Deakin University

Smart Watering Solution for Houseplants

Project Report

Negin Pakroohjahromi

222393187

Document Version 2.0

This project presents the Smart Watering System for Houseplants, an IoT solution designed to automate plant care by monitoring environmental conditions such as soil moisture and temperature. The system integrates cloud platforms, MQTT communication protocols, and modular software components to offer real-time monitoring, automated watering, and flexible manual controls. The solution uses AWS EC2 for backend services and MongoDB Atlas for cloud-based data storage, ensuring scalability and remote management. The project aims to address the challenges of inconsistent plant care through an intelligent system that can scale with minimal effort and provide alerts to users about plant conditions. This report outlines the system's architecture, implementation phases, and challenges encountered, along with the solutions applied. Screenshots and diagrams are provided to illustrate the system's functionality and design.

Problem Statement and Objectives

Houseplants require consistent care and proper watering schedules, but keeping track of soil moisture manually can be challenging, especially when the user is away from home. Inconsistent care, such as over-watering or under-watering, can affect plant health. To address these challenges, the project aims to develop a scalable IoT-based system capable of:

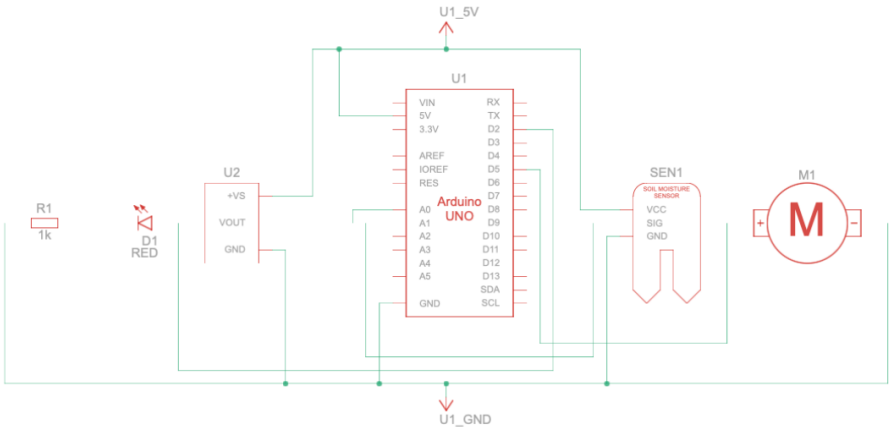
- Monitoring soil moisture and temperature in real-time.
- Providing manual and automatic control options for the water pump.
- Storing sensor data in the cloud for historical analysis and decision-making.
- Alerting users via email whenever the pump is activated.
- Scaling seamlessly to support multiple plants and sensors.

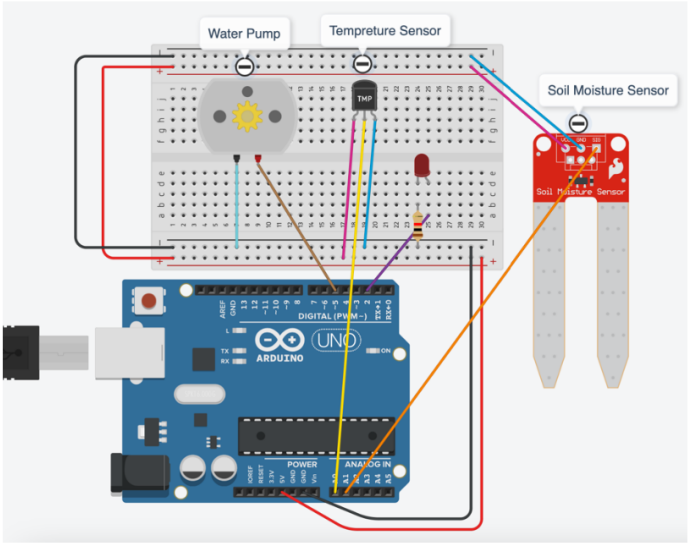
System Design and Architecture

The Smart Watering System for Houseplants is built using a combination of hardware sensors, a cloud-hosted backend, real-time communication protocols, and orchestration through Node-RED. This modular design allows it to expand easily with additional plants and sensors as needed.

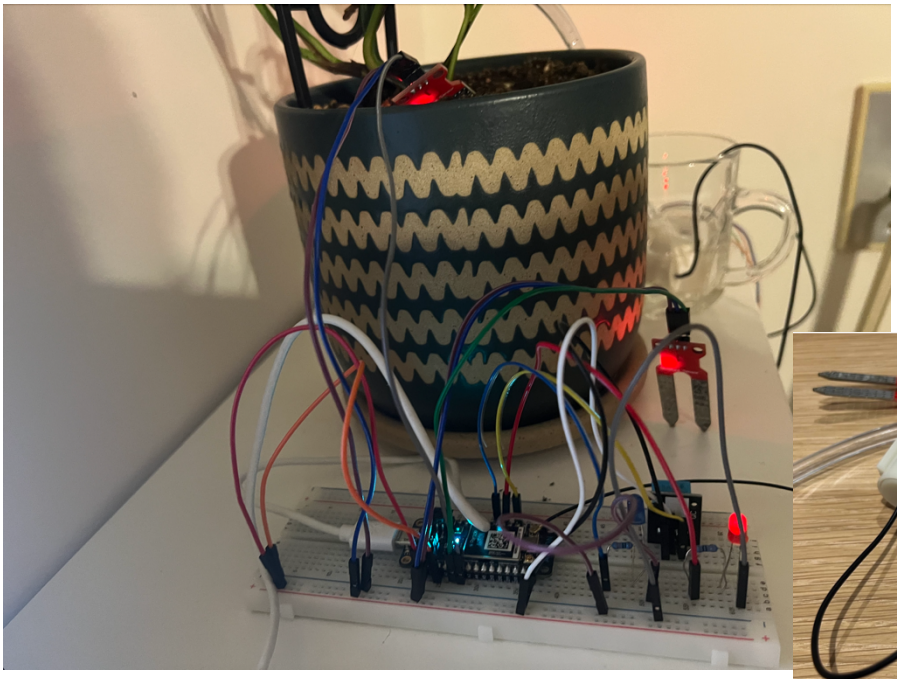
Hardware Components

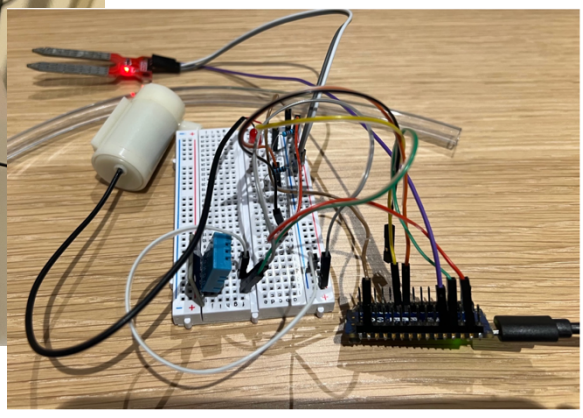
1. Arduino Nano 33 IoT – Collects soil moisture and temperature data from sensors and controls the pump.
2. DHT11 Sensor – Measures environmental temperature.
3. Soil Moisture Sensor – Monitors the soil's moisture level to determine if watering is required.
4. Pump and LED – The pump waters the plants, while the LED provides feedback on the system's status.





If DH22, connect it to Digital pin 2!





Software Components

1. Node-RED – Handles real-time data flow between the sensors, MQTT, backend, and cloud services.
2. MQTT Broker – Facilitates communication between the Arduino and backend.
3. MongoDB Atlas – Provides cloud-based storage for sensor data and plant watering schedules.
4. AWS EC2 – Hosts the backend services and provides API endpoints for manual and automatic control.
5. Postman – Simulates HTTP requests to test manual control modes for the pump.
6. Email Notifications – Alerts users when the pump is activated.

Implementation Phases

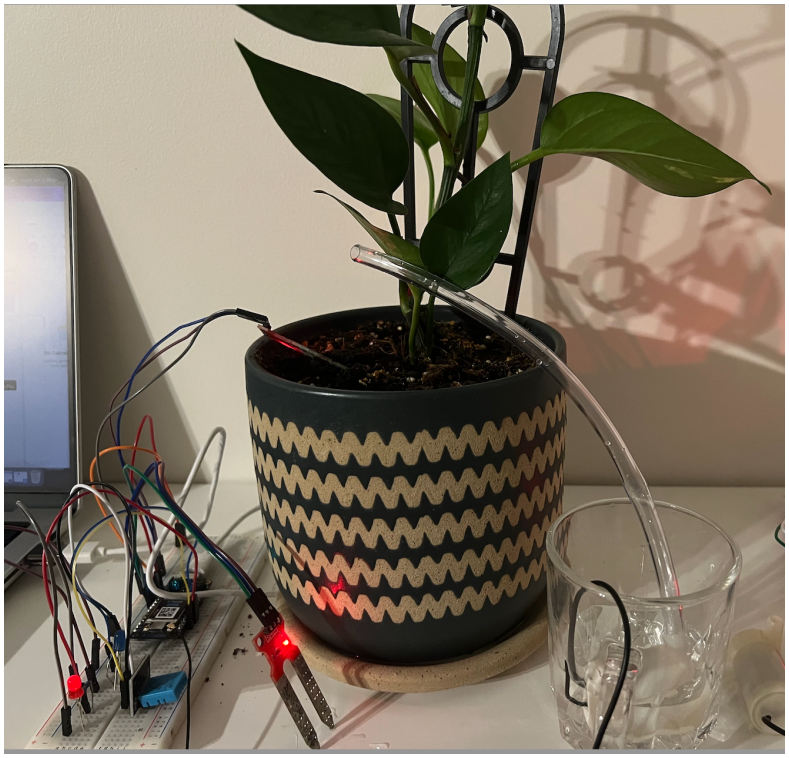
The project was implemented in several phases, each focusing on integrating a key component of the system.

