

# 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.

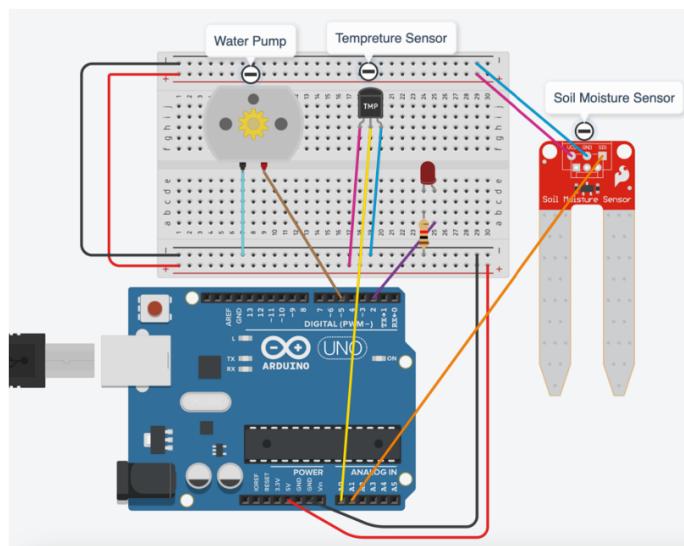
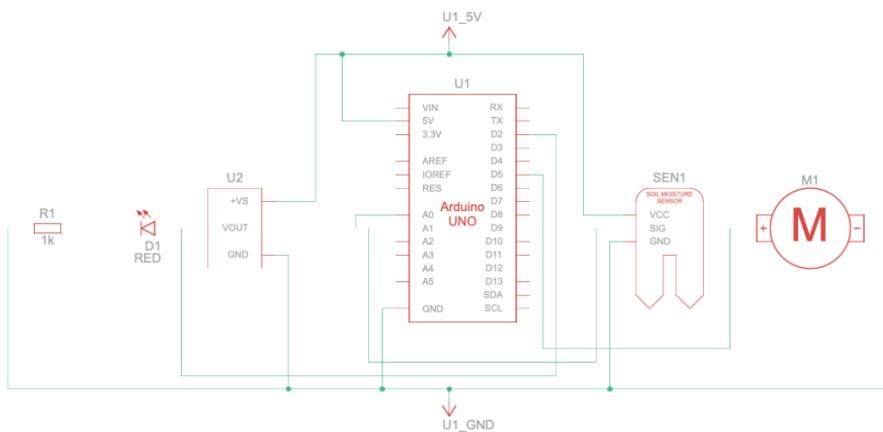
## Distributed Data Storage in MongoDB Atlas

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.

