DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 4293

Hardware Training

ARDUINO: AUTOMATED HYDROPONIC SYSTEM

by

2019510134 İlhan Gamze Saygı

2022510046 Sara Mahyanbakhshayesh

2022510013 Yasamin Valishariatpanahi

Lecturer

Assoc. Prof. Dr. Reyat Yılmaz

July, 2024

İzmir

ABSTRACT

This paper aims to introduce an Arduino project designed to measure and control specific environmental characteristics like temperature, humidity, pH, and water level automatically using an Arduino experimental setup. In this project, an automatic hydroponic system is simulated. Hydroponics is the cultivation of plants without soil, using nutrient-rich water to provide the necessary conditions and appropriate environment for plant growth. This method is expected to save time and space, leading to cost reductions. In the study, the Deep Water Culture technique, one of the hydroponic methods, was used.

The project's goal, as mentioned earlier, is to ensure the survival of the plant by maintaining environmental factors within fixed or specific ranges. The Arduino experimental setup will continuously measure these environmental factors, and control systems will be activated to restore desired values if deviations occur. This ensures that the plant grows in optimal conditions and is automatically maintained.

Data collected during the experiment will be transmitted to a server via a Wi-Fi system for record-keeping. Additionally, the grower will be able to monitor the plant's status in real-time through an LCD screen. Ultimately, the plant will be able to grow with monitoring, contributing to cost reduction.

Keywords:

Arduino project; Environmental characteristics; Temperature; Humidity; pH; Water level; Arduino experimental setup; Automatic hydroponic system; Hydroponics; Deep Water Culture; Nutrient-rich water; Plant growth; Time saving; Space saving; Cost reduction; Control systems; Optimal conditions; Data collection; Wi-Fi system; Server; Real-time monitoring; Real-Time Clock; LCD screen; Human intervention

TABLE OF CONTENTS

Page

FIGURE LIST

TABLE LIST

CHAPTER ONE INTRODUCTION

An automated hydroponic system is a method of growing plants without soil, using nutrient-rich water and advanced technology to maintain optimal growing conditions. Automation in hydroponics involves using sensors and controllers to monitor and adjust factors such as pH, water level, humidity, and temperature, ensuring that plants receive the ideal environment for growth with minimal human intervention. This technology-driven approach not only maximizes plant health and yield but also reduces labor and resource usage, making it an efficient and sustainable method of agriculture.

One popular type of hydroponic system is the Deep Water Culture (DWC). In a DWC system, plant roots are submerged in a continuously oxygenated nutrient solution, which promotes faster growth and healthier plants compared to traditional soil-based methods. Our project employs a DWC drip system, where the primary components include a plastic bucket that serves as the reservoir for the nutrient solution. Additionally, the system is equipped with various sensors such as pH sensors, water level sensors, humidity sensors, temperature sensors, and TDS (Total Dissolved Solids) liquid conductivity and quality sensors. These sensors are crucial for maintaining the precise conditions needed for optimal plant growth.

The automation in our DWC system is achieved through the integration of these sensors with timers and controllers, which regulate the delivery of water and nutrients. By continuously monitoring key parameters, the system can make real-time adjustments to ensure that plants are always in the best possible growing environment. This automated approach not only improves plant growth and yields but also significantly reduces the need for manual monitoring and adjustment, making it an ideal solution for both small-scale and large-scale agricultural applications.

CHAPTER TWO LITERATUR REVIEW

The article, "Nutrient Film Technique for Automatic Hydroponic System Based on Arduino," was presented at the 2020 2nd International Conference on Industrial Electrical and Electronics (ICIEE) by researchers Iswanto from Universitas Muhammadiyah Yogyakarta, Prisma Megantoro from Universitas Airlangga, and Alfian Ma'arif from Universitas Ahmad Dahlan. The article discusses the development and implementation of an automatic hydroponic system using the Nutrient Film Technique (NFT) based on Arduino technology. Hydroponics is a soil-free farming method where plant roots are exposed to a nutrient-rich water solution. The NFT system continuously circulates a shallow layer of this solution over the plant roots, providing water, nutrients, and oxygen efficiently.

