities comes at a high price, the team is confident that it can implement GardeNet using only the free features. It has also been used in many of existing applications that have reported its robustness and trustworthiness meeting the design

criteria and design norms the team put forth. Table 6 shows the decision matrix used to come to this decision.

Table 6: Server Decision Matrix

Factor	**Reliability**	**Cost**	**Scalability**	**Ease of Use**	**Connectivity**	**Weighted Score**
Weight (%)	40	25	10	15	10	100
Exosite	***8***	***7***	***9***	***7***	***9***	***7.8***
Aeris	8	3	10	3	9	6.3
Telit	8	5	8	8	5	6.95

6.3.1.2 Protocol Decision

The team decided that the CoAP protocol best fit the criteria specified. The protocol consumes small amounts of data reducing the cost to operate the system. Also, its ease of use with sensor data makes the protocol a prime candidate for sending sensor information in the GardeNet system. See Table 7 for the decision matrix.

Table 7: Protocol Decision Matrix

Factor	**Reliability**	**Bandwidth**	**IoT Focus**	**Ease of Use**	**Weighted Score**
Weight (%)	40	25	10	15	100
CoAP	***8***	***10***	***9***	***10***	***8.1***
HTTP	8	9	5	6	6.85
MQTT	10	9	10	5	8

6.3.2 Onsite Communication Decision

The team decided the nRF24L01+ radio best fit the design criteria. Wireless communication was chosen over wired communication because it provides better scalability and ease of use. The radio provides the user with as many zones as they need, without any long wires or extra I/O modules that would be needed with wired communication. The team decided specifically on the nRF24L01+ radio because it is the lowest cost option that still provides reliable communication at an acceptable range. ZigBee and LoRaWAN both offer longer range and higher data throughput, but at the cost of higher

power consumption, cost, and complexity. The team decided that a high throughput of data was unnecessary because GardeNet will not need to be transferring large amounts of data. Instead, nRF24L01+ radios will allow the sensors to essentially "chirp" their data when requested. See Table 8 for the decision matrix.

Table 8: Onsite Communication Decision Matrix

Factor	**Reliability**	**Cost**	**Scalability**	**Ease of Use**	**Power**	**Weighted Score**
Weight (%)	30	30	30	5	5	100
Wires	10	10	0	10	10	7
Wi/Mi-Fi	7	2	8	8	2	5.6
Bluetooth	5	4	4	6	2	4.3
ZigBee	8	4	10	8	10	7.5
LoRaWAN	9	2	10	8	10	7.2
nRF24L01+	***8***	***9***	***8***	***7***	***8***	***8.25***

6.3.3 Gateway Decision

The team decided that the Arduino Leonardo best fit the design criteria. The Leonardo was selected because it easily interfaces with the Nimbelink 3G modem that comes with already loaded Verizon and FCC certifications [11]. Scott Kerstein from Arrow Electronics pointed out that the integration of Exosite and the Nimbelink 3G modem should be quick and seamless to set up. (3G is a slower but lower cost version of today's common 4G network.) The Leonardo also met the design criterion for configurable pins, having extra pins on it that can configured to work as GPIO pins as well as communication and data lines. The Leonardo was also the most cost effective decision because Arduinos are open-source, meaning that the team will not have to purchase any licenses in order to flash the board with the GardeNet code. See Table 9 for the design matrix.

Table 9: Gateway Decision Matrix

Factor	**Reliability**	**Cost**	**Scalability**	**Connect-ivity**	**I/O Pins**	**Power**	**Weighted Score**
Weight (%)	40	25	5	20	5	5	100
Arduino	***8***	***10***	***8***	***8***	***10***	***10***	***8.7***
Microchip	7	8	8	8	5	10	7.55
MultiTech	10	4	10	10	5	5	8

6.3.4 Node Decision

The team decided that the Arduino Nano best fit the design criteria. The Nano provides the team with 20 configurable pins so that it can communicate with the radio, the valves, and the sensors without difficulty. The Nano is also draws a small amount of power and is relatively low cost. Because it is an Arduino, it has the same qualities as the main controller with the added benefit of ensured interoperability and code similarity with the main controller. See Table 10 for the decision matrix used to come to this decision.

Table 10: Node Decision Matrix

Factor	**Reliability**	**Cost**	**Scalability**	**I/O Pins**	**Power**	**Weighted Score**
Weight (%)	40	25	10	10	15	100
Arduino	***8***	***10***	***8***	***10***	***10***	***9***
Microchip	7	8	8	5	10	7.6
MultiTech	10	4	10	5	5	7.25

7 Development

This section describes the design changes made during development, the reasoning behind those changes, and technical design details for each system component.

7.1 Website

The finished website is a fully-functional and aesthetically pleasing component of the GardeNet system. Originally, the website was to be stored on Calvin College's web servers; however, this solution would not be sustainable, since GardeNet needs to be operational for Caledonia Community Garden for the years to come. Calvin's servers also had restricted usability for Calvin students due to security considerations. For this reason, the team chose to move the website to a Raspberry Pi that would be owned and operated by Caledonia Community Garden.

The website is composed of two main "views" with different accessible websites. These "views" are referred to as a guest view and an admin view. The guest view has limited access to web pages and cannot change schedules or delete historical data, which functionalities are exclusive to the admin view. Certain web pages are only accessible in the admin view. Administrators can access pages to change alert settings, for SMS and email, and change account settings such as the username and password used to access the admin view. Figure 3 shows an example of how the administrators could setup schedules on the GardeNet website. The website also allows the administrators to manage the historical data. In the admin view, historical data can be downloaded in the form of a text file; if the administrators would like to delete the data, they could do so and have new data sets starting the following day.

An essential component of the website is security. The Raspberry Pi hosts the website and the server and handles all interactions between the two system components. The team programmed it so that is not easily accessible to potential intruders. The web server is password protected by a number of safety protocols so that only the team can access and edit files. In the meantime, the website's IP address is masked by a domain name using a free DNS server, adding another security measure as the Raspberry Pi

has to be port forwarded while the user needs to type in the website's domain name, gardenet.ddns.net, without worrying about its actual IP address.

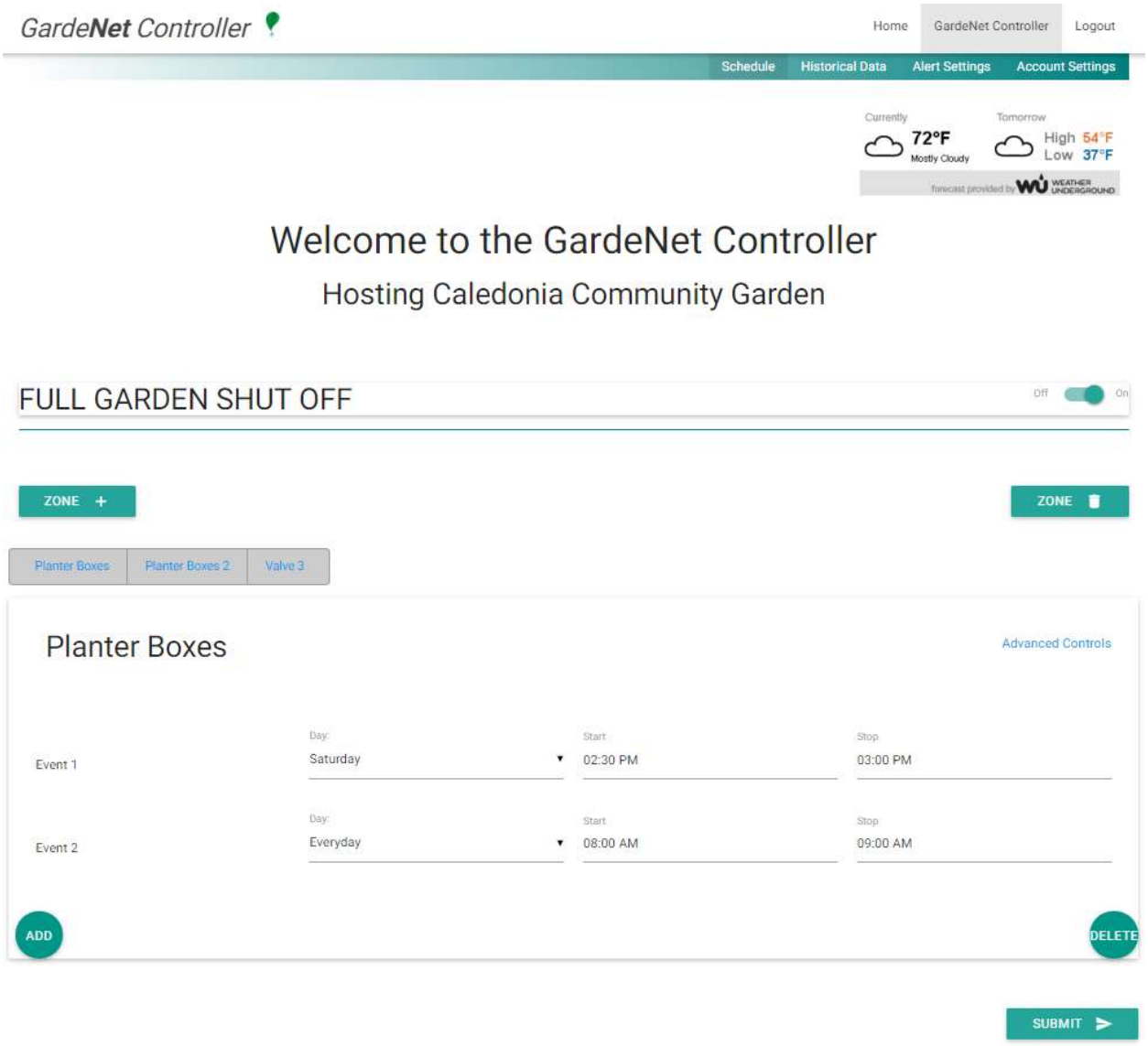

Figure 3: A Page on the GardeNet Website

7.2 Server

The choice to use Exosite as the GardeNet server and data host turned out to be the incorrect decision. After doing more research into Exosite, the team learned that they would have to learn a new scripting language that had several limitations without paying for Exosite's advanced features. Furthermore, the team also learned that due to the fact that a custom website was being built for the

system, Exosite would be wasteful and difficult to maintain. One of the main features of Exosite was the dashboard that it offered. Since GardeNet has its own website, the team would have to write scripts to essentially "virtually press buttons" on the Exosite dashboard. Not only was this approach wasteful, but it was also out of the team's realm of expertise.

