A Closed Loop Soil Moisture-Based Irrigation Controller

# Abstract

Irrigation has been pointed out as the catalyst to the development of engineering, organization and political aspects of our civilization. With the developments in technology, irrigation has become more automated. Closed-loop irrigation controllers actively monitor their surroundings, acting on this feedback to provide optimal irrigation water to the cultivated land. This automation has, however, experienced challenges especially in large scale farms in remote areas afflicted by availability of electricity. Such areas necessitate that irrigation controllers and systems be designed for scalability, and consideration to scarce power. As such, development of closed-loop soil moisture-based irrigation controller, with remote monitoring and control capabilities, and solar power was undertaken in this project. A client-server model making use of GSM and TCP/IP communication channels was implemented. A farmer was able to create an account on the website developed in this project, creating watering schedules for crops, and assigning these to clients found on their farm. It was found that the client could fetch this information based on the model developed, acting upon it to control the flow of irrigation water. Powering of client components by solar power was achieved. Finally, remote monitoring the the farm was achieved by logging farm data to be visualized by the farmer from the website developed.

# INTRODUCTION

## Objectives

1. To make use of sensor, actuators and microcontrollers to automate irrigation.

2. To enable remote scheduling of irrigation.

3. To enable data logging and visualization of farm parameters.

4. To make use of solar power to power the irrigation controller.

## Significance

The project would benefit the garden owner in allowing automation of the irrigation process. Through the system, the soil moisture and temperature would be automatically measured, and irrigation water supplied to the field as per the demand. This would free the farmer from the mindless process, freeing him/her to focus energies on other more intelligent activities.

The project would lead to reduction in wastage of irrigation water. The plants would only be supplied with the necessary amount of water needed for their optimal growth. Losses of water due to evaporation as in the case of flood irrigation and due to deep percolation would be as such minimized. This would not only behoove the famer in terms of water expenditure, abut also the water utilities at large due to the high amount of water drawn especially by large farms.

The project would allow the farmer to monitor the farm parameters through a data logging and display utility from their mobile phones. This would better inform the farmer on the dynamic conditions that are usually experienced in his/her farm.

# Literature Review

**… Redacted …**

# Proposed Approach

The project shall be undertaken in four steps: problem identification and research, overall and modular system design, hardware and software parts identification and actual implementation and testing of the system as follows:

First the project problem was identified in facilitating automated irrigation and monitoring of the farm. The scope of the problem was better understood by conducting literature review mostly from secondary sources including journal articles, periodicals, websites and books. It was established that soil moisture and temperature were crucial factors to the growth of crops. Irrigation methods were reviewed, and the operation of solar power and it’s benefit towards irrigation established. Similar systems developed to address the scope of the defined problem were then analyzed. Based on this knowledge and with the opportunities identified in the existing systems, the system design of the project would then be done.

The system design of the project would be done at an overall system and modular levels. A block diagram would be used to show how the different functional elements needed to actualize the objectives set out in the project would be interconnected and communicate. Flow charts would be used to model the program operation of the computing elements in the project. This would include a flow chart for the on-farm controller and mobile software needed to monitor the farm parameters. The operational constraints of the system would be identified in this section, and any other UML tools used to adequately model the performance expected from the system. In addition to this, the power requirements of the system shall be established.

The third stage would involve identification of hardware and software components to perform the functions expected as per the system’s block diagram and models. The soil moisture sensor and soil temperature sensors best suited for the project would be identified in this stage. The computing device to offer optimum performance at a suitable cost would be identified. The actuators needed to execute the irrigation of the farm as per the requirements would be identified. In addition to this, the software platforms on which the data logging and monitoring services are to be implemented shall be determined. The proper solar equipment to be used in order to power this project shall also be identified.

Finally, the system shall be actually developed based on the selected hardware and software. A modular design would be used to independently connect, program and test the functional parts of the project before integrating the complete system together.

# System Design

## System Block Diagram

The block diagram of the system is shown.

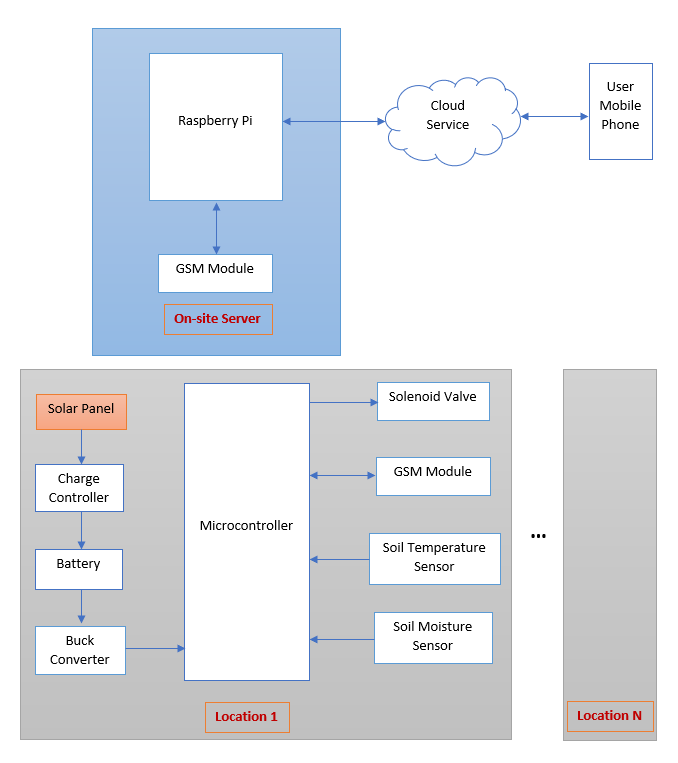


Figure 11: System block diagram

Scalability was a major consideration for this project. Whereas a simple monolithic system design could be used to cater to simple home garden irrigation project, extra consideration had to be given to ensure the system could scale up to large farms. As such, a system based on the client-server topology was deemed suitable to cater to this concern. Using this topology, the irrigation automation system could be broadly divided into four sections: on-site clients, an on-site server, a cloud service and the user mobile equipment. The discussion of each section is as follows:

### On-Site Clients

The clients for this system would be the different locations on the farm at which sensors and actuators would be installed to monitor and control the soil moisture and temperature. Using this approach, a large farm could be divided into different zones, which could be monitored and controlled independently from the other zones. This division could be used to cater to: different crops planted on different parts of the farm with different requirements, different micro-climates in different parts of an extensive farm, different topographies leading to different drainage patterns at different parts, and even provision of extra redundancy on the irrigation system.