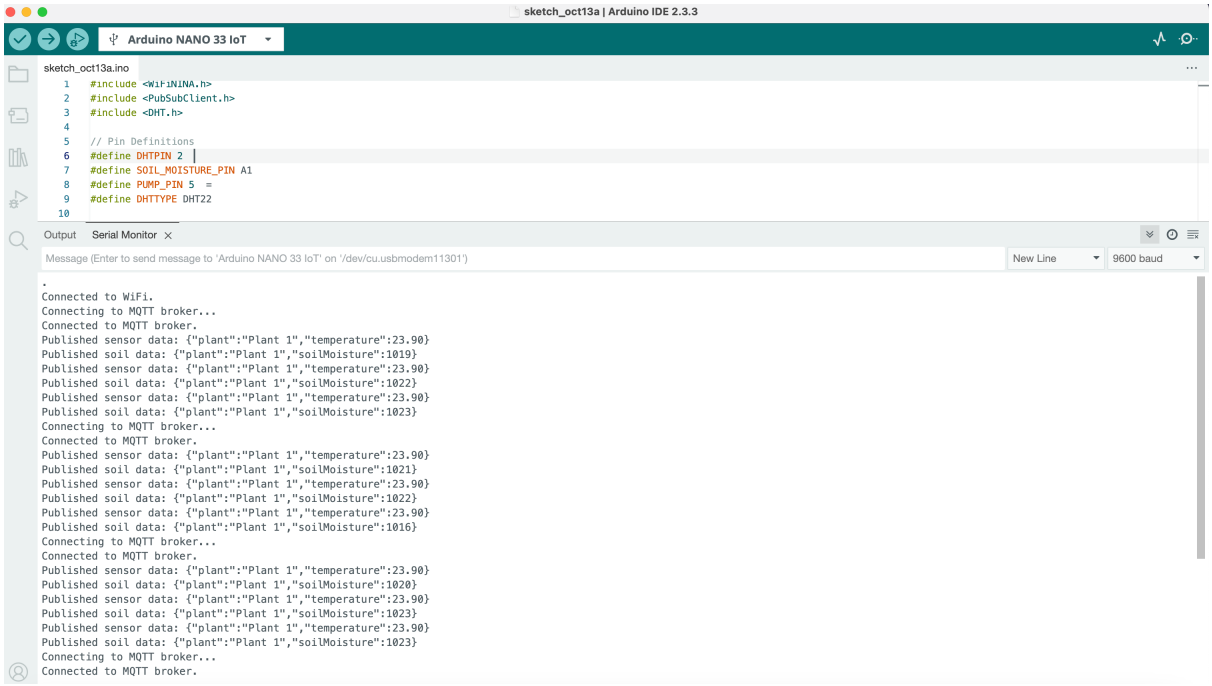
Phase 1: Sensor Setup and MQTT Communication

In the first phase, the Arduino Nano 33 IoT was configured to read temperature and soil moisture data. Using the WiFiNINA library, the Arduino published data to an MQTT topic (plant/sensor) every few seconds. Reconnection logic was added to the code to ensure reliable Wi-Fi connectivity.

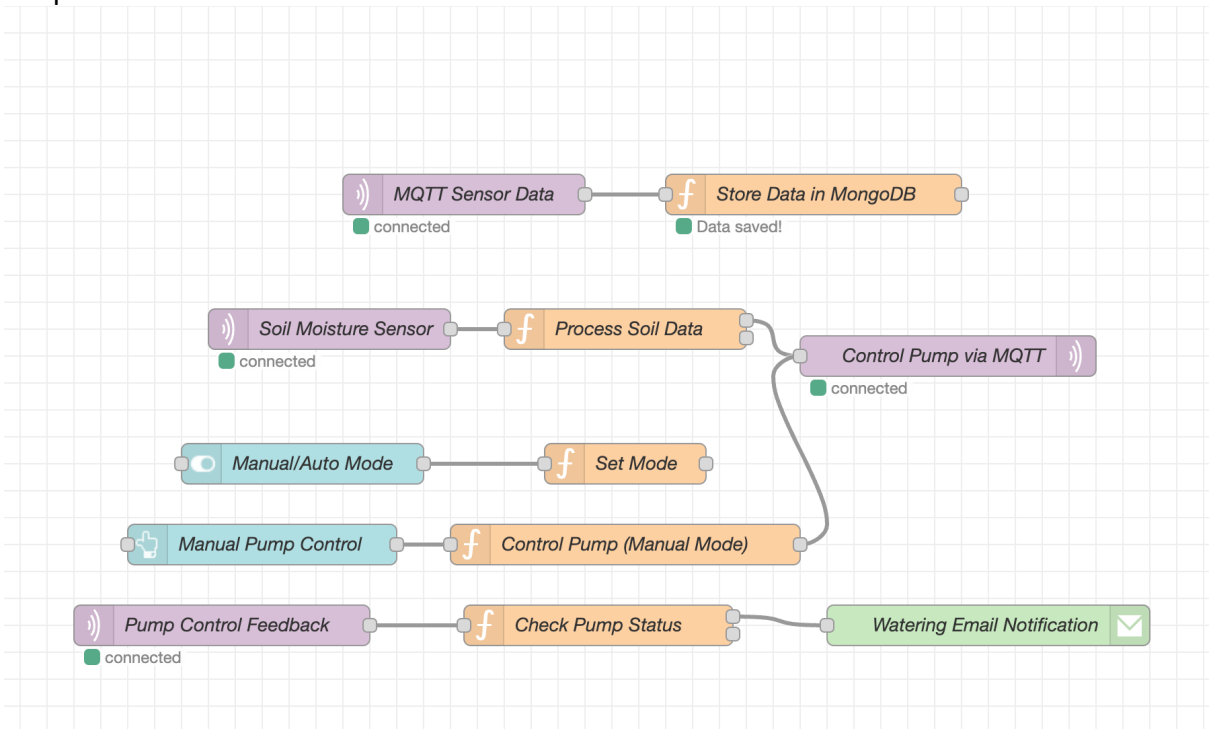
Outcome:

- The Arduino successfully transmitted sensor data to the MQTT broker running on the local network. Node-RED captured the MQTT messages and displayed the data.

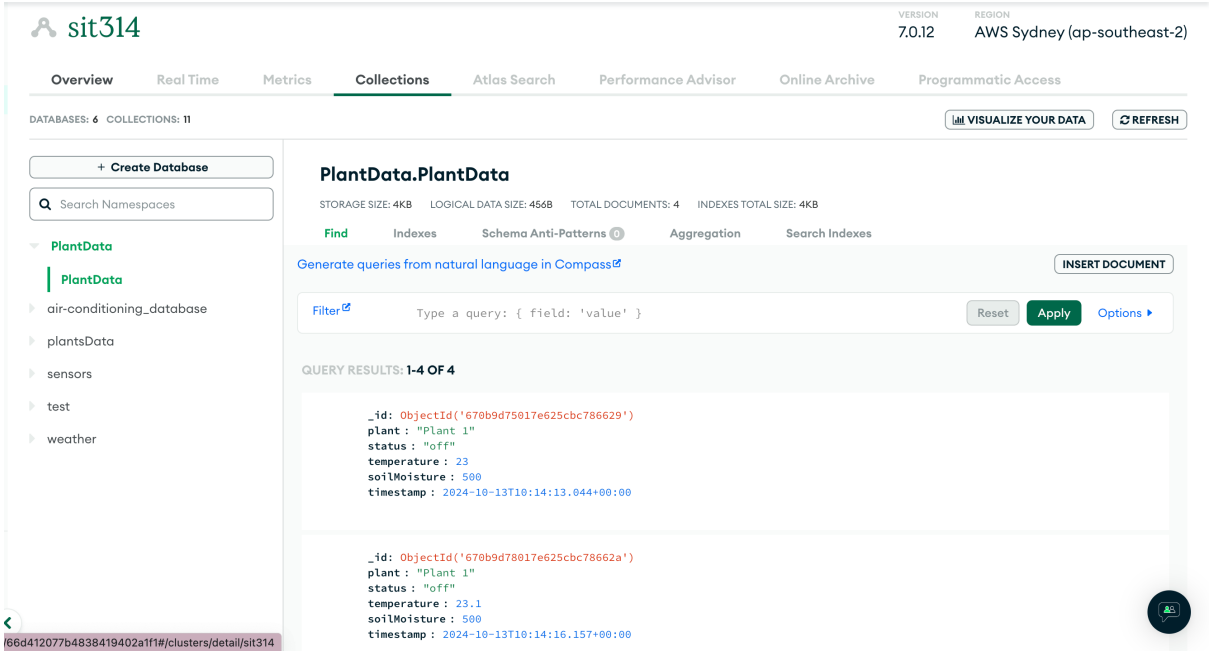




Please refer to the following phase 3 for node-red setup!
Output:



As you can see, the data is saved to the database!



