CSC453: Internet of Things
April 25th, 2021

Parker Killian
Justin Wald
Michael Barger

Individual Contributions

Component	Weight	Parker Killian	Justin Wald	Michael Barger
Architecture Design	0.1	33.3%	33.3%	33.3%
Frontend Design	0.05	10%	80%	10%
Wiring Design	0.05	20%	40%	40%
Hardware Assembly	0.05	33.3%	33.3%	33.3%
Web Server	0.15	50%	30%	20%
Garden Monitor	0.15	50%	25%	25%
Garden Manager	0.15	25%	65%	10%
Debugging	0.1	10%	10%	80%
Presentation	0.05	80%	10%	10%
Report	0.1	20%	40%	40%
Documentation	0.05	10%	10%	80%
Total Contributions		32.745%	34.995%	32.245%

Introduction

When attempting to grow an herb garden, it can be difficult to remember to water your plants or to keep the soil humidity levels at an appropriate level. The objective of this project is to simplify this process by Internet of Things automation

The goal of this project was to create an automated garden management system that uses an array of sensors and water pumps to monitor the environment and automatically water the plants in an herb garden. This system would provide detailed information to the user of the status of their plants as well as allowing for a large amount of user customization to tailor the system to the user’s plants’ needs

Design

Architecture

IoT Herb Garden is comprised of 3 primary python scripts:

Garden Manager	Garden Monitor	Garden Web Server
❖ Create database schema objects. ❖ Log sensor and event data to the database when received from the MQTT broker. ❖ Provide auto-watering functionality. ❖ Inform other components of configuration info read from the database.	❖ Configure defaults for sensors. ❖ Collect and publish sensor data. ❖ Activate water pumps when instructed to do so.	❖ Provide a web UI to interact with the Herb Garden ❖ Display current metrics for sensors ❖ Display a historic view of sensor values ❖ Allow configuration of plant and sensor objects ❖ Manually control pumps

Table 1: Script Overview

The separation of these three scripts gives the user flexibility in the hardware running each. The monitor script was kept lightweight to facilitate operation on an edge device such as a Raspberry Pi. Manager communicates with the monitor entirely through MQTT which means that it can run on an entirely different host, as long as they share the same broker. Web server does need access to the database to operate which ties it to running on the same host as manager since we used SQLite as our backend but with a little modification, web server could also run on its own host. Containerizing the entire system would be quite trivial thanks to this separation.

One other advantage of this architecture is that the more critical scripts, manager and monitor, could operate in “headless” mode without the web server running, only logging data to the DB. This also means that if the web server crashes for whatever reason, logging will continue.