One solution was to develop a custom server. This approach would require sufficient knowledge in networking concepts such as internet sockets, database development, and at least one scripting language. Although a custom server lacks the computational power a cloud server offers, it has no recurring cost and minimal restrictions on development.

Upon research into building a custom GardeNet server, the team discovered through Chris Wierenga and Professor Victor Norman that when the 3G modem connects to the Internet, it is assigned an IP address exactly like any other object is. The implication of this is that allows for the same functionality as any other service running on the Internet, meaning that a TCP socket could simply be opened between the server and 3G modem for communication. This meant that the team could easily communicate between the server and the gateway without using Exosite's capabilities. Lastly, the building of the custom server proved to be advantageous because it allowed for the team to build all of the backend data structures to support multiple additional features that the team was able to add. Upon considering all the facts, the team made a new decision matrix, seen in Table 11, and chose to go with the custom server.

Table 11: Updated Server Decision Matrix

Factor	**Reliability**	**Cost**	**Scalability**	**Ease of Use**	**Connectivity**	**Weighted Score**
Weight (%)	40	25	10	15	10	100
Exosite	8	7	9	7	9	7.8
Aeris	8	3	10	3	9	6.3
Telit	8	5	8	8	5	6.95
Custom Server	***8***	***10***	***10***	***10***	***10***	***8.2***

The server communicates using TCP sockets. There are three TCP sockets that the server maintains: an outside connection to the 3G modem and two loopback localhost sockets for communications with the website and testing services. The socket connected to the 3G modem is on port 5530 and is bidirectional. The socket is initiated by the 3G modem due to the fact that the server has been statically assigned a domain name whereas the 3G modem is dynamically assigned an IP address. The website communication resides on port 5538 and is used quite differently than the 3G modem connection. When a connection is made on port 5538, the socket is immediately shut and no data flows through the socket. Instead, the server then opens a file that the website writes before it makes the connection. This allows for both the website and application to read from the file. Lastly, the test service socket resides on port 5539. A script runs every 5 minutes to test if the server is on and functioning correctly by waiting for a message to be received back from the server. If the server is still correctly functioning, it sends a specified message back and closes the socket. If the server does not send the message back, the test script will timeout and call a bash script that kills the server and the resources it is still hogging, restarts the server, and writes to a log file marking a timestamp of when the server was restarted.

An important part of this server is the communication and structure. The structure is set up such that there are three types of scripts: interface, utility, and database. The server's architecture can be seen in Figure 4. The interface scripts allow the server to interact with other pieces of the system like the website and 3G modem. The interface scripts then call upon the utility scripts to correctly manipulate and respond to the interactions with these other systems. The utility scripts then call on the database functions to efficiently store data that is needed due to the fact that the scripts are set up in object oriented fashion. This means that the scripts are constantly calling instances of each other to pass information throughout the whole server correctly.

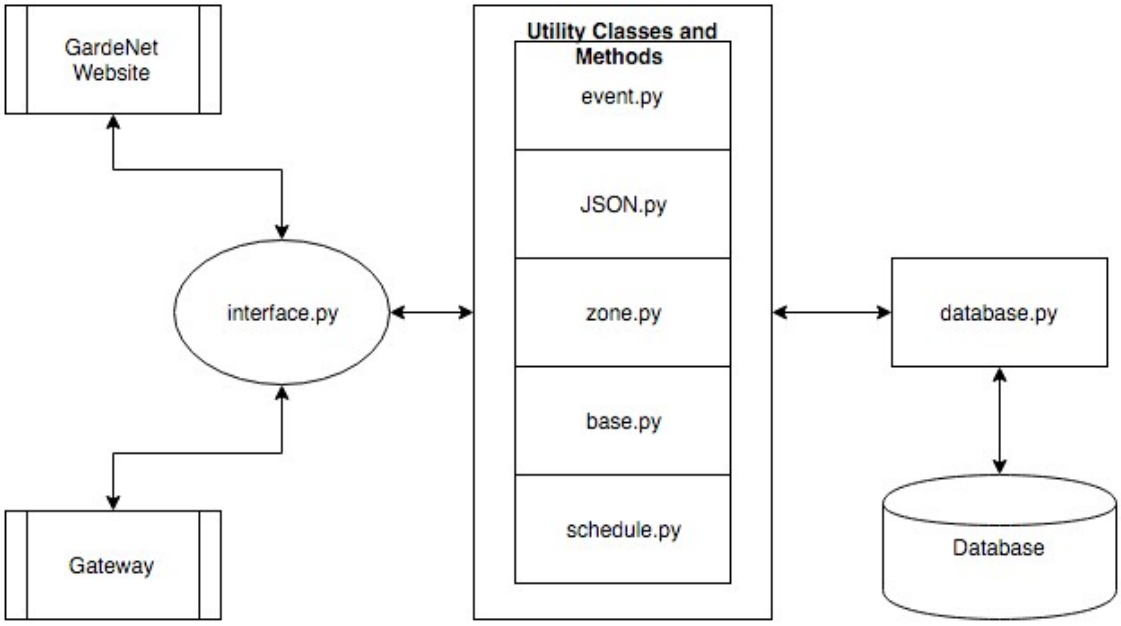

Figure 4: Server Architecture

7.3 Gateway

Although the Arduino Leonardo exhibited satisfactory performance during the early stage of the development, one issue arose quickly: the Leonardo has limited programming space—32 KB flash memory and 2.5 KB SRAM [12] — insufficient to support all functionalities the gateway requires. A program that establishes the basic mesh network and 3G communications already used 80% of the Arduino Leonardo's flash memory. Although the team attempted to optimize the gateway code by minimizing variable size (e.g. use type uint8_t in place of int) and reducing code length (e.g. convert multiple lines of code into a function and use function calls in the main loop), the result was only a few percent of the development board's programming space. After a meeting in early March, 2016, with Kurt Dykema, their industrial advisor, the team decided to use a development board with larger programming space to eliminate this limitation. Arduino MEGA 2560 has the largest programing space currently available in the Arduino lineup, featuring 256 KB flash memory and 8 KB SRAM [13]. Since Arduino MEGA 2560 is essentially a more powerful version of Arduino Leonardo with a much larger programming space, this upgrade is a minor change. The team have been developing their gateway

program on an Arduino MEGA 2560 ever since, and the final version of the gateway's code uses 15% of the Arduino MEGA 2560's flash memory.

Additionally, the team added an expansion shield to the gateway. The Nimbelink 3G modem requires both of the 3.3 V and 5 V pins on the Arduino development board. However, the RF24 radio also requires the 3.3 V pin for its operation. An expansion shield allows the Arduino development board to have multiple connections to each of its I/O pins and effectively solves this issue. The expansion shield also makes it possible to implement a self-reset circuit for the gateway. This circuit follows the same fashion as the one used on the node, which is discussed in Section 7.4. See Figure 5 for a picture of the gateway.

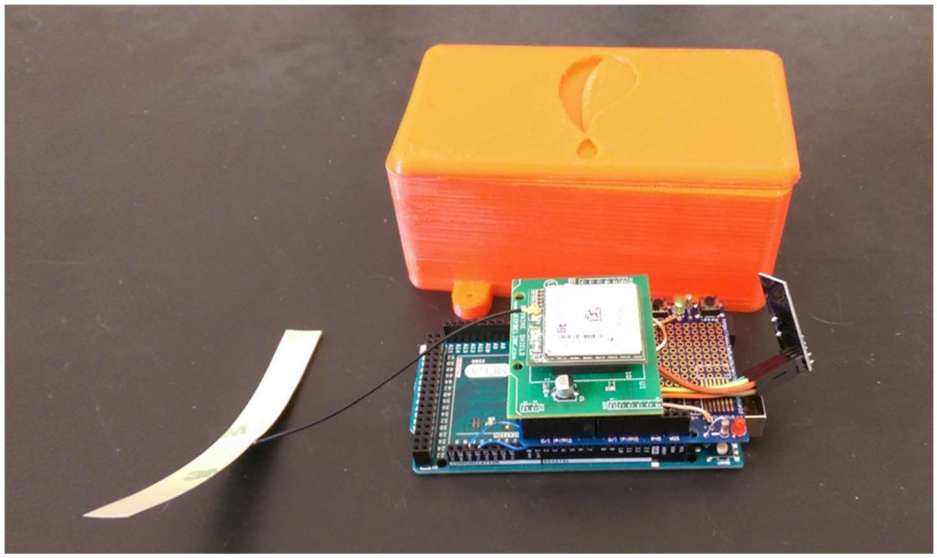

Figure 5: The Gateway

Another highlight of the gateway is its 3G connectivity. Following the Hayes Modem AT command set, the Nimbelink 3G modem makes the gateway-to-server bidirectional communication possible and easy to implement, regardless of the team's very limited knowledge in telecommunication. Following the AT command set's protocol, this communication is realized via a TCP socket, which passes data packets in either direction as defined in Nimbelink's documentation [14].

The team also encountered many challenges when developing the gateway software, most of which were related to robustness and system feedback. For example, the gateway would occasionally

receive a corrupted schedule event from the server with a few bytes of data missing. Since the performance of the gateway's TCP socket relies on the signal strength of the 3G cell network at a particular time and location, the team concluded that these corrupted data packets are inevitable. To cope with this issue, they added error handling routines in the gateway software, which compares the expected size and actual size of the data packets the gateway receives before parsing them; in the case of receiving a corrupted data packet, the gateway would request the server to resend that data packet.