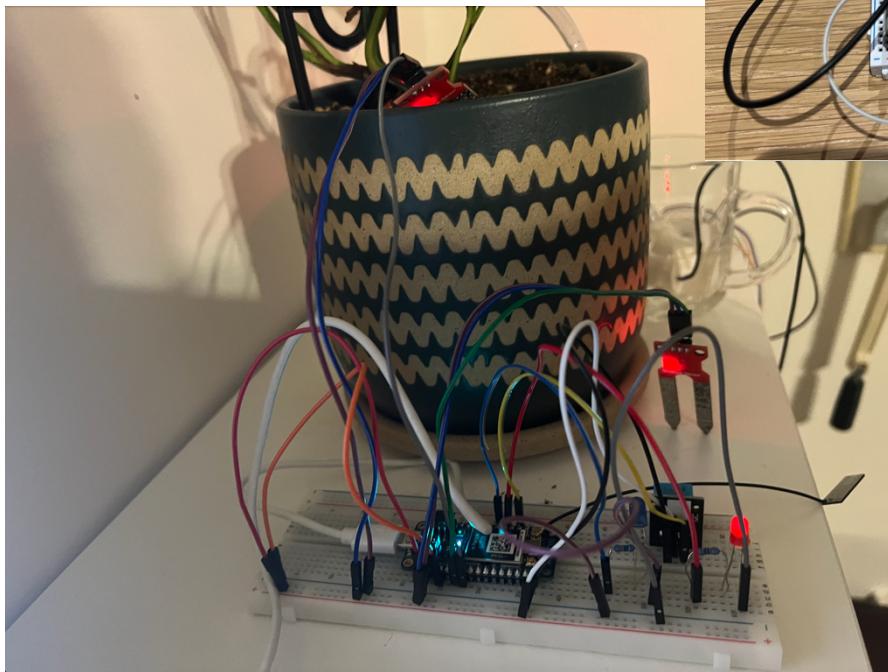
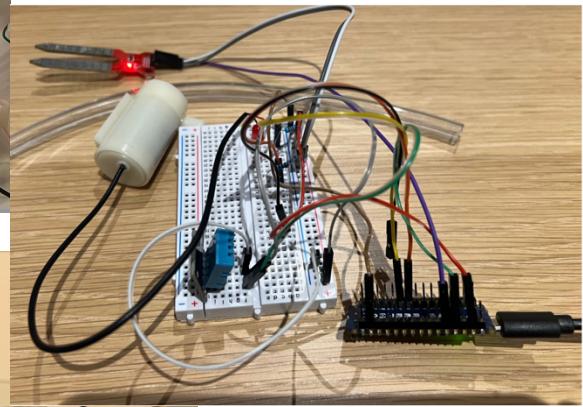
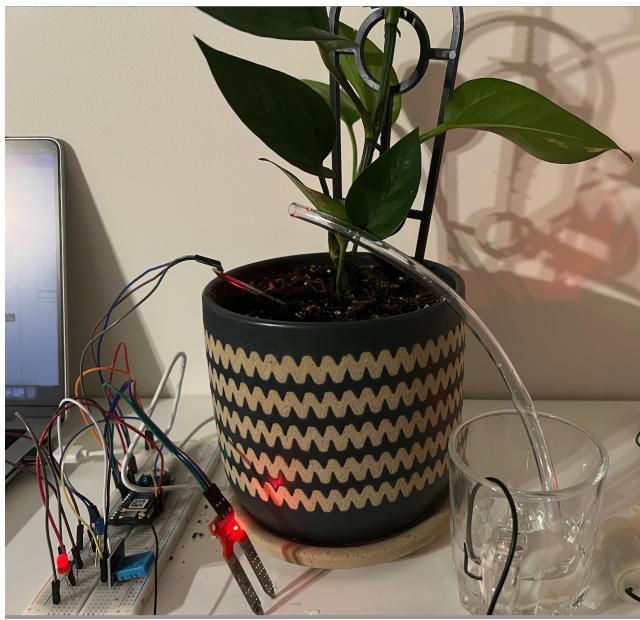
## Implementation Phases

### 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.

The screenshot shows the Arduino IDE interface with the sketch `sketch_oct13a.ino` open. The code includes definitions for WiFi, PubSubClient, and DHT libraries, and pin mappings for soil moisture and pump pins. The Serial Monitor window displays log messages indicating the connection to WiFi and MQTT brokers, and the publication of sensor data for a single plant across multiple topics (temperature, soil moisture) at different intervals.