The system described in the article utilizes an Arduino Uno microcontroller, relay modules, peristaltic pumps, solenoid valves, and sensors (ultrasonic and pH meter) to automate the process. The control algorithm manages nutrient and water levels, ensuring optimal conditions for plant growth without the need for constant human intervention. The system was tested on cayenne pepper plants, which successfully grew from seeding to flowering in nine weeks. The study highlights the benefits of using an Arduino-based NFT system for hydroponics, particularly in reducing the need for manual monitoring and adjustment of nutrient levels. However, the researchers noted a limitation with the solenoid valves, which were not suitable for low water pressure conditions, suggesting an area for future improvement. Overall, the automated NFT system presents a promising solution for enhancing hydroponic cultivation by leveraging technology to optimize plant growth and resource management.

The article "Hydroponic Farming Model Using Arduino" by NSV Jashwanth, Shrikanth CK, Raseety Sandeep, and Thanush RS from the Department of Electronics and Communication at K.S. Institute of Technology, Bangalore, India, was published in the International Journal of Research Publication and Reviews. This paper presents a hydroponic drip irrigation system integrated with sensors to monitor and manage essential parameters such as temperature and humidity. The system uses Arduino technology,

facilitating efficient indoor drip irrigation by ensuring optimal conditions for hydroponic cultivation. The design connects directly with sensors, allowing for easy monitoring, data management, and online settings adjustments.

The literature survey in the article reviews various hydroponic techniques and their implementations, such as ebb and flow, continuous flow solution culture, and IoT-based systems. These methods offer benefits like nutrient-rich environments, low risk of faults, and efficient monitoring, but also face challenges such as pump failures, internet connectivity issues, and cost inefficiency. The proposed system uses a network of feeder lines to deliver water and nutrients, making it suitable for large-scale operations. The system's high level of control and minimal water usage make it commercially preferable, though there are areas for potential improvement in reliability and cost efficiency.

The article "Automation of Greenhouse Hydroponic System using Arduino Uno and Resource Management Based on Crop Growth Monitoring with Camera" by Vibhu Sharma and Dr. V.K Srivastav from BMU Rohtak, explores the automation of greenhouse hydroponic systems using the Arduino Uno chipset. This research aims to optimize resource management for plants based on their growth by monitoring crop height with a camera and controlling various sensors accordingly. The study employed pH meters, temperature and light sensors, water and air pumps, and a camera to gather data and automate the system, resulting in faster growth and higher yields.

The literature review highlights the increasing popularity of hydroponic systems in greenhouses due to their potential for enhanced growth and yield. Previous studies have shown that using microcontrollers like the Arduino Uno and cameras for environmental monitoring and control can improve resource management and plant growth. The methodology of this study involved using the Arduino Uno to collect data from various sensors and adjust the hydroponic system to maintain optimal growing conditions. The results indicated that the Arduino Uno chipset is effective in automating greenhouse hydroponic systems, ensuring optimal conditions for plant growth and achieving faster growth and greater yields.

The article "Smart Cultivation: An Arduino-based IoT Aeroponics System for Indoor Farming" by Rashmi and Dr. Shilpa Shrigiri S from Sharnbasva University, Kalaburagi, Karnataka, India, addresses urgent global challenges posed by population growth and climate change on traditional agriculture. It advocates for vertical farming as a transformative solution, integrating advanced technologies like LED lighting, automated hydroponics, and innovative design approaches. Recent research emphasizes the importance of resilience and circularity in farming systems, highlighting the beneficial role of Plant Growth-Promoting Rhizobacteria (PGPR) in enhancing plant health and reducing reliance on chemical inputs.

