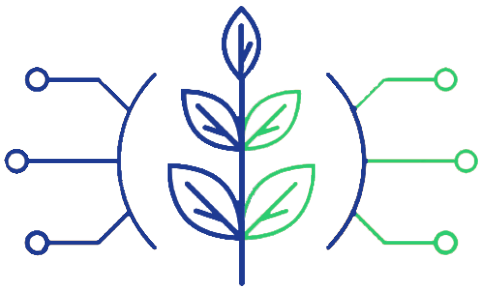
AMERICAN UNIVERSITY OF BEIRUT

Maroun Semaan Faculty of Engineering and Architecture
Department of Electrical and Computer Engineering



IoT Based Smart Agriculture System Using ESP32

EECE 425 Project
Final Report
Fall 2024-2025

by

Mohammad Al-Katranji ECE
Mostafa Kassem CCE

Professor Mazen Saghir

December 11, 2024

Contents

0.1 Introduction

A looming water crisis faces the world due to the explosive growth of the global population, primarily caused by the rise in demand for food production. Agriculture represents about 85% of global freshwater usage; thus, making farming efficient in water usage is crucial for sustainability [1]. Traditional irrigation systems often lead to over-watering or under-watering, which affects crop yield and results in resource wastage.

IoT technology in smart agriculture systems provides a solution by enabling precise monitoring of environmental and soil conditions, allowing for automated irrigation control based on real-time data. In this context, our project uses the ESP32 microcontroller along with a couple of sensors to optimize irrigation and help manage water supplies efficiently, especially in regions where water resources are limited.

The figure 1 below shows statistics about the crisis of water in 2050

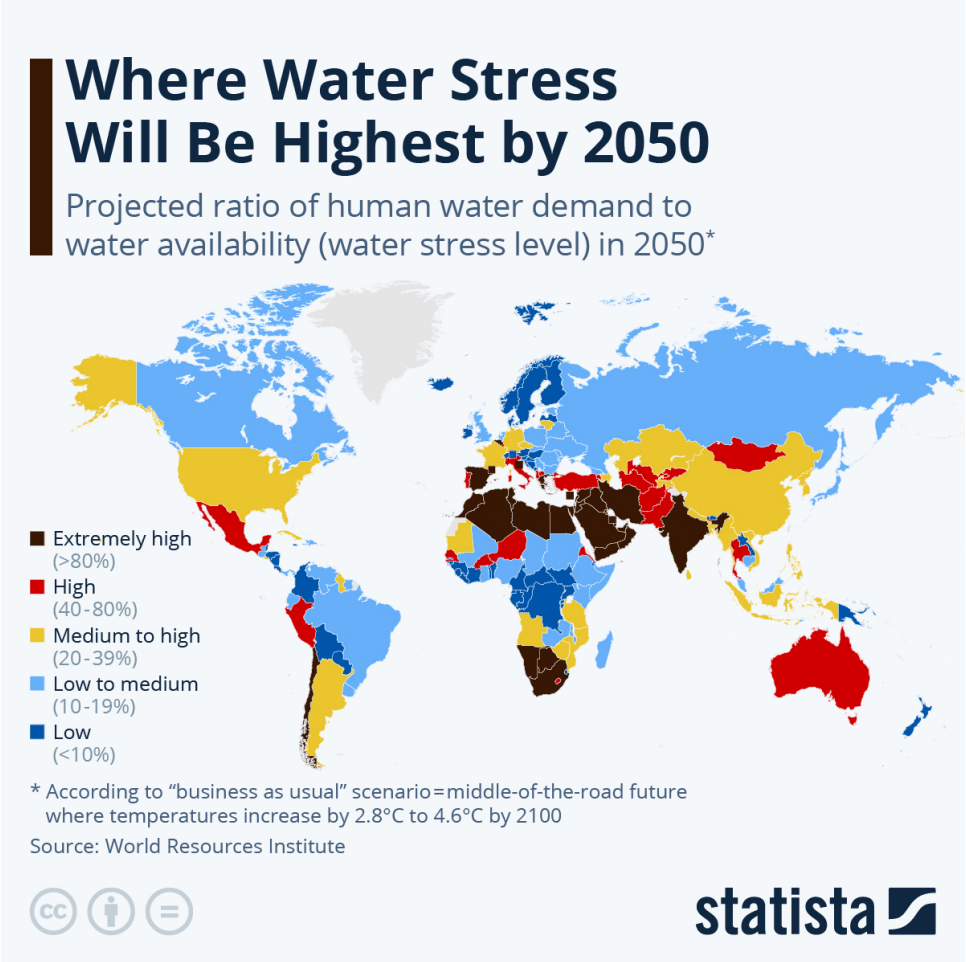


Figure 1: Where Water Stress Will Be Highest by 2050 [2]

