SIT315 HD Project Report: Smart Plant Watering System with Distributed and Parallel Programming

Student Name: Negin Pakroohjahromi
Student ID: 222393187

The Smart Plant Watering System is designed to automate plant care through IoT technologies, providing real-time monitoring and remote control. The system aims to solve the problem of inconsistent plant care, which can arise from user oversight or environmental factors, by automating the process of irrigation based on soil moisture and temperature. The project integrates microservices architecture, MQTT communication, and cloud-based storage, demonstrating the use of distributed and parallel programming concepts. The system includes both manual mode for user control and automatic mode for intelligent, data-driven watering based on sensor readings. Additionally, Node-RED provides a user-friendly GUI for system control and alerts.
This report focuses on the application of distributed computing principles, real-time communication protocols, and microservices with auto-scaling. The ability to simulate environmental conditions, store data in MongoDB Atlas, and integrate seamlessly with cloud services makes this system scalable, robust, and adaptable for future use.

Objectives

1. Real-time Monitoring and Control: Collect data from sensors and provide manual and automatic control modes.
2. Parallel Communication: Use MQTT for real-time, asynchronous communication between components.
3. Distributed Data Storage: Leverage MongoDB Atlas to store sensor readings and plant data.
4. GUI with Node-RED: Provide a visual interface for monitoring system status and controlling operations.
5. Simulated Data Handling: Generate CSV files from simulated data and publish it over MQTT for scalability testing.

System Architecture and Design

The system follows a microservices architecture, ensuring modularity and scalability. Each component of the system operates independently, improving fault tolerance and enabling future expansion.

Microservices Architecture with Auto-Scaling

- MQTT Broker: Manages real-time communication between the Arduino and backend services.
- MongoDB Atlas: Provides distributed data storage, ensuring high availability and data redundancy.

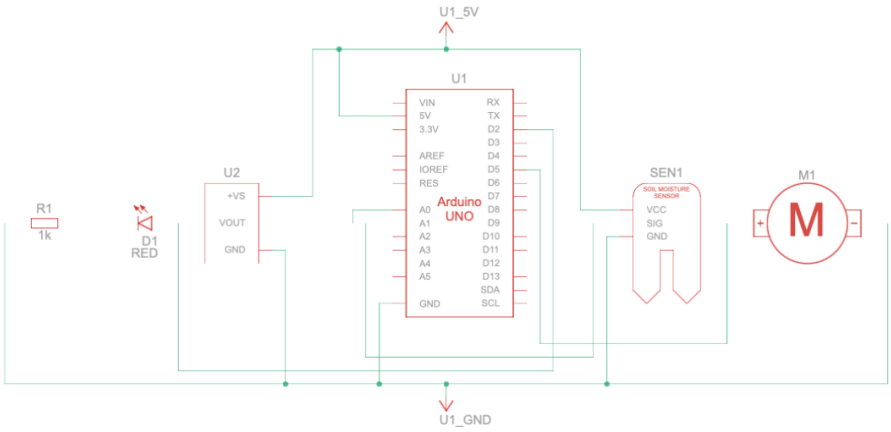
This architecture ensures that individual components can be developed, deployed, and scaled independently without affecting the entire system.

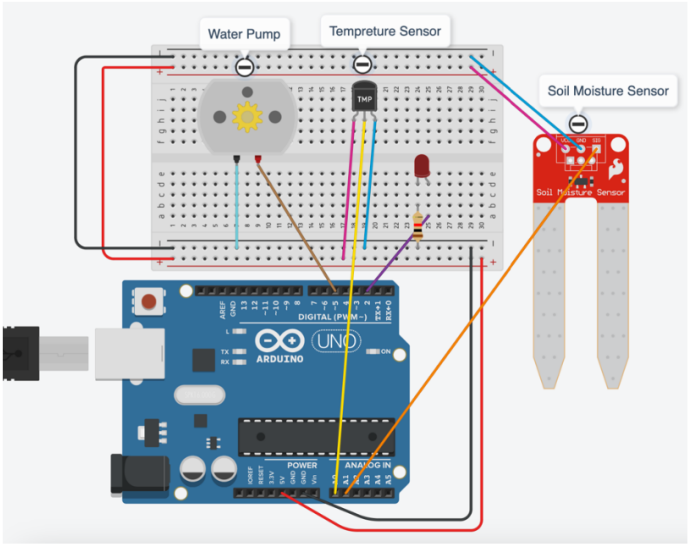
Parallel Communication with MQTT
The MQTT publish/subscribe model ensures parallel communication between the Arduino and the backend, preventing bottlenecks. Multiple sensors can publish data simultaneously to the MQTT broker, ensuring real-time data flow.
This asynchronous communication is essential for IoT systems, where multiple devices operate concurrently and need to transmit data without delays.

Distributed Data Storage in MongoDB Atlas
Sensor readings are stored in MongoDB Atlas, a distributed cloud database that supports automatic scaling. The database stores both real-time and historical data, enabling future analysis and AI-based predictions.

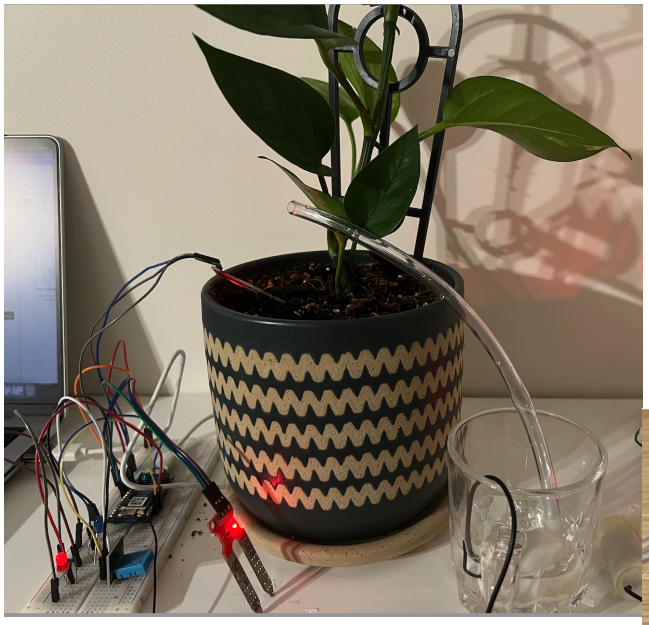
Implementation Phases

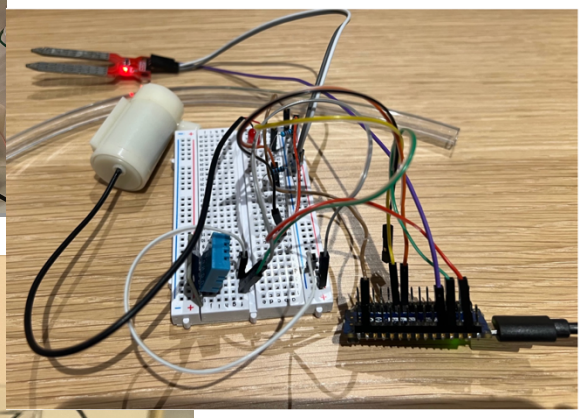
Phase 1: Sensor Integration and MQTT Communication
The Arduino Nano 33 IoT collects data from soil moisture and temperature sensors and publishes it to the MQTT broker.