A soil moisture sensor and soil temperature sensor would be installed in each client to measure the soil moisture and temperature respectively. These measurements would be periodically sampled using a microcontroller. The microcontroller would then send this data to the on-site server using a GSM module. The use of a GSM module would eliminate the need of using a wired connection to the client. This is because for a large farm, the wired connection would be inevitably susceptible to signal attenuation due to distance, wire tempering by animals and high cost of installation. In addition to this, GSM would highly improve the range at which the client and server can be separated. This would be advantageous compared to Bluetooth and Wi-Fi that are considerably shorter range.

Upon processing of the data by the on-sire server, the client microcontroller would be issued instructions on where to initiate irrigation of the client’s farm area or not depending on the decision made by the higher-intelligence server. The solenoid valve would provide the actuating action.

The power required by the sensors, actuators and microcontroller of the client would be met by using solar power. This approach would eliminate the need of installing power lines to remote sites. Not only would this approach provide a cheaper implementation, but also foster the use of green, renewable energy.

### On-Site Server

An on-site server would be central point to and from which the clients would send raw data and receive instruction on how to operate. By the use of a central server, the system would provide a higher scalability whereby a new client need only connect to the known server upon installation at whichever point on the farm. The design would also provide a convenient intelligence distribution option. If new operating soil moisture and temperature settings are desirable at different points of the plant’s growth, just a single database would need to be updated. This would eliminate the need of extracting, reprogramming and reinstalling each client microcontroller once installed. In addition to this, the server would have higher compute power and higher intelligence data and information in order to drive the clients more efficiently.

A Raspberry Pi would provide the compute power needed by the server. A GSM module would be used to receive raw soil moisture and temperature data from the different clients on the farms, and subsequently send the control actions to each client on how to provide actuation based on the desired optimum values.

The on-site server would have internet access. This would enable the server to log the farm parameters received from the client in order to be actively or passively monitored by the farmer from whichever remote location. In addition to this, internet access would open up the system’s database to being updated or modified remotely without the necessity of the farmer being actively on-site at the server.

### Cloud Service

The cloud service would be used to log the data received from the farm. This would provide the farmer the capability of remote monitoring of the farm, remotely detecting faults and remotely configuring the on-site server.

### User Mobile Equipment

This would provide an interface with which the farmer would be able to view the data logged from the farm for each client. In addition to this, an interface would be provided for configuring the on-site database in order to modify or update the optimum soil moisture and temperature for each client zone as desired. This would provide the flexibility of updating the desired parameters at different stages of crop’s growth in each zone.

## Flowcharts

The overall operation of the system can be represented by the following flowchart.

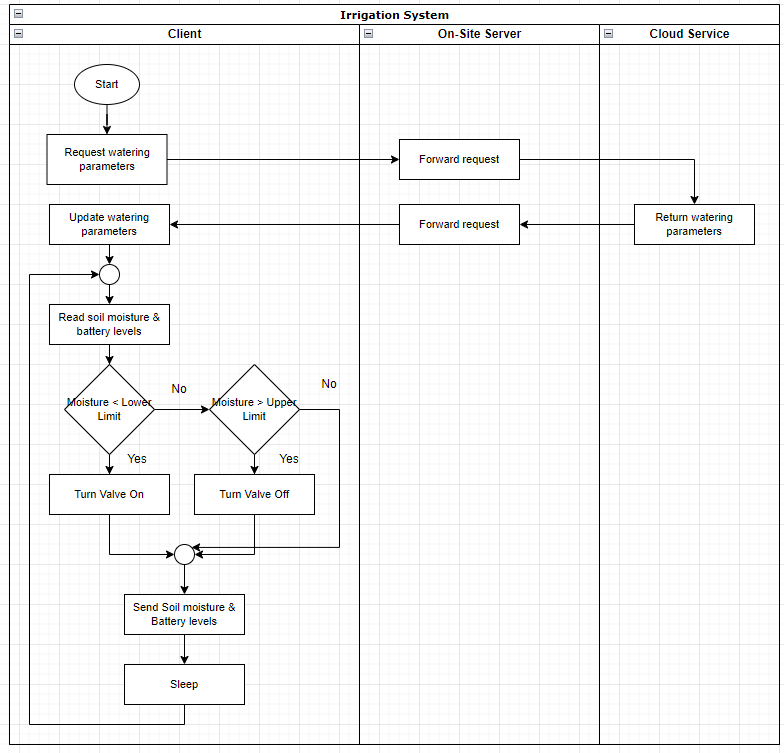


Figure 12: System flowchart

# System Components

## Hardware Components

Hardware components refer to the physical sensors, actuators, controllers and other electrical and electronics components used in the project. Factors such as cost, efficiency, power consumption, compactness and ruggedness are considered in selecting the hardware components. The discussion of the hardware components selected for this project can be divide into two: client components used in the different zones of the farm, and on-site server components.

**Client Components**

These are the components used in the different zones of the field. In order to ensure financial feasibility of the system on an upscaled implementation, cost considerations have to be made while at the same time providing reliable operation on the field.

***Soil Moisture Sensor***

A capacitive soil moisture sensor was used to measure the soil moisture content. This sensor works by measuring the dielectric strength formed by the soil, whose most important contributor is water. This sensor is advantageous compared to resistive soil moisture sensor in that the sensor is less prone to corrosion, while offering more accurate readings [10].



Figure 13: Soil moisture sensor

The sensor has the following specifications:

* Analog output
* Operating voltage: 3.3 – 5.5 VDC
* Output voltage: 0 – 3 VDC
* Operating current: 5mA

***Soil Temperature Sensor***

The LM35 sensor was selected as the temperature sensor for this project. The sensor offers low output impedance, linear output dependent on the ambient temperature and doesn’t require external calibration. In addition to these, the sensor is also highly affordable.

The specifications of the sensor are as follows:

* Operating voltage: 4V – 30V
* Operating current: 60uA
* Linearity: +10mV/°C
* Temperature range: -55°C - 150°C

***Microcontroller***

The ATmega 328P microcontroller was selected for this project. The microcontroller would be programmed using the Arduino development kit, extracted and used on a standalone bespoke PCB. This board would then be deployed to the field. This design would reduce the size of the deployed controller, improve ruggedness and reduce cost of the system. This is because this approach would only necessitate the use of a single Arduino board merely for programming of the many microcontrollers that would be deployed on an upscaled system, with the microcontrollers being deployed on low-cost PCB boards.