The paper introduces an IoT-enhanced aeroponics system driven by Arduino technology, designed to meticulously control essential growth parameters such as light, temperature, humidity, carbon dioxide levels, water, and nutrients. This comprehensive approach aims to ensure year-round production of high-quality crops, immune to external climate variations. The study explores diverse farming setups—from industrial-scale plant factories to modular container farms and household appliance farms—demonstrating scalability and adaptability for urban agriculture needs. Performance evaluations showcase remarkable efficiency gains, including a 40-50% reduction in water usage and a 25-35% decrease in chemical inputs compared to conventional farming methods. This reflects the system's success in optimizing resource utilization through real-time data monitoring and responsive automation. The integration of IoT technology with platforms like the BLYNK cloud app enables remote monitoring and control, empowering growers with actionable insights for maximizing productivity and sustainability.

In conclusion, the research underscores the transformative potential of IoT-driven aeroponics systems in fostering sustainable urban agriculture. By enhancing resilience, minimizing environmental impact, and ensuring consistent crop yields, this innovative approach contributes significantly to global food security efforts amidst evolving climate challenges.

The publication "Hydroponic Production of Edible Crops: Deep Water Culture (DWC) Systems" by specialists from Virginia State University and Virginia Tech presents an in-

depth exploration of DWC systems, pivotal in modern hydroponic agriculture since their development in the 1980s. DWC systems submerge plant roots in nutrient-rich solutions within ponds or reservoirs constructed from durable materials like lumber and polyethylene. These setups feature floating plant rafts that support crops such as lettuce and herbs, optimizing space and efficiency in controlled environments. Essential components include robust nutrient circulation systems powered by submersible or centrifugal pumps, supplemented by oxygenation techniques like venturi injectors. The publication underscores the importance of meticulous nutrient management and highlights DWC's suitability for a variety of crops, offering practical guidance to growers aiming for sustainable and high-yield indoor farming solutions.

The article “Arduino Controlled Smart Hydroponic Modular System” by Innovart Studio at Juan de Lanuza School developed a portable hydroponic system as part of the "Robotics in Family" activity at Juan de Lanuza School. This system uses mineral solutions instead of soil for plant growth, promoting water efficiency and higher yields while minimizing environmental impact by keeping fertilizers out of the water table. The system features an aluminum and PVC structure with six levels for plant placement, supported by 3D printed parts and equipped with sensors to monitor air and water parameters like temperature, humidity, pH, and conductivity.

Electronically controlled components include a water pump, LED grow lights, and a custom nutrient feeder managed by an Arduino MEGA board. Data from sensors are transmitted to a web server via WiFi (ESP8266 module) and an Android app for real-time monitoring and control. The project integrates automated watering cycles, light schedules, and nutrient dosing based on sensor readings, ensuring optimal plant health and growth. The system's code, libraries, and data visualization tools are open-source, facilitating replication and customization for educational and environmental initiatives. The "Lanuza Makers" team successfully implemented this modular hydroponic system, demonstrating its functionality and educational value within the school community.

The article titled "Design and Implementation of a Fuzzy Logic Controlled Ebb and Flow Hydroponic System" details the development and testing of a sophisticated hydroponic

setup. It was written by Suprijadi, N. Nuraini, and M. Yusuf, and published in the journal "Otomasi Kontrol dan Instrumentasi." The study focuses on utilizing fuzzy logic through an Arduino UNO microcontroller to oversee various environmental factors via sensors like the YL-69 soil moisture sensor and DHT11 temperature sensor. A DC water pump, managed by the L298N motor driver and regulated by Pulse Width Modulation (PWM), circulates nutrient water to plants based on real-time conditions. This integration ensures optimal growth conditions by adjusting water delivery as per the moisture and temperature levels in the planting medium.

The article presents the design, implementation, and testing of an ebb and flow hydroponic system controlled by fuzzy logic. The system uses an Arduino UNO microcontroller to manage various sensors and actuators, including the YL-69 soil moisture sensor, DHT11 temperature sensor, and a DC water pump driven by the L298N motor driver. The fuzzy logic control system is designed to adjust the pump speed based on the temperature and moisture levels of the planting medium, ensuring optimal growing conditions for the plants.