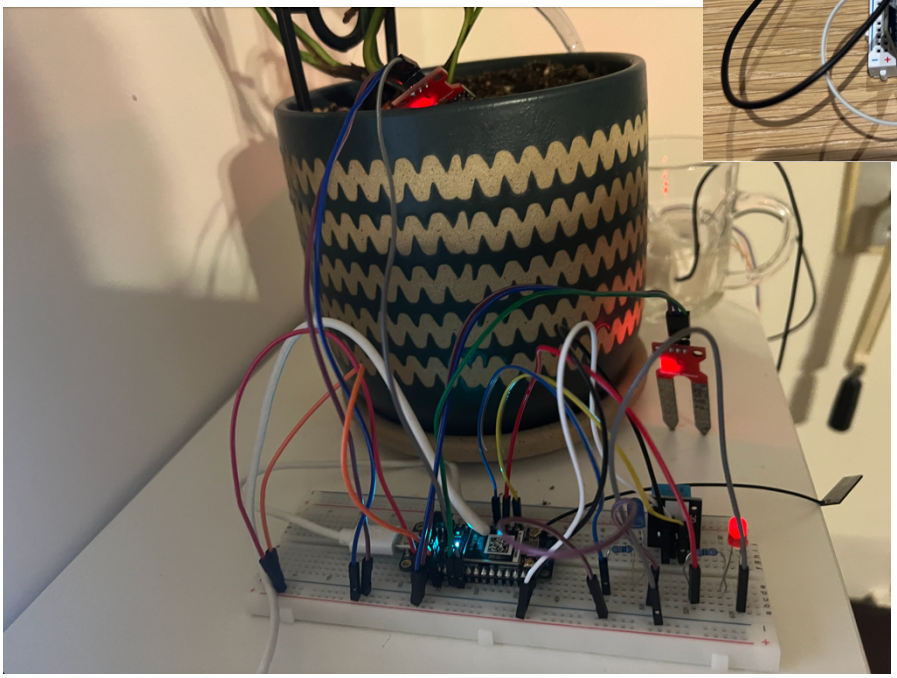




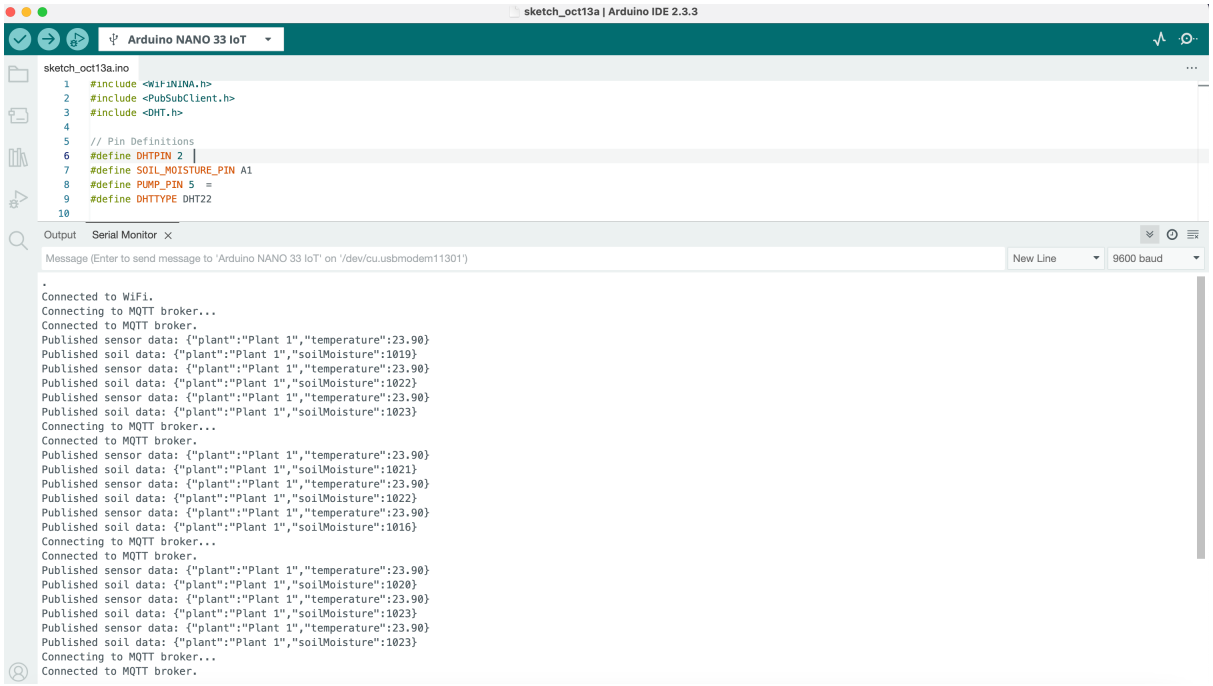
If DH22, connect it to Digital PIN 2!