The team also considered the need of system feedback. Although robustness was stressed throughout the project's design phase and various error-checking and self-healing mechanisms were added to each system component of GardeNet, the team believed that it is important to notify the user when the system is not behaving as expected. The gateway manages the system feedback since it communicates with the nodes and the server. If an issue occurs on a node (e.g. it is stuck on or off) or on the server (e.g. the server crashed), the gateway is able to send an alert message to the user. There are two ways for the user to receive alert messages: via email and via SMS messages. When a node issue is discovered, the gateway sends an SMS message and an encoded message to the server, which then notifies the user via email. The user has the option to choose what kind of alerts he or she would like to receive as SMS messages or as emails using the GardeNet website's alert setting page (see Section 7.1).

The other aspect of system feedback is concerned with historical data. The gateway sends a daily report to the server at midnight, including information from the nodes such as the percentage of time when the mesh network between the gateway and the nodes is good and how much water each node used. As mentioned in Section 7.1 and 7.2, this information is then stored in the database and displayed on the website.

7.4 Node

The Arduino Nano was an excellent choice for the platform for node development. The Arduino IDE made getting started fast and easy. The nRF24L01+ radios came with open-source mesh networking software that made building the wireless network relatively quick as well. Because they are open source,

the team has benefited from free to use, editable software, but has been challenged by the occasional lack of documentation and customer support. Once the pin configurations—the biggest obstacle in getting the network running—were settled, however, the radios have worked nicely.

One design change that has occurred is that the nodes are now designed to be able to control up to four valves and one flow meter each, whereas in the original plan they would control only one valve and flow meter each. Because the overhead costs for each node, such as the Arduino, radio, and power supply could be allocated across multiple nodes, the overall cost per node decreases. This is particularly practical for most gardens anyway, as water is often distributed and split into different zones in clusters anyway. The number four was chosen because the team already had a one-to-four splitter and CCG only needed three zones. The number of connections on any node remains variable, however, so a node can have between zero and four valves and a flow meter. For example, one node may have four valves and a flow rate meter, another may have just one valve, and another may have just a flow rate meter. An image of the physical setup of the node can be seen in Figure 6.

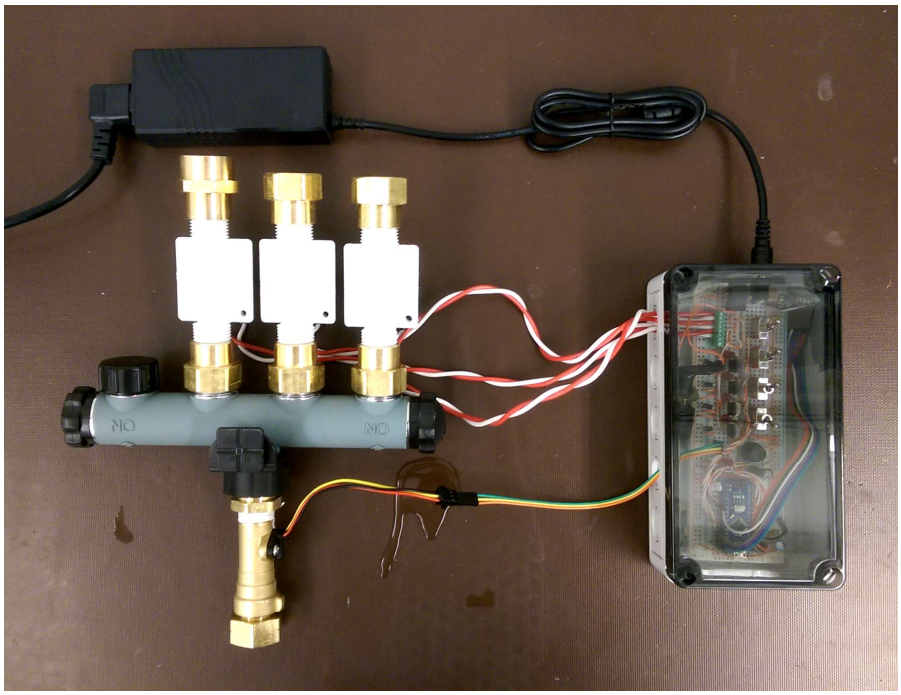

Figure 6: The Node

The code has been designed such that each node is flashed with the same program; the node ID, a unique identifying number, and the various connections are determined by the hardware configuration and are read by the Arduino upon startup. This way, the nodes can be programmed with the same exact code, but one will recognize that it is node 1 and has three valves and a flow rate meter while another will recognize that it is node 2 with two valves and no flow rate meter. Between controlling the valves, the flow rate meter, the radio, and node ID, the node uses every single available pin on the Arduino Nano.

An important part of the node is the self-reset circuit. Because of the team's commitment to reliability and simultaneous humility in admitting that they cannot hope to write fully-featured and absolutely crash-proof code in the course of this project, they determined to make sure that the system components fail—and recover—gracefully when those unforeseen errors occur. The self-reset circuit provides that solution. The circuit offers primary and secondary functions. The primary function is that if the program does not execute a certain section of code at least once every set period of time, the board will experience a reset; this catches any case where the software freezes from a bad instruction or an unintentionally infinite loop. The secondary function is that the program can restart itself; this is generally less useful, but catches cases such as the node getting in an odd state—if the gateway detects the node behaving oddly on the radio network, it can tell it to reset itself, hopefully resolving the issue. The circuit schematic is shown in Figure 7.

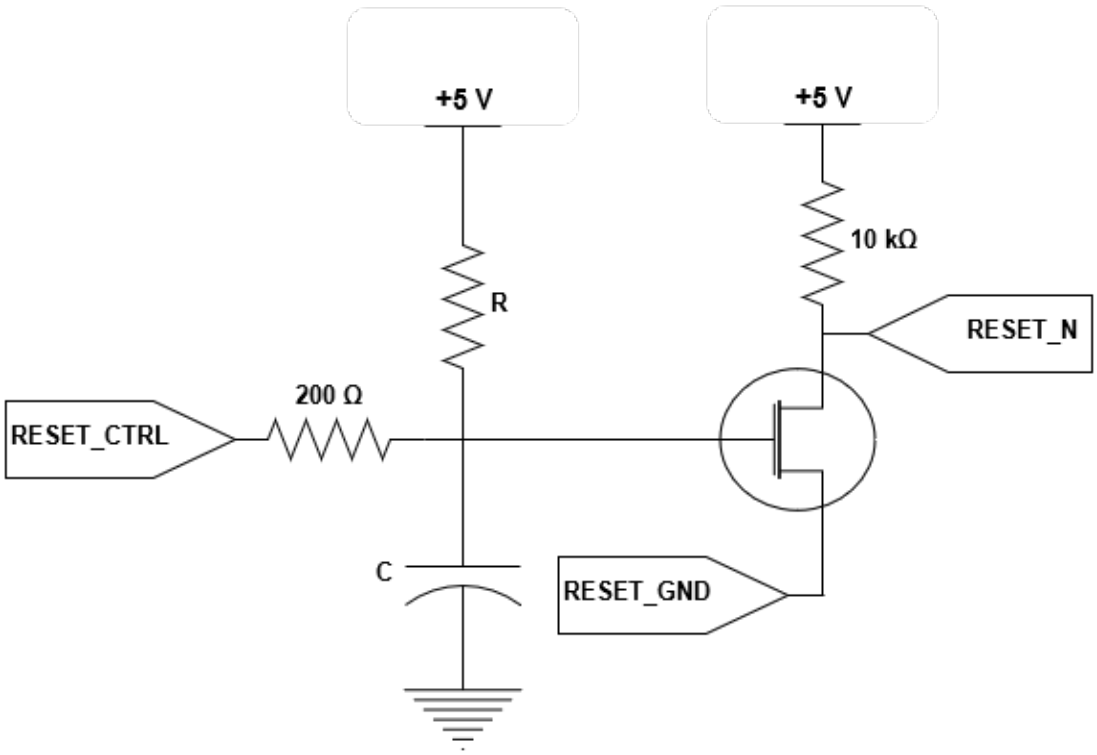

Figure 7: The Self-Reset Circuit

The basic structure is that of an RC timing circuit. During normal operation, the RESET_CTRL pin is configured as an input (high impedance) but at a certain frequency is configured by the microcontroller as an output and set "low" (ground) for a small period of time. When it is high impedance, the capacitor charges through resistor R. If it charges for long enough, the voltage becomes enough to turn on the transistor. The reset pin, RESET_N, is active low, and because the transistor is arranged as an inverter, turning on the transistor activates RESET_N. To prevent this from happening, RESET_CTRL must be periodically set to ground in order to discharge the capacitor. Thus, if the program freezes while the control pin is in a state of high impedance, it will stop setting RESER_CTRL to ground, and the Arduino will reset.

The input resistor was chosen to be 200 Ω because the Arduino documentation reports that GPIO pins can sink and supply at most 40 mA [15]; given a maximum voltage of 5 V, the 200 Ω resistor limits the current to 25 mA. The RESET_GND pin is used instead of a normal ground to prevent the circuit from constantly "pressing" reset once it has been activated. Without it (that is, with a normal ground

connection), the capacitor never discharges because the program restarts again before it can set RESET_CTRL to ground, so the Arduino resets infinitely; with it, the RESET_GND pin gets configured as high impedance by default during reset, so the transistor becomes ungrounded and RESET_N is deactivated, allowing the program to execute and discharge the capacitor through the RESET_CTRL pin. The 10 kΩ pullup resistor is internal to the Arduino board but is shown to complete the circuit.

