Exploration Project Report:

Industrial Plant cum Environmental Monitoring System

PC223

Group-6

Kartik Vyas ID: 202301003

Rishit Raj Jain ID: 202301167

Patel Harsh Nareshbhai ID: 202301192

Maheriya Harsh Prakashbhai ID: 202301470

Gori Faran Firozbhai ID: 202301209

MENTOR: Bakul Gohel

From Paper to Reality:

- First, we familiarized ourselves with the basics of the Arduino IDE software to work on the ESP32 microcontroller. This included learning how to integrate and utilize the functionality of libraries, such as the basic features of SoftwareSerial.h, writing simple programs for tasks like blinking an LED and implementing a counter, and displaying data on the serial monitor.

- As described in the document submitted for PC122, our main focus was on working with various sensors. These include:

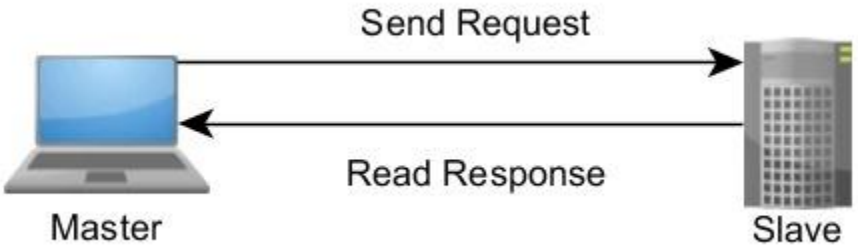
    - Temperature and Humidity Sensor (DHT22)
    - Pressure Sensor (BMP280)
    - Current and Voltage Sensor (ACS712)
    - Gas Sensor (MQ135)

- To integrate these sensors into a circuit with the ESP32, we referred to their respective documentation to understand their functionality and wiring requirements. We also learned how to implement sensor readings in the ESP32 code. These steps were primarily learned through instructional YouTube videos.

- We initially wrote code to use a single ESP32 microcontroller to read data from all the sensors and send it to the DWIN display. However, after a discussion with our mentor, Bakul Sir, it was suggested to use two ESP32 microcontrollers with a MODBUS protocol for a master-slave implementation. This approach simplifies the coding and debugging process, with each ESP32 dedicated to a specific task.

❖ **Integrating MODBUS PROTOCOL in our project:**

- Modbus is an industrial level request-response protocol implemented using a master-slave relationship. In a master-slave relationship, communication always occurs in pairs—one device must initiate a request and then wait for a response—and the initiating device (the master) is responsible for initiating every interaction.



- There are four types of data blocks available in esp32 in order to save values like integer and Boolean values.

Memory Block	Data Type	Master Access	Slave Access
Coils	Boolean	Read/Write	Read/Write
Discrete Inputs	Boolean	Read - only	Read/Write
Holding Registers	Unsigned Word	Read/Write	Read/Write
Input Registers	Unsigned Word	Read - only	Read/Write

- These blocks give you the ability to restrict or permit access to different data elements and also to provide simplified mechanisms at the application layer to access different data types.
- Data Addressing Ranges for these data blocks using prefix.

Data Blocks	Prefix
Coil	0
Discrete Inputs	1
Input Registers	3
Holding Registers	4

- Project is mainly dealing with Holding register.
- Standard function codes which we have used in our project to read and write holding register.