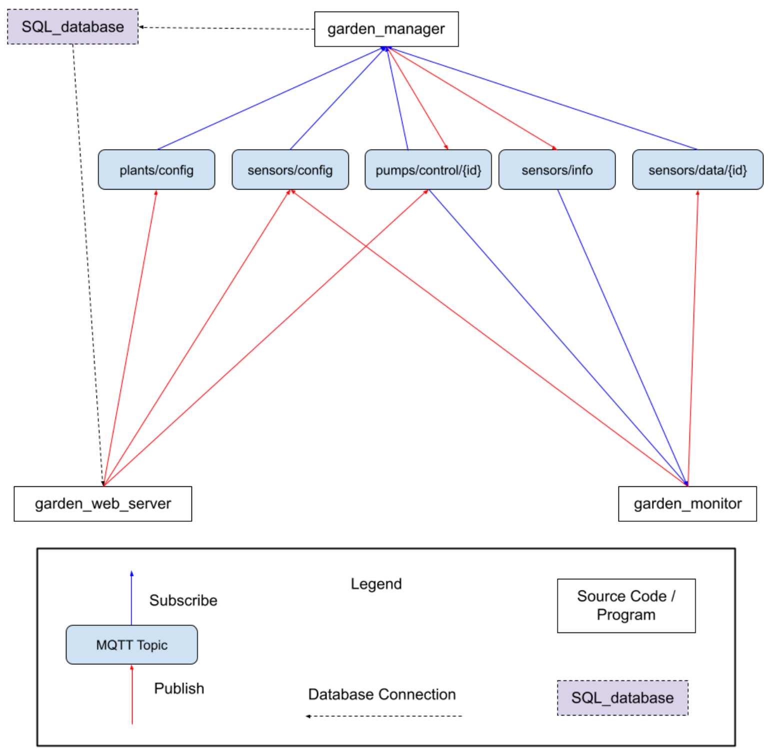


Figure 1: MQTT architecture between the scripts

MQTT

Communication between the different parts of our program is done primarily through MQTT, leveraging an outside broker. Below is a diagram showing the interaction between each component through MQTT messages. Each message contains a JSON payload (described in Appendix 1) to communicate the body of the messages.

A typical flow for the program is shown in figure 2 and follows these steps:

1. Monitor starts and begins logging sensor data with a default configuration file loaded from a static JSON File.
2. Manager reads any configuration data from the database and publishes this data to the sensors/info topic.
3. Monitor, which is subscribed to this topic, receives the information and compares it against the default values it has.
    a. If there are any differences, the information from the manager is preferred.
    b. If there are no sensors in the message published by the manager, such as in the case of a reset database, monitor publishes its default values to sensors/config, one at a time. Manager reads these messages and adds them to the sensors/info message.
4. Monitor begins sending sample messages to sensors/data.
5. Manager, subscribed to sensors/data, writes these messages to the database.
6. Manager also monitors sensor values to determine if watering is needed.

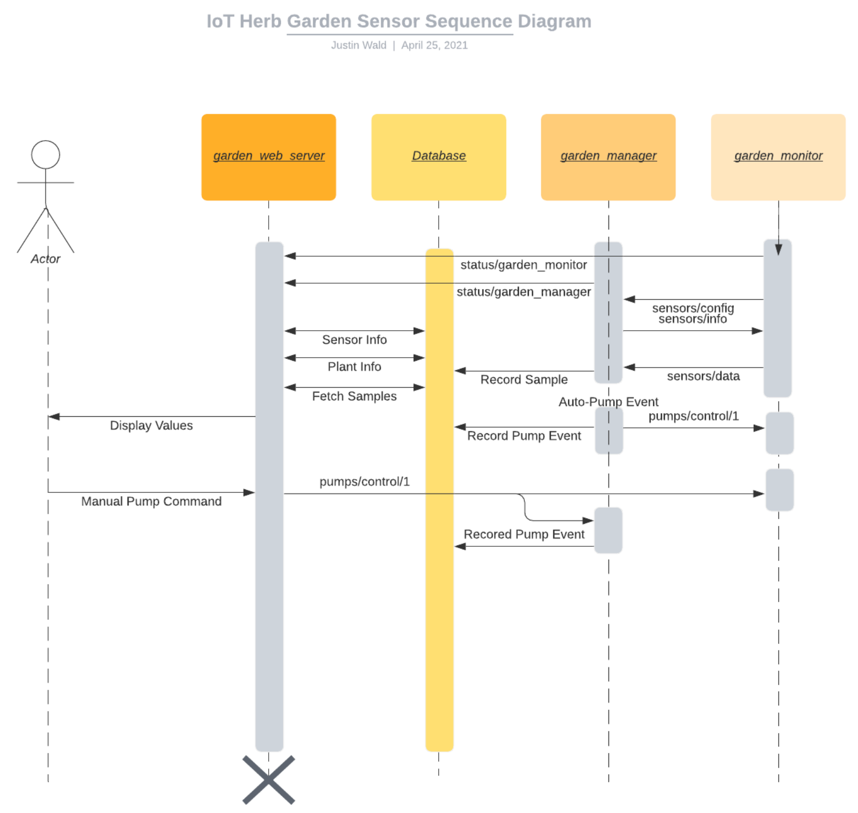


Figure 2: Sequence diagram of a basic flow through the system.

