COLLEGE CODE: 8107

COURSE: INTERNET OF THINGS

PHASE III: PROJECT SUBMISSION

PROJECT TITLE: Smart Water Fountains

TEAM MEMBERS DETAILS:

ABISHEK. C

HARISH. S

NAVEEN KUMAR. S

NAVAJEEVAN. S

VIJAY. V.S

Development Part-1:

○ IoT Device Selection:

In this step, we need to carefully choose the IoT devices that will be used in our smart water fountain project.

Consider factors such as the type of sensors (flow rate sensors, pressure sensors), microcontrollers (like Arduino or Raspberry Pi), and communication protocols (e.g., Wi-Fi, Bluetooth) needed for the project.

Ensure that the selected devices are compatible with the hardware requirements (e.g., power source, connections) and communication needs (e.g., Wi-Fi range, data transmission rates).

○ IoT Device Deployment:

After selecting the devices, physically install and deploy them on the smart water fountains.

Properly secure and position the devices so they can accurately collect data. For example, flow rate sensors should be placed in the water supply line of the fountain.

Pay attention to the durability of the installation, as the devices will be exposed to outdoor conditions.

○ Sensor Integration:

Connect the deployed sensors to the IoT devices (microcontrollers). This involves wiring and ensuring that the connections are secure.

Test the sensors to ensure that they are providing accurate data.

This task is crucial as the quality of data collected depends on the proper integration of sensors.

○ Python Script Development:

Write Python scripts to run on the IoT devices. These scripts will handle data collection, processing, and communication.

Develop algorithms within the scripts to process the sensor data in real-time. For example, create algorithms to calculate flow rates, detect pressure variations, and identify anomalies.

Writing clean and well-documented code is essential for maintainability.

- Data Transmission:

Implement data transmission protocols within the Python scripts to securely send sensor data from the IoT devices to the central IoT platform or cloud-based system.

Ensure that the data transmission is reliable and encrypted to protect the information as it travels from the devices to the central system.

- Testing and Calibration:

Conduct thorough testing of both the IoT devices and the Python scripts. This testing phase aims to verify that the devices and code are functioning as intended.

Calibrate the sensors to ensure their data is accurate and reliable in real-world conditions. Calibration may involve adjusting sensor settings to match the actual water flow or pressure.

- Integration with IoT Platform:

Integrate the IoT devices and Python scripts with the central IoT platform or cloud-based system. This connection allows data to be collected and stored in a centralized location.

Ensure that data is collected and stored correctly in the platform's database. The data should be structured and organized for easy access and analysis.

- User Interface Development:

If your project includes user interfaces, develop them during this phase. These interfaces could be mobile apps or web dashboards.

Ensure that the interfaces are user-friendly, allowing users and administrators to access real-time data effectively. Make sure that the interfaces are visually appealing and easy to navigate.

- ## Real-Time Monitoring and Alerts:

Develop a real-time monitoring dashboard for facility managers, enabling them to access up-to-the-minute information about fountain usage, maintenance recommendations, and water quality.

Integrate alert mechanisms within the system to notify facility managers when immediate action is needed. For example, generate alerts for low flow rates or water quality issues.

- ## Documentation:

Create comprehensive documentation that covers the deployment of IoT devices and the development of Python scripts. This documentation should include detailed instructions on device setup, software installation, calibration procedures, and data transmission protocols.

- ## Testing and Validation:

Thoroughly test the integrated system, including IoT devices, Python scripts, and user interfaces.

Use test cases to validate that the system meets the project requirements and objectives. This ensures that the system is functioning as expected.

- ## Troubleshooting:

We need to be prepared to identify and resolve any issues or errors that may arise during testing and validation.

Python Script:

```
import random
import time
import requests
import json

# Simulated IoT device with sensors
class SmartWaterFountain:
    def _init_(self, device_id, location):
        self.device_id = device_id
        self.location = location
        self.flow_rate_sensor = FlowRateSensor()
        self.pressure_sensor = PressureSensor()

    def collect_data(self):
        flow_rate = self.flow_rate_sensor.read_data()
        pressure = self.pressure_sensor.read_data()
        return {
            'device_id': self.device_id,
            'location': self.location,
            'flow_rate': flow_rate,
            'pressure': pressure
        }
```

```
class FlowRateSensor:
    def read_data(self):
        # Simulate flow rate data (for demonstration, you would interface with a real sensor)
        return round(random.uniform(0.5, 2.5), 2)  # Liters per minute

class PressureSensor:
    def read_data(self):
        # Simulate pressure data (for demonstration, you would interface with a real sensor)
        return round(random.uniform(30, 50), 2)  # PSI

# Replace with your actual IoT platform endpoint
API_ENDPOINT = 'https://your-iot-platform-api.com/data'

# Create smart water fountain instances
fountain1 = SmartWaterFountain(device_id='fountain1', location='Park A')
fountain2 = SmartWaterFountain(device_id='fountain2', location='Park B')

# Main data collection loop
while True:
    data1 = fountain1.collect_data()
    data2 = fountain2.collect_data()

    # Send data to the IoT platform (replace with your platform's API)
```

```
    data_to_send = [data1, data2]


    try:
        response = requests.post(API_ENDPOINT, data=json.dumps(data_to_send), headers={'Content-Type': 'application/json'})
        if response.status_code == 200:
            print("Data sent successfully.")
        else:
            print(f"Failed to send data. Status code: {response.status_code}")
    except requests.exceptions.RequestException as e:
        print(f"Error sending data: {e}")

    # Adjust the data collection frequency
    time.sleep(60)
```