0.2 System Design and Architecture

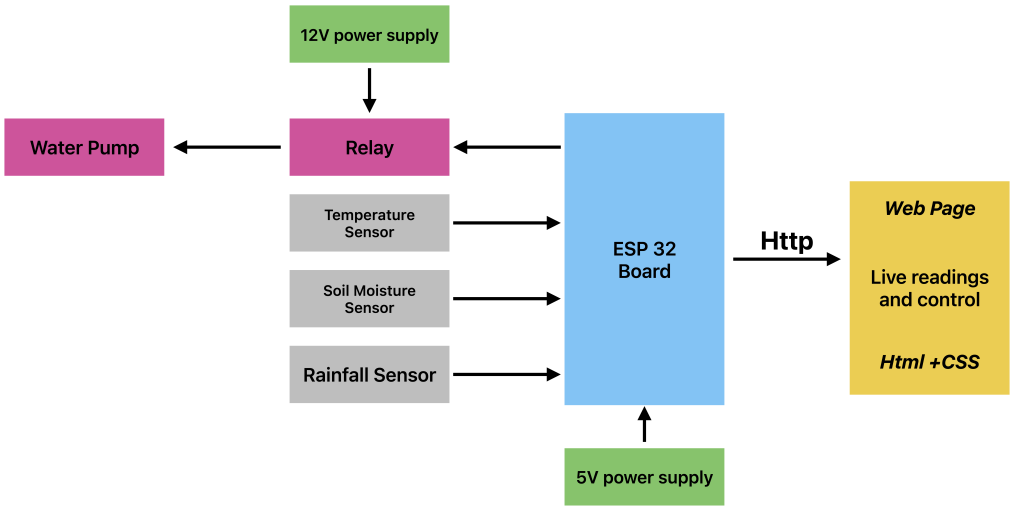


Figure 2: Hardware and Software Block Diagram

In this section, to better explain our proposed design, we provide a detailed block diagram and a flow chart.

Figure 2 explains the design architecture, showing the interaction between the hardware and software components of our system. We are using two analog sensors, the rainfall and soil moisture sensors, and an I2C temperature sensor. The output of those sensors is fed into the ESP32 (powered by a USB connection) where the decision making shown in Fig. 4 will be implemented. On the web page while using the HTTP protocol, the sensors readings are live streamed and displayed as shown in Fig. 5. Finally, the water pump will be activated using a relay connected to a 12V power supply to irrigate the plant.

First, it checks if it is raining; if rain is detected, the water pump remains off, as natural rainfall provides sufficient water. If no rain is detected, the system proceeds to check the soil moisture level. If the soil is wet, the pump is kept off. However, if the soil is dry, the system further evaluates whether the temperature falls within an ideal range for irrigation. If the temperature is unsuitable, the water pump remains off. Only if all conditions which are no rain, dry soil, and ideal temperature are met the system activate the water pump to irrigate the soil.

After executing the decision, the system waits for 10 minutes before re-evaluating the environmental conditions to ensure efficient and timely irrigation. This process continuously repeats, allowing the system to dynamically adapt to changes in environmental conditions.

To determine the threshold under which the soil is said to be dry or wet and to estimate whether the temperature is suitable for irrigation we based our thresholds on [2] and [3] papers. They tend to choose the temperature threshold to be between 25 and 27

degree Celsius and the soil moisture to be 30%. However, the type of crops planted affect the threshold values. This is due to the fact that various crops have varying water requirements. For instance, 5000 liters of water are needed in a paddy field to produce 1 kg of rice. Also, different crops have variable temperature thresholds and this is mainly effected by the season (summer or winter) and by the region of planting.