Code	Description
3	Read Multiple Register
16	Write Multiple Register

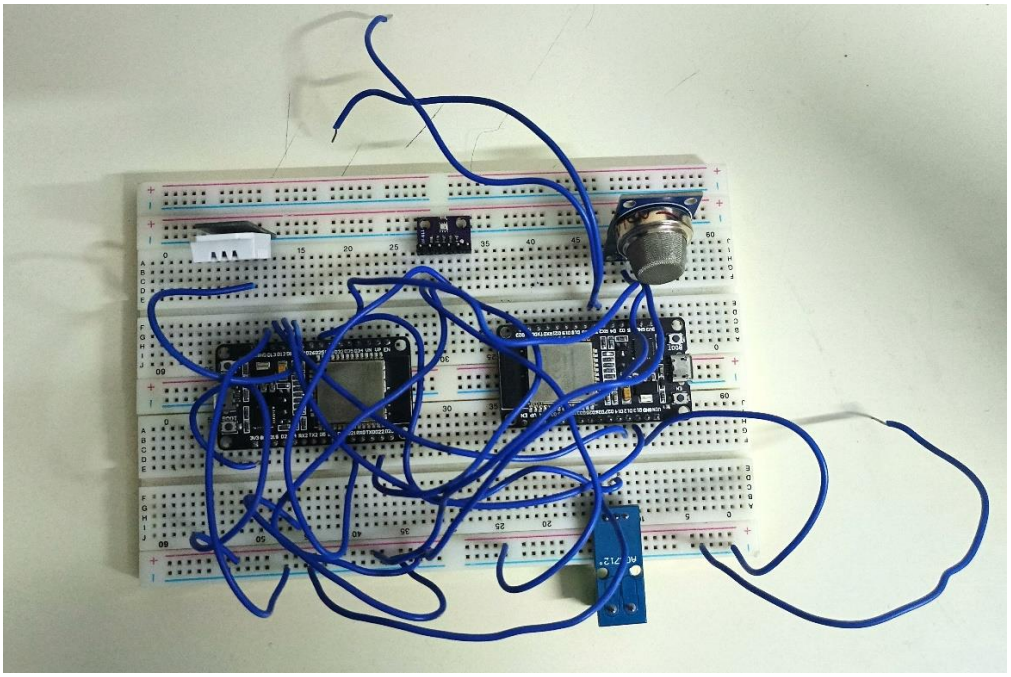
- There are mainly two sub part of the modbus communication which works on holding register in order to send integer values.

- Modbus TCP/IP and Modbus RTU.
    - Modbus TCP/IP communicate with IP addresses of the esp32.
    - Where else Modbus RTU communicate through serial line RX, TX.

- Modbus Ascii
    - It is also a sub-part of Modbus communication used to send character values, which was not favourable for us in this project.

- In our project, one ESP32 (slave) reads data from all the sensors and writes this data to the holding registers of another ESP32 (master) using the MODBUS RTU protocol. In the code, we utilized the ModbusRTU.h library, which provides the writeHreg() function to write data to the master ESP32. The master ESP32 then reads the data using the readHreg() function. At this stage, we also created a GitHub repository that includes our final code for the MODBUS implementation.
- GitHub repository Link: https://github.com/Harsh97120/Exploration.git

- Reference : Link.



❖ Integrating Node JS with Database:

- We used a Node.js server to fulfill our goal of connecting to the database and sending data from the HiveMQ MQTT broker through CRUD operations. We utilized the **Express** package for CRUD operations, the **cors** and **mqtt** packages for the MQTT protocol, and the **shortid** package to assign unique IDs to incoming data. Additionally, we used the **mongoose** package to interact with MongoDB. The data is transmitted in JSON format.

❖ MQTT:

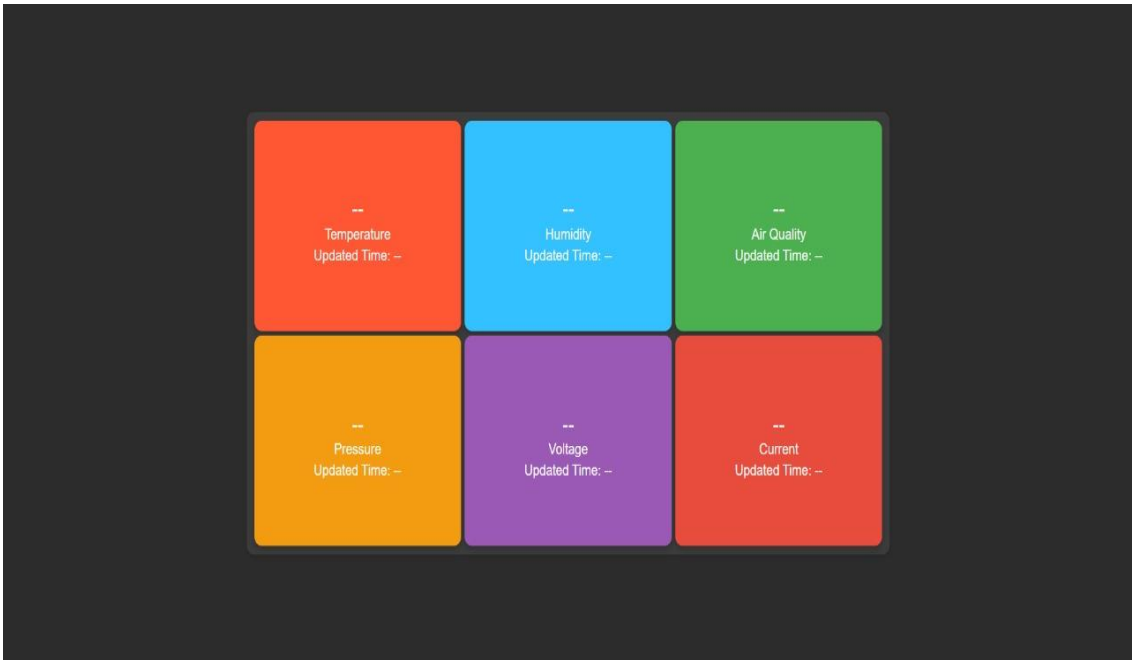
- MQTT (originally an initialism of MQ Telemetry Transport) is an Industrial level lightweight, publish-subscribe, machine to
machine network protocol for message queue/message queuing service. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth, such as in the Internet of Things (IoT).