Key components and their roles are described in detail. The L298N motor driver controls the DC pump, which circulates nutrient water to the plants. Pulse Width Modulation (PWM) is used to regulate the pump speed, with duty cycles determining the mean voltage and thereby the pump speed. The fuzzy logic system consists of fuzzification, rule evaluation, and defuzzification stages, converting sensor inputs into appropriate control signals for the pump. The system was tested and showed good performance in maintaining the desired moisture levels in the planting medium, with observations indicating healthy plant growth.

Overall, the study demonstrates the feasibility and effectiveness of using fuzzy logic control for hydroponic systems, providing a smart solution for automating nutrient delivery based on real-time environmental conditions. The use of Arduino and various sensors and actuators ensures a low-cost and scalable system suitable for small-scale farming applications.

CHAPTER THREE PROJECT DESCRIPTION

2.1. Project Objective

The primary objective of our project is to design and build an automated hydroponic system as part of our internship program. The main goal is to implement a system that grows plants in the best, most efficient, and healthy manner possible, leveraging advanced technology to optimize plant growth and minimize manual labor.

2.2. System Overview

Our system is a Deep Water Culture (DWC) drip hydroponic setup, a method where plant roots are suspended in a nutrient-rich, oxygenated solution. This type of system is ideal for promoting rapid and healthy plant growth, as it provides constant access to nutrients and oxygen directly to the roots. The integration of automation enhances this process by continuously monitoring and adjusting environmental conditions to maintain optimal growing conditions. The system is designed to be efficient, user-friendly, and scalable, suitable for both small-scale and large-scale agricultural applications.

2.3. Components and Materials

- 12V Relay Switch x 4 Channels
- 12V Peristaltic Liquid Pump - Silicone Tube, NKP-DC-S10
- 12V Aquarium Air Pump
- DHT22 Temperature & Humidity Sensor – Waterproof
- DS3231 Arduino Sensitive Real-Time Clock Module
- Plastic Bucket
- Net Pot for DWC
- Fertilizers Container
- ESP8266 Wi-Fi Module
- Arduino pH Sensor
- DS18B20 Temperature Sensor
- Gravity: Non-Contact Digital Water Level Sensor for Arduino

- Pumice Stone
- Arduino TDS Liquid Conductivity and Quality Sensor
- LCD Module 2x16
- Push Button
- Arduino Board
- Resistors
- Lightbulb
- Aquarium Air Stone
- Plastic Tube

2.4. System Design and Setup

The system design includes setting up the plastic bucket as the reservoir, placing net pots for holding plants, and arranging sensors to monitor various parameters. The air pump and aquarium air stone oxygenate the nutrient solution, while the peristaltic pump delivers nutrients. The Arduino board, equipped with Wi-Fi capabilities, controls the system based on real-time data from the sensors. Detailed diagrams and step-by-step instructions will be provided to illustrate the setup process.

2.5. Automation and Control

The automation and control of the system are managed using an Arduino board integrated with various sensors and relays. The ESP8266 Wi-Fi module allows for remote monitoring and control via an Android application. The sensors continuously monitor pH levels, water levels, temperature, humidity, and nutrient quality (TDS). Based on the sensor data, the Arduino board makes real-time adjustments to the nutrient delivery and environmental conditions, ensuring optimal plant growth.

2.6. Nutrient Solution Management

The nutrient solution is critical for plant health. Our system includes a fertilizers container that supplies nutrients to the plants through a peristaltic pump. The Arduino TDS sensor monitors the concentration of dissolved solids in the solution, ensuring the nutrient

levels are within the desired range. Regular checks and adjustments are automated to maintain the ideal nutrient balance.

2.7. Monitoring and Maintenance

The system's sensors provide continuous feedback on environmental conditions, which is displayed on the LCD module. This real-time data allows for immediate adjustments, reducing the need for manual intervention. Maintenance involves periodic calibration of sensors, cleaning of components, and ensuring the nutrient solution is replenished as needed. Alerts and notifications are set up to inform users of any issues that require attention.