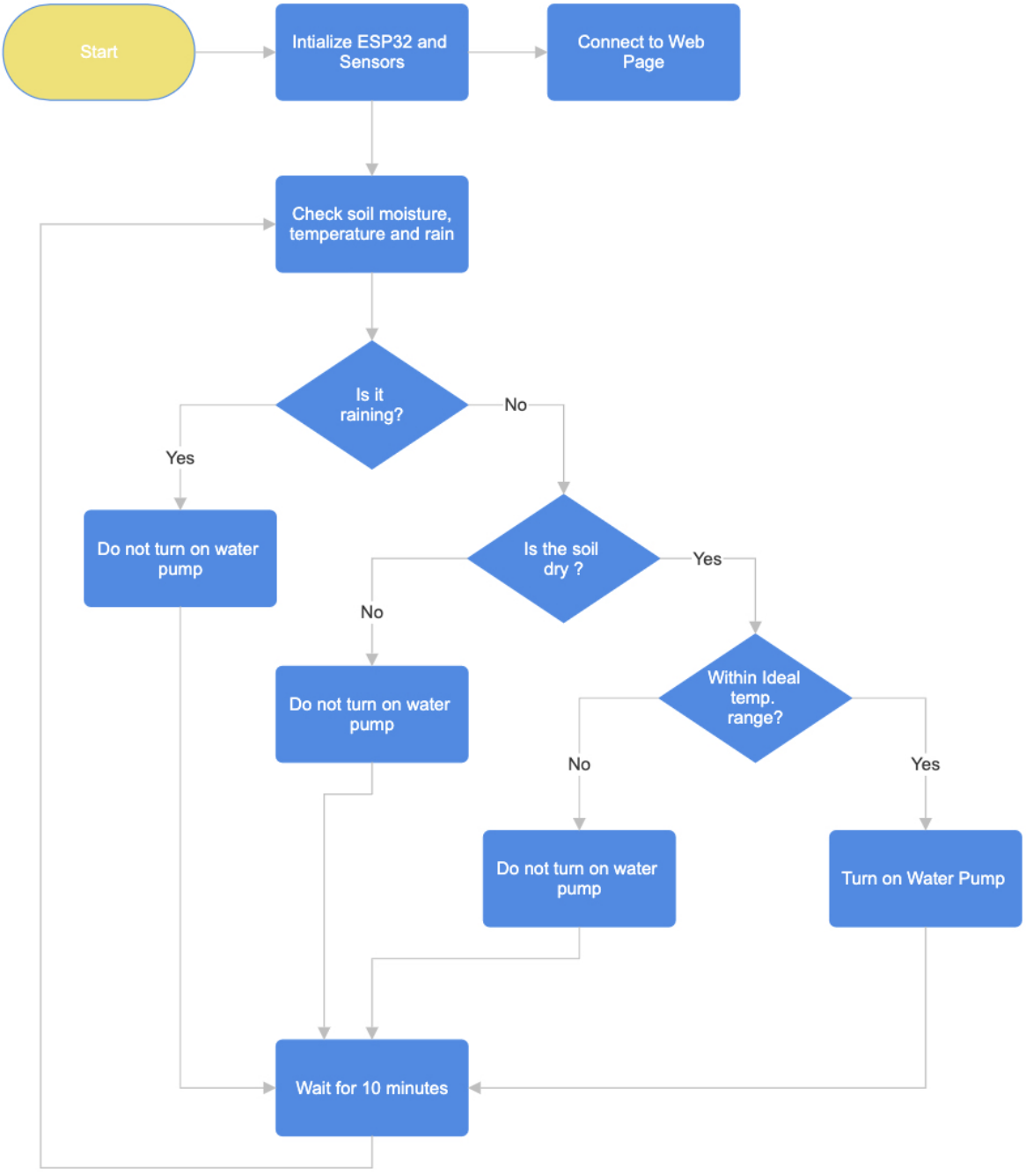


Figure 3: Decision-Making Flowchart

0.3 Hardware

The system utilizes various hardware components for environmental monitoring and control, interfaced with the ESP32 microcontroller:

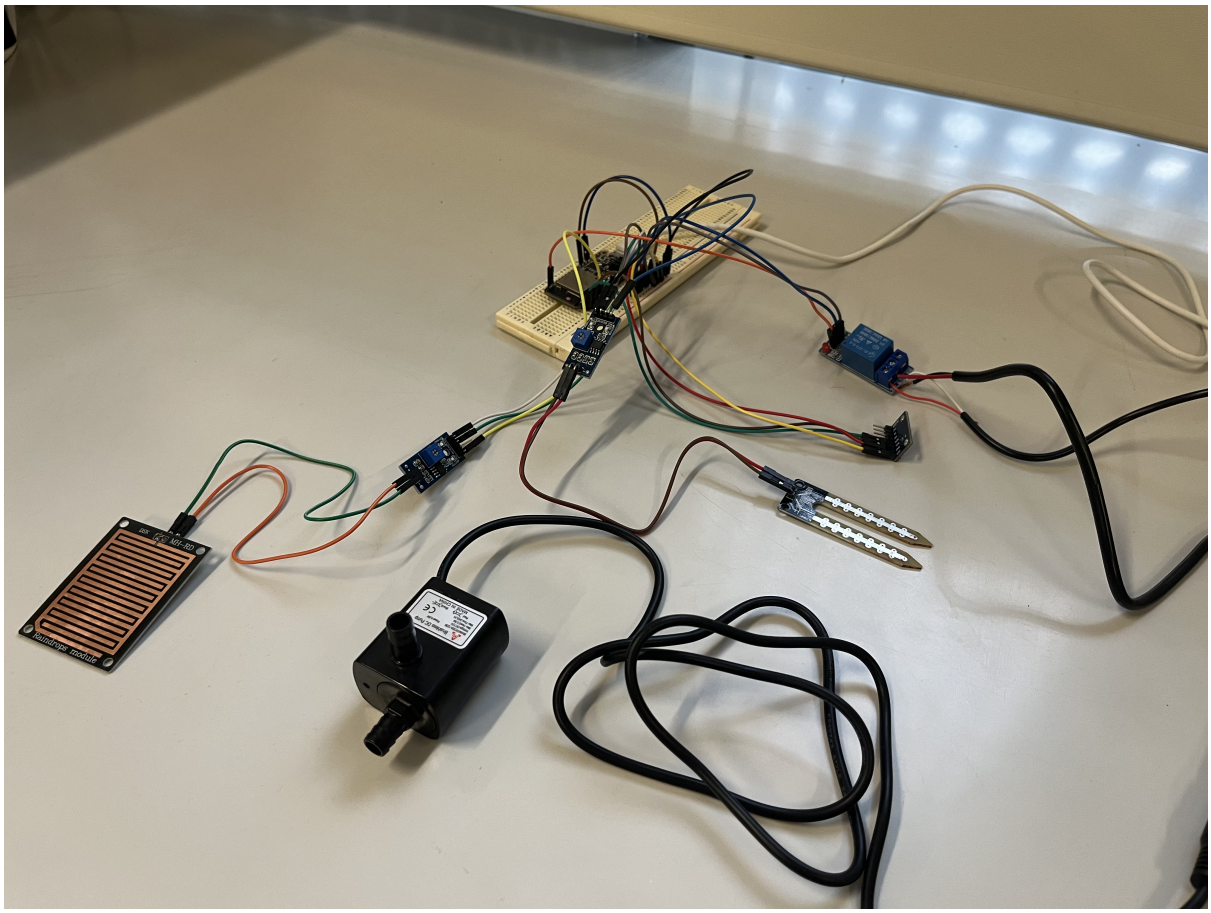


Figure 4: Hardware Setup

1. **Soil Moisture Sensor:**
   A soil moisture sensor was employed to measure soil dryness. The sensor's analog output is connected to the ESP32's ADC (Analog-to-Digital Converter) to read values in the range of 0 to 4095. Higher values indicate wetter soil, while lower values correspond to drier conditions.

2. **Rainfall Sensor:**
   Similarly, a rainfall sensor was integrated to detect precipitation levels. The analog output of the sensor is read via the ESP32's ADC, with higher values indicating increased rainfall intensity.

3. **Temperature Sensor:**
   For temperature measurements, the MCP9808 sensor was used. This I2C-based sensor communicates with the ESP32 through standard SDA (Serial Data Line) and SCL (Serial Clock Line) pins. The temperature readings are obtained directly in degrees Celsius, offering high accuracy and ease of integration.

4. **Water Pump Control with Relay:**
   A 1-channel relay module was connected to the ESP32 to control a water pump powered by a 12V adapter. The relay is activated by sending a digital HIGH (1) signal from the ESP32, while a LOW (0) signal turns it off. This setup allows for automated control of irrigation.

5. **Jumping Wires and BreadBoard**

6. **Irrigation System:**
   The water pump is connected to a plastic tube that distributes water into a container, mimicking a simple sprinkler system for efficient irrigation.

This integration enables real-time environmental monitoring and automated water delivery, enhancing the system's functionality and reliability for precision agriculture applications.