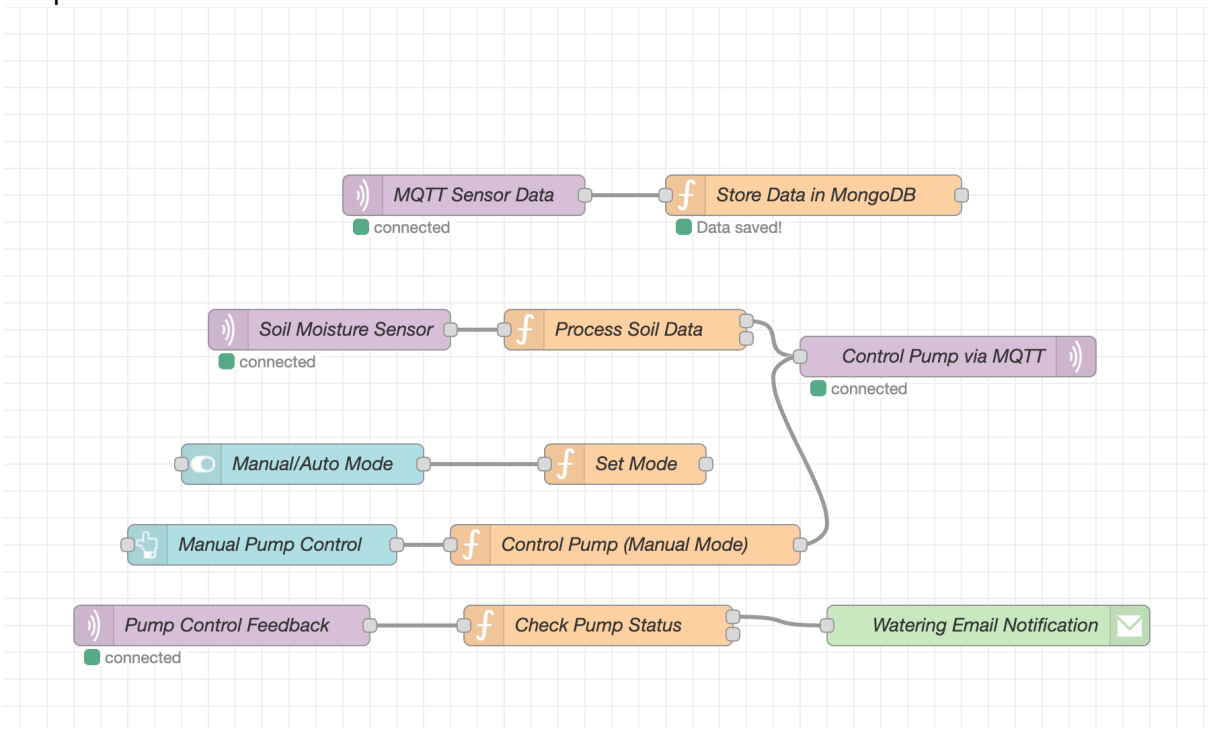




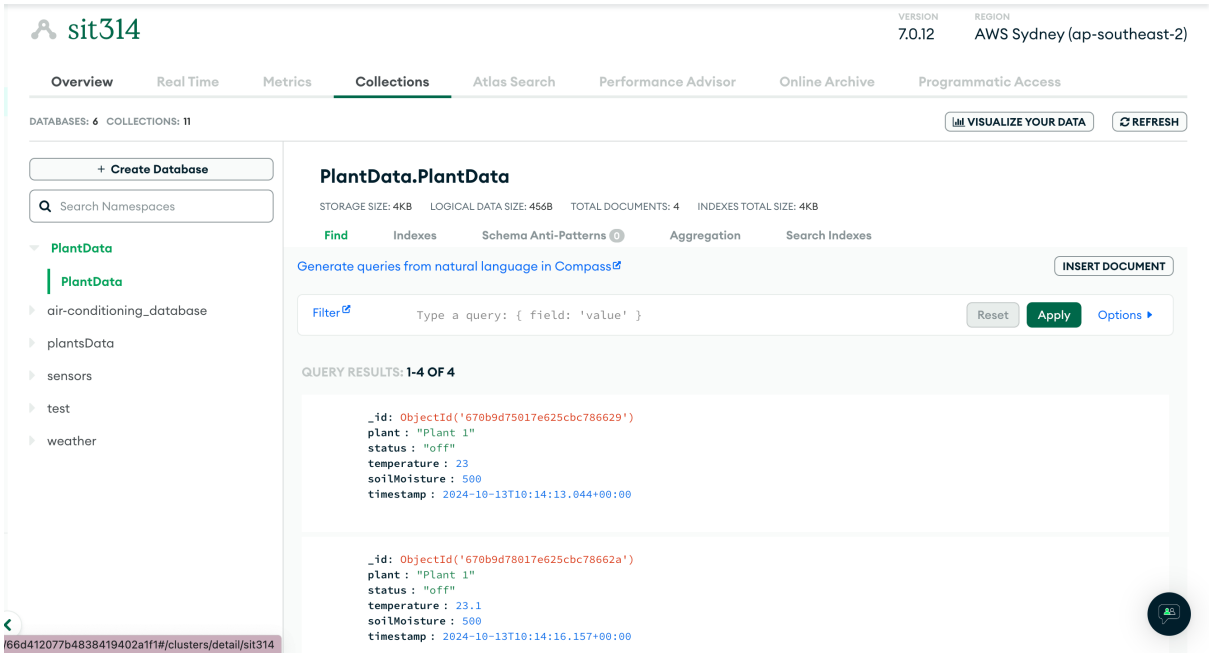
Phase 2: Backend Deployment and API Development
The backend was developed using Node.js. It provides RESTful APIs for manual control of the pump and monitoring plant data. The backend interacts with MongoDB Atlas for data storage.



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



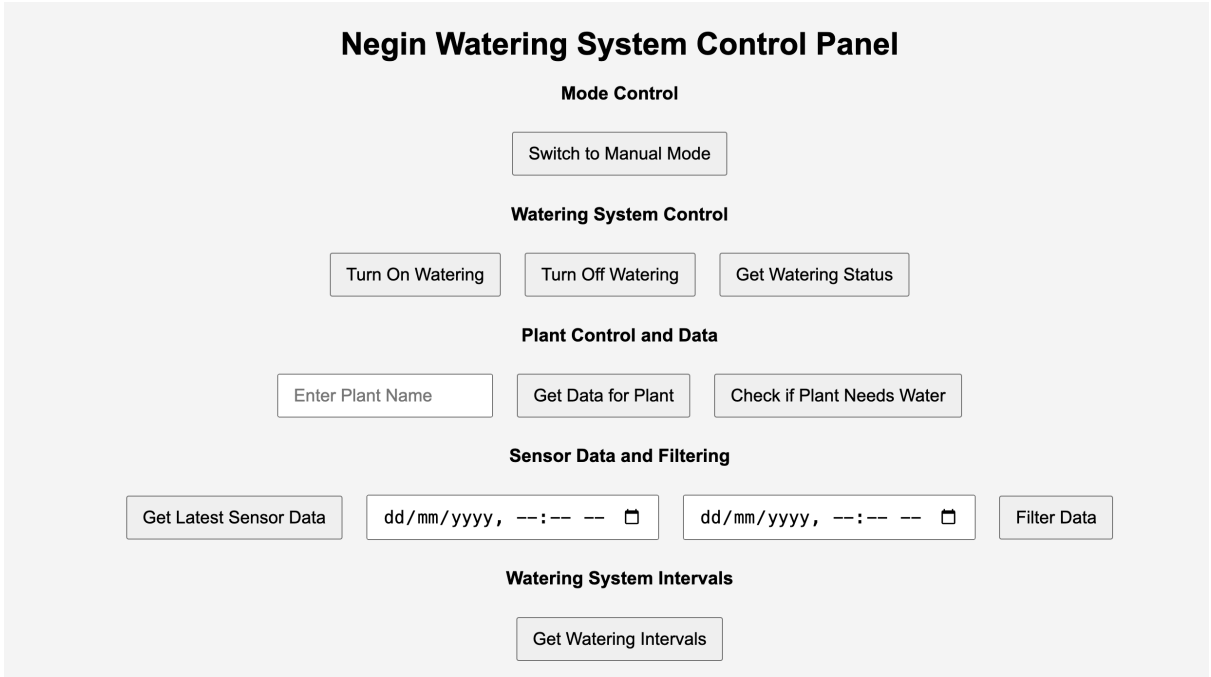
As you can see, the data is saved to the database!



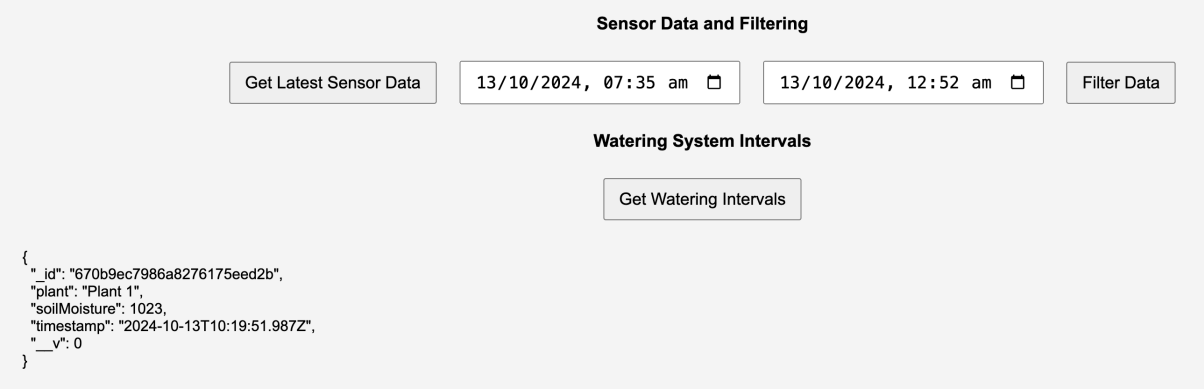
In the backend part (index.js):

```
[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 }
```