- Our ESP32 (master) acts as the publisher of sensor values on a specific topic called **Exploration_sensordata**, which it receives from the slave ESP32. This data is then sent to the HiveMQ MQTT broker. The Node.js server serves as the subscriber to the **Exploration_sensordata** topic in the project in order to receive data.

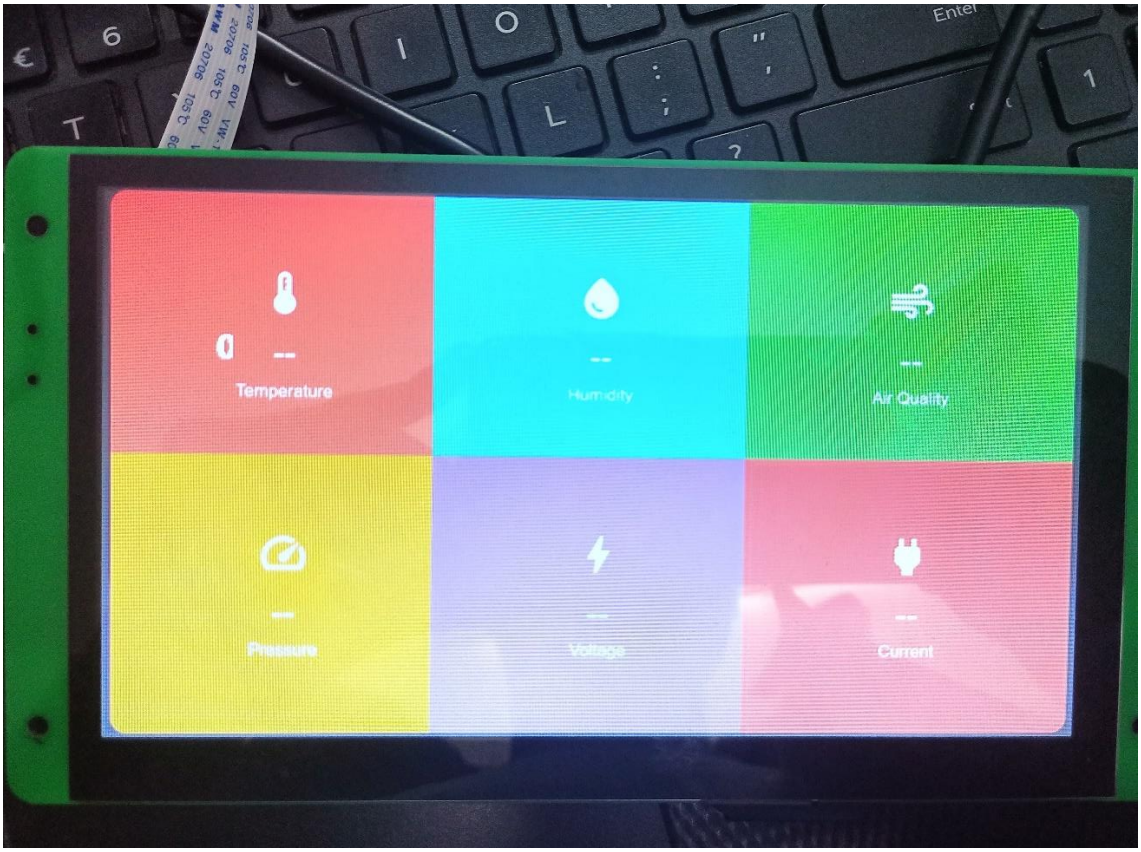
❖ Integrating Frontend and Dwin display:

- We have developed the frontend for our project using HTML, CSS, and JavaScript. It includes sections for each parameter, allowing us to display data on our laptop effectively.

