Smart Farming System Using Raspberry Pi

†Anastasiya Chilukuri, †Albrin Richard
†Department of Computer Science, California State University, Sacramento, CA 95819, USA
Email: albrinrichard@csus.edu, achilukuri@csus.edu

***Abstract*—This is the report for our project Smart farming using Raspberry-Pi for course CS275. In the current world of IOT, we want to leverage the technologies we have at hand, and create a Smart way to monitor our crop/plant's health, take actions based on the collected data, eventually automate the process.**

I. Introduction

The Internet of things (IoT) has gained huge popularity in recent years, and for good reasons. Imagine this scenario, you went on a trip, and on the trip you realized you might not have turned off the Air Conditioner(AC) in your house, wouldn't it be great if you can check the AC status while you are on trip? that's where an IoT solution comes in handy, you can get a thermostat can be linked to your WIFI at home, which you can control from any where in the world as long as you are connected to Internet.Availability of tiny, programmable, affordable computers like Raspberry-pi had opened the avenues for DIY projects to tech enthusiasts, students, hobbyists, and leading them to implement smart solutions for everyday problems, like the scenario above and these solutions are available at the tip of your hand by simple web/mobile app. Even tech giants are investing a lot into IoT technologies, and this trend does not seems to fade away.

We want to use the current available technologies, and implement our own IoT solution as part of project for course CS275. Our project is intended to address the problems faced by almost every plant enthusiast, who wants grow plants in their backyard/apartments, just like pets plant needs attention, they need to be watered, made sure not exposed to extreme temperatures, and not underexposed to light. But the difference between pets and plants is that plants cannot communicate with humans as pets does, so, what if we can interpret the data that is surrounding plants, like soil moisture, temperature, light exposure, and made decisions based on the interpreted data? wouldn't it be great if there is system that can determine the soil moisture around your plant, and water the plant if the moisture falls below a certain range? That's exactly our project. We created a plant watering system that determines the soil moisture, and determines if there is a need for watering. We also created a webapp, and mobile app through which user can monitor the moisture levels, and control the water pumps.

II. Project Equipment Description

In this project, we used Raspberry-pi2, Esp8266/NodeMCU, soil moisture sensors, 5v relays, 5v DC water pump. For details on the equipment used refer Table I.

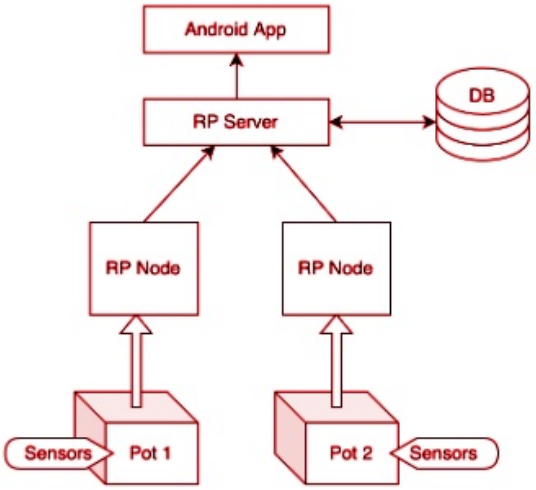


Fig. 1. High-level system architecture

III. Components Descriptions: Nodes

For the purpose of driving the soil moisture sensors, and pumps, we used ESP8266 micro controller with WIFI enabled. These nodes are powered using a 3v USB power supply. We have use ArduinoIDE to write the driver code, later to upload it to the esp8266.

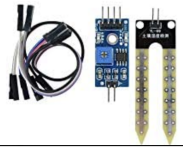
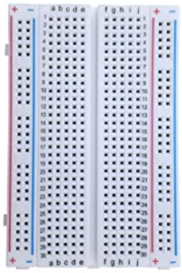
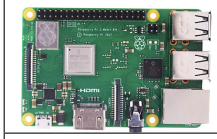
Esp8266's A0 pin is used to read the analog input from the sensor, and 3v power supply pin is used to power the soil moisture sensor.

A 5v relay is used to turn on/off the 5v DC water pump, which is connected to water reservoir. Esp8266's Vin pin is used to power up the 5v relay, and D0 pin is used as an input signal to the relay.