The frequency at which the capacitor must be discharged is determined by the values of R and C. By experiment, the team determined that the Arduino resets if the voltage on the capacitor reaches 2.3 V. By basic electrical theory, $V_{cap} = V_{DD}(1 - e^{-\frac{t}{RC}})$ for charging capacitors. By substituting 5 V for V_{DD}, 2.3 V for V_{cap}, and solving for t, one finds that $t_{reset} = -RC \cdot \ln\left(1 - \frac{2.3\ V}{5\ V}\right) = RC \cdot 0.616$. For this application, the team determined that a t_{reset} of 5 minutes would be short enough to prevent long downtimes while being long enough for it to not happen during correct operation. Therefore, the node (and gateway) uses a 4.7 MΩ resistor and 100 µF capacitor, resulting in $t_{reset} = 4.7\ M\Omega \cdot 100\ \mu F \cdot 0.616 = 290\ s = 4.8\ min$.

8 Integration, Test, and Debug

8.1 Integration, Test, and Debug Strategy

Components were tested on an individual basis according to system requirements as they were developed and features were added. When components were ready to interface, they were tested and debugged together. The strategy was to develop and test the basic functionality first and flesh out the system with its finer features second.

8.1.1 Website

The website was tested during the design phase of GardeNet. It was essential, due to it being the first component that the customer sees, to make sure every part of it was smooth, efficient, and understandable. During design, many different styles of the website were tested to reflect ease of use. Buttons and switches were changed from the basic HTML style to a sleeker, friendlier style thanks to the Materialize style sheet. Much of the early stages of testing the website, before any server connectivity was established, was clicking around each portion of the website to determine if it was functioning correctly and it did not break. Most issues came with trying to make the website scalable, as it was an essential component of GardeNet to be scalable for the customer. Sometimes HTML, JavaScript, CSS, and PHP do not like to operate as documentation would suggest or would simply stop working randomly. This made development very challenging, as operating with four coding languages a day meant that any one of the languages could be causing the problem.

To combat these problems, online documentation was a great help in making the website fully functional. When the time came to integrate with the server, the website seemed to almost seamlessly send data and did not have many problems. Data was sent in the form of JSON strings that were created by JavaScript and then sent to the database through sockets. To make sure that all data were sent and received in the correct form, results were logged in the console to make sure there were no errors or changes when transferring over the sockets. Overall, the website worked well even though it had four programming languages and the team came in with limited knowledge on web development.

8.1.2 Server

The server's utilities and database functions were developed and tested using unit testing – also known as test driven development. What this means, is that a test was written before writing any of the functions needed. This way of testing allows the developer to write the tests on the way a class should function as well as test the corner cases for each of its functions without having to worry about any of the actual implementation details. Then, the developer can begin to write the class and its functions. As the functions are written, they can be tested in parallel. Once all of the tests are passed, the developer can say with certainty that the class has been tested and functions correctly.

Once all of the utility and database classes were written, the team began to develop the server that used all of the functions as described in Section 7.2. To make sure that the server was fully tested correctly, the team decided to add a verbose mode to it so that they could monitor the servers output and connections when they needed to. The team also added functionality to the server that allowed it to write to log files so that if there was an issue while it was not being monitored, the team could go back and look at the log files to determine any issues.

8.1.3 Gateway

The gateway utilized the test-driven development strategy due to its software nature. Functionalities of the gateway were tested in the following order: 3G cell connection, time synchronization, scheduling algorithm, and system feedback. In addition, receiving and parsing data packets was tested throughout the development phase, since any gateway functionality requiring internet connection relies on it.

First of all, the gateway must have reliable internet connection. Testing in this area was accomplished by connecting the gateway to the NIST time server via a socket on the 3G modem. The NIST time server then sends back a time string containing the UTC time during the connection. The team also tested several Hayes commands for socket dialing found at [14] while developing the gateway's connection with the server.

Upon the success of establishing internet connection, the team then tested the gateway's time synchronization. The gateway utilizes Arduino's time library to run an internal timer synchronized with the NIST time server. The gateway parses the UTC time string sent by the NIST time server and converts it to EST time (which is four hours behind the UTC time).

While developing the scheduling algorithm, Arduino's serial monitor was used exclusively as the debugging tool. Essentially, the scheduling algorithm decodes the data packets sent by the GardeNet server. If it recognizes an uncorrupted schedule event, the algorithm needs to decode the event properly so that the gateway could control the correct valves at the correct nodes at the correct time. The serial monitor turned out to be the most efficient debugging tool since it allowed the team to monitor the gateway program without directly controlling the nodes and valves.

Lastly, the team tested system feedback to the server and to their cell phones. Based on a protocol they developed, they encoded different alert messages and successfully sent them in the form of emails and SMS messages as discussed in Section 7.3.

As mentioned in Section 7.3, the data receiving and parsing routine was a major challenge of the gateway development. The team went through numerous cycles of test and refinement before they minimized the occurrence of corrupted data packets. Arduino's serial monitor again proved to be a useful tool.

8.1.4 Node

The node was tested simultaneously with development, in the following order: mesh networking, opening and closing valves, reading from the flow rate meter, resetting itself with the reset circuit, and reporting data back through to the server through the gateway.

Once basic networking capability was established, the range and mesh functionalities were tested. With the gateway in the team's senior design work room (next to the wood shop in the engineering building), the node connected in every portion of the North bay, including the mezzanine and through a single layer of concrete walls. The connection became unstable in the lobby, and dropped in the South

bay due to the multiple walls. The team estimates the effective distance to be at least 100 feet. To test the mesh functionality, the node was placed out of range of the gateway in the South bay, and a second node was placed in the lobby. The node in the South bay was powered first, and did not connect to the gateway, but when the node in the lobby was powered, both became visible to the gateway, confirming that the mesh functionality was working.

Opening and closing the valves was done using standard solenoid control circuits and was tested by simply blowing through the valves to make sure that they were open and closed at the correct times. The flow rate meter was tested by using the system to fill a five-gallon bucket. At first, testing revealed that due to the way the flow rate is recorded, the flow rate was highly volatile, reporting variable rates depending on when the data was sampled. To fix this, the flowrate was changed from being and instantaneous measurement to a moving average of 100 samples. After making that change, both the rate and accumulated flow measurements were accurate. When the five-gallon bucket filled, the meter recorded precisely five gallons, at a believable rate. (The flow rate measurement was not tested as stringently as accumulated flow, because it is less important, being used only to indicate leaks or stoppages.)

Testing the reset circuit involved intentionally putting the program into an infinite loop and checking that it reset correctly and at the correct time. Several revisions led to the circuit described in Section 7.4, which does reset the program correctly in precisely 4.8 minutes, as predicted.

Sending information to the server through the gateway required a data structure to hold the necessary information and a protocol of when to send it. This function was tested by subjecting the node to various faults and seeing if (a) the node correctly diagnosed them and (b) the gateway was correctly alerted. Testing rooted out false positives and refined the communication protocol between the gateway and the nodes. In order to save bandwidth while maintaining low latency, the nodes only update their status to the gateway when something of interest changes, such as a low voltage error cropping up or begin resolved. Because many of the issues self-resolve, alerts are only generated if the issue has been present for a given amount of time.

8.1.5 Total System

Once each component of the system was tested to meet their respective requirements, the team began to conduct full system tests. This was first done by running the server in verbose mode as well as monitoring the output from both the gateway and node manually. When a bug was found, the team would analyze the issue and find the root cause. With the root cause found, the team then refined the design and continued to test manually. After the manual tests were confirmed from the system, the team decided to run GardeNet overnight on an accelerated timer. This allowed the system to speed up time and function as if it were running for days out in the field. The team signed up for all of the alert messages so that if there was an issue, it was seen by the team.

8.2 Results

The development and testing resulted in a complete and relatively robust system; Table 12 shows the results in terms of the requirements. A block diagram with full details can be seen in Figure 8.

Table 12: Requirements Revisited

Category	**Requirement**	**Results**
Power Requirements	AC and DC compatible; stretch goal to provide solar power	AC and DC: Met. Solar power: Not provided, but compatible, as both the gateway and node can run on 12V DC.
Control Type	Valves; stretch goal to control AC units, lights	Valves: Met. Other: Not implemented, but the node could be adapted easily to control alternative objects, as the output is simply 12 V DC at up to 1 A.
Control Zones	Arbitrarily large number	Up to 64 in this implementation, but extremely easily expandable.
Actuator-Controller Communication	Wireless	Met, with mesh networking capability.
Controller-Web Communication	3G cellular network connectivity	Met.
User Interface	Website, mobile-friendly website, onsite buttons and LEDs	Met.