0.4 Software

Using **Visual Studio Code IDE** with **PlatformIO**, we programmed the **ESP32** to integrate the hardware components of our system, which include the soil moisture sensor, rainfall sensor, temperature sensor, relay, and DC water pump. To create user-friendly system, we developed an **HTML/CSS web server** that displays sensor readings and the status of the water pump, as shown in Fig.5, and provides the option for manual control. Data is transmitted between the ESP32 and the web server using the HTTP protocol, with both the ESP32 and the device accessing the web server connected to the same local Wi-Fi network.

We have uploaded the code along with a clear README file to GitHub in the following repository:
`https://github.com/mostafa-IK03/IoT-Based-Smart-Agriculture-System-Using-ESP32.git`.

The `main.c` file contains the code for our system with comprehensive documentation. Below, we present a summary of the functions, which explains what each function in the code does, and we also describe how the code is executed.

0.4.1 Function Summary

1. `i2c_master_init`: Initializes the I2C master interface with specified configurations for communication.
2. `mcp9808_read16`: Reads a 16-bit value from a specified register of the MCP9808 temperature sensor via I2C.
3. `mcp9808_init`: Initializes the MCP9808 temperature sensor by verifying its manufacturer and device IDs.
4. `mcp9808_ambient_temp`: Retrieves the ambient temperature from the MCP9808 sensor and converts it to Celsius.
5. `mcp9808_delete`: Frees the memory allocated for the MCP9808 sensor handle.
6. `init_adc`: Configures the ADC channels for the rain and moisture sensors with appropriate resolution and attenuation.
7. `read_rain_sensor`: Reads the raw ADC value from the rain sensor and calculates the rainfall percentage.
8. `read_moisture_sensor`: Reads the raw ADC value from the moisture sensor and calculates the soil moisture percentage.
9. `get_req_handler`: Handles HTTP GET requests to serve the main HTML monitoring web page.
10. `sensor_data_handler`: Responds to HTTP GET requests with the latest sensor data in JSON format.
11. `pump_on_handler`: Handles HTTP GET requests to turn the water pump ON by activating the relay.