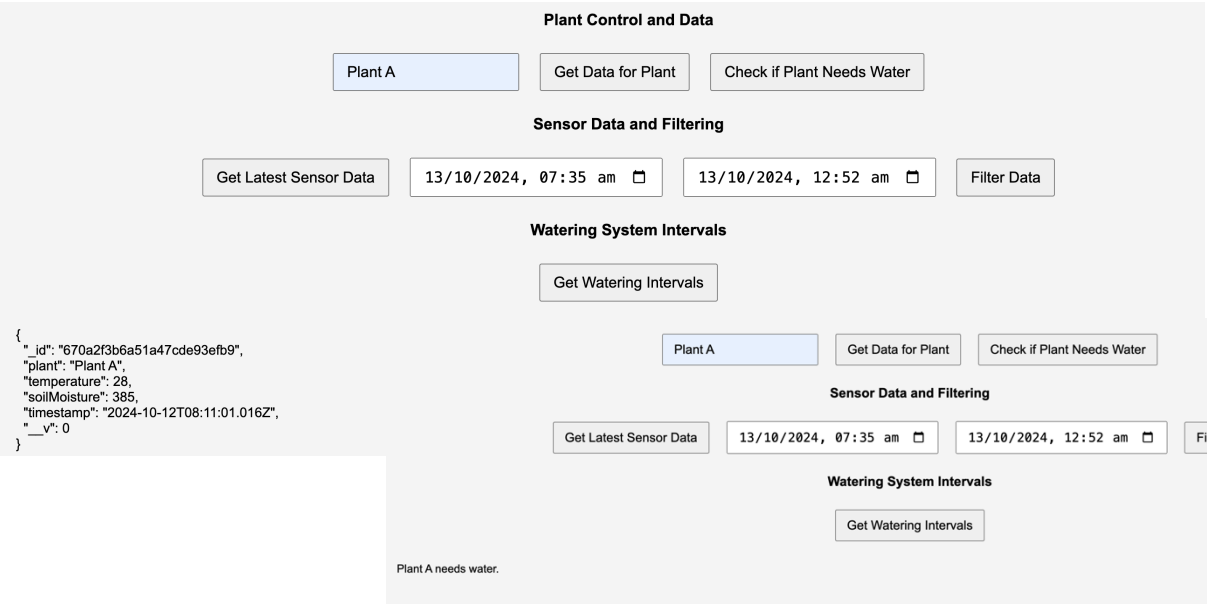
Some random data (plant A, etc.) was added to the database for testing:

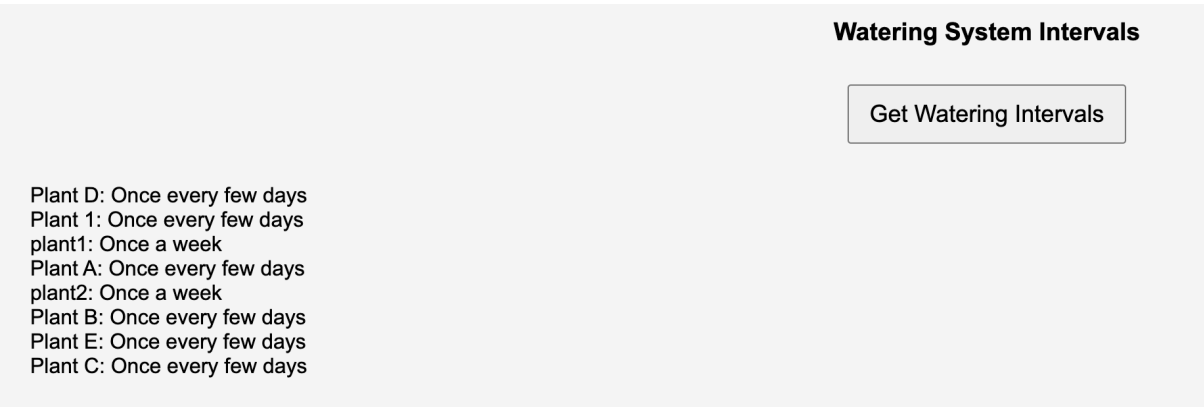


Filtering:

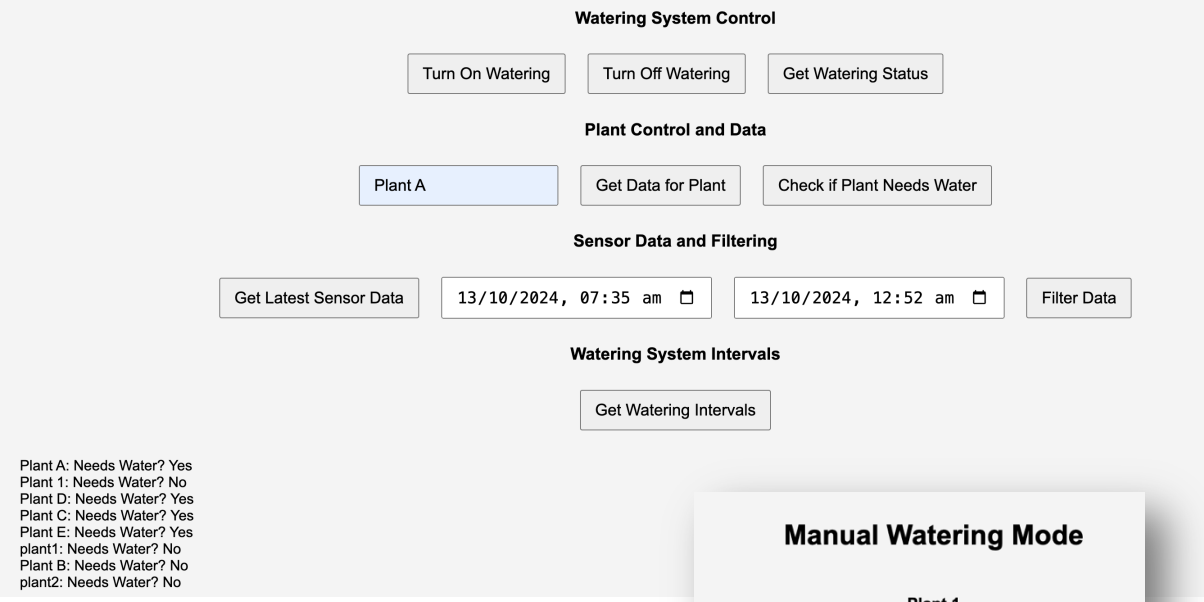


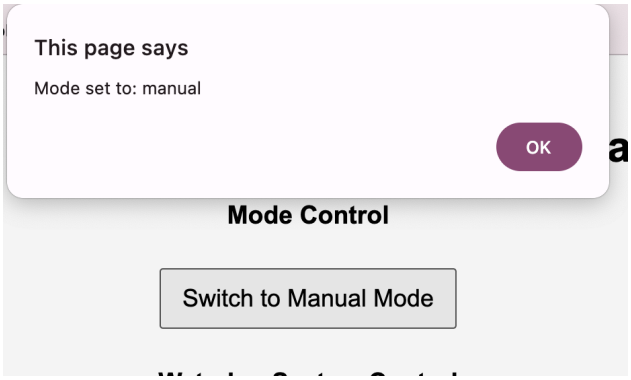
Plant control and its last data:

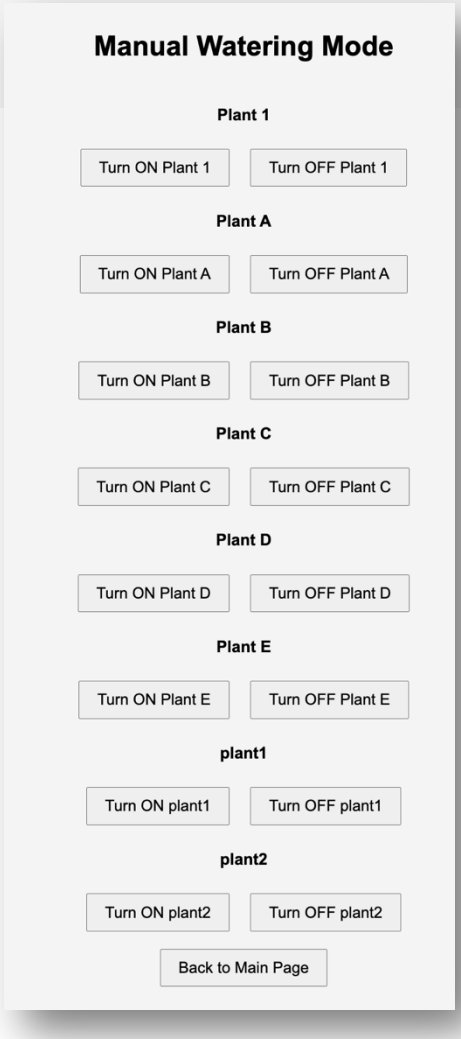




Get watering status:



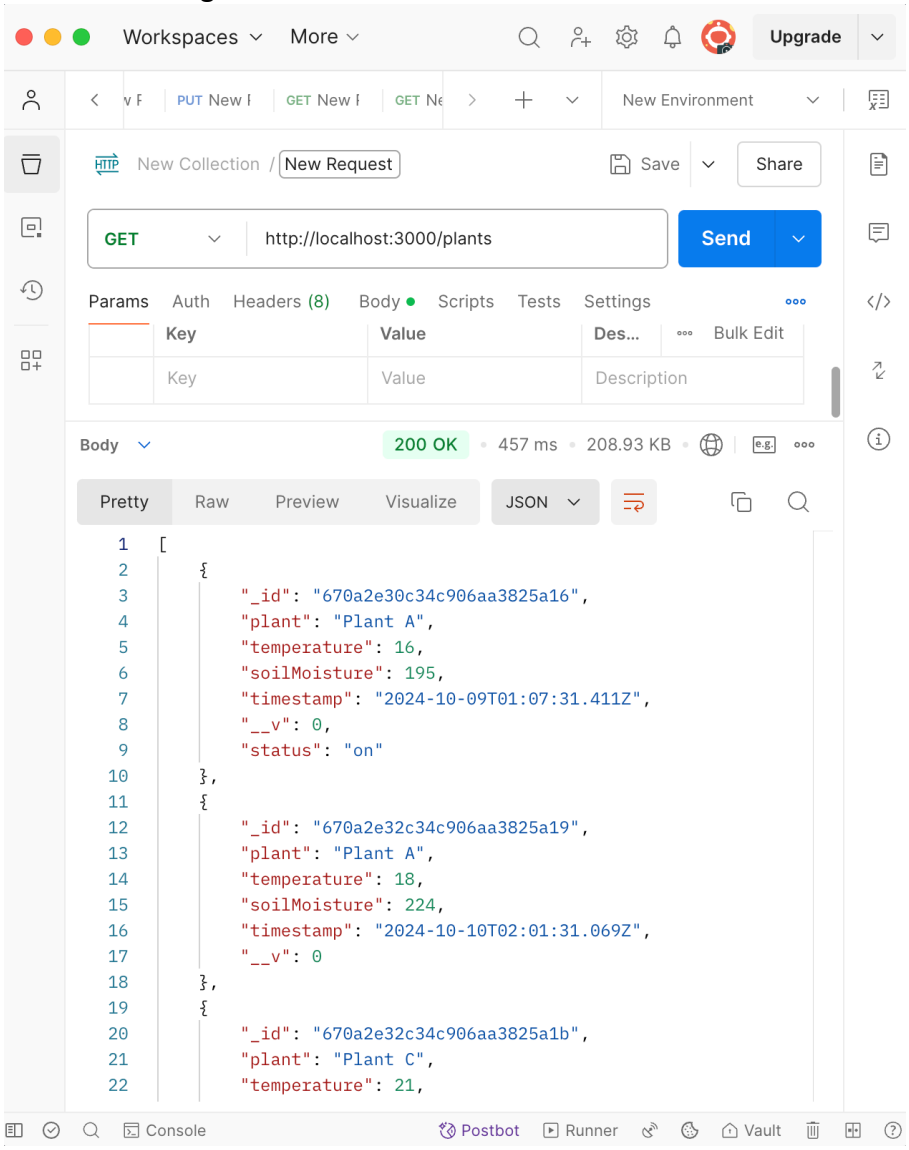


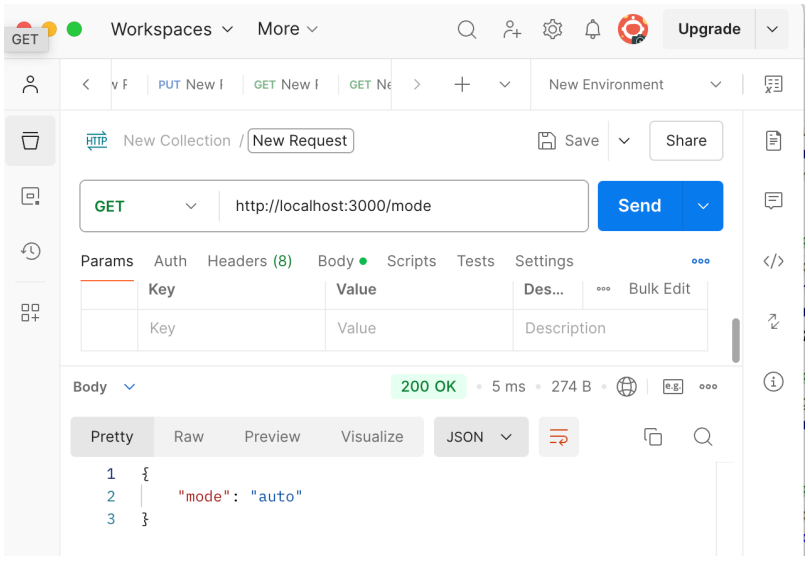


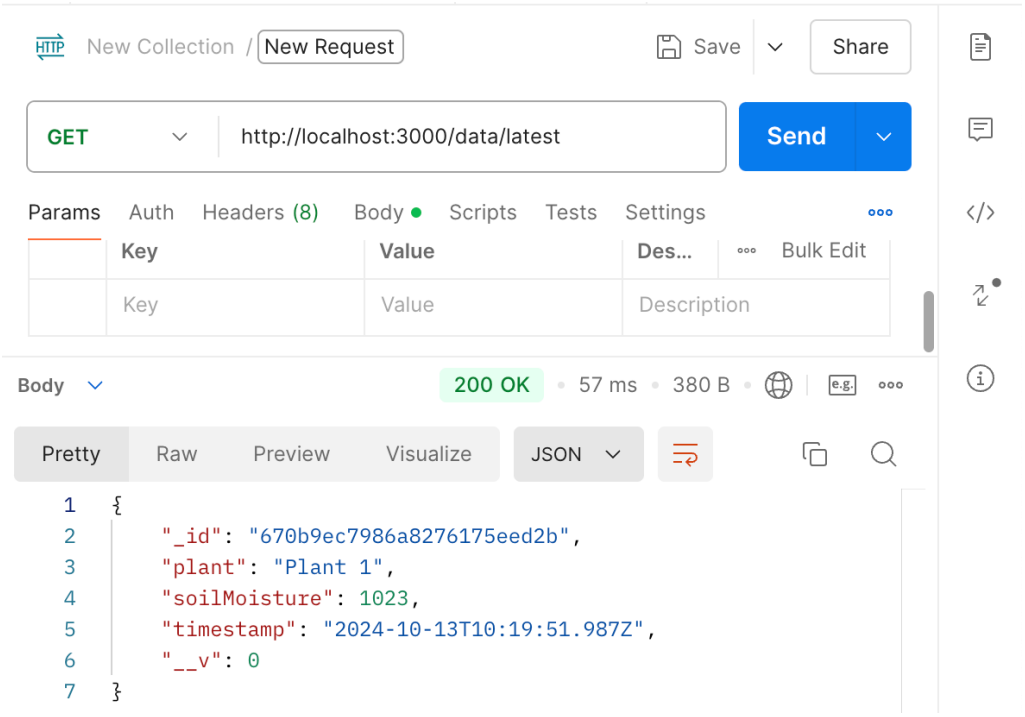
It opens a manual page for controlling:

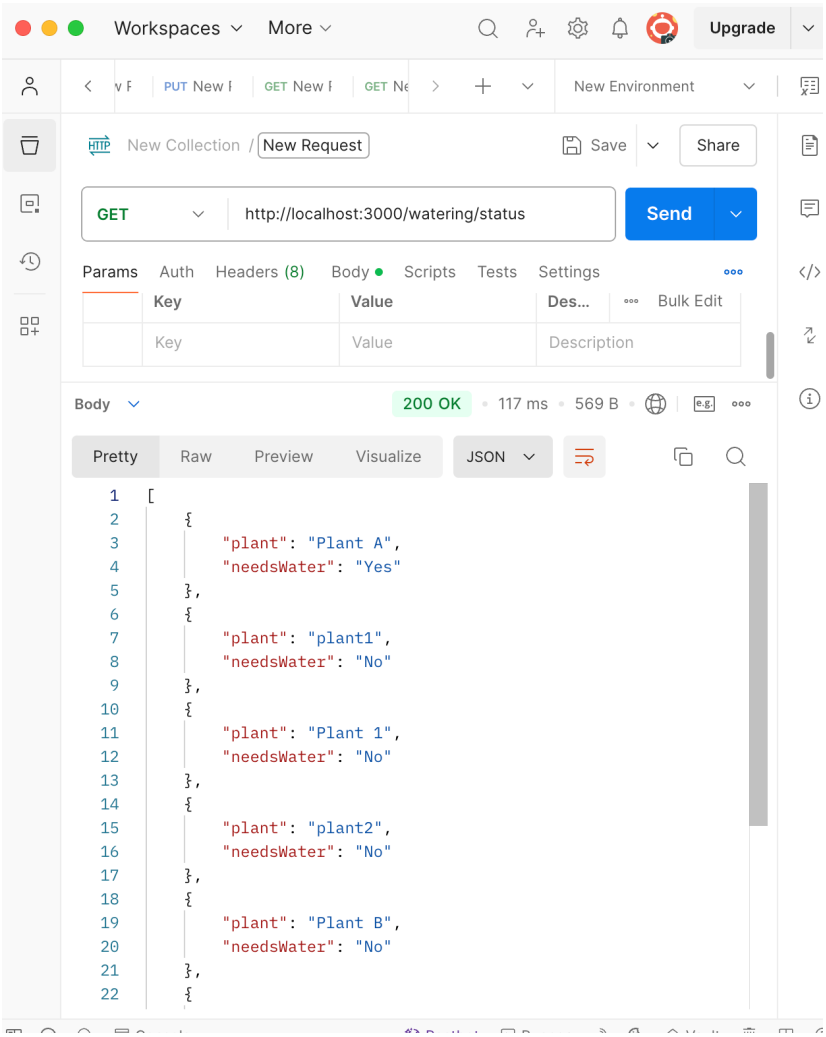
Water pump testing:
https://youtu.be/-dWv6v3rglk

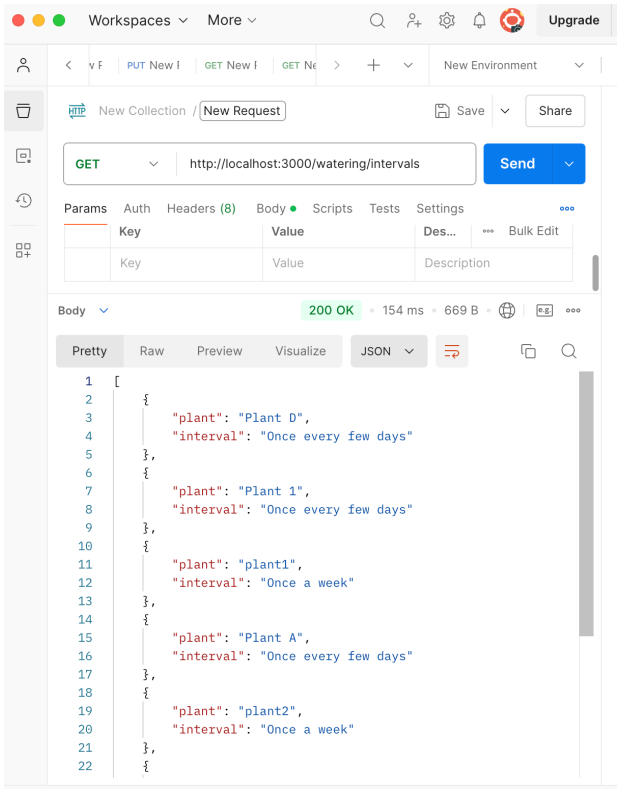
Postman Testing:

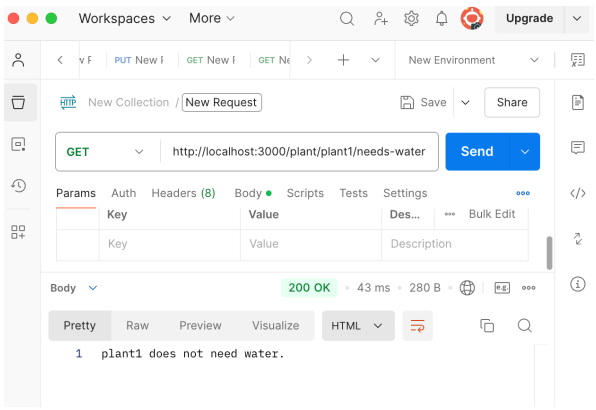


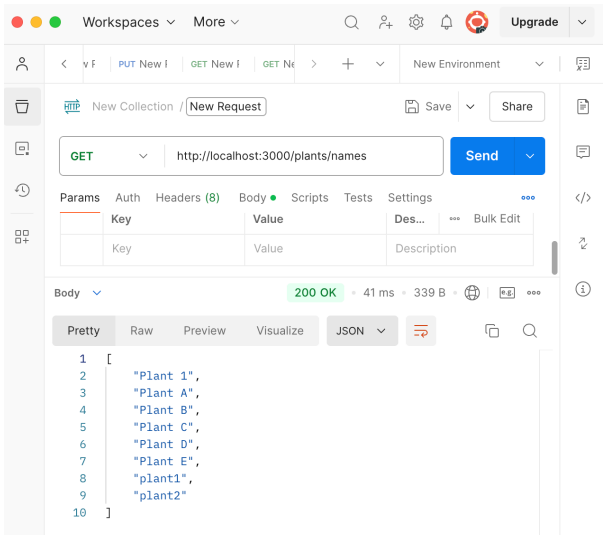



New Collection / New Request
Save
Share
GET
http://localhost:3000/data/latest
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 57 ms 380 B
Pretty Raw Preview Visualize JSON
{
"_id": "670b9ec7986a8276175eed2b",
"plant": "Plant 1",
"soilMoisture":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