The ATmega 328P microcontroller has the following specifications:

* IC type: AVR
* Operating voltage: 1.8V – 5.5 V
* Current: < 200mA full capacity (all I/O sensors used)
* Core size: 8 bit
* Number of I/O: 23
* Package: DIP-28
* Program memory: 32Kb flash memory

***GSM Module***

The SIM800L V2.0 GPRS GSM Module was selected as the GSM module for this project.



Figure 14: SIM800L V2 Module

The specifications of the module are as follows:

* Operating voltage: 4.6 – 5.2 VDC
* Sleep mode current: 1 mA
* Transmission burst current: 2A
* Interface: TTL UART
* SIMCARD: micro SIMCARD holder
* Control via AT Commands

***Solenoid Valve***

A 12V ½” BSPP solenoid valve was selected for this project. The valve is usually closed and opens when 12VDC is applied on its terminals. The valve is gravity fed removing the necessity of multiple pumps. The flow of water is in one direction.

Figure 15: 12V 1/2" solenoid valve

The specifications of the valve are as follows:

* Operating voltage: 12VDC
* Operating current: 250mA
* Operating pressure: 0 – 0.02 MPa
* Valve type: diaphragm (servo operated)

Power Requirements of Components

This section analyzes the power requirements of the components selected in the previous discussion for the client. The selection of solar-related equipment is based on the analysis obtained in this section.

The electrical usage of the components is calculated by multiplying the average operating voltage and current for each component as follows:

* Soil moisture sensor:
  + Average operating time per day: 2 minutes
* Temperature sensor:
  + Average operating time per day: 2 minutes
* ATmega 328P:
  + Average operating time per day: 10 minutes (not sleeping)
* GSM Module: Transmission burst power consumption -
  + - Sleep mode power consumption -
  + Average transmission time per day: 5 minutes
    - Average sleep time per day: 23 hours
* Solenoid valve:
  + Average operating time per day: 4 hours

Let the system be powered by 6VDC, 3.8Wp PV module

1. Power Consumption Demands

Total components use: (0.25W x 0.033h) + (0.0003W x 0.0033h) + (0.8W x 0.166h) +

(10W x 0.083h) + (0.005W x 23h) + (3W x 4h) = 13.1Wh/day

Total PV panel energy needed: 13.1 x 1.3 = 17W

2. Sizing panels

Total Wp of PV panel capacity: 17 / 3.4 = 5

3. Battery Sizing

Nominal battery voltage: 12V

Days of autonomy: 2

Battery capacity:

4. Solar charge controller sizing

Maximum current:

***Solar Panel***

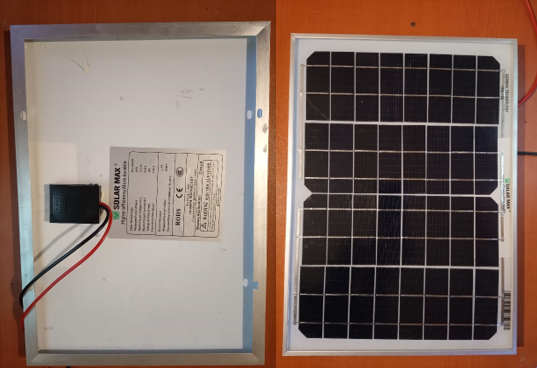
A single 12V 10Wp solar panel was selected for tis project. This was the solar panel found closest to the 5Wp specification found in the panel sizing section. 

Figure 16: 12V 10W solar panel

***Battery***

A 12V, 9Ah battery was selected to store charge for the client’s components. On full charge, the battery was selected to be capable of powering the components for two days autonomously without being charged by solar energy.

****

Figure 17: 12V 9ah battery

**Solar Charge Controller**

A 12V, 10A solar charge controller was selected for this project.



Figure 18: 12V 10A solar charge controller

**On-Site Server Components**

**Raspberry Pi**

Raspberry Pi Model B+ was selected to be computing device running on the on-site server. This device was preferred over an Arduino-based development kit on a microcontroller for: better capability of handling TCP/IP connections, multithreading to handle different clients alongside the networking functionality more efficiently and larger memory capacity and higher processing capability to handle higher intelligence operations.



Figure 19: Raspberry Pi B+

The specifications of the device are as follows:

* Operating voltage: 5V
* Maximum operating current: 2.5A
* Idle state operating current 400mA
* Processor: Broadcom BCM2837B0 64-bit ARM Cortex-A53 Quad Core Processor SoC running @ 1.4GHz
* Memory: 1 GB RAM LPDDR2 SDRAM
* I/O: 40-pin GPIO
* 2.4GHz and 5GHz IEEE 802.11.b/g/ac wireless LAN
* Bluetooth 4.2 and BLE

**GSM Module**

The SIM800L V2.0 GPRS GSM Module, similar to that employed in the clients, was selected for the on-site server.

## Software Components

Python was chosen to be the programming language to program the Raspberry Pi. This language offered extensive documentation paramount to the smooth development of any system, short and concise syntax to lessen the programming burden and open-source libraries necessary to avoid the writing of boiler-plate code but instead focusing more on the system logic.

C was chosen to be the programming language to program the ATmega328 microcontroller. The language provided a more programmer-friendly language to program compared to assembly language, while easing the debugging process, and quickening development time by abstracting lower-level operations by making use of libraries.

HTML, CSS, Javascript and PHP were selected for creating the website with which a user could remotely monitor and administrate their farms. These programming languages were selected for their extensive documentation and support.

# Implementation

## On-Site Server Side

The diagram below shows the wiring diagram of the on-site server. The Raspberry Pi B+ PC was connected to the Sim800L module. Power was provided to the server components via 5V USB connection. An ethernet connection was used to provide internet access to the PC and execute commands and monitor the device via SSH.