Database

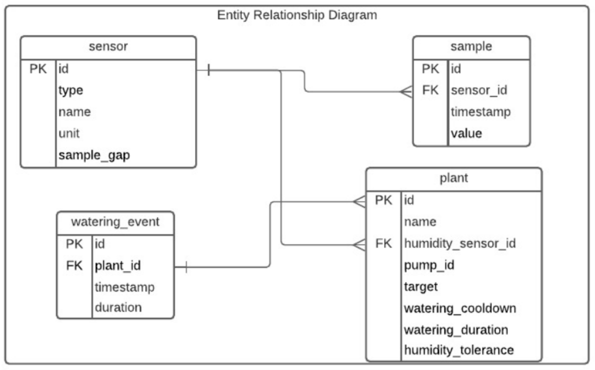


Figure 3: ERD diagram for the database.

The database is based on a SQLite backend which we accessed via SQLAlchemy for consistency and reliability. The database was used primarily to record data so that historical viewing was possible but it also enabled persistent settings to be specified by the user about the elements of the system.

Web Server

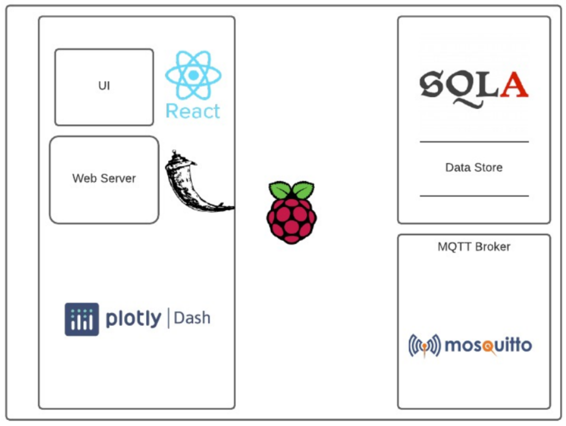


Figure 4: Technology overview.

The web server uses the Dash library which makes use of the highly popular Flask library for its web server operations while providing a robust React-based UI toolkit that includes the Plotly graphing library.

Implementation

Our project runs on Python, using many public libraries for specific operations including SQLAlchemy, Dash, Dash Bootstrap Components, Pandas, Paho MQTT, and GPIOZero.

The hardware for our project included:

- Raspberry Pi 3b+
- A breadboard
- A prototyping board
- 4 capacitive soil humidity sensors
- 4 5V impeller water pumps
- 4 PCB-mounted 5V relays
- A DHT22 (thermistor / hygrometer combo)
- A photoresistor
- An 8 channel ADC
- 22 Awg wires

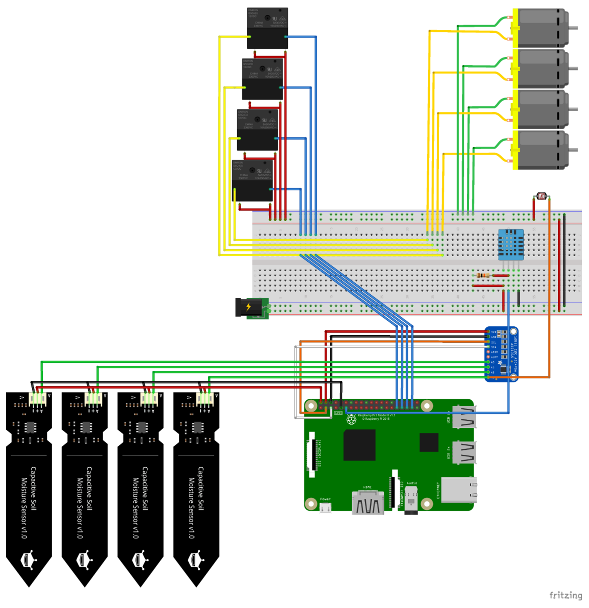


Figure 4: Wiring Schematic

To run the IoT_Herb_Garden system, it is necessary to run 3 programs, garden_manager.py, garden_monitor.py, and garden_web_server.py. While it is necessary for the garden monitor to run on the Raspberry Pi, it is recommended that manager and web server both run on a more powerful computer to increase performance.