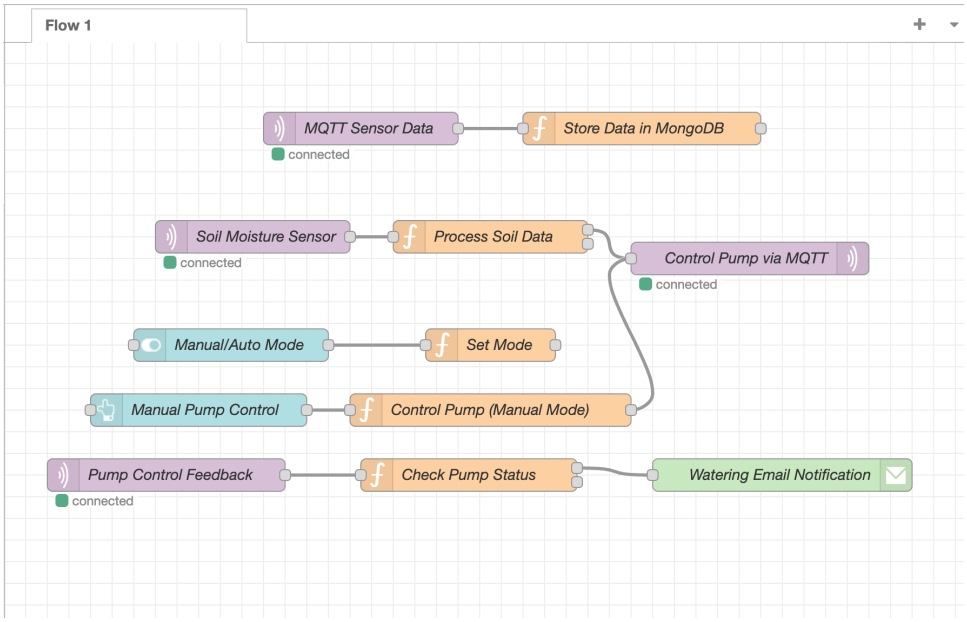

Workspaces
More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/watering/intervals
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 154 ms 669 B
Pretty Raw Preview Visualize JSON
[
{
"plant": "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.

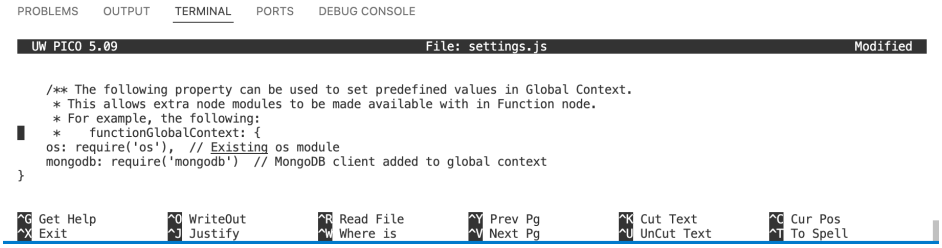

Workspaces
More
Upgrade
New Environment
New Collection / New Request
Save
Share
GET
http://localhost:3000/plants/names
Send
Params Auth Headers (8) Body Scripts Tests Settings
Key Value Des... Bulk Edit
Key Value Description
Body
200 OK 41 ms 339 B
Pretty Raw Preview Visualize JSON
[
"Plant 1",
"Plant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

Phase 3: Node-RED Orchestration and GUI Development
Node-RED orchestrates the system's operations and provides a GUI for monitoring. It processes MQTT messages, stores data in MongoDB, and sends email alerts when the pump is activated.

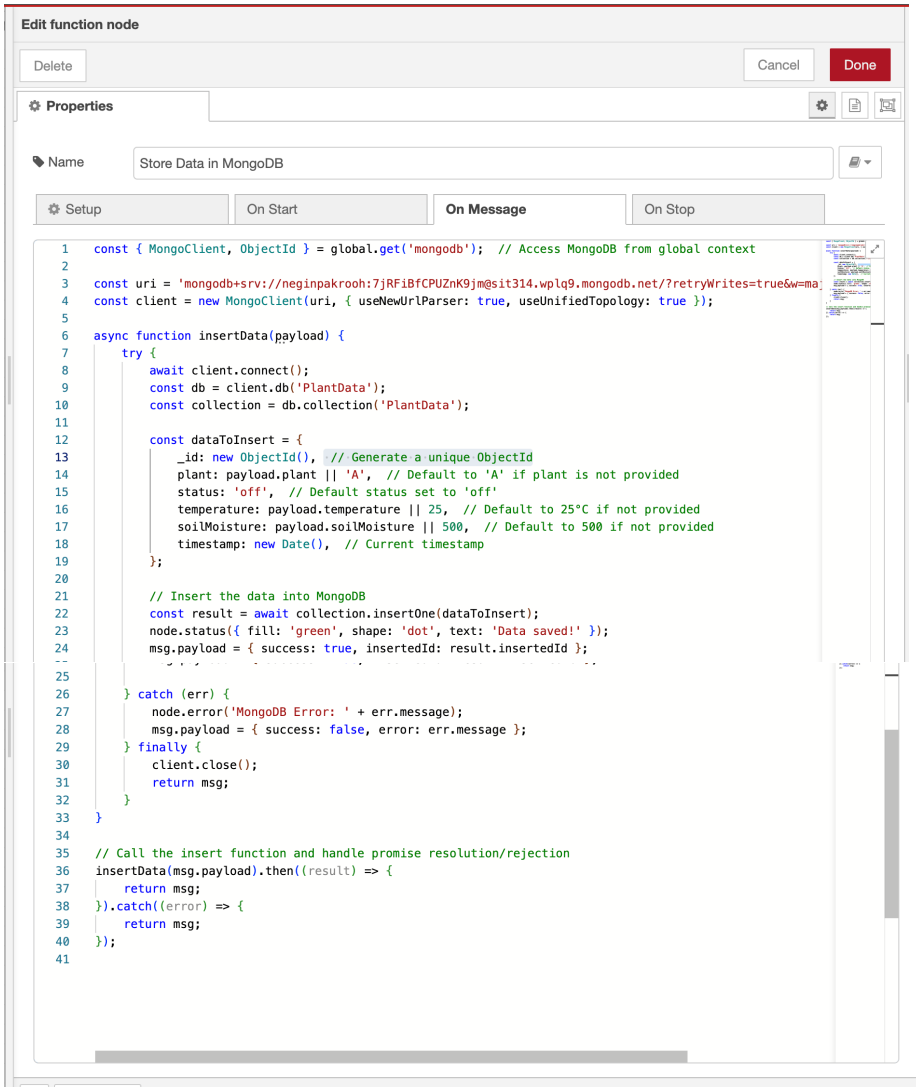
- Manual Mode: Users can switch to manual mode and control the pump through the GUI.
- Email Notifications: Node-RED sends notifications whenever the pump is turned on.