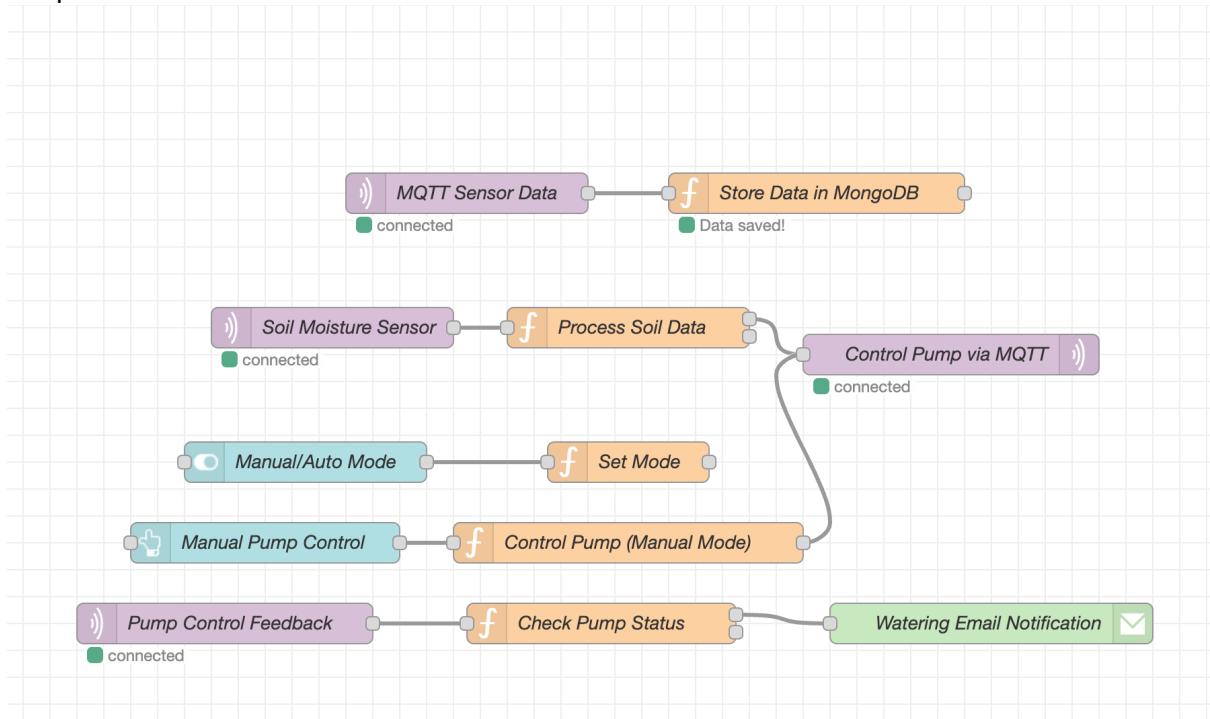
```
#include <WiFiNINA.h>
#include <PubSubClient.h>
#include <DHT.h>

// Pin Definitions
#define DHTPIN 2
#define SOIL_MOISTURE_PIN A1
#define PUMP_PIN 5
#define DHTTYPE DHT22

Connected to WiFi.
Connecting to MQTT broker...
Connected to MQTT broker.
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1019}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1022}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1023}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1016}
Connecting to MQTT broker...
Connected to MQTT broker.
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1021}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1022}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1017}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1016}
Connecting to MQTT broker...
Connected to MQTT broker.
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1020}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1023}
Published sensor data: {"plant": "Plant 1", "temperature": 23.90}
Published soil data: {"plant": "Plant 1", "soilMoisture": 1023}
Connecting to MQTT broker...
Connected to MQTT broker.
```

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

Output:



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

DATABASES: 6 COLLECTIONS: 11

**PlantData.PlanData**

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 456B TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 4KB

**Find** Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

**INSERT DOCUMENT**

**Filter** Type a query: { field: 'value' } **Reset** **Apply** **Options**

QUERY RESULTS: 1-4 OF 4

```
_id: ObjectId('670b9d75017e625cbc786629')
plant: "Plant 1"
status: "off"
temperature: 23
soilMoisture: 500
timestamp: 2024-10-13T10:14:13.044+00:00

_id: ObjectId('670b9d78017e625cbc78662a')
plant: "Plant 1"
status: "off"
temperature: 23.1
soilMoisture: 500
timestamp: 2024-10-13T10:14:16.157+00:00
```

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:

### Negin Watering System Control Panel

**Mode Control**

[Switch to Manual Mode](#)

**Watering System Control**

[Turn On Watering](#) [Turn Off Watering](#) [Get Watering Status](#)