garden_manager.py

Garden manager acts as the brains of the system. It handles recording MQTT messages to the database and automatically triggering pump operations based on sensor data.

garden_monitor.py

Garden monitor runs all of the sensors, pumps, and hardware for the project. On startup, and when a message is received from “sensors/info”, it updates its local values for sensor sample rates. Its sensors then sample based on their set sample rate, and the values are published to the MQTT topic “sensors/data/{id}”. When it receives a message from “pumps/control/{id}”, a locking thread is created that runs the pump for the provided duration. If multiple pump operations are received in quick succession, the locking threads ensure that only one pump runs at a time, limiting the current draw on the Raspberry Pi.

garden_web_server.py

Garden web server works in tandem with **app.py** and the files in **apps/** to create and run the website for the IoT Herb Garden system. The Overview and History pages provide graphical views of the sensor and pump history, while Controls and Configuration allow for user interaction with the system. Through the Controls page, users can provide pump durations to send manual pump operations through MQTT over the “pumps/control/{id}” topic. These messages are stored in the database and trigger the indicated pumps to operate for the given duration. The Configuration page allows for the user to change the settings of each parameter of the Plants and Sensors in the system. The parameters affect the displays on the Overview and History pages as well as the operating settings of the Raspberry Pi and garden manager. Through the web server, all aspects of the system can be controlled and monitored remotely.

utils/

Each source file in the **utils/** directory is used by at least one of the three main programs. **adc_library.py** is used to interface with our adc. **common.py** contains common functions that can be used system wide. **db_interactions.py** is used primarily by garden_manager to create database entries. **sensors.py** is used primarily by garden_monitor to interface with the other sensors in the system. **ui_elements** is used to create graphics on garden web server pages.

Results and Discussion

While we did not intend too much in the way of analysis, we have noticed some interesting trends in our sensor setup. The above graphic shows the history of sensor values over several hours. With these readings, you can see a very clear relationship between ambient temperature and ambient humidity. As temperature increases, the air is able to hold more humidity which decreases the relative humidity. Notably, these changes don't have a significant short-term effect on the humidity of any of the soil humidity sensors. Visible in the history graph at approximately 16:00 is a pump operation. The large spike in soil humidity comes from the water soaking into the soil at the level of the humidity sensor. The gradual settling into a lower level comes from the water soaking in and spreading out in the soil.