Circuit diagram is shown in Fig. 2

The Flowchart representing the logic running on Nodes is shown in Fig. 3

TABLE I
LIST OF EQUIPMENT USED IN THIS PROJECT

Module	Name	Quantity
	ESP8266 ESP-12E Development Board WiFi WLAN Wireless Module CP2102	2
	Soil Hygrometer Moisture Detection Water Sensor Module YL-69 Sensor and HC-38 Module	2
	400 Pin Breadboard	2
	Relay Module DC 5	2
	9v battery	2
	Mini Water Pump DC 3V 5V	2
	Raspberry pi 2	2
	Wires	As needed.

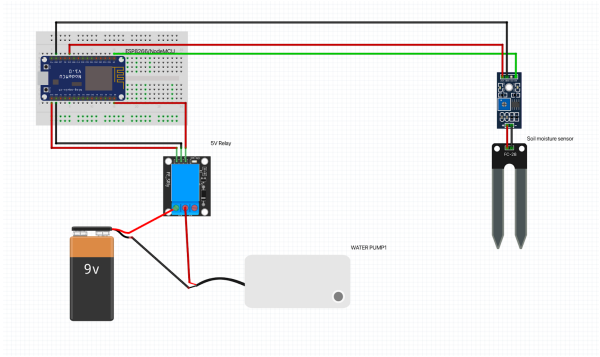


Fig. 2. Circuit diagram indicating node setup

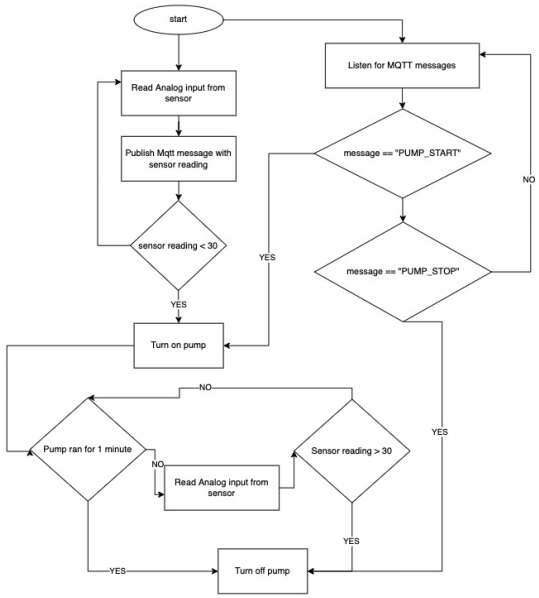


Fig. 3. Flowchart representing logic running on Nodes

IV. Components Description: Raspberry Pi as Server / Web Server

In our project Raspberry Pi servers multiple roles, we will go over the details on each role, and the purpose of the role. A system architecture from the Raspberry Pi point of view is shown in Fig. 5

A. MQTT Broker

MQTT (originally an initialism of MQ Telemetry Transport) is a lightweight, publish-subscribe, machine to machine network protocol. [1]

We have used MQTT as a means of communication between different components of our system. Nodes (Esp8266), to which soil moisture sensors are connected, publish sensor reading as MQTT messages, and also listens for message that control pump. Web dashboard, which displays the sensor data listens to the MQTT messages from sensor to display latest reading, and also publishes MQTT messages to turn on/off pump based on the user input.

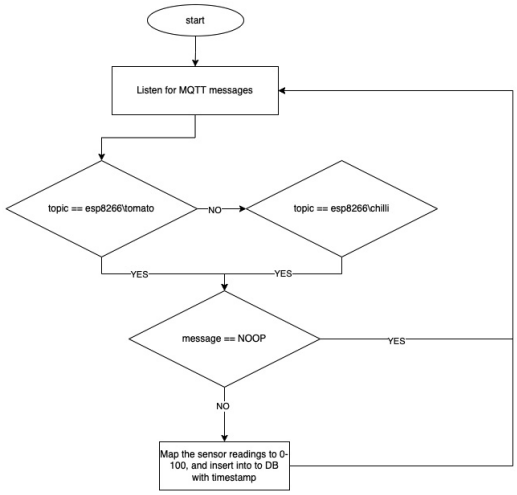


Fig. 4. Flowchart representing Python MQTT client logic

We used mosquitto version 2.0.11, mosquitto is an MQTT v5.0/v3.1.1/v3.1 broker.