Figure 20: On-site server picture

The main program running on the onsite server was a python program named “server.py”. The complete program is provided in the appendix of this report. The main functions of the program were:

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| def initalizeSim() | Initialized a global object to the serial interface attached by the Sim800L module to the Raspberry Pi PC. |
| def checkNewMessage() | Polled the GSM module for new messages |
| def respond\_to\_incoming\_sms(msg\_id) | Was called when a new message was received. Received the message’s index as parameter. |
| def sendMessage(phone\_number, message) | Was invoked to send the message specified by the second parameter to the phone number specified by the first parameter |
| def validateClient(phone\_number) | Was invoked to: validate a client communicating with the server and send the soil moisture parameters specified for that client. This function invoked an exposed URL on the cloud server to validate the client from the universal database, and receive the corresponding parameters, via a POST method. |
| def logClientData(phone\_number, message) | Posted the soil moisture readings and battery level received from each client to the universal database by calling an exposed URL on the cloud server. |

Table 1: On-site server functions

## Client Side

**Schematic Design**

The client circuit was based the ATMega328P microcontroller. The microcontroller was in charge of reading the soil moisture and battery values, communicating with the higher order components via the SIM800L module and turning on or off the solenoid valve. The microcontroller was connected to the other components on the client side as shown in Figure 21.

The user-defined pinouts of the microcontroller were as follows:

|  |  |
| --- | --- |
| Pin 17 | Sim800L TX |
| Pin 16 | Sim800L RX |
| Pin 23 | Soil moisture sensor analog input |
| Pin 24 | Battery voltage analog input |

Table 2: Client pinout

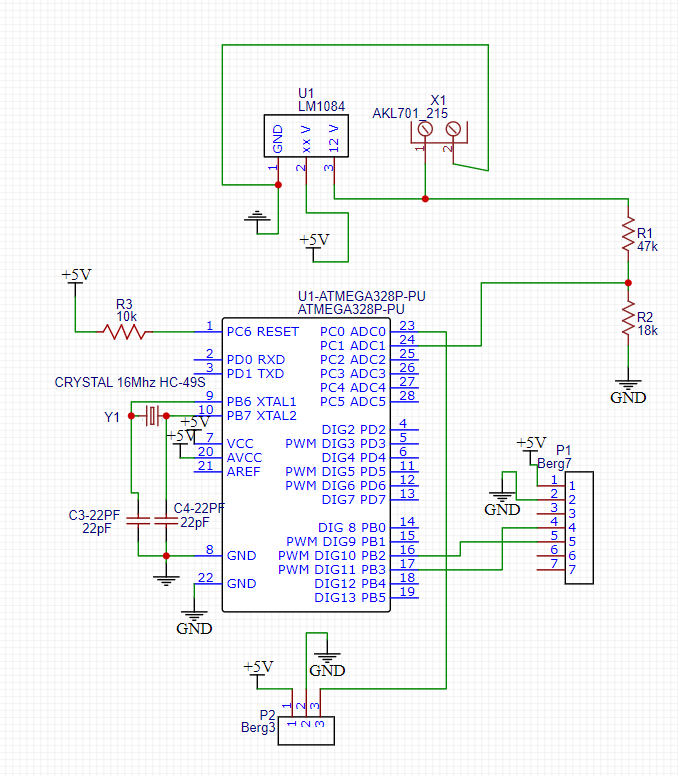


Figure 21: Client schematic diagram

The components comprising the client were connected as shown in Figure 22 and 23. The 10W solar panel was connected to the solar charge controller, which controlled the charging of the battery. Output from the solar charge controller was fed to the LM1084 voltage regulator that produced constant 5V DC to power the electronic components of the client. A relay was used to provide 12V signal to turn the solenoid valve on or off depending on the input signal received from the microcontroller. The solenoid actuated gravity fed water – simulated by the column of water in the jerrycan. The Sim800L module enabled communication via SMS with the on-site server.