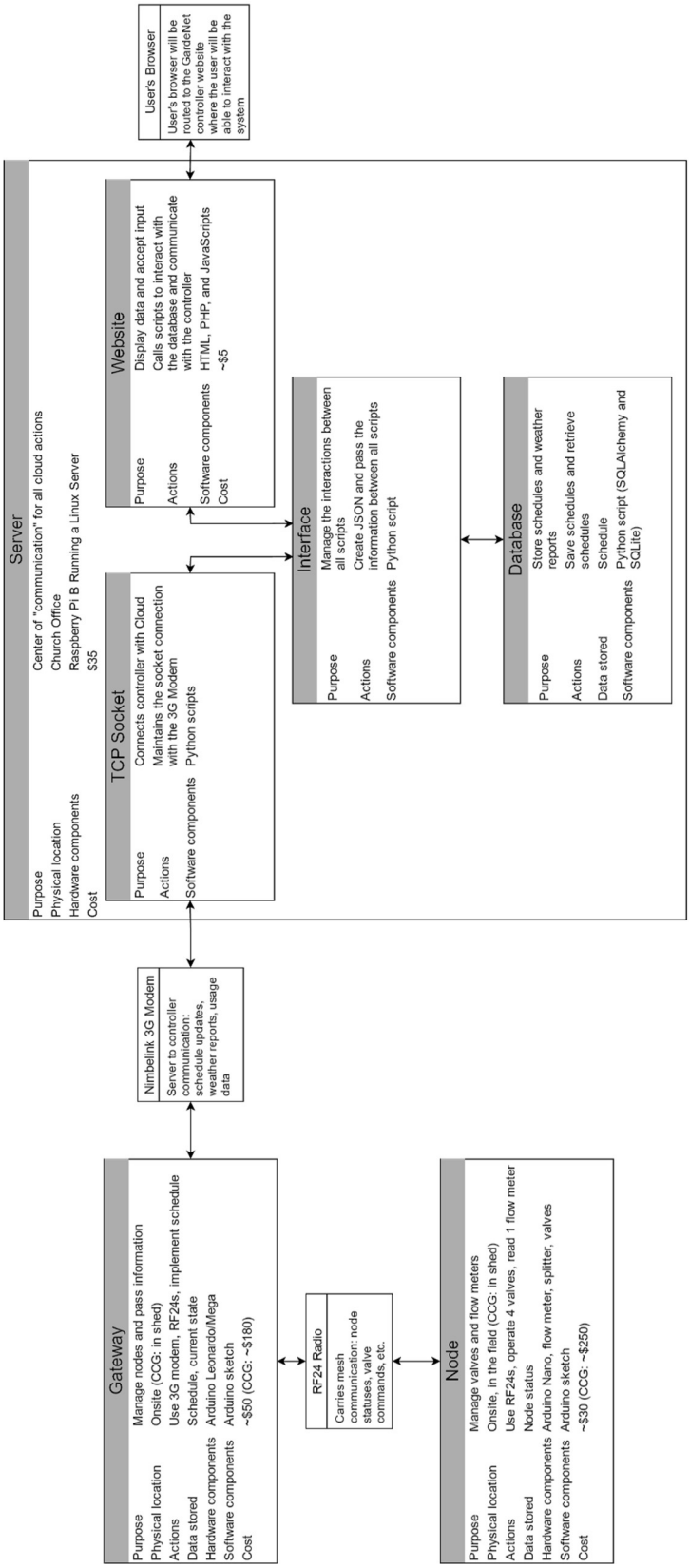

Figure 8: A Detailed GardeNet Block Diagram

An image of the system is shown in Figure 9. It accomplishes all of the goals that the team established, but is not perfect. The team met their basic requirements; in order to integrate robustness and reliability to their system, however, they did not have the time and resources to accomplish their stretch goals. Future work on the project includes adding more control options for the onsite gardeners so that they do not have to rely on the website, improving reliability issues throughout the system (even though it recovers gracefully in the case of crash, it should never crash in the first place), and improving the weather processing scheme, either using more complex heuristics or onsite sensors.

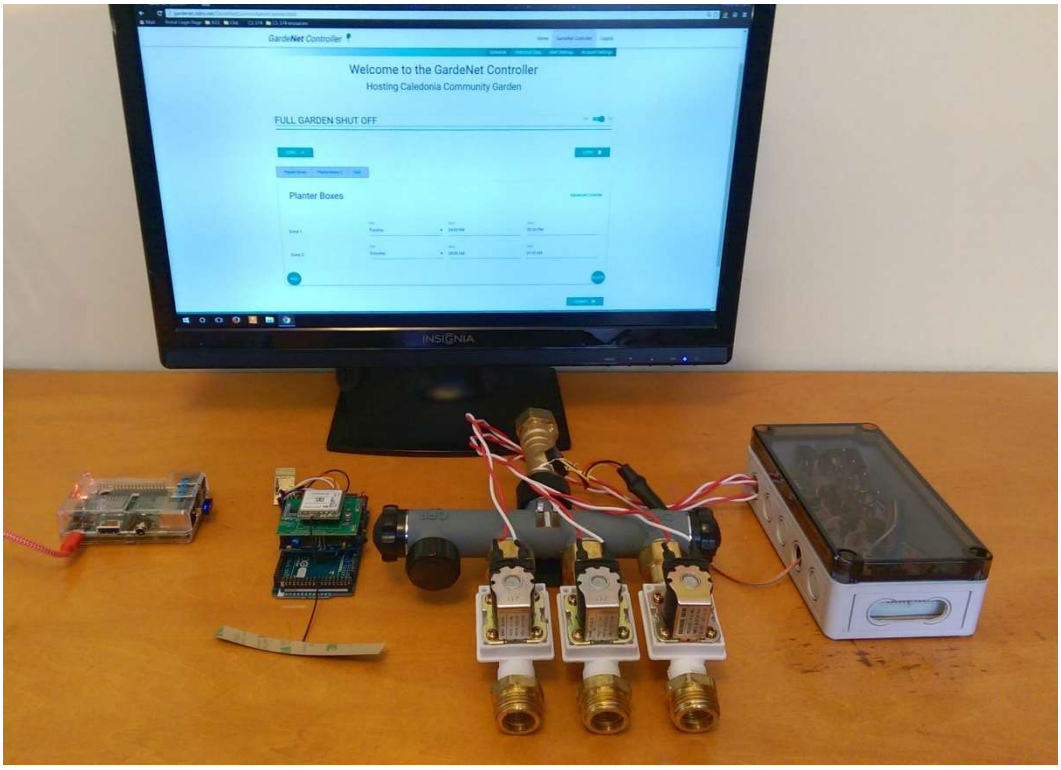

Figure 9: The Complete GardeNet System

9 Business Plan

9.1 Marketing Study

In the realm of IoT, a steady stream of products is released every year. An automated gardening system, while an interesting idea, is nothing new to the market. GardeNet gives the same basic functionality as a few other competitors but has advantage of running on 3G data. Using 3G data allows the system to be used in more locations than its competitors that only use Wi-Fi. Although the prototype can use a 3G connection only, offering GardeNet in Wi-Fi connected as well as 3G versions would give the system a cost advantage that would appeal to everyone.

9.1.1 Competition

As it stands, GardeNet has two major competitors in the market, the SkyDrop Smart Sprinkler Controller and HydraWise Irrigation System. Both of these systems cost upwards of $250 not including additional payments needed to control more zones [16] [17]. Both systems promise yearly water savings and environmental friendliness. However, the scalability of these systems come at a great cost to the consumer.

The HydraWise controller offers a 6 and 12 zone controller with a 12 zone expansion module for up to 36 zones total. The total retail cost of owning 36 individual zones costs about $675. If the customer wants to measure the flow out to the zones, it is about $175 or $235 per flow meter depending on pipe size. HydraWise also adds a monthly payment option to give SMS alerts and advanced features [18].

The SkyDrop Sprinkler Controller can control up to 16 zones. The base controller costs $280 for eight zones plus an extra $60 to control the maximum 16 zones. This costs a bit less than HydraWise but is not as scalable. SkyDrop also does not currently support rain and flow sensors as it requires “a pending controller software update” [17]. The pending flow meters and rain sensors add another larger cost comparable to the HydraWise system.

GardeNet, equipped with 3G service, will cost $250 plus $100 per four zones. Because GardeNet prices per four zones, the service is more customizable to the number of zones that the gardener actually

needs. If brought to market, GardeNet would supply a Wi-Fi version to directly compete with these competitors. At current estimates, the retail price of the Wi-Fi system will be $150 plus $100 per four zones. Figure 10 shows how the 3G and Wi-Fi versions of GardeNet compare in price with HydraWise and SkyDrop.

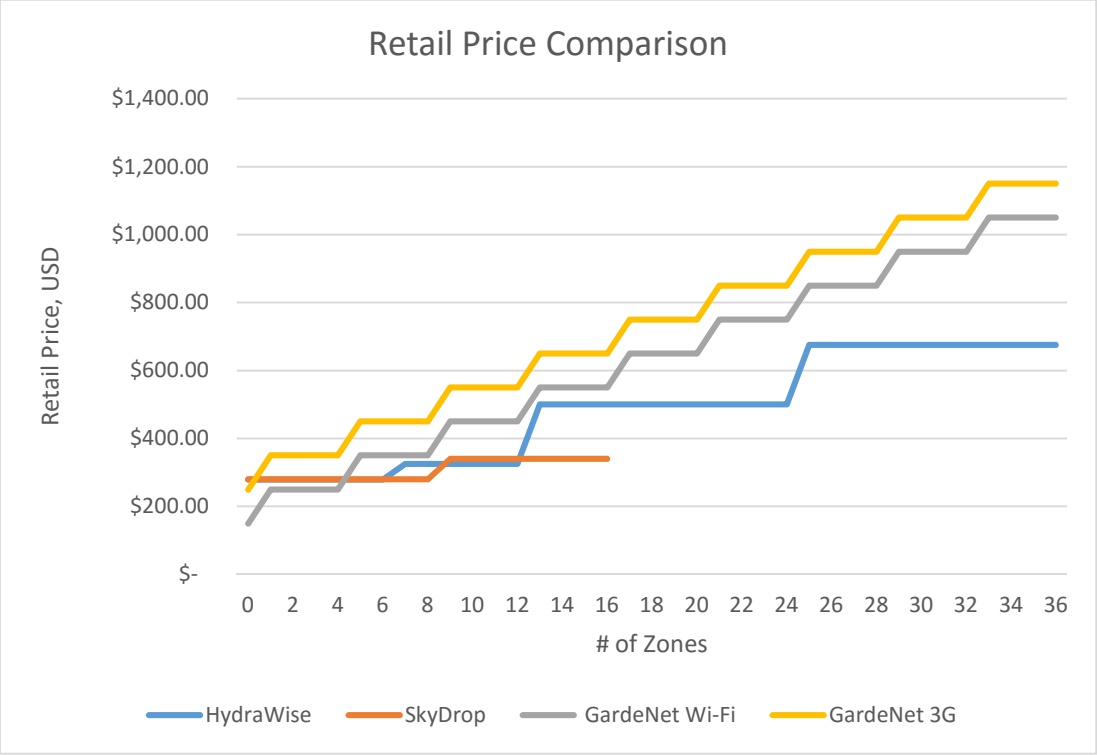

Figure 10: Retail Price Comparison of Automated Watering Systems

GardeNet is relatively price competitive. For four or fewer zones, the Wi-Fi version of GardeNet is actually cheaper. While GardeNet is more expensive above four zones, it actually offers more feature than the competition, and can support 64 valves in the prototype version and many more in the final version. The most difficult part would be developing GardeNet's brand over the established brands and increasingly all-encompassing home automation systems.