2.8. Expected Outcomes and Benefits

The expected outcome is a fully functional automated hydroponic system that promotes efficient and healthy plant growth. The benefits include reduced manual labor, consistent and optimal growing conditions, and higher yields. The system is designed to be scalable and adaptable, making it suitable for various applications from small home gardens to larger agricultural setups.

2.9. Challenges and Solutions

The most challenging part of the project is integrating the code with the hardware. While both coding and building the system are manageable separately, combining them has presented significant difficulties. To address this, we have focused on rigorous testing and iterative troubleshooting to ensure smooth operation when the system is assembled.

2.10. Future Improvements

Currently, our system's remote monitoring and control capabilities are limited to an Android application. A key future improvement would be to extend these capabilities to a web-based platform, allowing for broader access and control. Additionally, we plan to enhance the system's user interface and add more advanced features for better customization and scalability.

CHAPTER FOUR DESIGN

Under this title, drawings of the experimental setup design and the necessary materials list have been provided.

3.1. Drawing of the Automated Hydroponic System

The experimental setup for the automated hydroponic system, as previously described, is depicted as follows. The positions of the relay, pump, and sensor elements are indicated in the figure.

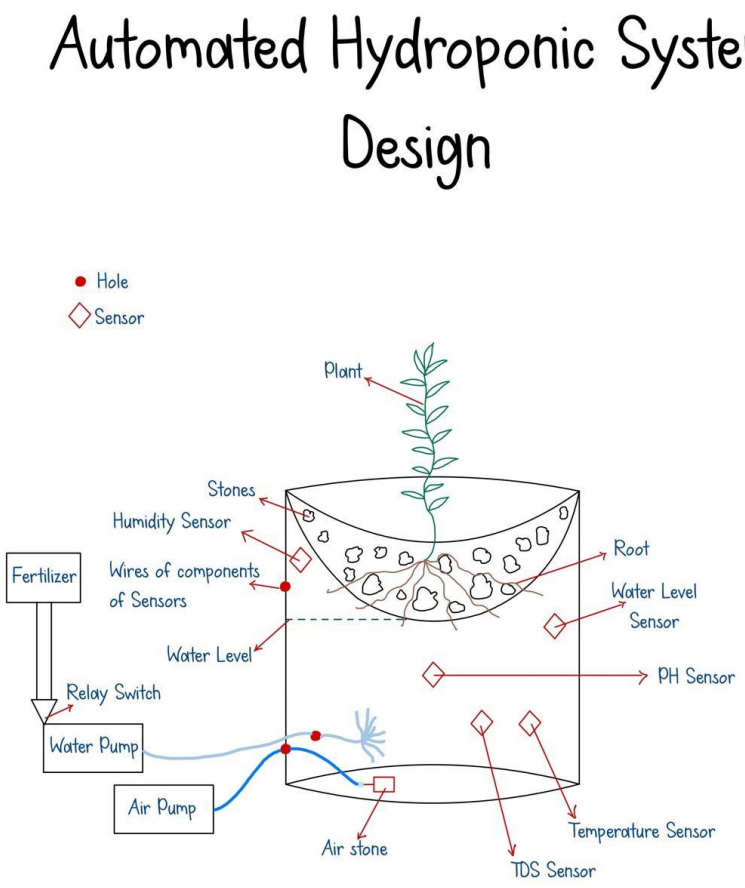

Figure 4.1. Automated Hydroponic System Visualization

3.2. Required Materials List

The material list for the described project that will be used throughout includes:

- 12V Relay Switch x 4 Channels
- 12V Peristaltic Liquid Pump - Silicone Tube, NKP-DC-S10
- 12V Aquarium Air Pump
- DHT22 Temperature & Humidity Sensor – Waterproof
- DS3231 Arduino Sensitive Real-Time Clock Module
- Plastic Bucket 10 L
- Net Pot for DWC
- Fertilizers Container 1 L
- ESP8266 Wi-Fi Module
- Arduino pH Sensor
- DS18B20 Temperature Sensor
- Water Level Sensor
- Pumice Stone (2 L)
- Arduino TDS Liquid Conductivity and Quality Sensor
- LCD Module 2x16
- Arduino Board
- Resistor
- Lightbulb
- Jumper Wires
- Aquarium Air Stone
- Plastic Tube (2 meters)
- Transistor BC547

12V Relay Switch x 4 Channels: This relay module allows for the control of high voltage devices with the Arduino. It's essential for automating various aspects of the hydroponic system, such as pumps and lights, providing the necessary interface between the Arduino and external devices.