Figure 22: Client picture

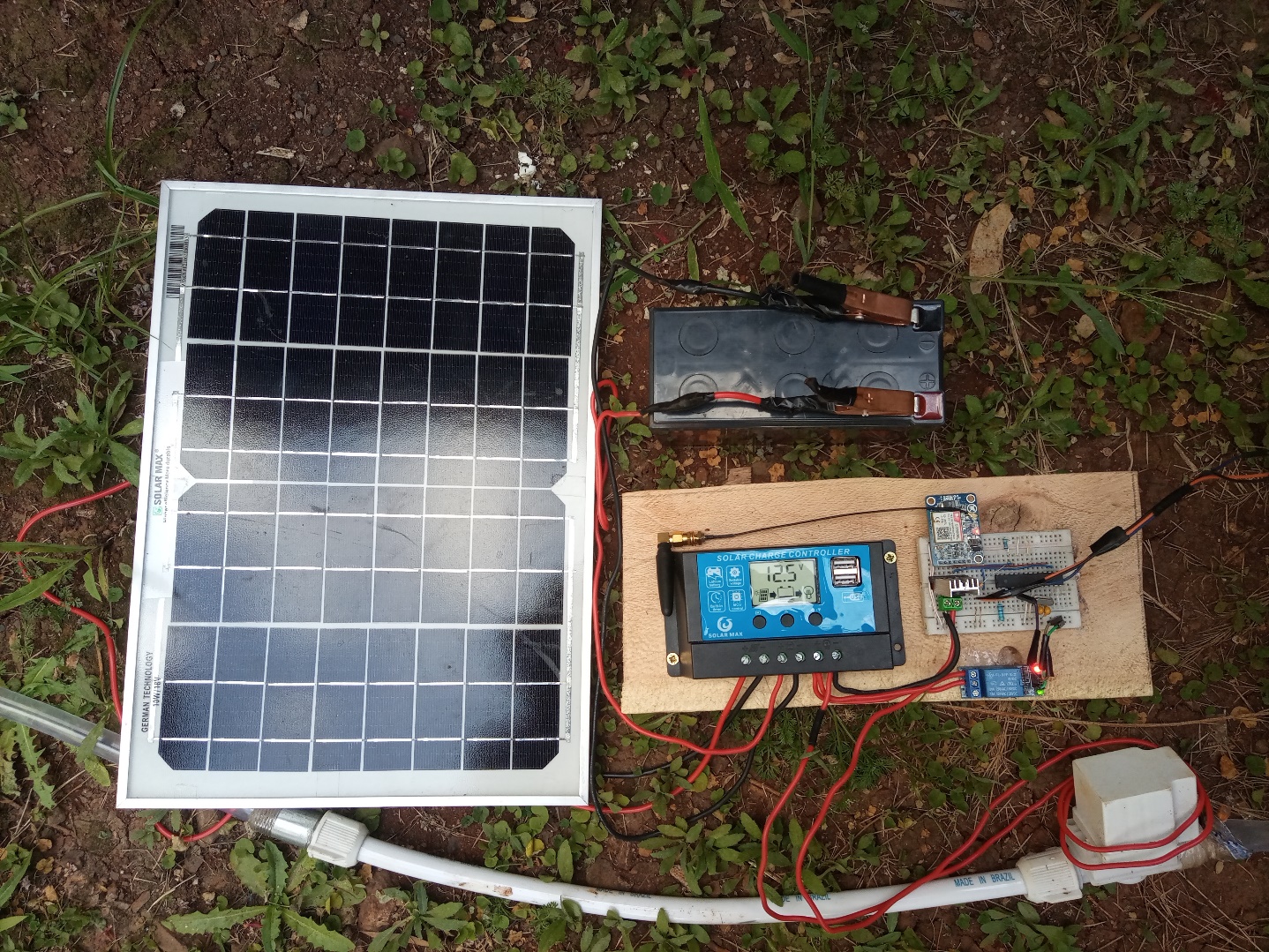


Figure 23:Client picture - zoomed

The microcontroller was programmed in C. The complete program run on the client is provided in the appendix of this report. The main functions of the program were:

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| void sendGSMMessage(String message) | Accepted a string argument and sent this via the Sim800L module |
| void GSMHandler() | This was the central handler for GSM functionality, coordinating the execution of various functions. The function:  1. called void waitForResponse() to check if an AT notification had been received by the Sim800L module.  2. invoked bool isNewMessage(String notification) to check if the notification was a new message.  3. if the notification was for a new message, invoked int getMessageIndex(String notification) to get the index of the message.  4. read the contents of the message at the gotten index by invoking String readMessageAtIndex(int index)  5. parsed the message header to determine if the message originated from the phone number of the on-site server assigned to the client  6. If the message originated from the client’s server, checked the command so sent by the server.  7. If the command was in response to the client making an initialization request, the moisture parameters so received in the response were applied to the client.  8. Alternatively the response could be an acknowledgement by the cloud server on receiving the client’s data log instance. |

Table 3: Client functions

## Cloud Side

A free web hosting service called “000webhost.com” was used to host the website. This enabled the website to be publicly available from whichever location, and accessible by the Raspberry Pi B+ PC. The web host enabled hosting of the php files needed to create the website and the MySQL database used to store information about clients and servers. The primary URL to the website was <https://irrigation-project.000webhostapp.com>.

Several php files were used in coding the website:

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| config.php | This file contained the credentials needed to access the MySQL Database used to store information related to the system. |
| login.php | This file was used to enable a user to login to his/her dashboard. Session variables were used to enable persistent login sessions of the user. |
| logout.php | This file was used to enable a user to logout. This effectively deleted the session variable associated with the user. |
| create\_server.php | This file was used by a new user to sign up to the service, enabling him to create a fresh server. |
| my\_server.php | This was the dashboard accessed by the user on successful login. From this dashboard, the user could: get an overview of clients associated with their on-site server, and get an overview of the crops’ watering parameters created by the user. |
| create\_client.php | This file was used to create a new client associated with the server |
| view\_client.php | This file was used to: view the soil moisture and battery level logs of the client, edit the client’s settings and delete the client at wish. |
| create\_crop.php | This file was used to create the watering parameters of a particular crop |
| edit\_crop.php | This file was used to edit the watering parameters of a particular crop |

Table 4: Cloud server functions

In addition to these file that enabled a user to interact with the website via the browser, an additional service was created to enable the cloud server to communicate with the clients. These files were contained in a directory named “web\_service”. The functionality of these were as follows:

|  |  |
| --- | --- |
| **File** | **Function** |
| authenticate.php | This file was called by the on-site server on receiving the initialization command from the client. The file received the client phone number, verified that it existed in the system’s database, fetched the irrigation parameters associated with the crop planted in that particular client’s region, and sent these back to the on-site server to be forwarded back to the client. |
| post\_log.php | This file was used to record the soil moisture and battery level log instances sent by the client via the on-site server to the universal database. This subsequently enabled the user to view the logs graphically on the website. |

Table 5:Cloud server files

A MySQL database was used in order to persistently store the information entered by the farmer on the website and logged from the client. The database was comprised of four tables: servers, clients, crops, and logs. The brief schema of the database showing the tables and their columns was as shown.

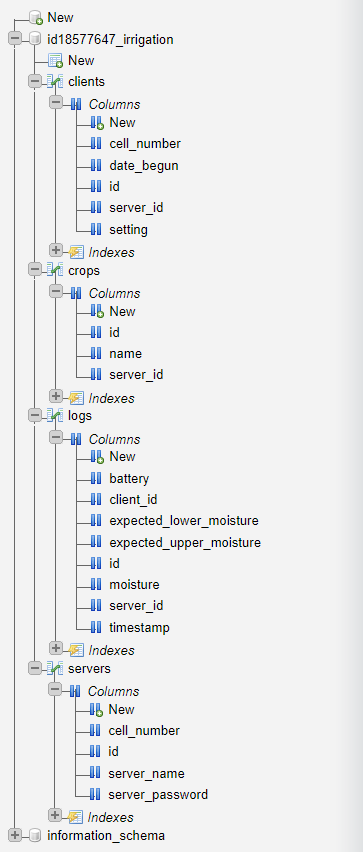


Figure 24:Database schema

# Results

**1. Creating a server**

A new user was ale to open an account with the service and create a fresh server by accessing “<https://irrigation-project.000webhostapp.com/create_server.php>”. The user name, phone number of the sim card placed in the on-site server, and password to secure access to the user’s dashboard where requested.