9.1.2 Market Survey

The market for GardeNet consists of home gardens up to large community gardens and urban farms. Given the increase in community and urban gardening, and the stable market of home gardens, the teams is comfortable estimating sales of 500, 1000, and 2000 units in the first three years.

10 Cost Estimate

10.1 Development Costs

If the result of this project were to be taken to market, developing a professional, sale-ready product would drive several new costs. Three primary factors drive these new costs: design, test, and administrative costs. All costs are to be considered rough order of magnitude (ROM) estimates.

10.1.1 Design Costs

First, a final product would be designed to use its own circuit boards, not the open source development boards used for this project. That is for two reasons: one, the development boards contain several components that the product does not use, and two, each component would have a lower per-unit cost. Further, the design would have to be refined to a professional level.

According to an *IEEE Spectrum* article from 2010, printing a custom designed printed circuit board (PCB) only costs $10 per board [19]. Converting the design to an integrated circuit (IC), however, would cost $3000 per board [20]. Neither of these include non-recurring engineering costs (NRE) or components to populate the board. For the volumes being produced, developing an IC solution is not worth the cost. Therefore, given two board designs (one for the gateway and the nodes), four design iterations per board, and a budget of $10 per board for components, there is $360 of parts cost.

NRE in terms of engineering design hours is more difficult to estimate. A low-range estimate for design hours is three months of four team members working 40 hour weeks, or around 4000 hours. If the team's time is valued at $80 per hour, that is $160,000.

10.1.2 Test Costs

Second, a final product would require thorough testing of a complete system in various conditions. That requires several copies of each component, test equipment, and a test garden. In order to test the system properly, the team would need at least two gateways and 64 nodes. At $20 per board, $200 additional per gateway, and $100 additional per node, the total cost is $2 \times (\$20 + \$200) + 64 \times (\$20 + \$100) = \$8120$. If the team use their backyards as test zones, they would still need pipes and

water to test with; thus if the team have three test facilities running a total of 100 gallons per day for a period of three weeks, the total cost is $100\frac{gal}{day} \times \frac{\$2}{750gal} \times 21days = \$655$. Thus the total testing costs are $8775.

10.1.3 Administrative Costs

Third, developing a final product would be fruitless without a business backing it up, and businesses require administration, floor space, and utilities. Assuming the team manage the administration on their own, utilize their basements and garages as workspaces, continue to use exclusively open source software, and use their own utilities, they would only be optionally paying for additional time worked and for additional used utilities; thus the costs would be approximately $15,000 for additional labor and at most $1,000 for additional utilities.

10.1.4 Total Development Costs

Summing the design, test, and administrative costs of bringing the product to market results in approximately $10,135 in design and test as well as $175,000 worth of labor. By including the administrative costs and rounding up expenses to account for unforeseen troubles, the total costs are approximately $190,000. Note that for the sake of the business plan, the team are assuming that this money would be raised through family, friends, and crowdfunding, and would not have to be repaid during the first three years of operation while the business becomes profitable.

10.2 Production Costs

Production costs consist of fixed costs such as facilities, insurance, and legal fees, and variable costs such as such as parts production, labor, and sales commissions. The following financial analysis is based on a three-year operating period and a manufacturing and sales volume of 500 units during the first year of operations and 100% growth rate in the next two years. For simplicity, the effect of inflation is neglected.

10.2.1 Fixed Costs

10.2.1.1 Equipment and Facilities

In order to provide technical support to the customers and be able to debug the products before delivery, GardeNet needs to have necessary electronic test equipment such as oscilloscopes, power supplies, multimeters, function generators, and various probes and accessories. According to the offers on an online electronics retailer, the costs of one set of equipment are: $300 for a bench top oscilloscope, $330 for a power supply, $450 for an industrial-standard multimeter, $750 for a function generator, and an additional $200 annual replacement costs for probes, test jacks, bread boards, jumper wires, and various kits [21]. The total cost is then $1,830 per set of equipment. Assume that the GardeNet venture needs two sets of these electronic equipment, the total costs on equipment are $3,660. Equipment maintenance costs are estimated to be $200 for the first year and $1,000 for the second and third year, assuming $200 annual replacement cost of the equipment and $800 for other needs in this area.

The cost of maintaining a facility is neglected because the business is modeled to utilize the team's garages and basements in the first few years while sales volumes remain small.

10.2.1.2 Insurance and Taxes

While it is difficult to accurately define the industry GardeNet falls into, a sample quote from TechInsrance for an independent contractor/web designer is used as a reference [22]. For this kind of small business, Commercial General Liability costs between $425 to $900 and is limited to bodily injury or property damage. The insurance company also offers other specialized liability insurances such as worker's compensation and professional liability, but they are more expensive. At minimum, the annual insurance costs are estimated to be $900.

As for the federal and state taxes, according to a document released by the Federal Government's Small Business Administration in 2009, the estimated average effective tax rate of small businesses is at 19.8% [23]. To use a conservative estimation and assume that the GardeNet venture will be profitable, the income tax is estimated to be 40%.

10.2.1.3 Legal Fees

Legal fees vary from company to company, but for a small business such as GardeNet, the example of the entrepreneur Guy Kawasaki from Truemors is appropriate [24]. For his start-up, he was charged $4,824.13 for the following areas:

- Trademarking
- Drafting a Term of Use
- Discussion of copyright, liability, infringement, intellectual properties, and insurance issues
- Organizational resolutions and bylaws
- Stock purchase agreements

Since most of these items will likely apply to GardeNet and the venture will need an attorney to handle legal issues, the cost of legal fees is estimated to be around $5,000. After the first year, it should decrease to about $1,000, incurred mainly by the cost of attorney.

10.2.1.4 Administrative Costs

Assume that for the first two years of its operation, GardeNet will need a weekly administration of 10 hours; as the sales volume increases, the administration time will increase to 15 hours per week during the third year to meet the new demand. Again set the hourly cost to be $80, the administrative costs are $41,600 for the first two years, and $62,400 for the third year.

10.2.1.5 Total Fixed Costs

In conclusion, GardeNet's total fixed costs are shown in Table 13. On top of this, the business should expect a 40% income tax if it is profitable.

Table 13: Total Annual Fixed Costs of the GardeNet Venture

YEAR OF OPERATIONS	TOTAL FIXED COSTS
1	$51,360
2	$44,500
3	$65,300

10.2.2 Variable Costs

GardeNet consists of two main physical components: gateway, and nodes. In addition, the company will offer both a 3G and a Wi-Fi version of the gateway. Table 14 shows the costs and price for each of these components. Note that the overhead was calculated by increasing raw material costs by 10%, and is meant to cover shipping and warranty.

Table 14: GardeNet Component Costs and Prices

Component	*Materials*	*with Overhaed*	*Labor*	*Utilities*	*Total Variable Cost*	*Price*
3G Gateway	$ 113.18	$ 124.50	$ 4.00	$ 0.10	$ 128.60	$ 250.00
Wi-Fi Gateway	$ 40.06	$ 44.06	$ 4.00	$ 0.10	$ 48.16	$ 150.00
Node (4 Zones)	$ 43.67	$ 48.04	$ 20.00	$ 0.10	$ 68.14	$ 100.00

10.2.2.1 PCB and Parts Outsourcing

Since the venture will be too small to support its own parts manufacturing workshop and assembly line, the production has to be outsourced. Referring to Table 14, component costs are expected to be lower in the production phase than those experienced during this project and in the development phase, assuming reasonable cost reduction when mass produced.

10.2.2.2 Labor

Assuming it takes 0.1 hours to assemble the gateways and 0.5 hours to assemble the nodes, and given an hourly cost of labor of $40, the corresponding labor costs are shown in Table 14.

10.2.2.3 Production Volume and Product Costs

Since customers' requirements on watering zones will very likely differ, the average case should be considered when estimating production costs, as shown in Table 15. In summary, a typical GardeNet

system costs $216.61 to produce and $390 to the consumer. The annual production costs, revenues, and gross profits are shown in Table 16.

Table 15: Typical GardeNet System Costs and Prices

Component	Number	Cost	Price
3G Gateway	0.4	$ 51.44	$ 100.00
Wi-Fi Gateway	0.6	$ 28.90	$ 90.00
Node (4 Zones)	2	$ 136.27	$ 200.00
Total		$ 216.61	$ 390.00

Table 16: Annual Production Costs of the Typical GardeNet System

Year	Production Volume	Variable Production Cost	Total Revenue
1	500	$ 108,304.97	$ 195,000.00
2	1,000	$ 216,609.94	$ 390,000.00
3	2,000	$ 433,219.88	$ 780,000.00

10.2.2.4 Customer Service and Warranty

Costs incurred in this area is included in the 10% overhead as demonstrated in Table 14.

10.2.2.5 Sales Commissions and Shipping Costs

Assuming that each sale takes two hours of processing, and the corresponding cost is $40 per hour, the annual sales and administrative costs are then $40,000 for the first year, $80,000 for the second year, and $160,000 for the third year.

10.2.2.6 Total Variable Costs

The total variable costs of the business are shown in Table 17.

Table 17: Total Annual Variable Costs of the GardeNet Venture

YEAR OF OPERATIONS	TOTAL VARIABLE COSTS
1	$148,304.97
2	$296,609.94
3	$593,219.88

10.2.3 Total Production Costs

Based on the analysis of the fixed and variable operating costs, the total annual production costs are shown in Table 18.

Table 18: Total Annual Costs of the GardeNet Venture

YEAR OF OPERATIONS	TOTAL COSTS
1	$199,664.97
2	$341,109.94
3	$658,519.88

10.3 Profitability

Given the sales numbers of 500, 1000, and 2000 units at the prices listed in Table 14, GardeNet would be profitable be profitable after a net loss in the first year as shown in Table 19. Net profit is calculated based on the assumptions that GardeNet will receive an income tax credit in the first year and the income tax is 40% for the second and third year. Since the production is outsourced and the cost of maintaining a facility is neglected, the depreciation of fixed costs is not considered.