B. MQTT Client

Now that we established the fact that Nodes use MQTT protocol to publish sensor data, we need some way to store that data, for that reason, we created a python MQTT client which runs on Raspberry Pi. The responsibility of the client is to listen to message from Nodes, parse those messages, and insert them into a database with timestamps. This data is used to plot moisture level trends over time.
The Flowchart representing the Python MQTT client logic running on Raspberry PI is shown in Fig. 4

C. WebServer

We have collected data from Nodes, stored them in to Database, now we need a way to monitor that data, and also control the pumps if we want to, for that reason, we created a web dashboard. To view the data on any device connected to the network (for now on LAN), we need a webserver. We have use Apache webserver.

- To read the database, we used PHP. On the page load, we invoke the PHP script which reads the database, and then use the data read by PHP to plot graphs.
- To make the web dashboard more appealing to the user, who might be using a phone/tablet/laptop, we use bootstrap.js/CSS.
- To plot the graphs we use Chart.js.
- To handle the button events we use Jquery/JavaScript.

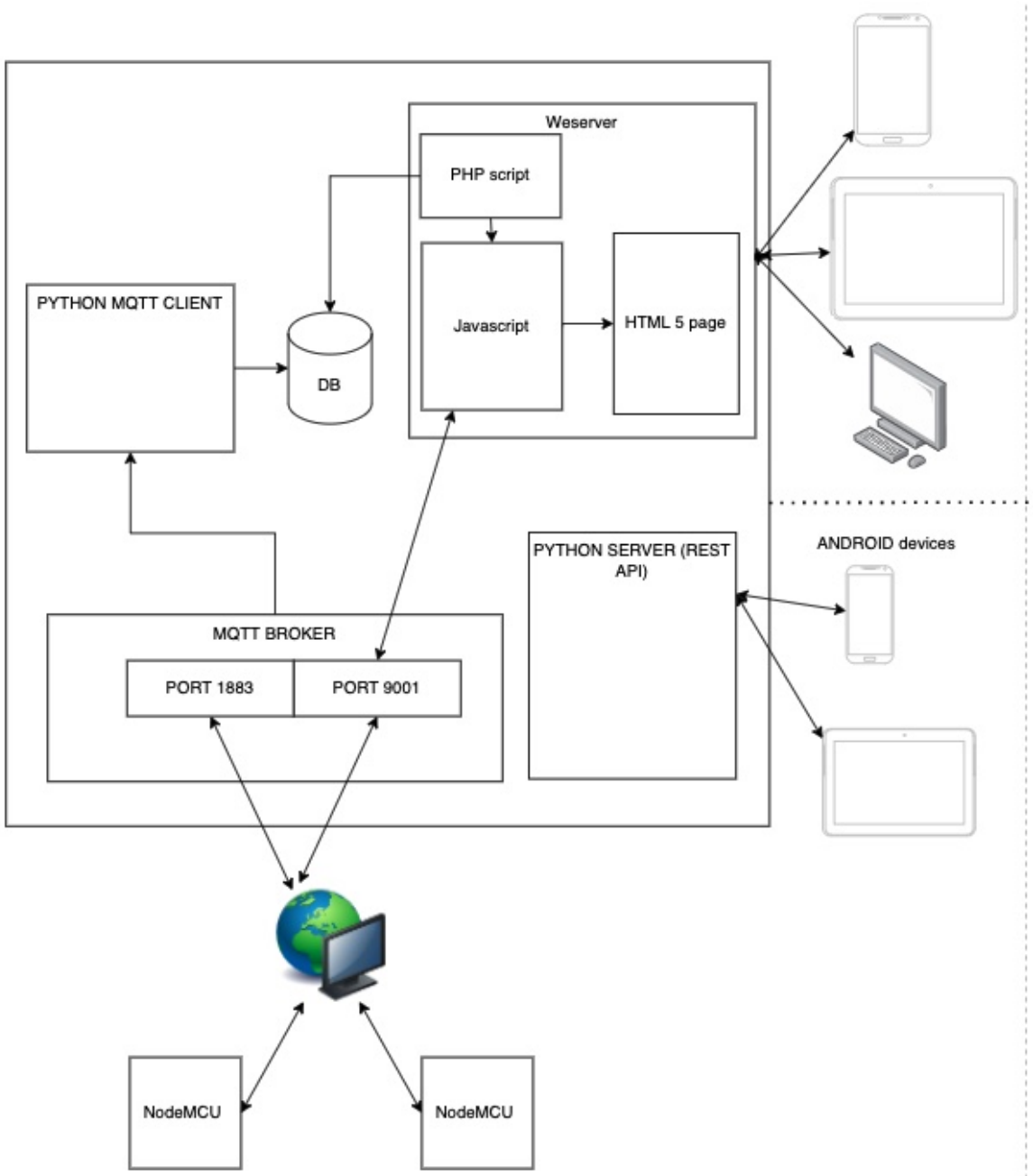


Fig. 5. System architecture

```
sqlite> .open sensor_data.db
sqlite> .schema
CREATE TABLE tomato (date_time TEXT PRIMARY KEY, mositure_level INTEGER);
CREATE TABLE chilli (date_time TEXT PRIMARY KEY, mositure_level INTEGER);
```

Fig. 6. Tables used to store sensor data

D. Database

The collected sensor data is stored in a SQLite Database, which eventually accessed by our web dashboard, Android app. Fig. 6 shows schema of the tables that are used to store data:

E. The REST API (also known as RESTful API)

The Rasberry Pi serves as the back-end server and database. The database used in this project is SQLite DB. The Back-end architecture as shown in Fig. 7 is based on REST APIs and is built using the Flask micro-web framework. Flask code is written in Python Programming Language. We chose the REST API-based back-end because of its flexibility in acquiring data through HTTP requests over the internet. Due to the increasing number of IoT devices of all forms, from complex smartphones to small embedded devices, all can send and receive data through HTTP requests. Responses from the

server are sent as JSON (JavaScript Object Notation) string, which is parsable by all devices to retrieve data.

1) Processing HTTP Requests at Back-end Server (server.py): The back-end processes all HTTP requests in three steps by first associating the HTTP request with a class based on the URL, acquiring data from DB and processing the data accordingly, and converting the output into a JSON response and sending it to the requested device.

