**Project Artifact Report**

**--- Project Title: Real-Time Sensor Monitoring System Using Arduino and Raspberry Pi with MQTT Integration**

**Author:** Aruj Punia

**1. Project Overview**

**1.1 Background**

As the need for remote monitoring and real-time data tracking increases, sensor systems that provide environmental data become critical. This project explores a solution for monitoring environmental and physiological data, such as temperature, humidity, air quality, and pulse rate, using an Arduino Nano as the sensor node and a Raspberry Pi as the gateway to display the data. MQTT, a lightweight messaging protocol, facilitates communication between the devices, providing real-time updates to a graphical user interface on the Raspberry Pi.

**1.2 Existing Work**

Existing systems often use complex setups requiring proprietary software or closed systems for sensor monitoring. This project leverages open-source technologies and widely available hardware, offering an accessible and cost-effective alternative for real-time environmental monitoring.

**1.3 Problem Statement**

In critical environments where data monitoring is essential—such as healthcare, laboratories, and smart homes—there is a need for reliable, real-time monitoring systems. Most existing solutions lack customization, are costly, or have complex deployment requirements. This project aims to develop a simple, modular, and easily deployable system for real-time sensor monitoring with features like data storage, graphical visualization, and remote data access.

**1.4 Requirements**

- **Real-time data monitoring** for multiple environmental parameters.

- **Wireless communication via MQTT** protocol for scalability and flexibility.

- **User-friendly interface** for data visualization.

- **Data persistence** to save sensor readings locally.

- **Cost-effectiveness** by using affordable hardware like Arduino and Raspberry Pi.

- **Customizability** for potential expansion and additional sensor integration.

**2. Design Principles**

The design follows these principles:

**- Modularity**: Each component (sensor, communication protocol, GUI) is modular, making it easier to upgrade or replace components as needed.

**- Simplicity and Efficiency:** The system is built with minimal hardware, keeping costs low and efficiency high.

**- Reliability:** MQTT was chosen for its reliability in sending messages even over low-bandwidth networks.

**- User-Centricity:** The GUI is designed to be intuitive, making it accessible even for non-technical users.

**3. Prototype Architecture**

**3.1 System Components**

- **Arduino Nano:** Functions as the main sensor node, collecting temperature, humidity, air quality, and pulse data and publishing it to the MQTT broker.

**- Raspberry Pi:** Acts as the gateway and visualization platform, receiving data from the MQTT broker and displaying it on a Tkinter-based GUI.

**- EMQX MQTT Broker:** A cloud-based broker that enables seamless communication between the Arduino and Raspberry Pi.

**- Tkinter GUI:** Built on the Raspberry Pi, this interface displays real-time sensor data, providing a clear visualization of environmental and physiological metrics.

**3.2 Data Flow**

**1. Sensor Readings:** The Arduino Nano collects data from various sensors.

**2. MQTT Publish:** The Arduino publishes the collected data to the EMQX MQTT broker.

**3. MQTT Subscribe:** The Raspberry Pi subscribes to the relevant MQTT topics and receives data updates.

**4. Data Display:** The Tkinter-based GUI on the Raspberry Pi updates to show the latest data and stores the information locally.

**4. Link to Prototype Code on GitHub**

- <https://github.com/puniaruj/SIT-210--Embedded-Systems/tree/main/PROJECT/Arduino%20Code/arduino_code>

**5. Testing Approach**

Testing was carried out across different stages:

**1. Connectivity Testing:** Verified the Arduino’s ability to publish data to the MQTT broker and the Raspberry Pi’s ability to receive it.

**2. Data Accuracy Testing:** Cross-referenced sensor data with standalone sensors to ensure the accuracy of temperature, humidity, and air quality readings.

**3. GUI Responsiveness Testing:** Assessed the GUI's responsiveness to frequent data updates to ensure smooth user interaction.

**4. Failure Testing:** Simulated network outages to evaluate system behavior and reconnection efficiency.

**5. User Testing:** Gathered feedback on the GUI layout and ease of use to make improvements in accessibility.

**6. User Manual**

**6.1 Hardware Requirements**

- Arduino Nano

- Raspberry Pi (model 3 or higher)

- Sensors (temperature, humidity, air quality, pulse)

- WiFi Module for Arduino Nano (e.g., ESP8266)

**6.2 Software Requirements**

- Python 3.7 or higher on Raspberry Pi

- Tkinter, Matplotlib, Paho-MQTT libraries installed on the Raspberry Pi

- Arduino IDE for programming the Arduino Nano

- MQTT broker credentials (EMQX or a local broker)

**6.3 Installation Instructions**

**1. Set up the Arduino Nano:**

- Connect the sensors to the Arduino Nano and upload the provided Arduino code from the GitHub repository.

- Ensure that the MQTT broker details are correctly set in the Arduino code.

**2. Set up the Raspberry Pi:**

- Clone the GitHub repository to the Raspberry Pi.

- Run `pip install -r requirements.txt` to install the necessary Python libraries.

- Open the main GUI code and update the MQTT broker details if necessary.

**3. Running the Program:**

- Power on the Arduino and ensure it has internet connectivity.

- Run the Python script on the Raspberry Pi to start the GUI and begin monitoring.

**6.4 Usage Instructions**

- Upon launching, the GUI will display temperature, humidity, air quality, and pulse data.

- To save data, click the ‘Save Data to JSON’ button in the GUI.

- To exit, close the window, which will stop the data display and MQTT connection.

---

**7. Link to Demonstration Video**

- <https://www.youtube.com/watch?v=sRD5gJ7Thes>

---

**8. Conclusion**

**8.1 Reflection**

This project presented valuable learning experiences. The major challenges included integrating multiple sensors and ensuring consistent data flow over MQTT. The biggest learning came from setting up a robust communication channel between the Arduino and Raspberry Pi, as network interruptions were common. Additionally, building a responsive and informative GUI using Tkinter and Matplotlib helped reinforce Python programming skills.

**8.2 Future Improvements**

If given a second chance, I would:

- Consider using a more powerful microcontroller that supports WiFi directly, reducing setup complexity.

- Implement data logging to a cloud-based database for remote monitoring.

- Add more complex error handling for increased robustness.

- Experiment with advanced visualization libraries to enhance the data presentation.