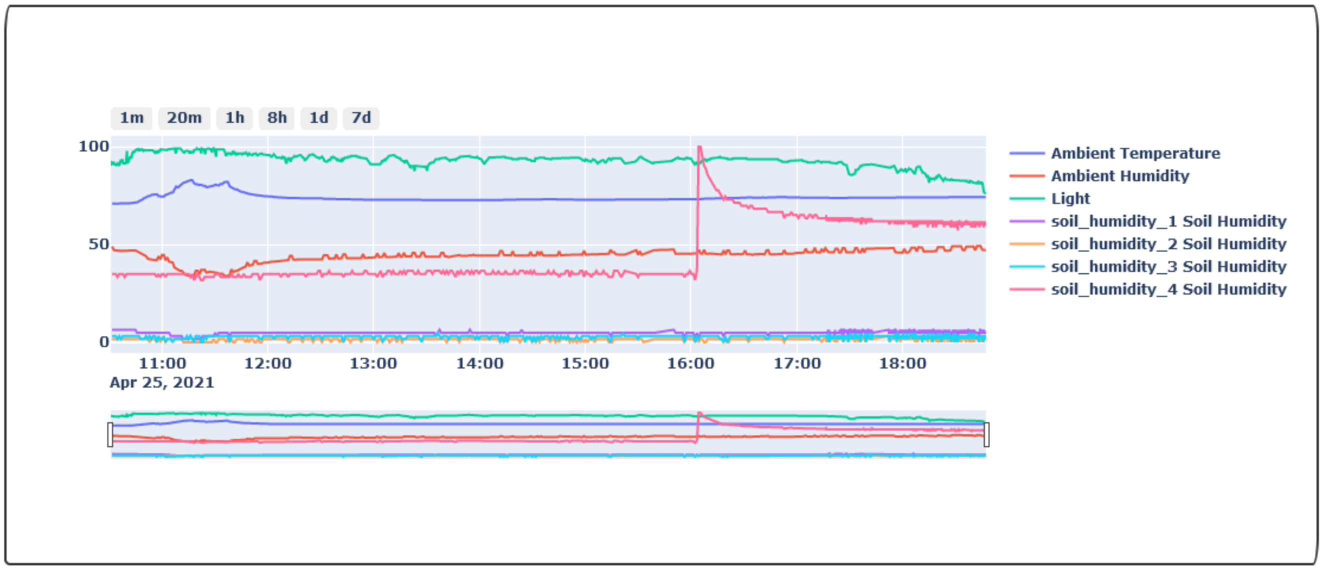


Figure 5: Data from History Page Graph

Appendix 1 - MQTT

status/garden_manager

Retained: True

Format:
`online`

Goodwill_message:
`offline`

status/garden_monitor

Retain: True

Format:
`online`

Goodwill_message:
`offline`

sensors/info

Retained: True

Purpose: Tell the other programs about what sensors are connected to garden_monitor.py

Format:

```
[
    {
        id: <int>,
        type: <string>,
        name: <string>,
        unit: <string>,
        sample_gap: <int>
    },
    {
        id: <int>,
        type: <string>,
        name: <string>,
        unit: <string>,
        sample_gap: <int>
    },
    ...
]
```

sensors/data/{id}

Retained: False

Purpose: Publish as single data sample

Format:

```
{
    timestamp: <ISO>,
    value: <value>
}
```

pumps/control/{pump_id}

Retained: False

Purpose: Instruct garden_monitor to run the pumps for a set duration.

Format:

```
{
    duration: <seconds>
}
```

plants/config

Retained: True

Purpose: Tell garden_manager about the configured plants.

Format:

```
{
    id: <id>
    name: <name>,
    humidity_sensor_id: <sensor_id>,
    pump_id: <pump_id>,
    target: <[0, 100]>,
    watering_cooldown: <[5, 300]>,
    watering_duration: <(0, 5]>,
    humidity_tolerance: <[0, 50]>
}
```

sensors/config

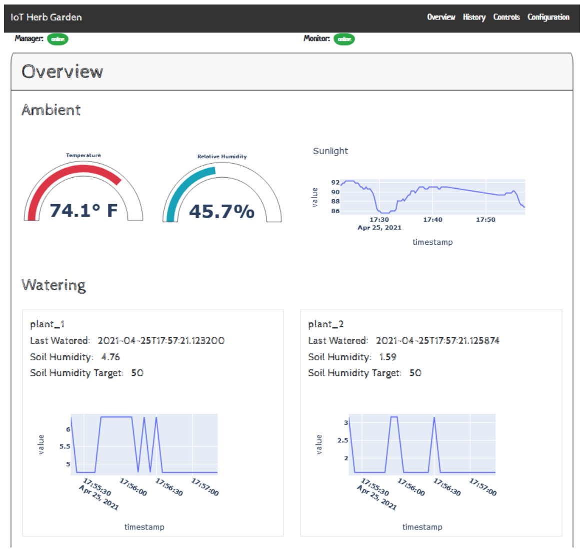
Retained: True

Purpose: Tell garden_manager about the configured plants.