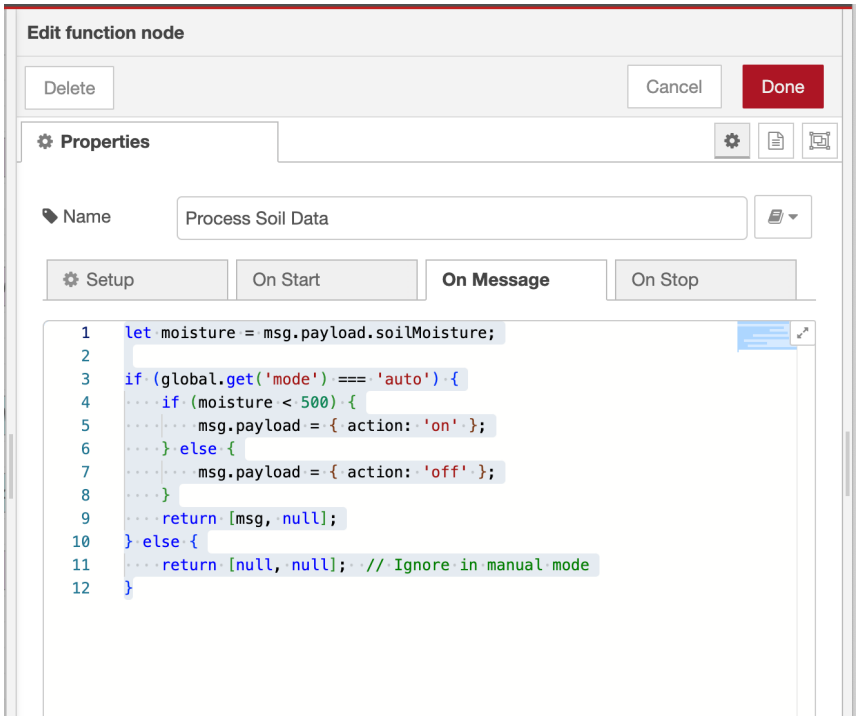


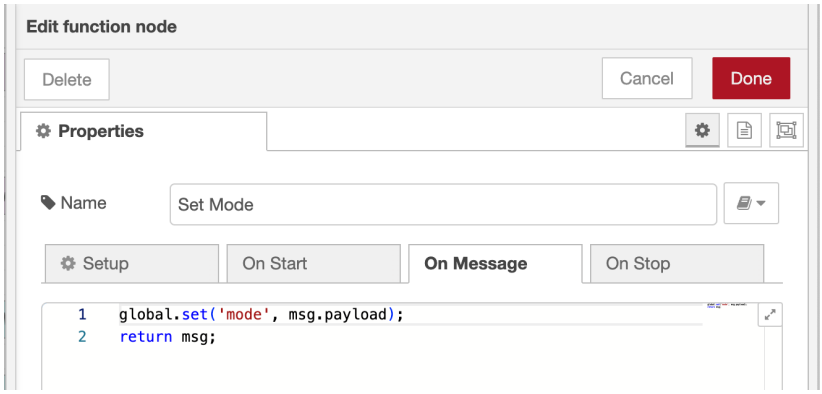
Add MongoDB to the node-red setting:

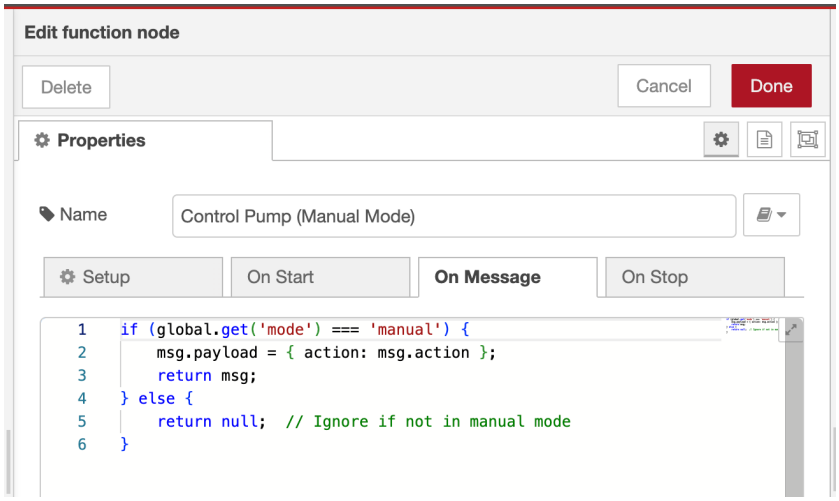


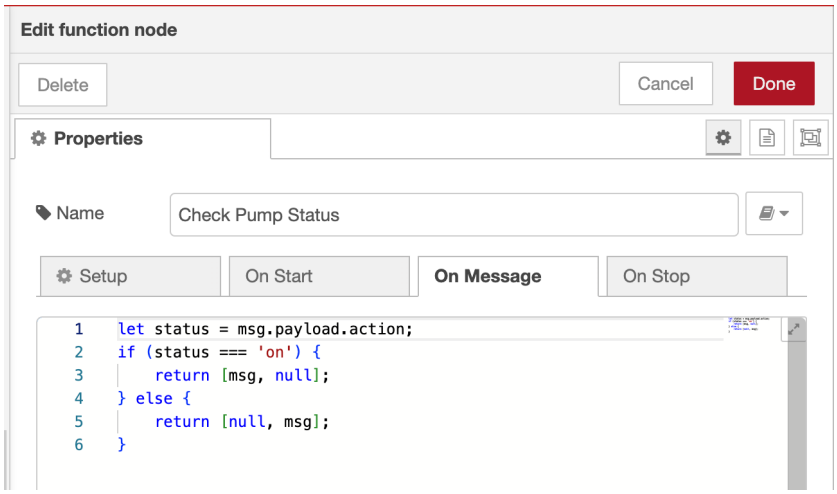
Functions:


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
}


Edit function node
Delete
Cancel
Done
Properties
Name
Set Mode
Setup
On Start
On Message
On Stop
1 global.set('mode', msg.payload);
2 return msg;


Edit function node
Delete
Cancel
Done
Properties
Name
Control Pump (Manual Mode)
Setup
On Start
On Message
On Stop
1 if (global.get('mode') === 'manual') {
2 msg.payload = { action: msg.action };
3 return msg;
4 } else {
5 return null; // Ignore if not in manual mode
6 }


Edit function node
Delete
Cancel
Done
Properties
Name
Check Pump Status
Setup
On Start
On Message
On Stop
1 let status = msg.payload.action;
2 if (status === 'on') {
3 return [msg, null];
4 } else {
5 return [null, msg];
6 }

Phase 4: Simulating Data and CSV Export
The system simulates plant movements and sensor data can be exported from the database. This data then can be published over MQTT for analysis and visualisation for getting each plant's watering intervals enabling the scalability of the architecture.

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
```


Results and Achievements

The Smart Plant Watering System successfully demonstrates:

- Real-time communication with MQTT.
- Scalable storage using MongoDB Atlas.
- Parallel data handling with multiple sensors.
- Seamless user interaction through the Node-RED GUI.

These achievements align with this units HD-level requirements, demonstrating proficiency in distributed and parallel programming.

Challenges and Solutions

- Wi-Fi Connectivity Issues: Reconnection logic was added to ensure stable communication.
- Sensor Calibration: Thresholds were adjusted to prevent over-watering or under-watering.
- Security Enhancements: Password-protected MQTT connections and secure database access were implemented.

Future Directions

- AI-Based Predictions: Use machine learning to optimize watering schedules.
- Mobile App Development: Create a mobile app for easier remote control.
- Additional Sensors: Integrate light and humidity sensors for more comprehensive monitoring.

The Smart Plant Watering System demonstrates how IoT technologies, distributed computing, and parallel programming can automate plant care. The system's modular design, scalability, and real-time capabilities make it a robust solution, meeting the SIT315 High Distinction criteria.

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

Thank You!