Example of data saved:

```
_id: ObjectId('670b9d7d017e625cbc78662b')
plant : "Plant 1"
status : "off"
temperature : 23.2
soilMoisture : 500
timestamp : 2024-10-13T10:14:21.145+00:00
```

```
_id: ObjectId('670b9d82017e625cbc78662c')
plant : "Plant 1"
status : "off"
temperature : 23.3
soilMoisture : 500
timestamp : 2024-10-13T10:14:26.213+00:00
```

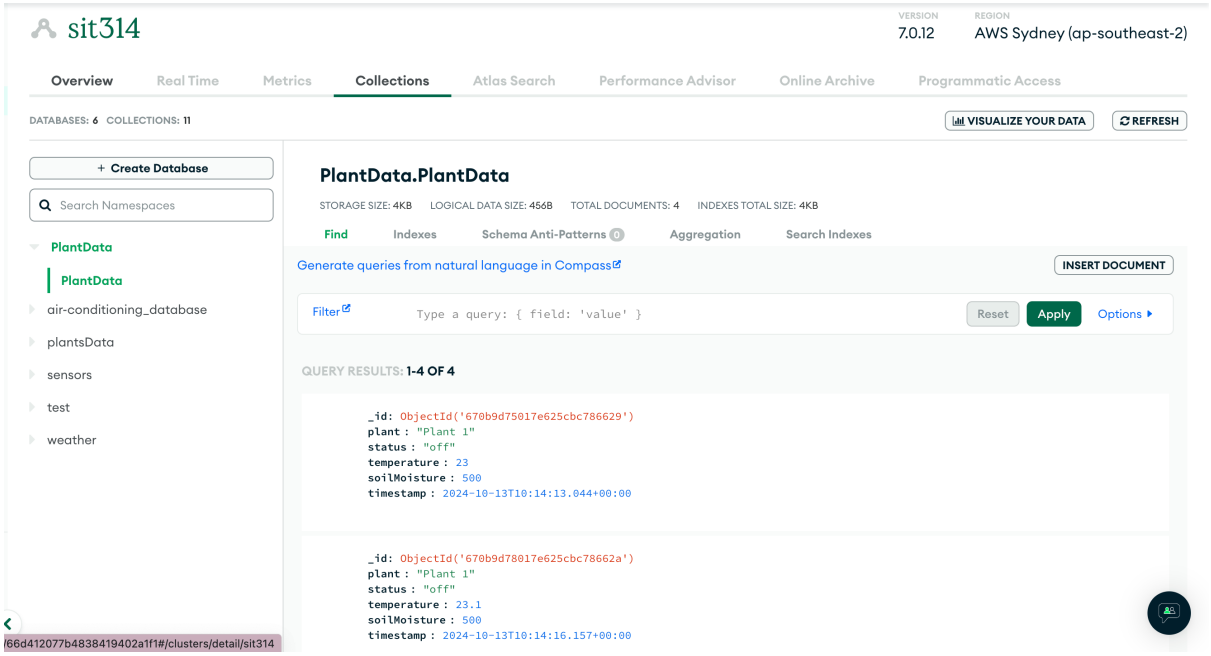
We can continue this setup for different plants.
As I was limited to a soil moisture sensor, plant A-E + Plant 2 were added manually to the database.

Phase 2: Backend Development on AWS EC2 and MongoDB Atlas
The second phase focused on developing a backend using Node.js and deploying it on an AWS EC2 instance. This backend provided API endpoints for:

- Switching between manual and automatic modes.
- Controlling the pump remotely.
- Storing sensor data in MongoDB Atlas.

The backend interacted with MongoDB Atlas to store and retrieve data. MongoDB provided persistent storage, ensuring that all plant data and watering schedules were available for analysis.
Outcome:

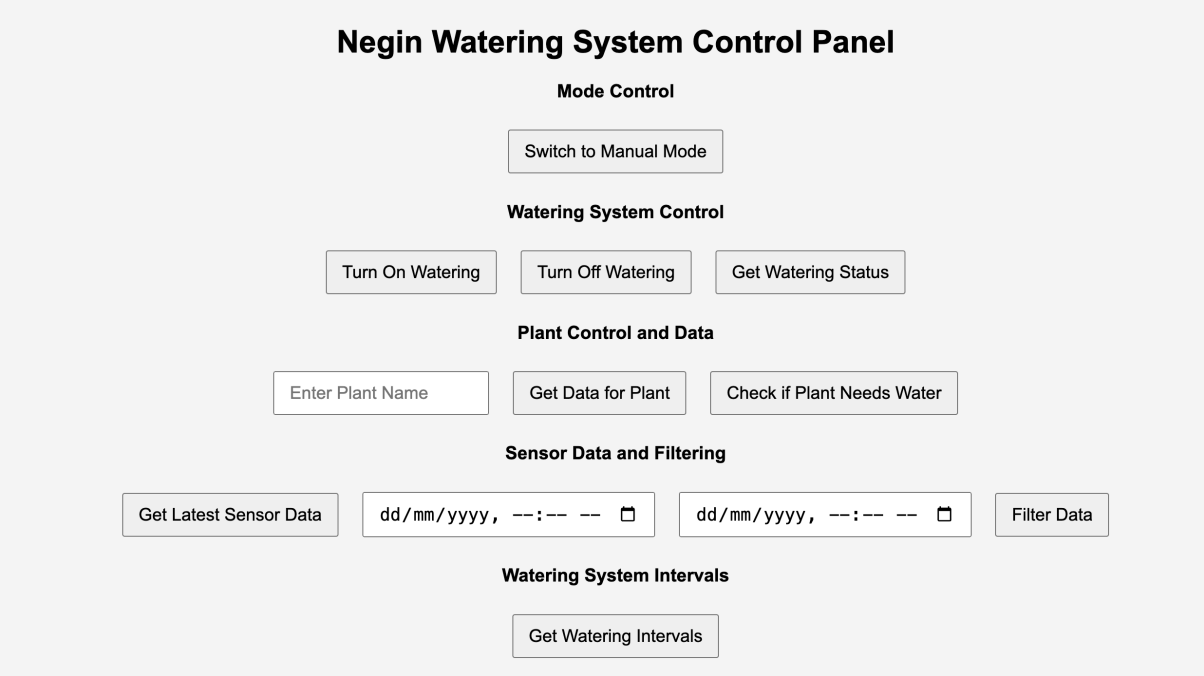
- The backend responded accurately to HTTP requests via Postman and stored data in MongoDB Atlas without issues.



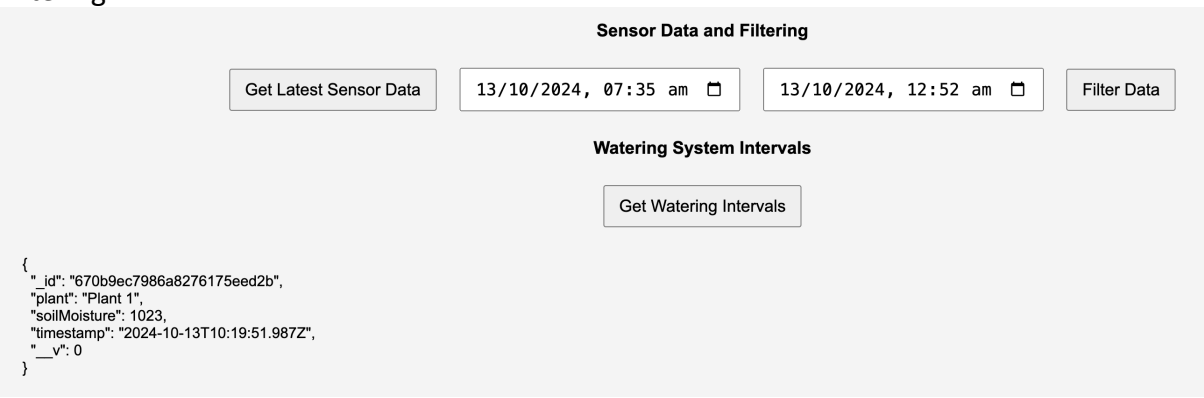
In the backend part (index.js):