**Plant Control and Data**

[Get Data for Plant](#) [Check if Plant Needs Water](#)

**Sensor Data and Filtering**

[Get Latest Sensor Data](#)  [Filter Data](#)

**Watering System Intervals**

[Get Watering Intervals](#)

## Filtering:

Sensor Data and Filtering

Get Latest Sensor Data    13/10/2024, 07:35 am    13/10/2024, 12:52 am    Filter Data

Watering System Intervals

Get Watering Intervals

```
{  
  "_id": "670b9ec7986a8276175eed2b",  
  "plant": "Plant 1",  
  "soilMoisture": 1023,  
  "timestamp": "2024-10-13T10:19:51.987Z",  
  "__v": 0  
}
```

## Plant control and its last data:

Plant Control and Data

Plant A    Get Data for Plant    Check if Plant Needs Water

Sensor Data and Filtering

Get Latest Sensor Data    13/10/2024, 07:35 am    13/10/2024, 12:52 am    Filter Data

Watering System Intervals

Get Watering Intervals

```
{  
  "_id": "670a2f3b6a51a47cde93efb9",  
  "plant": "Plant A",  
  "temperature": 28,  
  "soilMoisture": 385,  
  "timestamp": "2024-10-12T08:11:01.016Z",  
  "__v": 0  
}
```

Plant A needs water.

Plant Control and Data

Plant A    Get Data for Plant    Check if Plant Needs Water

Sensor Data and Filtering

Get Latest Sensor Data    13/10/2024, 07:35 am    13/10/2024, 12:52 am    Filter Data

Watering System Intervals

Get Watering Intervals

Plant D: Once every few days  
Plant 1: Once every few days  
plant1: Once a week  
Plant A: Once every few days  
plant2: Once a week  
Plant B: Once every few days  
Plant E: Once every few days  
Plant C: Once every few days

## Get watering status:

**Watering System Control**

**Plant Control and Data**

**Sensor Data and Filtering**

**Watering System Intervals**

Plant A: Needs Water? Yes  
 Plant 1: Needs Water? No  
 Plant D: Needs Water? Yes  
 Plant C: Needs Water? Yes  
 Plant E: Needs Water? Yes  
 plant1: Needs Water? No  
 Plant B: Needs Water? No  
 plant2: Needs Water? No

This page says  
Mode set to: manual

**Mode Control**

**OK**

**Manual Watering Mode**

**Plant 1**

Turn ON Plant 1   Turn OFF Plant 1

**Plant A**

Turn ON Plant A   Turn OFF Plant A

**Plant B**

Turn ON Plant B   Turn OFF Plant B

**Plant C**

Turn ON Plant C   Turn OFF Plant C

**Plant D**

Turn ON Plant D   Turn OFF Plant D

**Plant E**

Turn ON Plant E   Turn OFF Plant E

**plant1**

Turn ON plant1   Turn OFF plant1

**plant2**

Turn ON plant2   Turn OFF plant2

**Back to Main Page**

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

## Postman Testing:

The screenshot shows the Postman application interface. At the top, there are workspace and more options, followed by a search bar and various icons. The main area shows a collection named "New Collection" with a "New Request" button. A "GET" request is selected with the URL "http://localhost:3000/plants". The "Send" button is highlighted in blue. Below the request, the "Params" tab is active, showing a table with one row. The "Body" tab is selected, showing a "Pretty" JSON response with three objects representing plants. The response status is "200 OK" with a duration of 457 ms and a size of 208.93 KB.

```
1 [  
2 {  
3   "_id": "670a2e30c34c906aa3825a16",  
4   "plant": "Plant A",  
5   "temperature": 16,  
6   "soilMoisture": 195,  
7   "timestamp": "2024-10-09T01:07:31.411Z",  
8   "__v": 0,  
9   "status": "on"  
10 },  
11 {  
12   "_id": "670a2e32c34c906aa3825a19",  
13   "plant": "Plant A",  
14   "temperature": 18,  
15   "soilMoisture": 224,  
16   "timestamp": "2024-10-10T02:01:31.069Z",  
17   "__v": 0  
18 },  
19 {  
20   "_id": "670a2e32c34c906aa3825a1b",  
21   "plant": "Plant C",  
22   "temperature": 21,
```