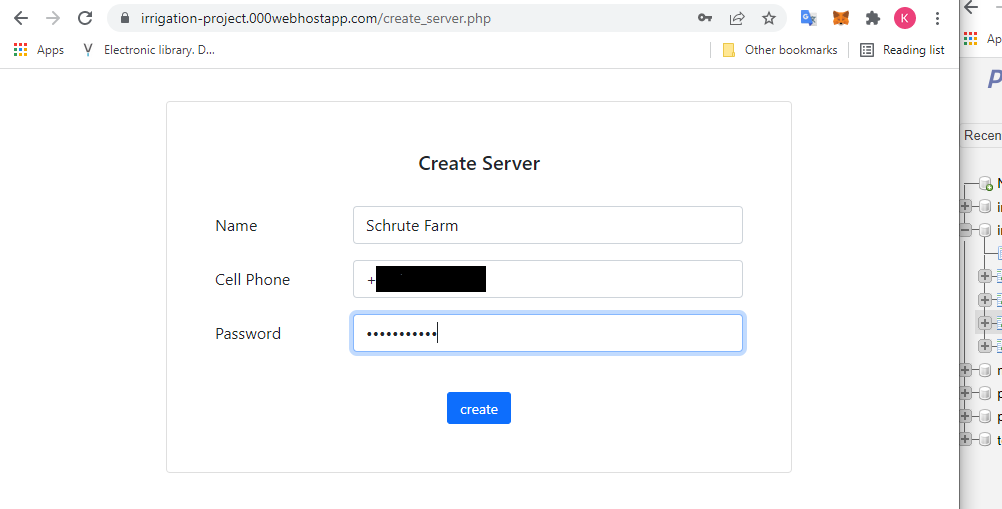


Figure 25:Creating server definition on cloud

**2. Logging in**

A user was only granted access to his/her dashboard on entry of registered user name and password as shown in Figure 26. This mechanism was enforced to provide access control to the confidential information of farmer’s farms stored on the platform. On successful login, the farmer was granted access to his dashboard as shown in Figure 27. From this dashboard, the farmer had the flexibility to define clients to control different zones of the farm, access data logged concerning different clients, and define watering schedules for the different plants cultivated by the farmer.

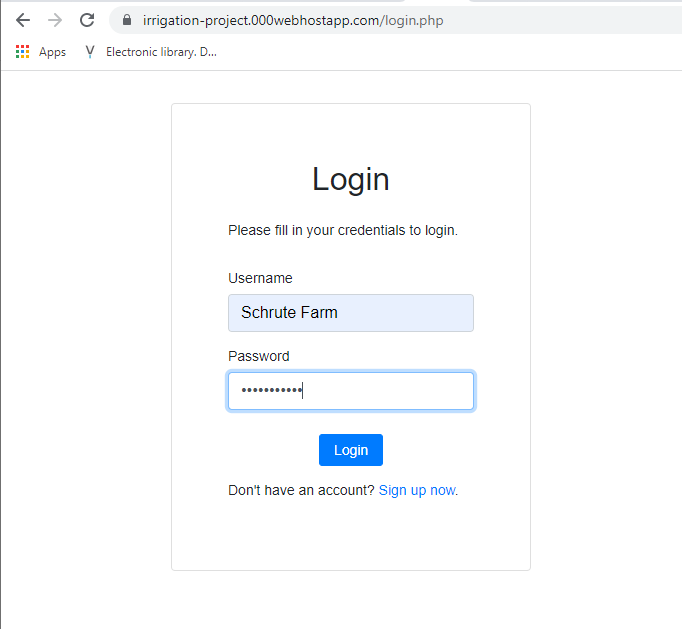


Figure 26:Logging into cloud account

**3. Creating a crop’s watering schedule**

The website allowed a farmer to define watering schedules as shown in Figure 28. Different watering schedules could be defined for different crops. Flexibility was provided in in the definition of this schedules, allowing the user to define different optimal soil moisture values desired at different stages of the plant’s growth. The interface presented to create the watering schedule of a plant required the farmer to input the crop’s name, and define an array of lower soil moisture and upper soil moisture level for a span of days.