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Server running on http://localhost:3000
Connected to MQTT Broker
Subscribed to sensor topics
Sensor data saved: { plant: 'Plant 1', temperature: 23.9 }
Sensor data saved: { plant: 'Plant 1', soilMoisture: 1019 }
Sensor data saved: { plant: 'Plant 1', temperature: 23.9 }
Sensor data saved: { plant: 'Plant 1', soilMoisture: 1021 }
Sensor data saved: { plant: 'Plant 1', temperature: 23.9 }
Sensor data saved: { plant: 'Plant 1', soilMoisture: 1020 }
```

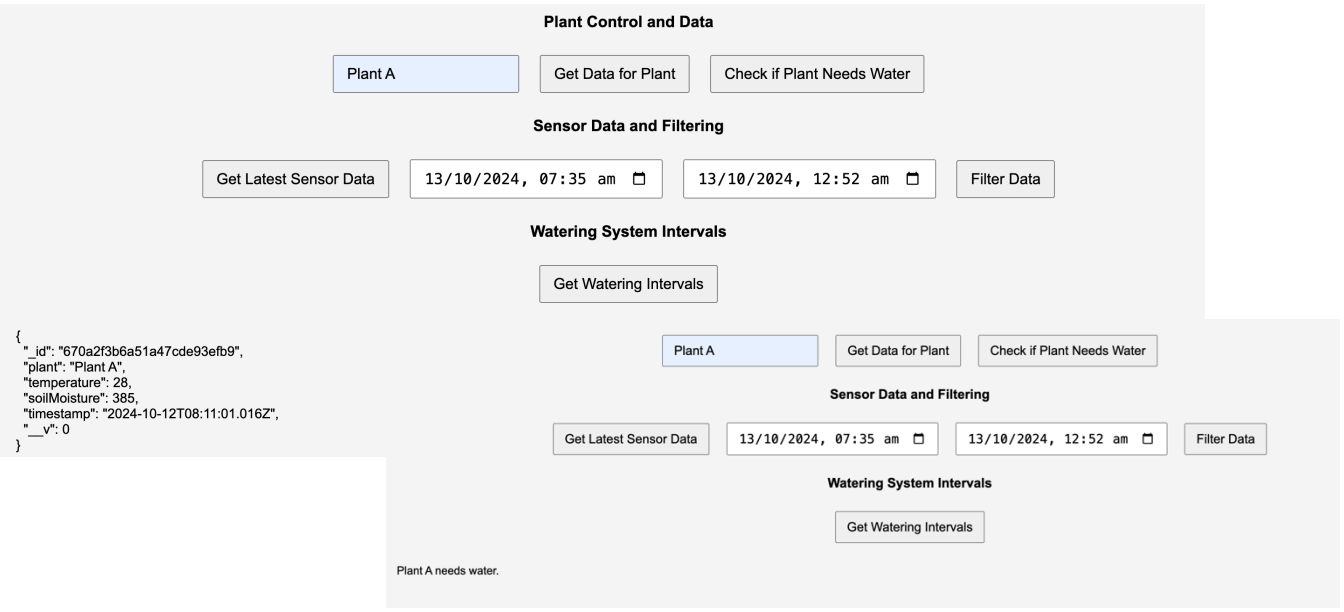
Some random data (plant A, etc.) was added to the database for testing:

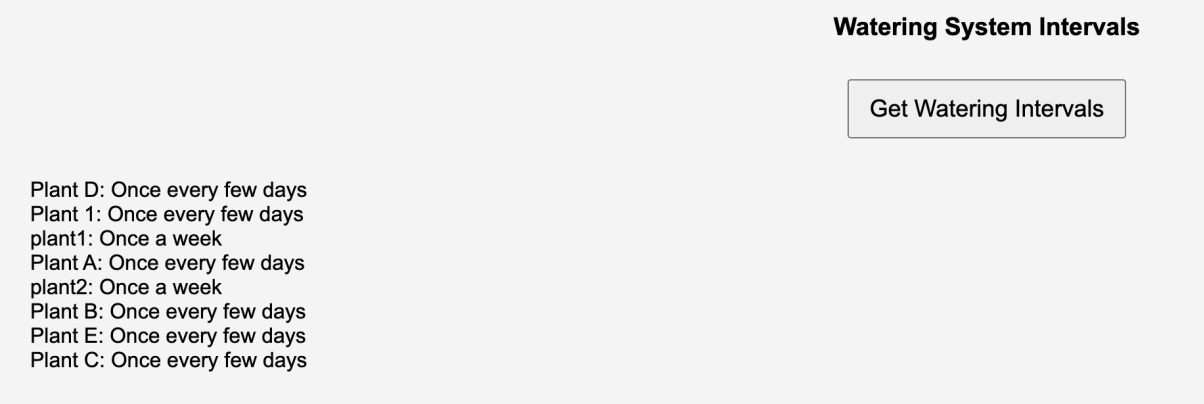


Filtering:

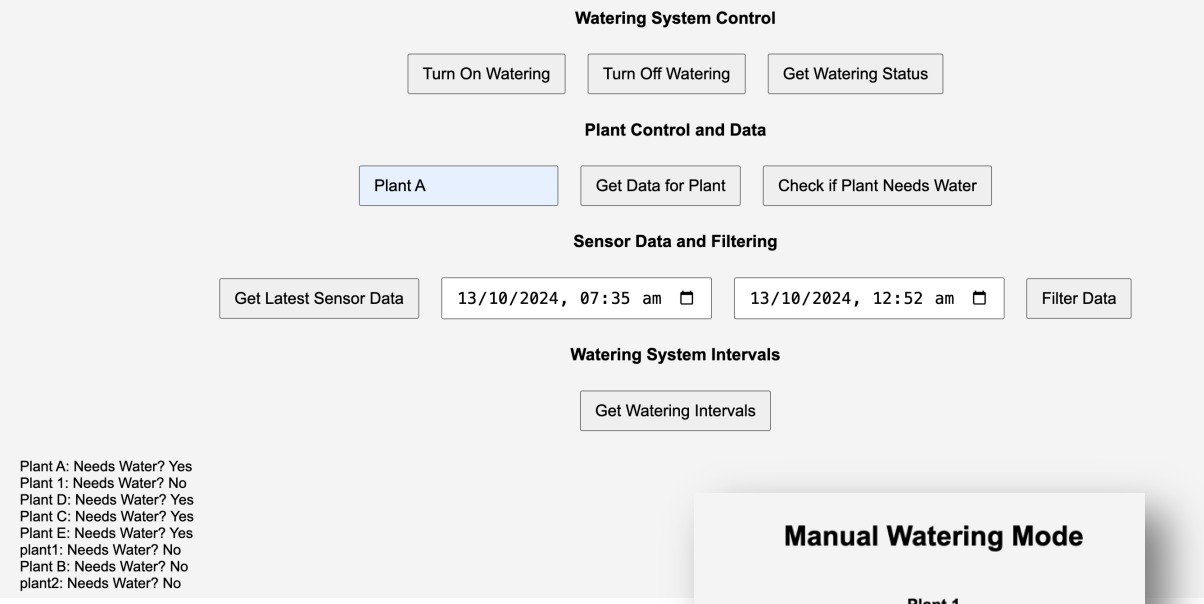


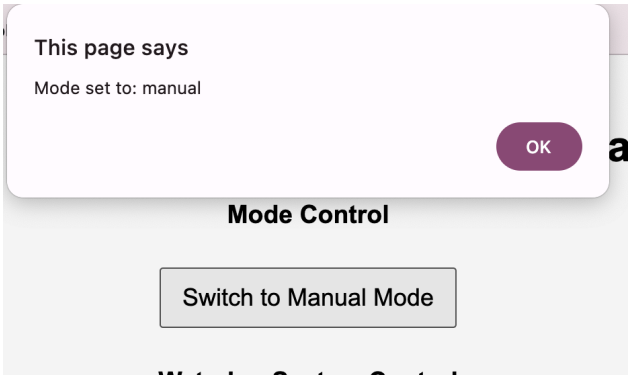
Plant control and its last data:

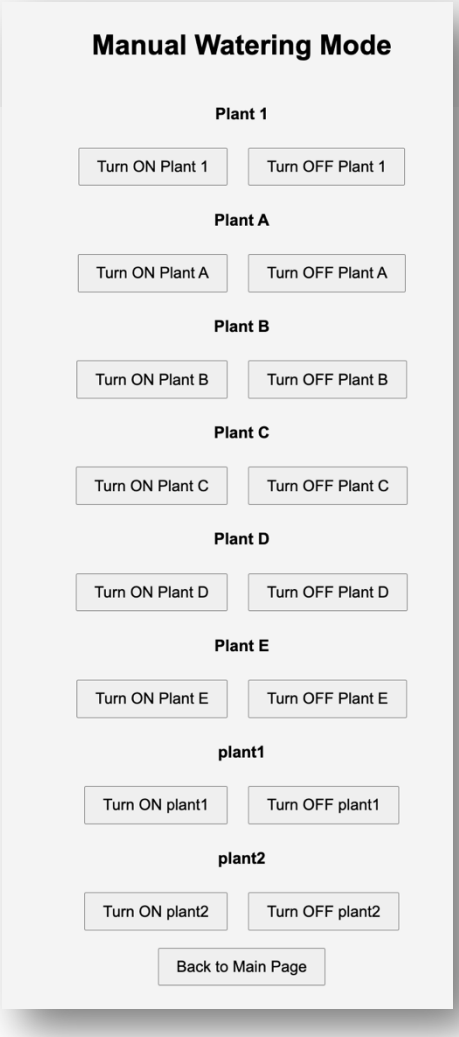




Get watering status:







It opens a manual page for controlling:

Water pump testing:
https://youtu.be/-dWv6v3rglk

In EC2 instance terminal:

```
ubuntu@ip-172-31-81-114:                $ npm install express mongoose body-pars
er

added 85 packages, and audited 86 packages in 5s

14 packages are looking for funding
  run `npm fund` for details

2 low severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
ubuntu@ip-172-31-81-114:                $ nano app.js
ubuntu@ip-172-31-81-114:                $
```

By copying the index.js program in app.js and running it using node app.js; you can successfully get a backup of the backend server.

You can use npm start if:

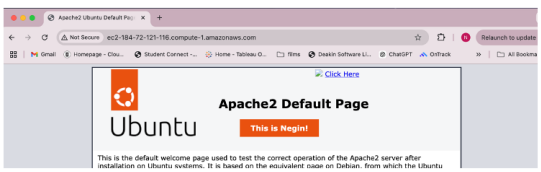
```
ubuntu@ip-172-31-81-114:                $ nano package.json
```

module5_credit — ubuntu@ip-172-31-81-114: ~/module5_credit — ssh -i...

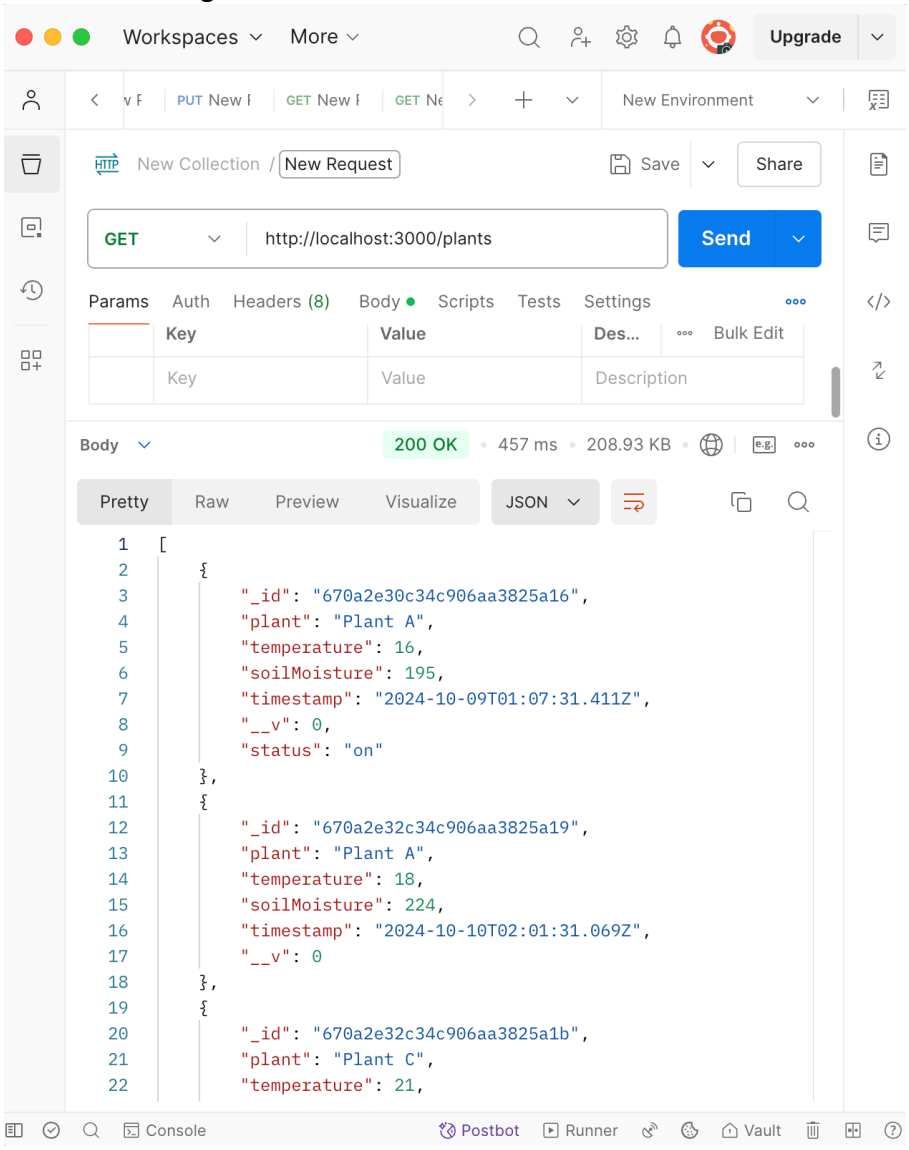
```
 GNU nano 7.2                          package.json *
{
  "name": "module5_credit",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "body-parser": "^1.20.3",
    "express": "^4.21.0",
    "mongoose": "^8.7.0"
  }
}
```

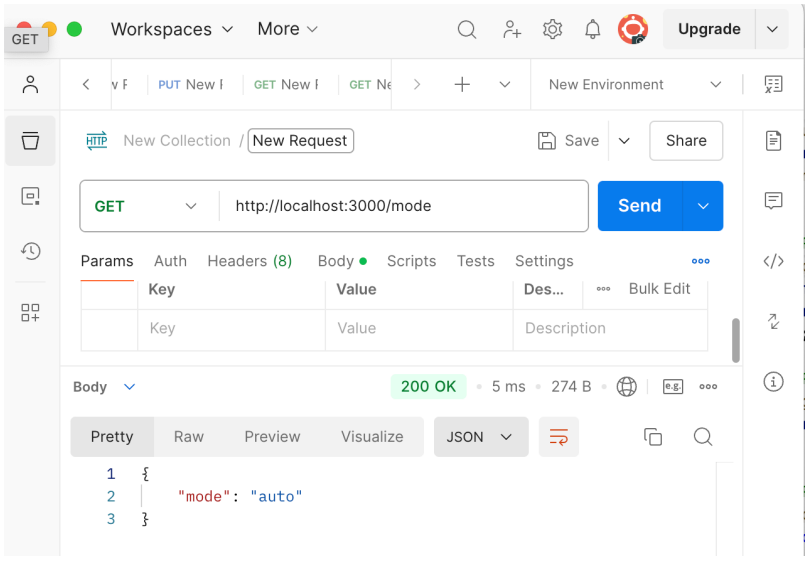
Testing:

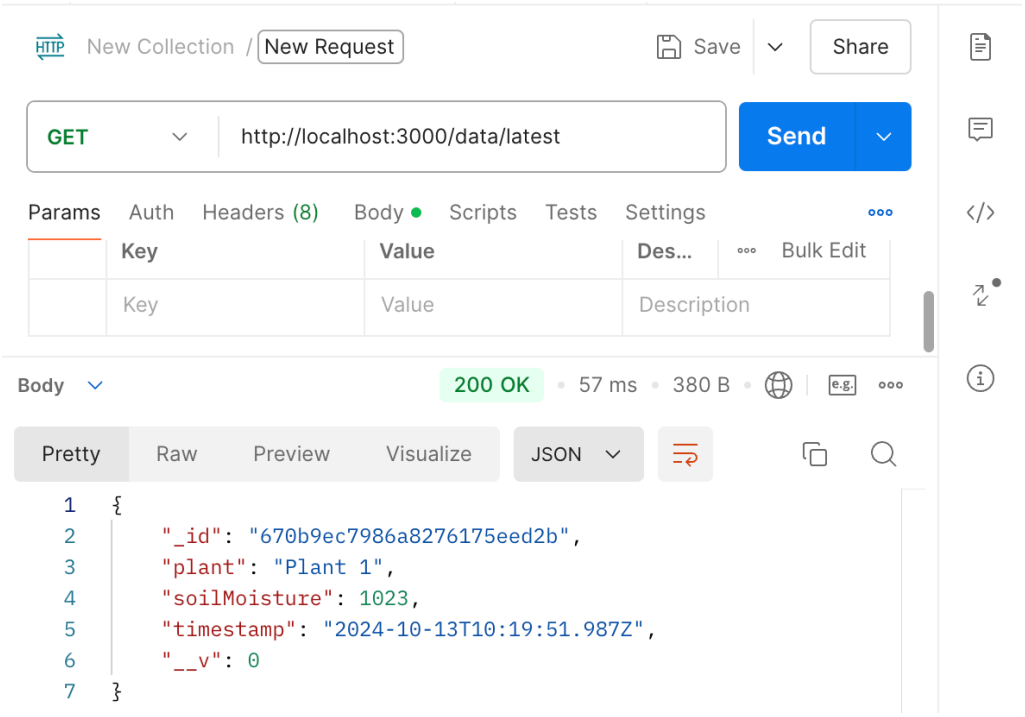
```
ubuntu@ip-172-31-81-114:                $ node app.js
Server is running
Connected to MongoDB
```

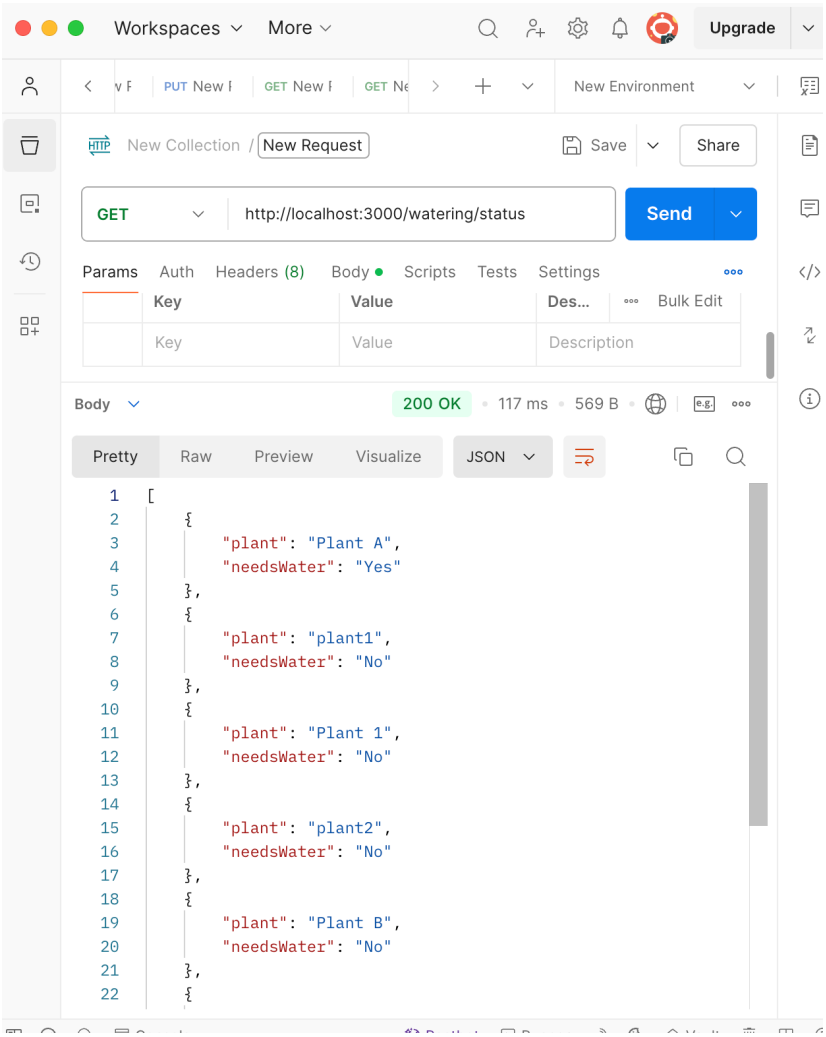


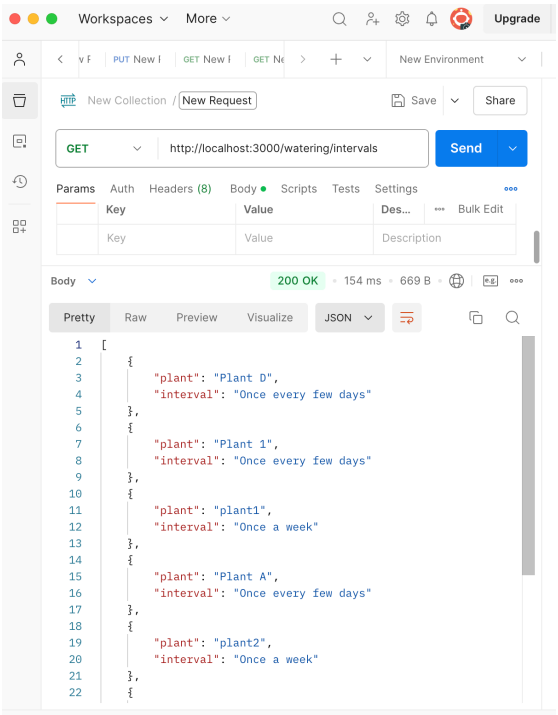
Postman Testing:

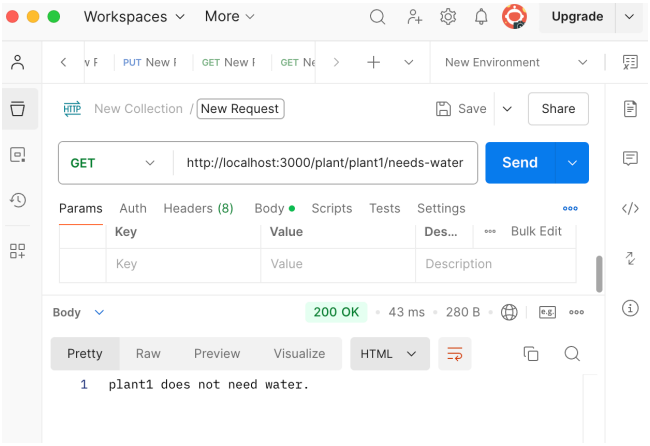


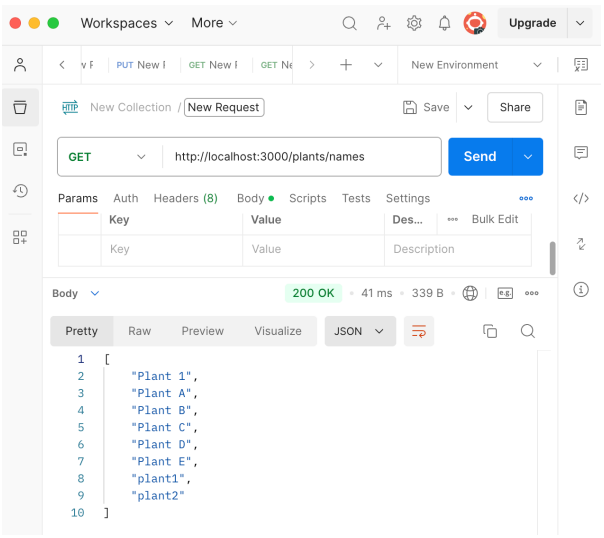



New Collection / New Request
Save
Share
GET
http://localhost:3000/data/latest
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 57 ms 380 B
Pretty Raw Preview Visualize JSON
{
"_id": "670b9ec7986a8276175eed2b",
"plant": "Plant 1",
"soilMoisture": 1023,
"timestamp": "2024-10-13T10:19:51.987Z",
"__v": 0
}


Workspaces More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/watering/status
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 117 ms 569 B
Pretty Raw Preview Visualize JSON
[
{
"plant": "Plant A",
"needsWater": "Yes"
},
{
"plant": "plant1",
"needsWater": "No"
},
{