In the first step of associating the HTTP request with a class based on the URL, there are three types of URL in our project: the dashboard URL, chilli plant URL, and tomato plant URL. The dashboard URL gets the latest moisture value of the chilli plant, tomato plant, and the current pump state. The chilli plant URL is responsible for retrieving the moisture value of chilli based on the time of day(hours) over a multiple-day period. The tomato plant URL is responsible for retrieving the moisture value of tomato based on the time of day(hours) over a multiple-day period. The chilli plant URL and tomato plant URL are used to get moisture data, which can be plotted as graphs.

The second step is processing the data based on the request, which can be of three classes: the dashboard class, chilli plant class, and tomato plant class. The Dashboard class queries and obtains the latest readings of the chilli and tomato plants. If the moisture value of chilli or tomato is less than 50, then it sets the pump state ON, else sets the pump state OFF. The chilli plant class queries and obtains all the chilli table entries from the DB and processes the raw data. The chilli plant logs readings every two minutes, so the readings are averaged based on the time of day to obtain the average moisture value for the specific hour of the day. The tomato plant class queries and obtains all the tomato table entries from the DB and processes the raw data. The tomato plant logs readings every two minutes, so the readings are averaged based on the time of day to obtain the average moisture value for the specific hour of the day.

The third and final step is converting the results into a JSON response. The outputs from the above three classes are converted into JSON responses before sending them to the requested device. JSON is a lightweight data-interchange format used for exchanging data over the HTTP protocol. The JSON response is a string representation as shown in Fig. 9 and is preferred because it is easy for humans to read and machines to process.

We can see in Fig. 8 that the application made a dashboard request, a chilli request, and a tomato request. The dashboard request returns the current moisture values and state of pumps to the Android Application. The chilli request queries all the chilli entries and returns the average moisture value based on the hour of the day for all dates. The tomato request queries all the tomato entries and returns the average moisture value based on the hour of the day for all dates. All the responses are in JSON format as seen in the Fig. 9.

2) The SQLite Database: The database used here is the SQLite database. The database is currently deployed on Raspberry Pi. The database is responsible for logging and storing all the data sent by the plant nodes to the Raspberry Pi. The database contains separate tables for chilli plant, tomato plant, and pump states. The back-end queries the database to obtain the required data based on the user request.