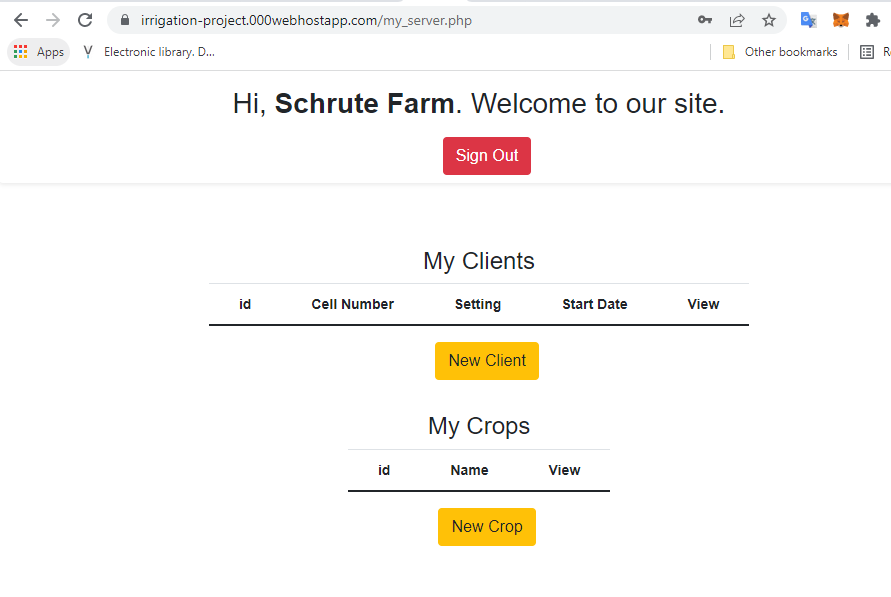


Figure 27:Farmer's dashboard on initial login

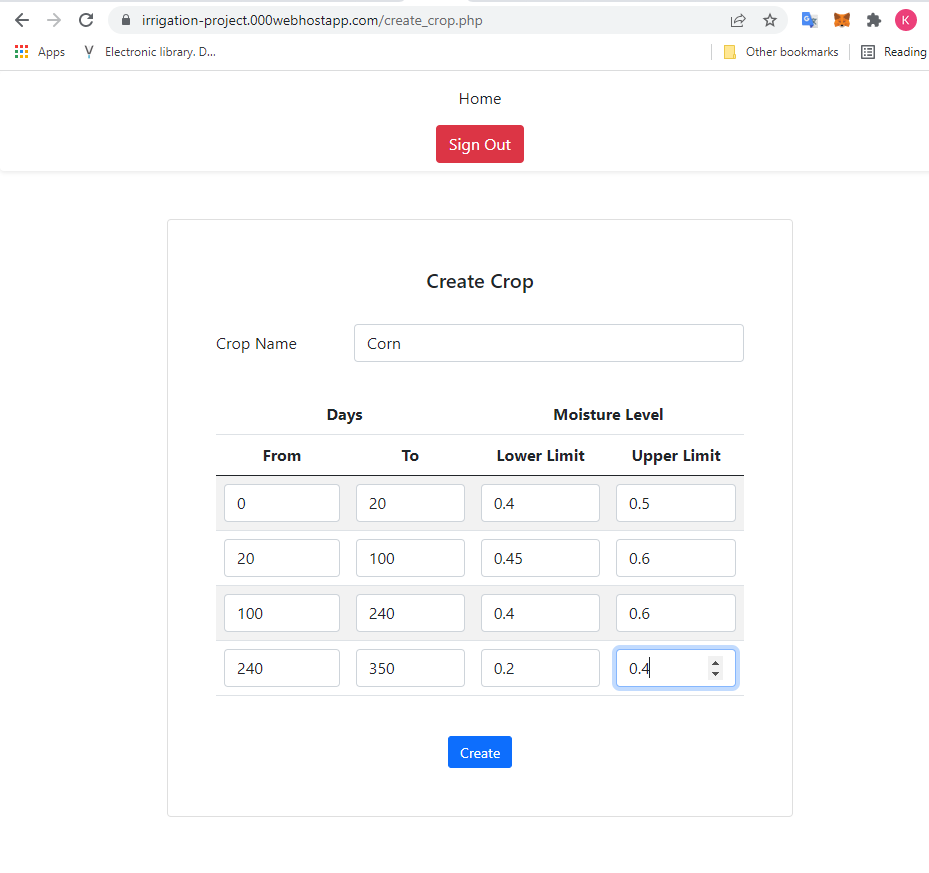


Figure 28:Creating crop's watering schedule

**4. Creating Clients**

The farmer was able to define the different clients to be found on his farm. Each client would contain the PV equipment, microcontroller and GSM module as described in the Implementation chapter. In order to create a client, the user was required to enter the phone number of the sim card attached to the GSM module of the particular client, the watering schedule to be applied to that client, and the date the operations of the client are to begin.

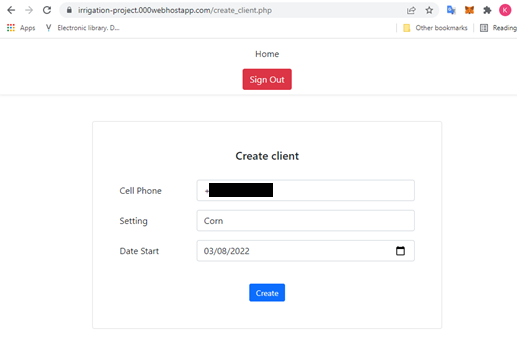


Figure 29:Creating new client

The dashboard reflected the creation of watering schedule of a crop and definition of a client to make use of the pre-defined watering schedule. The dashboard offered the user the ability to edit or delete a watering schedule or client defined in the user’s system.

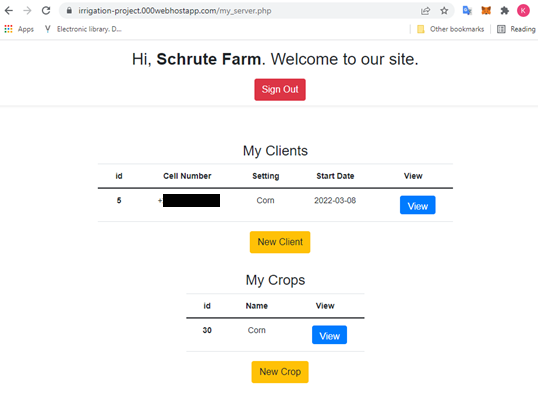


Figure 30:Dashboard after populating crop and client

**4. Client-Server interaction**

The server and client were powered on. The server begun listening to incoming SMSs via the “server.py” program described in the Implementation chapter. The client on powering on, begun its initialization by requesting the server for soil moisture parameters with which to operate as described in the Implementation chapter. The on-site server received this SMS, parsed it and requested for authentication for that client. The client was then authenticated by the cloud server, and sent its soil moisture parameters via the on-site server.

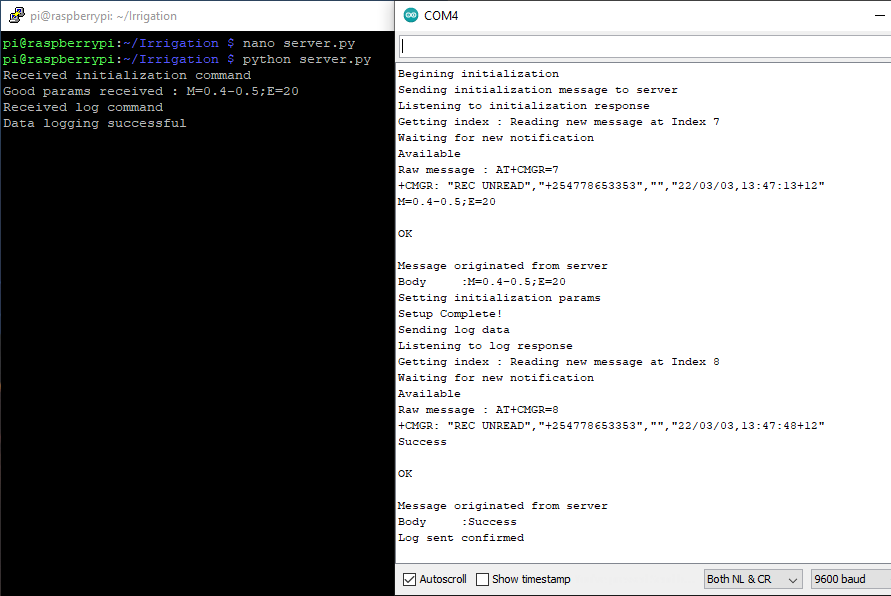
This sequence of interaction between the client, on-site and cloud server is shown in the picture below. Here, the COM port was used during testing to visually show the internal state of the microcontroller. The state of the on-site server was assessed by using SSH to access to Raspberry Pi PC. ****

Figure 31:Client-server interaction captured

It was seen that the parameters received by the on-site server was 0.4 – 0.5 with and expiry of 20 days as expected from the corn watering schedule for days 0 – 20, with the date of creating of the client being the same as the day the soil moisture parameters were requested in the above instance.

The client then begun its infinite loop by reading the current soil moisture value and battery level, and sending that to the cloud server for logging via the on-site server every 30 minutes. The data log was successfully received by the cloud server and confirmed back to the client via the on-site server as shown in the last line of the COM4 serial monitor.

The solenoid valve was turned on when the soil moisture fell below the lower threshold received from the initialization step. When the soil moisture rose above the upper threshold received from the initialization step, the solenoid valve was turned off.