The screenshot shows the Postman application interface. At the top, there are workspace and more options, followed by a search bar and various icons. The main area shows a collection named "New Collection" with a "New Request" button. A "GET" request is selected with the URL "http://localhost:3000/mode". The "Send" button is highlighted in blue. Below the request, the "Params" tab is active, showing a table with one row. The "Body" tab is selected, showing a "Pretty" JSON response with one object. The response status is "200 OK" with a duration of 5 ms and a size of 274 B.

```
1 {  
2   "mode": "auto"  
3 }
```

HTTP New Collection / New Request

Save Share

GET http://localhost:3000/data/latest Send

Params Auth Headers (8) Body Scripts Tests Settings

Key	Value	Description
Key	Value	Description

Body 200 OK 57 ms 380 B ⓘ

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "670b9ec7986a8276175eed2b",
3   "plant": "Plant 1",
4   "soilMoisture": 1023,
5   "timestamp": "2024-10-13T10:19:51.987Z",
6   "__v": 0
7 }
```

Workspaces More

PUT New f GET New f GET New f + New Environment

HTTP New Collection / New Request

Save Share

GET http://localhost:3000/watering/status Send

Params Auth Headers (8) Body Scripts Tests Settings

Key	Value	Description
Key	Value	Description

Body 200 OK 117 ms 569 B ⓘ

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "plant": "Plant A",
4     "needsWater": "Yes"
5   },
6   {
7     "plant": "plant1",
8     "needsWater": "No"
9   },
10  {
11    "plant": "Plant 1",
12    "needsWater": "No"
13  },
14  {
15    "plant": "plant2",
16    "needsWater": "No"
17  },
18  {
19    "plant": "Plant B",
20    "needsWater": "No"
21  },
22  {
```

The screenshot shows the Postman interface with a successful JSON response. The URL is `http://localhost:3000/watering/intervals`. The response body is:

```
1 [  
2   {  
3     "plant": "Plant D",  
4     "interval": "Once every few days"  
5   },  
6   {  
7     "plant": "Plant 1",  
8     "interval": "Once every few days"  
9   },  
10  {  
11    "plant": "plant1",  
12    "interval": "Once a week"  
13  },  
14  {  
15    "plant": "Plant A",  
16    "interval": "Once every few days"  
17  },  
18  {  
19    "plant": "plant2",  
20    "interval": "Once a week"  
21  },  
22 ]
```

The screenshot shows the Postman interface with a simple text response. The URL is `http://localhost:3000/plant/plant1/needs-water`. The response body is:

```
1 plant1 does not need water.
```