V. Components Description: Android App

The architecture of the Android application to obtain data from the back-end server can be seen in Fig. 10. The Android application sends an HTTP GET request to the server and receives a JSON response, which is then parsed and displayed. The HTTP GET request is determined by the current page and the values needed by the page. The server sends the requested data to the application in JSON format. Then the app parses the JSON response to obtain values and display them in the app.

The Android application consists of three pages: the dashboard, the chilli plant graph, the tomato plant graph.

A. The Dashboard

The dashboard page can be seen in Fig. 11. The dashboard page shows the current moisture level of the chilli plant and tomato plant. At the top of the page, we have to enter the link to the back-end server. After that, we can see the current moisture level of the chilli plant and tomato plant. The red color background of moisture value denotes the moisture level is low, and the green color background of moisture value denotes the moisture level is high. We can see at the bottom of the page the UPDATE button, which is responsible for updating the latest moisture values. When the UPDATE button is pressed, the page is updated with the latest value from the server. Clicking the chilli plant takes us to the chilli plant graph page, and clicking the tomato plant takes us to the tomato plant graph page.

B. The Chilli plant Page

The chilli plant page can be seen in Fig. 12. The graph at the top of the page shows the moisture values of the chilli plant. The horizontal axis of the graph represents the date with time, and the vertical axis represents the moisture level. The graph plots the moisture value for each hour of the day. In the graph, we can see the moisture value of the chilli plant over multiple days. Below the graph, we can find the current pump state as ON or OFF. At the bottom of the page, we can find the toggle button to turn the pump ON or OFF. Upon pressing the TURN ON button, an HTTP request is sent to the server and the pump is turned on. When the moisture level of the plant increases we turn OFF the pump.

C. The Tomato plant Page

The tomato plant page can be seen in Fig. 13. The graph at the top of the page shows the moisture values of the tomato plant. The horizontal axis of the graph represents the date with time, and the vertical axis represents the moisture level. The graph plots the moisture value for each hour of the day. In the graph, we can see the moisture value of the tomato plant over multiple days. Below the graph, we can find the current pump