**5. Viewing logged data**

The system was left to operate for a day. The data log received by the cloud server from the clients could then be viewed by clicking the “View” button of that client from the dashboard.

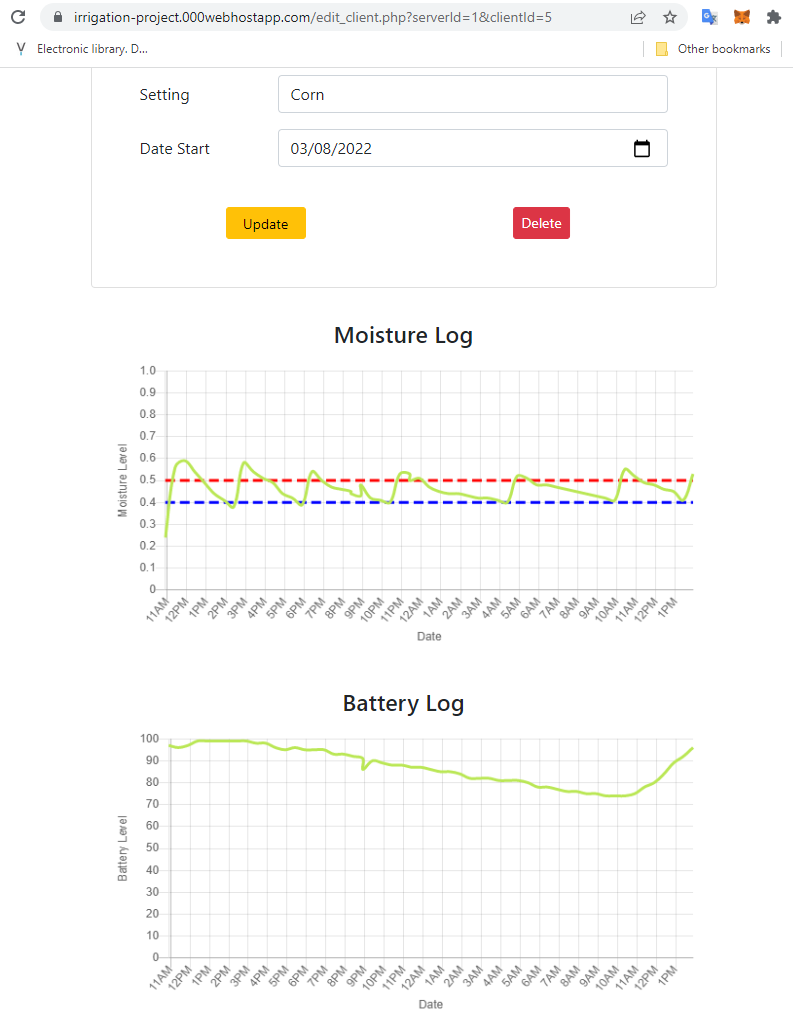


Figure 32:Viewing logged data

The line chart below the “Moisture Log” heading contained the soil moisture logged for the client. The green curve represented the actual moisture logged, the dashed red line the upper soil moisture threshold expected at an instance, and the dashed blue line the lower soil moisture threshold expected at an instance.

The line chart below the “Battery Log” heading contained the battery level logged for the client.

# Discussion

This chapter evaluates the results obtained in relation to the objectives set out for the project.

## Automating irrigation

It was important for the system developed in the project to minimize the intervention needed by an operator to perform irrigation. As such, the irrigation controller developed made use of a capacitive soil moisture sensor to read the soil moisture level, and based on thresholds received, automated the release of irrigation water onto the garden via a solenoid valve.

An emergent issue discovered in the project was the positioning of the soil moisture sensor. Tests found that placing the sensor too close to the irrigating source produced misleading results. On the other hand, placing the sensor too far from the irrigation source led to wastage of water before percolating to the sensor. Calibrating the sensor was also found to be susceptible to error. This is because two extremes were needed to calibrate the sensor – value in air, and in water. Different ionic contents of water produced different readings.

Having obtained suitable distance from the irrigating source, the irrigation controller was however able to turn on or off the solenoid valve automatically to water the soil such that the soil moisture remained within acceptable limits of the thresholds set. Little overshoots were noticed in the soil moisture level. This was attributed to the sleep periods iterated by the microcontroller prior to polling the soil moisture level in order to minimize power consumption.

## Data logging and remote visualization

The project intended to provide the farmer access to remote monitoring of the farm parameters. In order to achieve this, the soil moisture and battery levels were read every 30 minutes and transmitted to the cloud server for storage in a series of steps. First, the client packaged the values read in an SMS and sent these to the on-site controller. The on-site controller received this, parsed it and transmitted the values to the cloud server via the internet. The cloud server then finally received this data, and stored it in the log table on the database for persistent storage.

Visualization of this data was enabled by the use of line charts. The values read for soil moisture were plotted on a chart, together with lines showing the expected limits for that client on the same chart. This enabled the farmer to view the fluctuations of the soil moisture on the farm with time. The battery level was similarly plotted, from which the impact of sunshine on the charge could be inferred.

## Remote scheduling of irrigation

The website provided the farmer with an interface to schedule the watering plans for crops defined in the system. Using this interface, the farmer could set the optimal soil moisture thresholds at different stages of the crop’s growth. The farmer could define watering schedules for a number of crops. This provided the flexibility for different clients, operating in different parts of the farm, with different crops in each, to have different watering schedules.

It was observed that using soil moisture level provided an easy way for the watering conditions of crops to be specified. The development, however, noticed that the use of water volume, or interconversion between soil level and water volume, could be a beneficial addition to providing farmers different ways of defining the watering needs of their crops according to their intuition.

## Powering the irrigation controller using solar power

The development of the project envisioned such a system being deployed in remote areas that could be afflicted by power availability, or vast farms in which running power lines from the consumer unit to the remote client sites would not be an attractive solution. As such solar power was considered in powering the clients found in the system.

Based on the total Watt-hour per day projected for the client components of 13.1 Wh/day, it was found that the total PV energy needed was 17W by factoring losses, from which the peak watt panel capacity needed was found to be 5W by factoring the panel generating factor. Using a 12V battery, it was found that 8.3Ah was required by the battery factoring the battery loss factor and depth discharge factor. Finally, the charge controller was calculated to be handle 0.83A.

Based on these specifications, the closest components found were: a 10W solar panel, 12V nominal voltage 8.3Ah battery, and 10A solar charge controller. The components were found to be suitably power the client.