Table 19: Revenues, Costs, and Profit of GardeNet in the First Three Years of Operation

YEAR OF OPERATIONS	TOTAL REVENUE	TOTAL COSTS	POST TAX PROFIT
1	$195,000.00	$199,664.97	$(4,664.97)
2	$390,000.00	$341,109.94	$29,334.04
3	$780,000.00	$658,519.88	$72,888.07

11 Conclusion

The previous sections outline the method of approach, system requirements, system architecture, design alternatives, and business plan for an automated garden watering system to help large gardens and small farms, implemented with a website, server, 3G enabled gateway, and wirelessly controlled nodes. Over the course of the year, guided by their education and the design norms of integrity, trust, humility, and stewardship, they have produced a viable prototype that brings new capabilities to the market. Given the competitive situation outlined in the business plan and the costs outlined in the cost estimate section, the team feels confident that given the right circumstances, funding, word of mouth marketing, and dedication and hard work, GardeNet could be a profitable venture. While the team is not pursuing that avenue, they look forward very much to installing the prototype at CCG in the coming weeks.

12 References

[1] "Systems Engineering Process," Wikipedia, [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e8/Systems_Engineering_Process_II.svg/420px-Systems_Engineering_Process_II.svg.png. [Accessed 14 11 2015].

[2] C. Bormann, "CoAP," [Online]. Available: http://coap.technology/. [Accessed 07 11 2015].

[3] "Exosite Documentation," Exosite, [Online]. Available: http://docs.exosite.com/coap/. [Accessed 09 11 2015].

[4] "Hypertext Transfer Protocol," Wekipedia, [Online]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Accessed 09 11 2015].

[5] "MQTT," MQTT.org, [Online]. Available: http://mqtt.org/faq. [Accessed 08 11 2015].

[6] "MiFi," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/MiFi. [Accessed 09 11 2015].

[7] "Bluetooth Marketing," bluAir, [Online]. Available: http://www.bluair.pl/bluetooth-range. [Accessed 12 11 2015].

[8] "ZigBee Wireless Standard," DIGI, [Online]. Available: http://www.digi.com/resources/standards-and-technologies/rfmodems/wireless-zigbee. [Accessed 10 11 2015].

[9] "LoRa Technology," LoRa Alliance, [Online]. Available: https://www.lora-alliance.org/What-Is-LoRa/Technology. [Accessed 12 11 2015].

[10 "nRF24L01," Nordic Semiconductor, [Online]. Available: http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01. [Accessed 12 11 2015].

[11 NimbeLink , [Online]. Available: https://nimbelink.com/skywire-evdo/. [Accessed 10 12 2015].

[12 Arduino.cc, "Arduino Leonardo," [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardLeonardo. [Accessed 24 April 2016].

[13 Arduino.cc, "Arduino MEGA 2560," [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardMega2560. [Accessed 24 April 2016].

[14 Nimbelink, "Skywire Embedded 3G Modem with CDMA/EVDO," [Online]. Available: http://nimbelink.com/skywire-evdo/. [Accessed 22 April 2016].

[15 Arduino.cc, "Arduino Nano," [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardNano. [Accessed 24 April 2016].

[16 S. LLC, "SkyDrop - Smart Sprinkler System," SkyDrop, 2015. [Online]. Available: http://www.skydrop.com/product/. [Accessed 13 November 2015].

[17 S. LLC, "SkyDrop Expansion," SkyDrop, 2015. [Online]. Available: http://www.skydrop.com/product/skydrop-expansion/. [Accessed 13 November 2015].

[18 HydraWise, "HydraWise," HydraWise, 2015. [Online]. Available: https://hydrawise.com/pricing/. [Accessed 13 November 2015].

[19 J. Turner, "Build a Custom-Printed Circuit Board," IEEE Spectrum, 31 March 2010. [Online]. Available: http://spectrum.ieee.org/geek-life/hands-on/build-a-customprinted-circuit-board. [Accessed 11 11 2015].

[20 S. Elder, "The REAL Cost for a Custom IC," Planet Analog, 14 5 2013. [Online]. Available: http://www.planetanalog.com/author.asp?section_id=526&doc_id=559840. [Accessed 11 11 2015].

[21 "MCM Electronics," [Online]. Available: http://www.mcmelectronics.com/. [Accessed 12 11 2015].

[22 "TechInsurance Sample Quotes," TechInsurance, [Online]. Available:

http://www.techinsurance.com/sample-quotes/independent-contractor/. [Accessed 13 11 2015].

[23 L. Quantria Strategies, "Small Business Administration," April 2009. [Online]. Available: https://www.sba.gov/sites/default/files/rs343tot.pdf. [Accessed 12 11 2015].

[24 G. Kawasaki, "guykawasaki.com," 5 June 2007. [Online]. Available: http://guykawasaki.com/482413_for_lega/. [Accessed 13 11 2015].

13 Acknowledgements

Team GardeNet would like to acknowledge the support of several people without whom this project could not be a success: professor and advisor Mark Michmerhuizen; technical advisors Victor Normand and Chris Wieringa; mentor Kurt Dykema of Twisthink; industrial advisor Eric Walstra of Gentex; component advisor Scott Kerstein of Arrow Electronics; co-chair David Benjamin of Caledonia Community Garden; town center manager Kyle VanEerden of Eighth Day Farm; Calvin College's engineering support staff including Robert DeKraker, Phil Jasperse, and Michelle Krul; and the countless others who have supported the project in one way or another.

14 Appendix I: Requirements List

1. Power
    1.1. Input Power Type
        1.1.1. Primary power source shall be 110-120 VAC ..... Met
        1.1.2. Should 110-120 VAC be unavailable, the user is responsible for providing a power source ..... Met
        1.1.3. It will be a stretch goal to make the system function solely on 12 VDC ..... Met
        1.1.4. It will be an additional stretch goal to design a solar power system capable of providing reliable power for the system ..... Not met
    1.2. Power Quality
        1.2.1. Surge Protection—power surges shall blow a fuse on the system before they cause permanent damage ..... Met
        1.2.2. Power Failure—the system shall respond to loss of power in a predictable and recoverable manner ..... Met
    1.3. Power Modes
        1.3.1. On—the system shall have a simple power-on procedure and be oriented towards minimizing power consumption ..... Met
        1.3.2. Sleep—the design shall have “sleep mode” for extended periods when the controller will not be required for control or data collection ..... Met
        1.3.3. Off—the design shall have a simple power-down procedure. ..... Met
2. Water Supply
    2.1. Control—the water distribution system shall be controllable via 12 VDC valves on pipe sizes up to 1.5” and up to 100 PSI ..... Partially met
    2.2. Scope—all else regarding water sourcing, pressurization, and distribution is outside the scope of this project ..... N/A
3. Controller
    3.1. Valve Control
        3.1.1. Valve Communications—the valves shall be controlled by the gateway via wireless communication ..... Met
        3.1.2. Valve States—the valves shall have only two states: ON and OFF ..... Surpassed
        3.1.3. Valve Behavior
            3.1.3.1. The valves shall turn on and off based on explicit signals from the microcontroller ..... Met
            3.1.3.2. If the valves do not receive a “continue” signal from the controller once per five minutes, the valves shall shut off to prevent unwanted behavior if the controller loses power ..... Met
    3.2. Local I/O
        3.2.1. Button Inputs
            3.2.1.1. The controller shall have the following button inputs: Stop, Start, Delay 30 Min., Power ..... Partially met
            3.2.1.2. The buttons shall always be enabled and override cloud controls ..... Met
        3.2.2. LED Outputs
            3.2.2.1. The controller shall minimally have the following LED outputs
                3.2.2.1.1. Green “running” light: solid if on, blinking if delayed, off if off ..... Partially met

3.2.2.1.2. Red “connection” light: solid if connected to cloud server, blinking if unconnected, off if system is off ..... Partially met

3.3. Sensor Control

3.3.1. Sensor Communications

3.3.1.1. The sensors shall send data to the controller via direct RF communication ..... Met

3.3.1.2. The controller shall request a reading from the sensor at regular periods depending on the sensor type ..... Met

3.3.1.3. The controller shall request a reading from the sensor when queried from the cloud ..... Not met

3.3.2. Sensor States—the sensors shall have two states: ON and OFF ..... Surpassed

3.4. Gateway Operation

3.4.1. Gateway Communications

3.4.1.1. The term “gateway” refers to the device via which the controller communicates with the cloud server ..... N/A

3.4.1.2. The gateway shall operate on cellular data service which balances speed and longevity with economy (e.g., 2G will soon be unsupported, 4G is expensive) Met

3.4.1.3. Transmission shall be encrypted ..... Not met

3.4.2. Data Transmitted

3.4.2.1. Current State

3.4.2.1.1. The gateway shall update the cloud server whenever it changes state ..... Met

3.4.2.1.2. State changes are any alteration or deviation from the expected schedule ..... N/A

3.4.2.2. Sensor Data—the gateway shall transmit sensor data when (a) every half hour during standard operation and (b) when queried from the server ..... Partially met

3.4.2.3. Control Input—the gateway shall update the cloud when input settings have changed due to onsite user input ..... Met

3.4.3. Data Received

3.4.3.1. Cloud Input—the gateway shall receive control input from the website or mobile app though the cloud server ..... Met

3.4.3.2. Query for information—the gateway shall receive queries for data from the cloud server when users go online to check the status of the garden ..... Partially met

4. Sensors

4.1. Number of Sensors

4.1.1. Each valve shall have one pressure sensor and one flow rate sensor ..... Partially met

4.1.2. Each installation shall have one weather sensor ..... Not met

4.2. Sensor Communications with the controller—sensor shall communicate with the controller via direct RF communication when queried by the controller ..... Met

4.3. Power—sensors shall run for at least two weeks on battery power. It shall be a stretch goal to generate power via a micro solar panel ..... Not met