"plant": "Plant 1",
"needsWater": "No"
},
{
"plant": "plant2",
"needsWater": "No"
},
{
"plant": "Plant B",
"needsWater": "No"
},
{

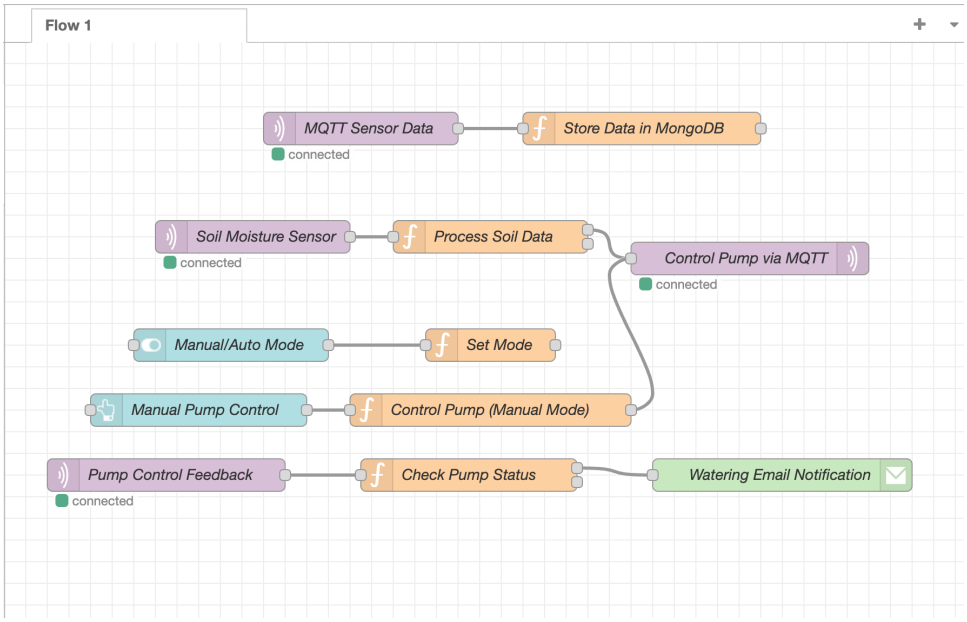

Workspaces
More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/watering/intervals
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 154 ms 669 B
Pretty Raw Preview Visualize JSON
[
{
"plant": "Plant D",
"interval": "Once every few days"
},
{
"plant": "Plant 1",
"interval": "Once every few days"
},
{
"plant": "plant1",
"interval": "Once a week"
},
{
"plant": "Plant A",
"interval": "Once every few days"
},
{
"plant": "plant2",
"interval": "Once a week"
},
{


Workspaces
More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/plant/plant1/needs-water
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 43 ms 280 B
Pretty Raw Preview Visualize HTML
plant1 does not need water.

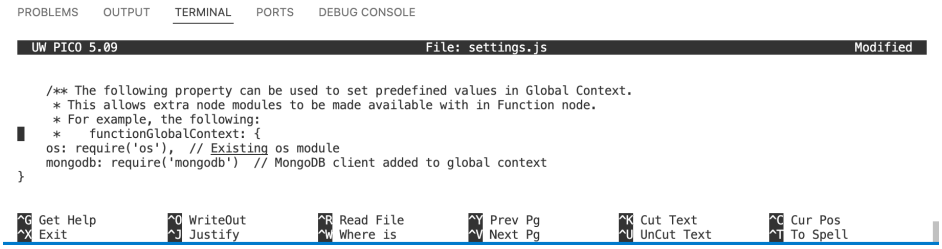

Workspaces
More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/plants/names
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 41 ms 339 B
Pretty Raw Preview Visualize JSON
[
"Plant 1",
"Plant A",
"Plant B",
"Plant C",
"Plant D",
"Plant E",
"plant1",
"plant2"
]

Phase 3: Node-RED Integration and Email Notifications
In this phase, Node-RED was used to capture MQTT messages and store the data in MongoDB Atlas. Node-RED also managed the triggering of email notifications when the pump was activated. Users could switch between manual and automatic modes from the Node-RED dashboard, providing real-time control over the system.
Outcome:

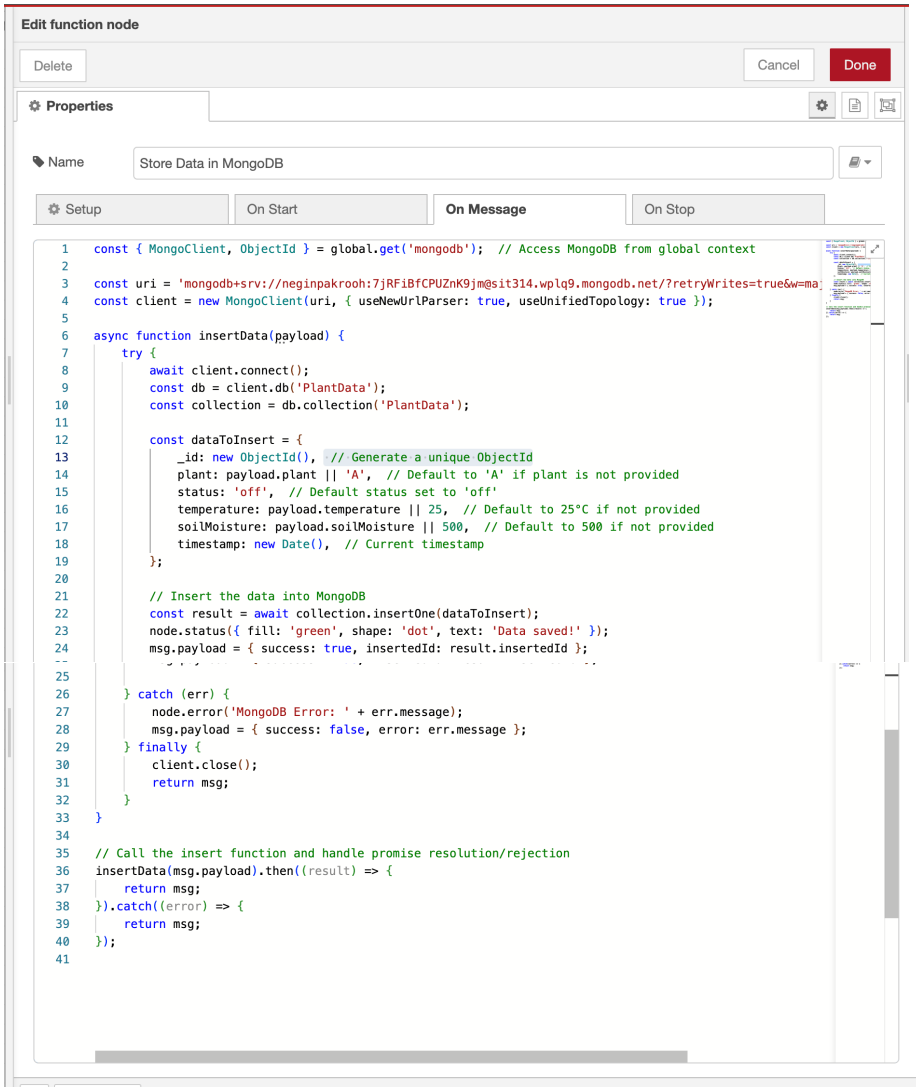
- The Node-RED flow successfully orchestrated the system's operation, ensuring smooth communication between sensors, MQTT, and the cloud backend.

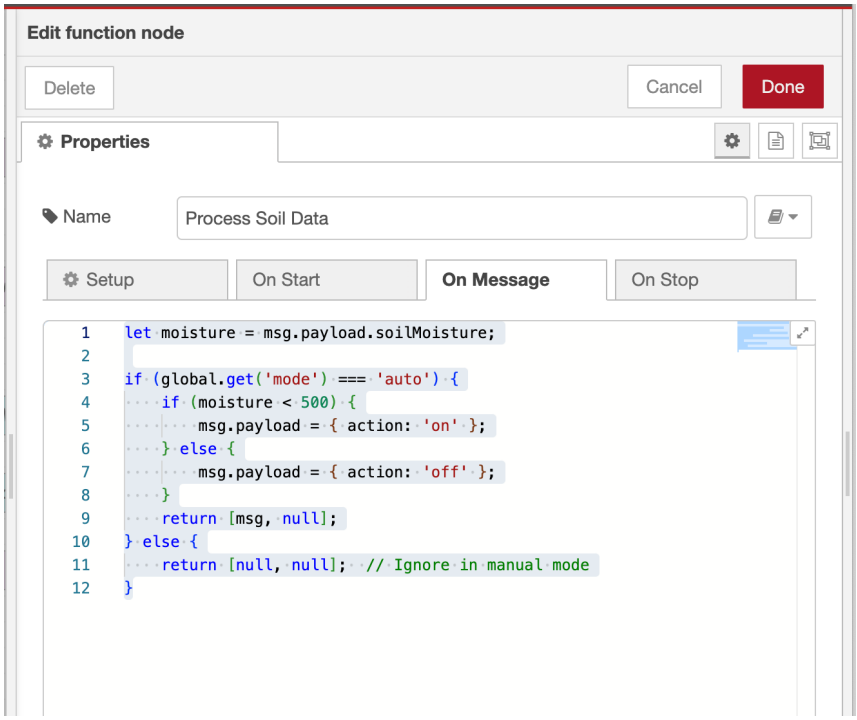


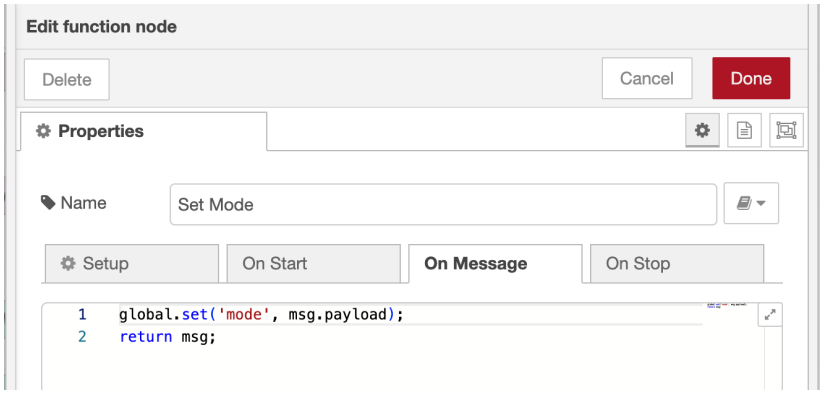
Add MongoDB to the node-red setting:

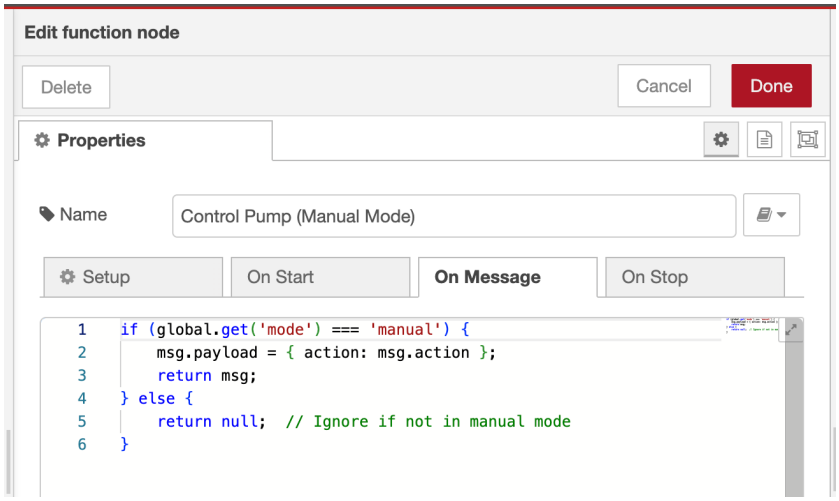


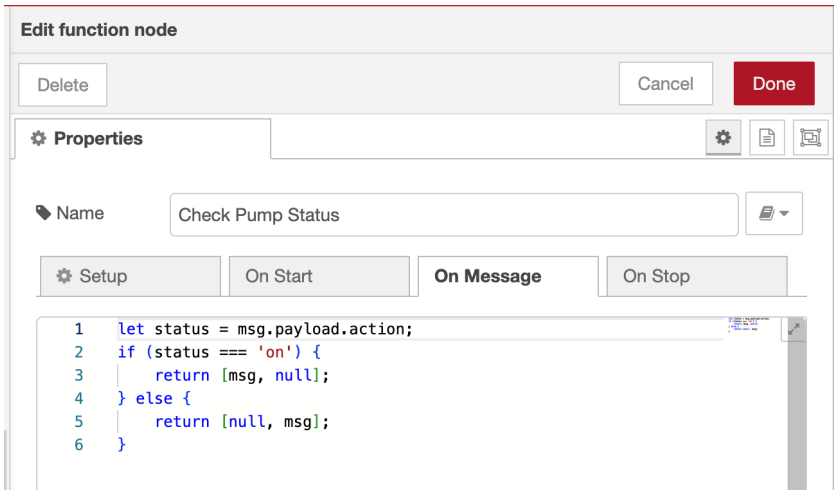
Functions:


Edit function node
Delete
Cancel
Done
Properties
Name
Store Data in MongoDB
Setup
On Start
On Message
On Stop
const { MongoClient, ObjectId } = global.get('mongodb'); // Access MongoDB from global context
const uri = 'mongodb+srv://neginpakrooh:7jRFiBfCPUZnK9jm@sit314.wplq9.mongodb.net/?retryWrites=true&w=ma
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });
async function insertData(payload) {
try {
await client.connect();
const db = client.db('PlantData');
const collection = db.collection('PlantData');
const dataToInsert = {
_id: new ObjectId(), // Generate a unique ObjectId
plant: payload.plant || 'A', // Default to 'A' if plant is not provided
status: 'off', // Default status set to 'off'
temperature: payload.temperature || 25, // Default to 25°C if not provided
soilMoisture: payload.soilMoisture || 500, // Default to 500 if not provided
timestamp: new Date(), // Current timestamp
};
// Insert the data into MongoDB
const result = await collection.insertOne(dataToInsert);
node.status({ fill: 'green', shape: 'dot', text: 'Data saved!' });
msg.payload = { success: true, insertedId: result.insertedId };
} catch (err) {
node.error('MongoDB Error: ' + err.message);
msg.payload = { success: false, error: err.message };
} finally {
client.close();
return msg;
}
}
// Call the insert function and handle promise resolution/rejection
insertData(msg.payload).then((result) => {
return msg;
}).catch((error) => {
return msg;
});


Edit function node
Delete
Cancel
Done
Properties
Name
Process Soil Data
Setup
On Start
On Message
On Stop
let moisture = msg.payload.soilMoisture;
if (global.get('mode') === 'auto') {
if (moisture < 500) {
msg.payload = { action: 'on' };
} else {
msg.payload = { action: 'off' };
}
return [msg, null];
} else {
return [null, null]; // Ignore in manual mode
}







The purpose is to change pump status to on if (soil moisture <500 and mode:auto) or (mode:manual and the user enter pump on in html)

Node-red output:

```
13 Oct 16:42:39 - [info] Started flows
13 Oct 16:42:41 - [info] [mqtt-broker:Hive pump out MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 16:42:41 - [info] [mqtt-broker:Hive pump in MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 16:42:41 - [info] [mqtt-broker:Hive sensor MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 16:42:41 - [info] [mqtt-broker:Hive soil MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:29 - [info] [mqtt-broker:Hive soil MQTT] Disconnected from broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:29 - [info] [mqtt-broker:Hive sensor MQTT] Disconnected from broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:29 - [info] [mqtt-broker:Hive pump in MQTT] Disconnected from broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:29 - [info] [mqtt-broker:Hive pump out MQTT] Disconnected from broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:45 - [info] [mqtt-broker:Hive sensor MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:45 - [info] [mqtt-broker:Hive pump in MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:45 - [info] [mqtt-broker:Hive pump out MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
13 Oct 17:10:45 - [info] [mqtt-broker:Hive soil MQTT] Connected to broker: mqtt://broker.hivemq.com:1883
```

Challenges and Solutions

Wi-Fi Connectivity Issues
During testing, the Arduino occasionally lost Wi-Fi connectivity, causing delays in data transmission. To resolve this, reconnection logic was added to the Arduino code to automatically reconnect to the network if disconnected.

Sensor Calibration
The soil moisture sensor initially provided inconsistent readings. After multiple tests, the moisture threshold was fine-tuned to prevent over-watering, particularly during high temperatures.

Security Concerns
Storing data in the cloud introduced security concerns. Password-protected MQTT connections and secure MongoDB Atlas access were implemented to ensure the safety of transmitted data. Future iterations will explore end-to-end encryption.

Testing and Validation

Extensive testing was conducted to ensure the system performed as expected.

1. Unit Testing:
   Each sensor was tested independently to verify accurate readings and responses.
2. Integration Testing:
   The interaction between the Arduino, MQTT, Node-RED, and MongoDB Atlas was tested to ensure smooth data flow.
3. User Testing:
   HTTP requests were simulated through Postman to confirm the backend's functionality and the system's ability to switch between manual and automatic modes.
4. Email Notification Testing:
   Email alerts were tested by activating the pump, ensuring users received notifications promptly.

Outcome:
The system performed reliably across all tests, meeting both functional and non-functional requirements.

Outcomes and Achievements

The Smart Watering System achieved all project objectives, providing a scalable and reliable solution for automated plant care. Key achievements include:

- Real-time monitoring and control using MQTT and Node-RED.
- Cloud-based data storage with MongoDB Atlas for long-term analysis.
- Manual override functionality, allowing users to control the pump remotely.
- Email notifications to keep users informed about the system's status.
- Scalable architecture that can support additional plants and sensors with ease.

Future Improvements

Several improvements are planned for future iterations of the system:

1. Mobile App Integration:
   Developing a mobile application will provide a more user-friendly interface for monitoring and control.
2. AI-Based Predictions:
   Historical data stored in MongoDB Atlas can be used to predict watering schedules using machine learning.
3. Additional Sensors:
   Adding light and humidity sensors will further optimize the watering schedule.
4. Enhanced Security:
   Implementing end-to-end encryption will ensure all data transmissions are secure.

The Smart Watering System for Houseplants demonstrates how IoT technologies can be used to automate routine tasks and provide real-time insights. By leveraging AWS EC2, MongoDB Atlas, MQTT, and Node-RED, the system offers a flexible, scalable, and reliable solution for plant care. The project successfully integrated hardware, software, and cloud services, delivering a comprehensive platform for both manual and automatic operation. With future enhancements such as mobile integration and AI-based predictions, the system has the potential to become even more effective.

Appendices

- Hardware Used: Arduino Nano 33 IoT, DHT11 sensor, soil moisture sensor, pump, LED
- Software Used: Arduino IDE, Node-RED, AWS EC2, MongoDB Atlas, Postman

You can find the codes and video link from GitHub:
https://github.com/Negpkr/My-Final-IoT-Project.git

Thank You!