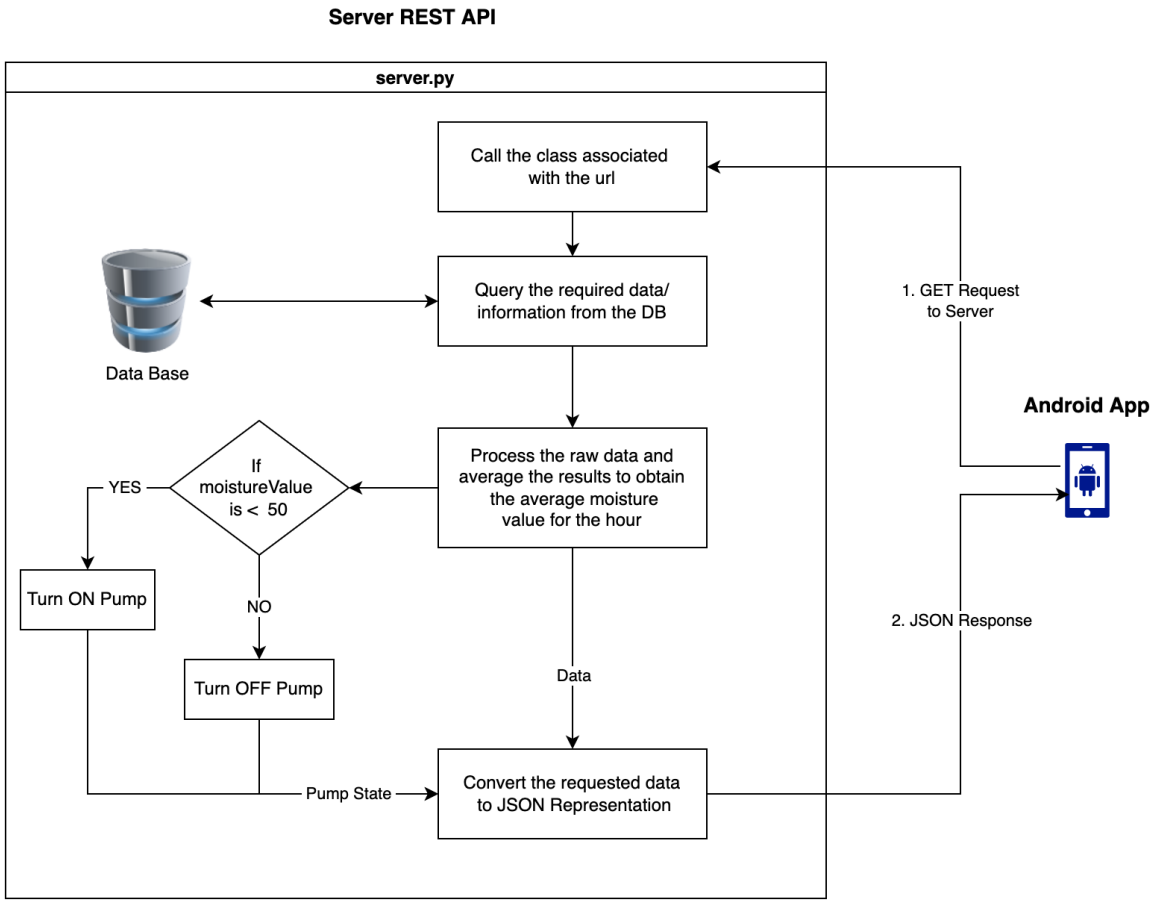


Fig. 7. The Back-end Server Architecture.

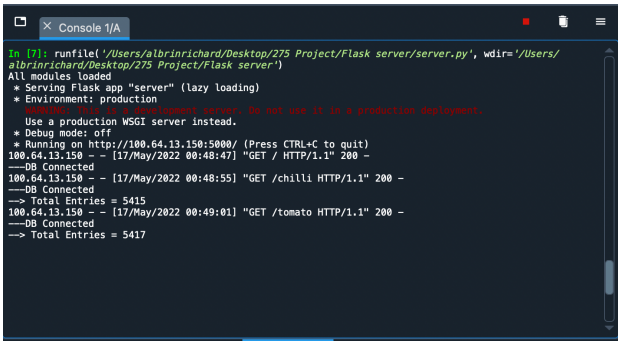


Fig. 8. The server console response when the Application makes a GET request.

Fig. 9. The JSON Response from back-end server.

state as ON or OFF. At the bottom of the page, we can find the toggle button to turn the pump ON or OFF. Upon pressing the TURN ON button, an HTTP request is sent to the server and the pump is turned on. When the moisture level of the plant increases we turn OFF the pump.

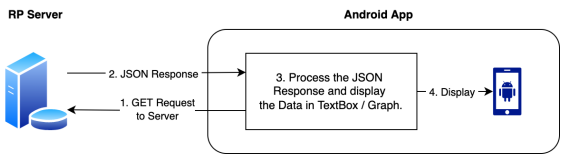


Fig. 10. The Application Architecture.

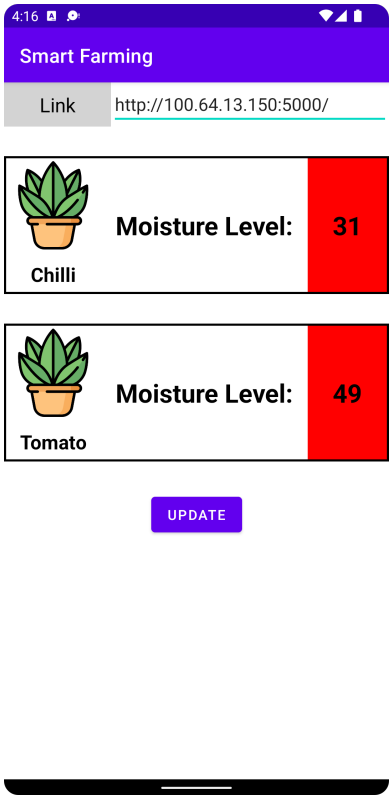


Fig. 11. The Dashboard of the Android App.

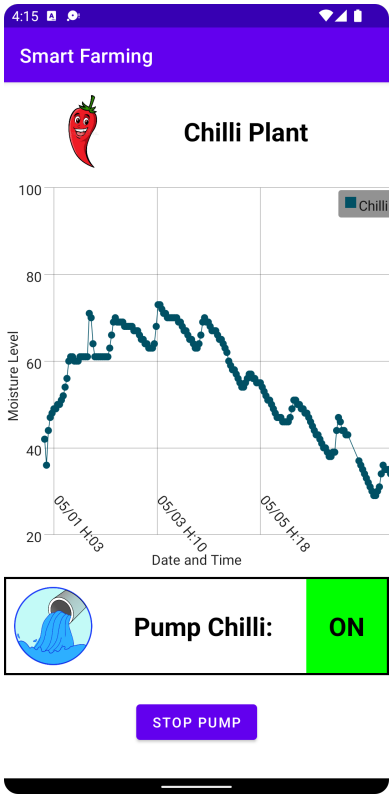


Fig. 12. The Chilli plant moisture value Graph.