5. Valves

5.1. Amount of Valves

5.1.1. There shall be one valve per zone ..... Surpassed

5.1.2. The system shall handle an arbitrarily large number of zones ..... Partially met

5.2. Valve Control Signals

5.2.1. All valve control signals shall be initiated by the controller ..... Met

5.2.2. If the controller does not send out a “continue” signal at least once per five minutes, then the valves shall shut off in order to prevent flooding in the case the controller loses power.....Met
5.2.3. Valves shall be designed so that they have minimum static power consumption.....Met
5.2.4. Signals shall be sent wirelessly up to at least 500m .....Met
5.3. Valve Capabilities
5.3.1. Flow rate—the valves shall handle flow rates of up to 50 GPM .....Met
5.3.2. Leakage—the valves shall have zero leakage at pressures below 100 PSI.....Met
5.3.3. Pressure—the valves shall handle pressure up to at least 100 PSI .....Met
5.3.4. Power—the valves shall run for at least two weeks on battery power. It shall be a stretch goal to generate power via a micro solar panel.....Not met
6. Cloud Server
6.1. Rules Engine
6.1.1. Onsite commands shall overrule online controls.....Met
6.1.2. Rule Types—the user shall be able to specify:
6.1.2.1. Run for X number of minutes .....Met
6.1.2.2. Run until X gallons of water delivered .....Not met
6.1.2.3. Combinations of (1) and (2).....Not met
6.1.3. Design Goals
6.1.3.1. A fully functioning rules engine will keep the user updated on all things going on in the garden.....Met
6.1.3.2. The rules engine shall not result in “positive feedback” results; runtimes and control signals shall be controlled and predictable .....Met
6.2. Alerts
6.2.1. Type
6.2.1.1. The system shall support email alerts .....Met
6.2.1.2. The system shall support SMS text message alerts.....Met
6.2.2. Frequency
6.2.2.1. Maximum
6.2.2.1.1. Emails < 20/day.....Partially met
6.2.2.1.2. SMS text message < 10/day.....Partially met
6.2.3. Type and frequency shall be modifiable by the user .....Met
7. Website and Mobile App
7.1. Views
7.1.1. Administrator View—shall contain:
7.1.1.1. All components available in the Passive View .....Met
7.1.1.2. Interface for control.....Met
7.1.2. Passive Viewer View—shall contain:
7.1.2.1. Overall system state .....Met
7.1.2.2. Historical usage data .....Met
7.2. Controls
7.2.1. The system shall have an interface to monitor and control individual valves .....Met
7.2.2. The system shall have a master switch to control system power.....Met
7.2.3. The system shall have an interface to setup watering schedules .....Met
7.3. Usability
7.3.1. Website

7.3.1.1. Detailed and aesthetically pleasing interface .....Met
7.3.1.2. Shall be usable on first attempt by a new user .....Met
7.3.2. Mobile Devices
7.3.2.1. Shall have simplified version of website functionality .....Surpassed
7.3.2.2. Shall consist minimally of a mobile-friendly version of the full website .....Met
7.3.2.3. It shall be a stretch goal to develop a standalone app .....Not met

15 Appendix II: Work Breakdown Structure

Table 20: Work Breakdown Structure with Hours Percent Completed

Section Number	Title	Description	Time Estimate
1.0	***Develop Requirements***		20 hours, 100%, (11/4/2015)
1.1	***Power***	Choose the kind of power used for the system and develop "modes" for efficiency	20 hours, 100%
1.2	***Water Supply***	Determine the kind of source, the output flow rate, and quality of water	10 hours, 100%
1.3	***Controller***	Find a controller to control the LCD, sensors, and valves and provide sufficient gateway operations.	20 hours, 100%
1.4	***LCD***	Choose a user friendly GUI on the LCD and provide communications to controller	20 hours, 100%
1.5	***Sensors***	Find sensors that can detect rainfall and give data to the controller	20 hours, 100%
1.6	***Valves***	Use valves that can take in control signals in order to operate with the system	20 hours, 100%
1.7	***Website--Server***	A cloud server is needed that can handle large amounts of data sent between the controller and the website	20 hours, 100%
1.8	***Website--User Interface***	The user interface of the website must be able to handle scheduling, commands, and alerts to the user	10 hours, 100%
2.0	***Design***		240 hours, 100%, (4/15/2015)
2.1	***Implement Requirements***		150 hours, 100%
2.1.1	***Hardware Installation***	Connect the controller to the valves and sensors	30 hours, 100%
2.1.2	***App Development***	Configure an application to control the controller from the cloud	30 hours, 100%
2.1.3	***Enable remote control***	Remote control connected to controller to send signals to valves and sensors	30 hours, 100%
2.2	***Low Level Debug***		180 hours, 100%
2.2.1	***Functionality***	Actively reconfigure system so that the system remains functional	50 hours, 100%
2.2.2	***Communication***	Fix communication errors that may occur internally and externally	60 hours, 100%
2.2.3	***Security and Reliability***	Create dedicated admin users to make all scheduling decisions	10 hours, 100%
2.3	***Purchase Materials***		20 hours, 100%

2.3.1	***Development Boards***	Purchase the researched development board	20 hours, 100%
2.3.2	***Hardware***	Purchase all other necessary hardware	20 hours, 100%
2.3.3	***Others***	Purchase miscellaneous things to operate the system	20 hours, 100%
2.4	***Website***		100 hours, 10%
2.4.1	***Admin/Viewer Use***	Create different views of the website for admin and viewer use. Viewers can only see data from garden. Admins can send data to the controller.	90 hours, 100%
2.4.1.1	***Garden stats***	Real-time stats sent to the viewer and admin	30 hours, 100%
2.4.1.2	***Remote Control***	Control the garden valves from the website.	60 hours, 100%
2.4.2	***About team***	Create a website about the team	10 hours, 100%
2.4.2.1	***339 Requirements***	Make sure the website contains all necessary requirements by the Engineering 339 class	10 hours, 100%
3.0	***Test***		30 hours, 100%, (5/6/2016)
3.1	***Test Bottom Level Requirements***	Test website interface usability, test website interaction with cloud, cloud interaction with controller, controller interaction with devices	10 hours, 100%
3.2	***Test Middle Level Requirements***	Test website control of individual devices (e.g. successful data collection, able to open/close valves)	10 hours, 100%
3.3	***Test Top Level Requirements***	Test in actual use-case--with running water, controlled remotely	10 hours, 100%
4.0	***Refinement***		50 hours, 100%, (5/7/2016)
4.1	***Troubleshoot Bugs***	Root out problems found in testing	10 hours, 100%
4.2	***Improve Scalability***	Improve code and system design for maximum generalization	50 hours, 100%
4.3	***Include WOW Factors***	Add special features to happily surprise clients	50 hours, 100%
5.0	***Develop Documentation***		10 hours, 50%, Pending Completion
5.1	***Technical Documentation***		10 hours, 0%
5.1.1	***Functional diagrams***	Draft diagrams showing the functional workings of the system	4 hours, 0%
5.1.2	***Schematics***	Show schematics, pseudocode, and full system specifications	2 hours, 0%

5.2	***User Documentation***		10 hours, 100%
5.2.1	***Website***		2 hours, 100%
5.2.1.1	***Use***	Have website map and describe intended usage	2 hours, 100%
5.2.1.2	***Maintenance***	Hand over website control	2 hours, 100%
5.2.2	***Controller & Device***		6 hours, 100%
5.2.2.1	***Installation***	Describe installation process	2 hours, 100%
5.2.2.2	***Use***	Describe proper use	2 hours, 100%
5.2.2.3	***Maintenance***	Describe necessary maintenance (e.g. safe power-down)	2 hours, 100%
5.2.2.4	***Storage***	Describe portions that require special long term winter storage	2 hours, 100%
6.0	***Onsite Installation***	Install	3 hours, 0%, Pending Completion
7.0	***339 Requirements***		218 hours, 100%, (12/12/2015)
7.1	***Work Breakdown Structure***	Identify and quantify tasks, put into schedule	20 hours, 100%
7.2	***Project Brief***	Prepare document introducing project to an industrial reviewer	10 hours, 100%
7.3	***Oral Presentation I***		8 hours, 100%
7.3.1	***Project Overview***	Present to the class about the project in its beginning stages	8 hours, 100%
7.4	***Oral Presentation II***		16 hours, 100%
7.4.1	***Progress Report***	Update given progress since last presentation	8 hours, 100%
7.4.2	***Draft***	Prepare draft	8 hours, 100%
7.4.3	***Final Submission***	Prepare final version	8 hours, 100%
7.5	***Fridays at Calvin Poster***	Have poster describing the work for high school visitors	4 hours, 100%
7.6	***PPFS***		60 hours, 100%
7.6.1	***Draft***	Write first draft of PPFS	40 hours, 100%
7.6.2	***Final Submission***	Make edits, add sections, and improve on draft PPFS for final submission	20 hours, 100%

8.0	***340 Requirements***		72 hours, 100%, (5/12/2016)
8.1	***Contemporary Issues Paper***	Write about a contemporary issue relating to the project	8 hours, 100%
8.2	***Team Blurb***	Small poster with team description and visuals	4 hours, 100%
8.3	***Oral Presentation III***		8 hours, 100%
8.4	***Team Description***	Write a one-page description of the team	2 hours, 100%
8.5	***CEAC Project Review***	Prepare and present for Calvin Engineering Advisory Committee.	4 hours, 100%
8.6	***Fridays at Calvin***	Prepare and demo for Fridays at Calvin visitors	4 hours, 100%
8.7	***Oral Presentation IV***		8 hours, 100%
8.8	***Project Posters***	Finish posters for senior design	2 hours, 100%
8.9	***Senior Design Night***	Present final project to parents, professors, and friends	10 hours, 100%
8.10	***Final Design Report***		
8.10.1	***Draft***	First draft of final design report	14 hours, 100%
8.10.2	***Final Submission***	Final draft of design report	8 hours, 100%