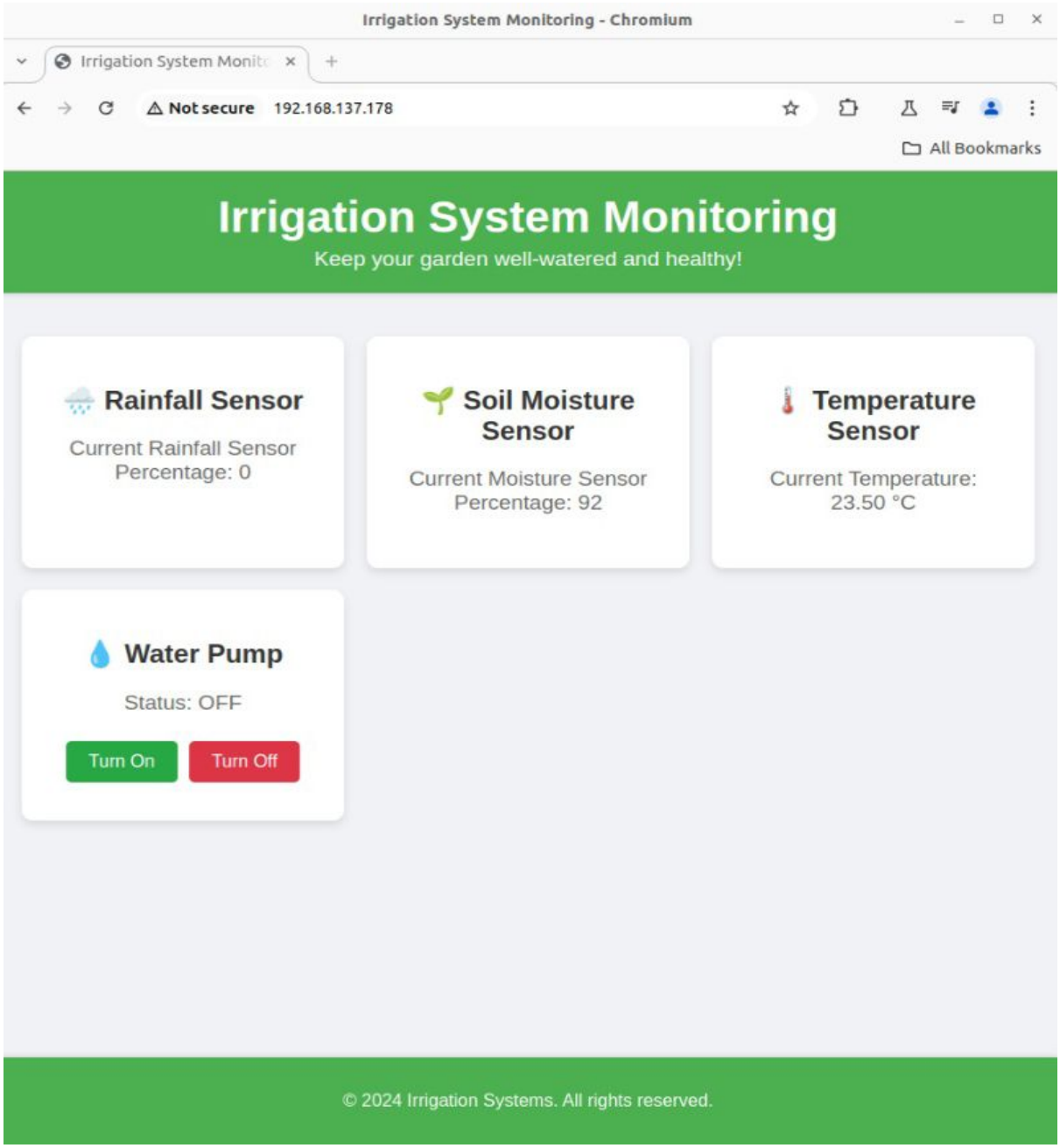


Figure 5: HTML/CSS Web server using HTTP protocol

12. `pump_off_handler`: Handles HTTP GET requests to turn the water pump OFF by deactivating the relay.
13. `start_webserver`: Initializes and starts the HTTP server, registering all necessary URI handlers.
14. `event_handler`: Manages Wi-Fi and IP-related events, handling connections and retries.
15. `connect_wifi`: Configures and initiates the connection to the specified Wi-Fi network.
16. `temperature_task`: Runs as a FreeRTOS task to periodically read and update the ambient temperature from the sensor.
17. `init_pump_relay`: Sets up the GPIO pin connected to the water pump relay and initializes it to the OFF state.
18. `app_main`: Serves as the main entry point, orchestrating initialization, task creation, sensor reading, pump control, and server operation.

0.4.2 Execution Logic Summary

1. **Initialization Phase**: Initializes Non-Volatile Storage (NVS), connects to Wi-Fi, configures ADC channels for rain and moisture sensors with predefined thresholds (rainfall $\geq 20\%$ which reflect very little rain and no rain, soil moisture $\leq 70\%$) as defined in section 2 of the report, sets up GPIO for the pump relay, and initializes the I2C interface for the temperature sensor with a temperature threshold less than 27°C as explained in section 2.
2. **Task and Server Setup**: Starts a FreeRTOS task to continuously read temperature and sensor data, and launches the HTTP server to handle web interface interactions.
3. **Main Operational Loop**: Continuously reads sensor data, logs the readings, and every minute makes watering decisions based on sensor thresholds (rainfall less than 20%, soil moisture greater than 70%, temperature less than 27°C) to control the water pump.
4. **Web Interface Interaction**: Provides real-time sensor data and allows manual control of the pump via the web interface through periodic data fetching and pump control requests.
5. **Continuous Operation**: Ensures ongoing monitoring, automated pump control based on sensor data thresholds, and user-friendly web interactions for effective irrigation system management.

Figure 6 shows the how the web server is showing error message on the sensor readings when the esp32 is not on, or while uploading