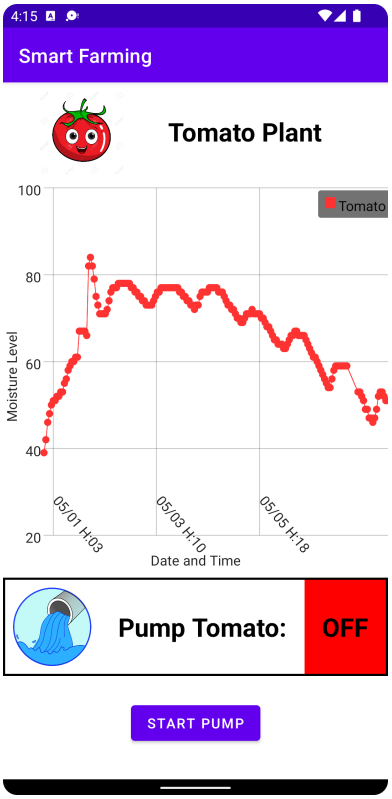


Fig. 13. The Tomato plant moisture value Graph.

VI. Web Dashboard

As described in Webserver subsection(IV-C), we have created a web dashboard for the convenience of the user, through which they can monitor the moisture levels and turn on/off pumps.

A. Current moisture level

The dashboard is split vertically into two divisions, each division contains sensor reading from one type of plant, in out project we used tomato and chilli plants, and placed them in two different pots.The top most ”meter” shows the current moisture level collected by the sensors.

B. PUMP control

Under the meter that indicated the moisture level you can see the Turn on pump button, this button can be used to turn on/off pump. Once the pump is turned on it will automatically gets turned off in 60 seconds, and the timer on the button indicates that.

C. Overall moisture trend

Represents the overall moisture level, from the day the we started reading projects.

D. Daily moisture trend

Presents the hourly moisture level data on a daily basis.

A video recording of the web dashboard and its working has been recorded and uploaded to youtube, a link of it is mentioned in the System Testing section.

VII. System Testing

From the start to the end of the project we have tested each and every component, and fixed bugs/issues identified along the way.

Final working project demo can be found here: https://www.youtube.com/watch?v=5ltGwt-I5Is

VIII. Code

We checked out code on to a github repository and the link is here :

https://github.com/AnastasiyaChilukuri/smart-farming-with-raspberryPi/

IX. Limitations and Future Work

Due to the limitation of time we employed one type of sensor, but to track the condition of our farm there are other potentially useful sensors, as such: temperature sensor, sensor of water level for water reservoir, sensor of light consumption. Apart from that, to make sure that the owner of Smart Farm can track their farm condition in real time we could employ camera supervision and movement detectors for safety and unwelcome guests (like rodents, rabbits, etc.) detection.

Conditioning facilities also can be improved. At this moment only pump is used, but we can also install growth lamps with remote control of light emission, and also water distribution systems, and sprinklers.

Constructing our solution for the Smart Farming the individual plant-based conditions were not taken into account. The future work in this direction is to generate database with individual special requirements for different plant species (e.t. tomato, cucumber, cabbage, etc.) and automate the work of the conditioning facilities (currently pump, in future temperature conditioning, sprinklers) to work as per the need of a specific plant.

X. Conclusion

Thus in this project, we implemented a RasberryPi based smart farming system. The Raspberry Pi serves as the backend server for hosting the web server and database. Each plant has a nodemcu connected with a moisture sensor that sends values to the Raspberry Pi. The moisture values are processed on the server and sent to the website and Android app. On both the Website and the Application, we can see the current moisture level of the plants. We can turn the pump on and off via the website and application. Plant moisture values can be seen as a graph plotted by hours and moisture values over days. Automating these processes with IoT will allow us to monitor the plants more effectively and have better results on farming with fewer interventions

References

[1] Wikipedia, *MQTT*. Wikimedia Foundation, 2019.