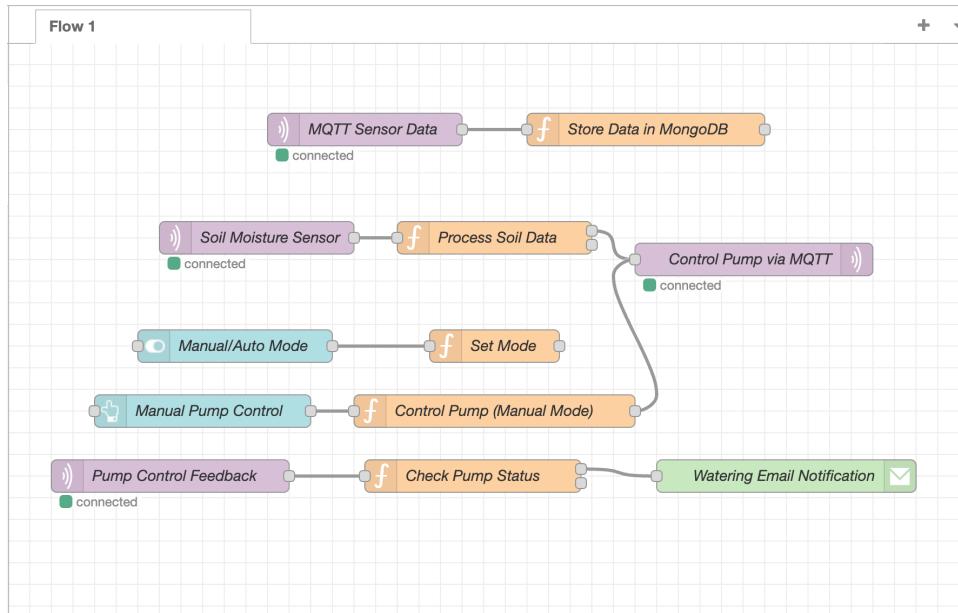
The screenshot shows the Postman interface with a JSON response. The URL is `http://localhost:3000/plants/names`. The response body is:

```
1 [  
2   "Plant 1",  
3   "Plant A",  
4   "Plant B",  
5   "Plant C",  
6   "Plant D",  
7   "Plant E",  
8   "plant1",  
9   "plant2"  
10 ]
```

### 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:

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE  
File: settings.js Modified  
UW PICO 5.09  
/* The following property can be used to set predefined values in Global Context.  
 * This allows extra node modules to be made available with in Function node.  
 * For example, the following:  
 *  
 *     functionGlobalContext: {  
 *         os: require('os'), // Existing os module  
 *         mongodb: require('mongodb') // MongoDB client added to global context  
 *     }  
  
Get Help WriteOut Read File Prev Pg Cut Text Cur Pos  
Exit Justify Where is Next Pg Uncut Text Cur Pos To Spell
```

Functions:

Edit function node

Delete Cancel Done

**Properties**

Name: Store Data in MongoDB

Setup On Start **On Message** On Stop

```

1 const { MongoClient, ObjectId } = global.get('mongodb'); // Access MongoDB from global context
2
3 const uri = 'mongodb+srv://neginpakrooh:7jRFiBfCPUZnK9jm@sit314.wplq9.mongodb.net/?retryWrites=true&w=majority';
4 const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });
5
6 async function insertData(payload) {
7   try {
8     await client.connect();
9     const db = client.db('PlantData');
10    const collection = db.collection('PlantData');
11
12    const dataToInsert = {
13      _id: new ObjectId(), // Generate a unique ObjectId
14      plant: payload.plant || 'A', // Default to 'A' if plant is not provided
15      status: 'off', // Default status set to 'off'
16      temperature: payload.temperature || 25, // Default to 25°C if not provided
17      soilMoisture: payload.soilMoisture || 500, // Default to 500 if not provided
18      timestamp: new Date(), // Current timestamp
19    };
20
21    // Insert the data into MongoDB
22    const result = await collection.insertOne(dataToInsert);
23    node.status({ fill: 'green', shape: 'dot', text: 'Data saved!' });
24    msg.payload = { success: true, insertedId: result.insertedId };
25
26  } catch (err) {
27    node.error(`MongoDB Error: ${err.message}`);
28    msg.payload = { success: false, error: err.message };
29  } finally {
30    client.close();
31    return msg;
32  }
33}
34
35 // Call the insert function and handle promise resolution/rejection
36 insertData(msg.payload).then(result => {
37   return msg;
38 }).catch(error => {
39   return msg;
40 });
41

```

Edit function node

Delete Cancel Done

**Properties**

Name: Process Soil Data

Setup On Start **On Message** On Stop

```

1 let moisture = msg.payload.soilMoisture;
2
3 if (global.get('mode') === 'auto') {
4   if (moisture < 500) {
5     msg.payload = { action: 'on' };
6   } else {
7     msg.payload = { action: 'off' };
8   }
9   return [msg, null];
10 } else {
11   return [null, null]; // Ignore in manual mode
12 }

```

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 unit's 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!