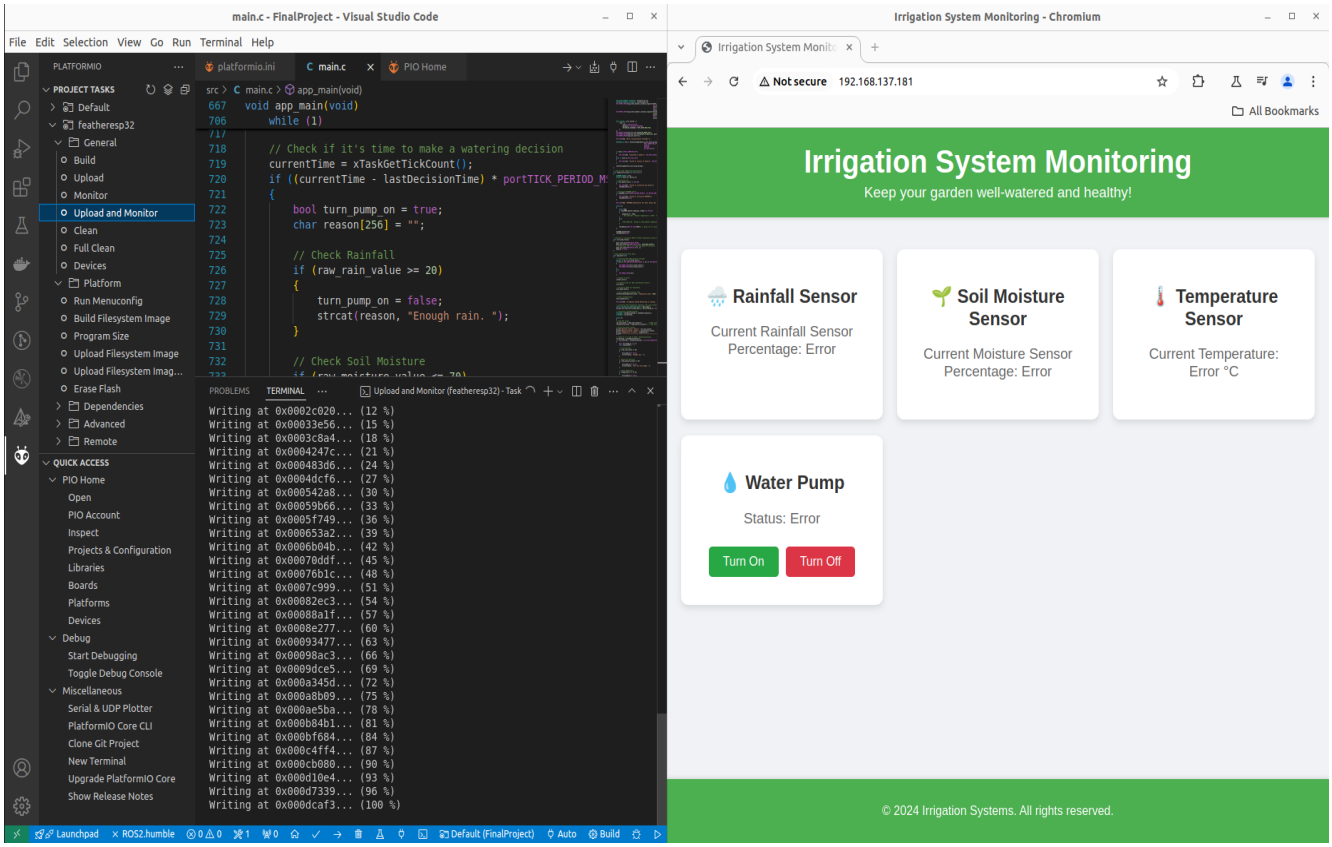


Figure 6: Web server and vs code terminal while uploading for esp32

Figure 7 shows the vs code terminal printing the live sensor readings which are streamed on the web server on the right