Format:

```
{
    id: <int>,
    type: <string>,
    name: <string>,
    unit: <string>,
    sample_gap: <int>
}
```

Appendix 2 - Web Pages



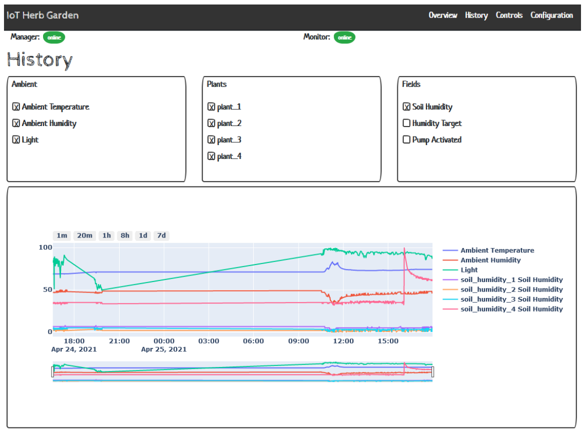
Overview

Ambient

Watering

plant_1
Last Watered: 2021-04-25T17:57:21.123200
Soil Humidity: 4.76
Soil Humidity Target: 50

plant_2
Last Watered: 2021-04-25T17:57:21.125874
Soil Humidity: 1.59
Soil Humidity Target: 50

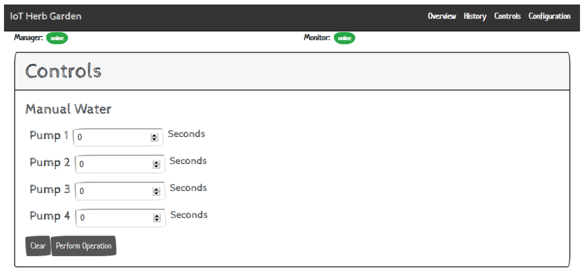


History

Ambient
- ☒ Ambient Temperature
- ☒ Ambient Humidity
- ☒ Light

Plants
- ☒ plant_1
- ☒ plant_2
- ☒ plant_3
- ☒ plant_4

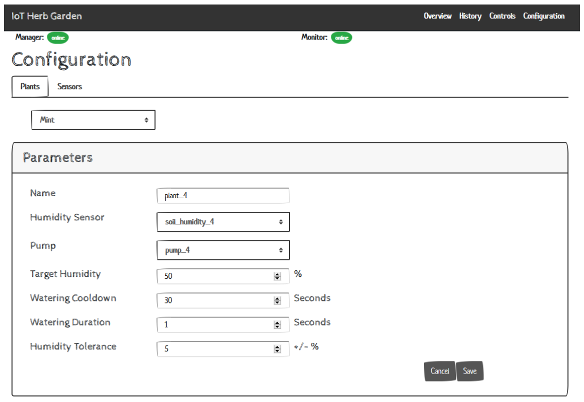
Fields
- ☒ Soil Humidity
- ☐ Humidity Target
- ☐ Pump Activated



Controls

Manual Water

Pump 1	0	Seconds
Pump 2	0	Seconds
Pump 3	0	Seconds
Pump 4	0	Seconds

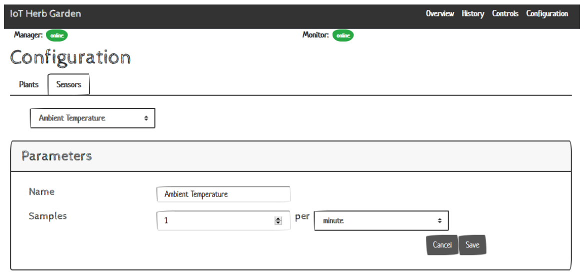


Configuration

Mint

Parameters

Name	plant_4	
Humidity Sensor	soil_humidity_4	
Pump	pump_4	
Target Humidity	50	%
Watering Cooldown	30	Seconds
Watering Duration	1	Seconds
Humidity Tolerance	5	+/- %



Configuration

Ambient Temperature

Parameters

Name	Ambient Temperature		
Samples	1	per	minute