- In a DWIN display, we first need to create all the functional files for the desired touch functionality, including images. These files are created using the DGUS software and then placed in a folder named 'DWIN_SET.' This folder is copied to an SD card, which is then inserted into the SD card slot of the DWIN display. Following these steps, we successfully included our frontend along with the desired touch functionality into the DWIN display.



Reference Video Links: ----
https://drive.google.com/file/d/142LCPgsGcXPa7UEWSxEUH4AP4H2hRro4/view?usp=drive_link
DWIN displays - YouTube

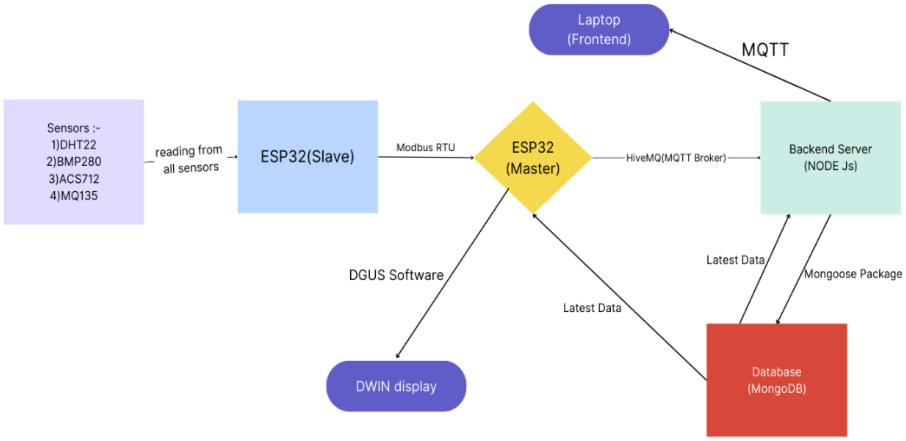


❖ **<u>Problems while executing the Project</u>**:

- These are some of the problems we identified in the project:
- **Issues with uploading code to the ESP32**
  Solution: We discovered that we needed to press the boot button on the ESP32 every time to successfully upload the code.
- **The asynchronous nature of the ESP32 when sending multiple data points to another ESP32 via Modbus RTU**
  Solution: Initially, we searched online resources, including YouTube videos, but found limited information on Modbus RTU. Therefore, we reached out to our senior, Aditya Makwana, who had worked on a similar project using Modbus RTU. He suggested using a boolean value for each data point in the code to address the issue.
- **Challenges with connecting to the MQTT broker, initially using Mosquitto, then EMQX, and finally settling on the HiveMQ broker**
  Solution: After trying several brokers, none of which worked, we sought advice from our senior, Ruturaj. He recommended switching to HiveMQ, which resolved the connection issues.
- **Incorrect wiring in the circuit, leading to connectivity issues**
  Solution: We tried replacing the wires, as using the same type of wire repeatedly seemed to cause persistent connectivity issues.

Industrial IoT Framework

Industrial Plant cum
Environnmental Monitoring system



❖ Contributions:

- **Patel Harsh Nareshbhai (202301192)** worked on sending data from an MQTT broker to a database using Node.js and wrote the Node.js code for this functionality.

- **Gori Faran Firozbhai (202301209)** worked on integrating the ModBus protocol, wrote the code for the slave ESP32, and established a connection between two ESP32 devices using the ModBus protocol.

- **Maheriya Harsh Prakashbhai (202301470)** worked on integrating the DWIN display into the project, including the frontend and touch functionality, and worked with DGUS software for this integration.

- **Kartik Vyas (202301003)** worked on sending data from the master ESP32 to the MQTT broker and wrote the code for the master ESP32.

- **Rishit Raj Jain (20230167)** worked on creating the frontend and establishing the hardware connection between the sensors and the ESP32.

❖ **Special Thanks**

- Ramesh Prajapati
- Aditya Makwana (4th year student)
- Ruturaj (4th year student and TA of IOT subject)
- Bakul Gohel