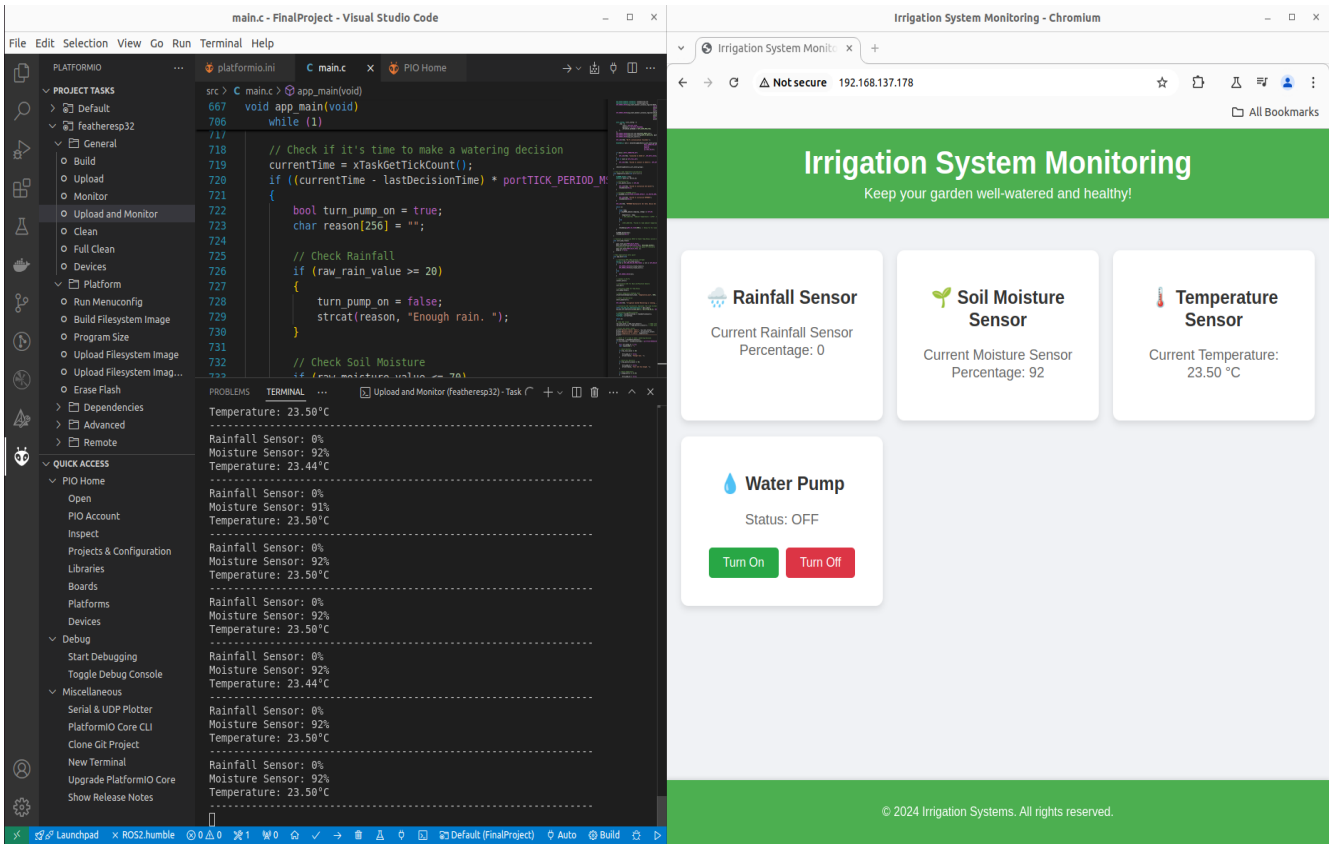


Figure 7: Web server and vs code terminal showing live sensor readings

0.5 Tasks Division

Mohammad:
Mostafa:
Together:

0.6 Conclusion and Future Work

Throughout this project, we successfully developed an **ESP32-Based Smart Irrigation System** using **Visual Studio Code IDE** and **PlatformIO**. We integrated essential hardware components—including soil moisture sensors, rainfall sensors, temperature sensors, relay modules, and a DC water pump—into the ESP32 microcontroller. To enhance user-friendliness, we implemented an **HTML/CSS web server** that displays real-time sensor readings and the status of the water pump, as illustrated in Fig. 5. This interface also provides the option for manual control of the pump, offering both automated and user-driven irrigation management. Data transmission between the ESP32 and the web server is facilitated through the HTTP protocol, with both devices connected to the same local Wi-Fi network. The system effectively optimizes water usage based on environmental conditions, demonstrating the potential of IoT solutions in promoting sustainable agricultural practices.

For future work, we aim to enhance the system's safety by incorporating advanced protective mechanisms such as overcurrent protection, thermal shutdown, and waterproof enclosures for all electronic components to prevent hardware damage and ensure reliable operation in various environmental conditions. Additionally, we plan to design and implement a custom Printed Circuit Board (PCB) to replace the current breadboard setup, providing a more compact, reliable, and scalable solution that reduces wiring complexity and enhances the overall durability of the system. Furthermore, we intend to expand the system's connectivity beyond the local Wi-Fi network by integrating cellular modules or leveraging cloud platforms, enabling remote monitoring and control. This enhancement will allow users to access the irrigation system from anywhere, facilitating data logging and analysis for improved decision-making and further optimizing water usage for sustainable agricultural practices.

Bibliography

[1] G. Sushanth and S. Sujatha, "IOT Based Smart Agriculture System," IEEE Xplore, Mar. 01, 2018. Available: https://ieeexplore.ieee.org/abstract/document/8538702

[2] G.P. Pereira, M.Z. Chaari, and F. Daroge, "IoT-Enabled Smart Drip Irrigation System Using ESP32," IoT, vol. 4, pp. 221–243, 2023. Available: https://doi.org/10.3390/iot4030012

[3] https://www.statista.com/chart/26140/water-stress-projections-global