Figure 4.2. 12V Relay Switch x 4 Channels

12V Peristaltic Liquid Pump - Silicone Tube, NKP-DC-S10*:* This pump, used for precise liquid transfer, is integral to the system for nutrient delivery. It operates at 12V and uses a silicone tube to prevent contamination. The pump ensures a consistent and controlled flow of nutrient solution to the plants, which is vital for maintaining their growth and health.

Figure 4.3. 12V Peristaltic Liquid Pump - Silicone Tube, NKP-DC-S10

12V Aquarium Air Pump: Used to aerate the water in the hydroponic system, this pump ensures that the nutrient solution is well-oxygenated. This is crucial for the roots of the plants, as it prevents root rot and promotes healthy growth.

Figure 4.4. 12V Aquarium Air Pump

DHT22 Temperature & Humidity Sensor: This sensor measures the ambient temperature and humidity levels in the growing environment. It can be safely used that the plants are kept within their ideal temperature and humidity range for air condition of plants.

Figure 4.5. DHT22 Temperature & Humidity Sensor

DS3231 Arduino Sensitive Real-Time Clock Module: This module provides accurate timekeeping for the Arduino system. It's used to schedule various automated tasks, such as lighting cycles and nutrient delivery, ensuring that the plants receive consistent care at the right times.

Figure 4.6. DS3231 Arduino Sensitive Real-Time Clock Module

Plastic Bucket 10 L: The bucket serves as the main reservoir for the hydroponic nutrient solution.

Figure 4.7. Plastic Bucket

Net Pot for DWC: The net pot holds the plants in place. Plastic bucket and net pot form the core structure of the Deep Water Culture (DWC) hydroponic system together.

Figure 4.8. Net Pot for DWC

Fertilizers Container 1 L: The fertilizer container stores the nutrient solution that is delivered to the plants.

Figure 4.9. Fertilizers Container

ESP8266 Wi-Fi Module: The ESP8266 Wi-Fi module allows for wireless monitoring and control of the hydroponic system, enabling remote access to the data and settings.

Figure 4.10. ESP8266 Wi-Fi Module

Arduino pH Sensor: This sensor measures the pH level of the nutrient solution, which is critical for maintaining the optimal pH range for plant growth. It ensures that the plants can effectively absorb the nutrients from the solution.

Figure 4.11. Arduino pH Sensor

DS18B20 Temperature Sensor: This sensor provides precise temperature measurements of the nutrient solution or the environment. Maintaining the right temperature is essential for the health and growth of the plants in the hydroponic system.

Figure 4.12. DS18B20 Temperature Sensor

Water Level Sensor: The Funduino water level sensor measures the depth of water by using a series of exposed copper traces on the sensor board that act as resistive elements. When water makes contact with these traces, it creates a variable resistance that the sensor

translates into an analog signal, which can be read by a microcontroller like an Arduino to determine the water level. This sensor is commonly used in projects requiring water level detection, such as automated hydroponic systems or water tanks.

Figure 4.13. Water Level Sensor

Pumice Stone (2 L): These are the growing media used to support the plants' roots in the hydroponic system. They provide aeration and help retain moisture and nutrients, promoting healthy root development and overall plant growth.

Figure 4.14. Pumice Stone

Arduino TDS Liquid Conductivity and Quality Sensor: Priced at 307.69 TL, this sensor measures the total dissolved solids (TDS) in the water, indicating its quality and conductivity. It helps monitor the nutrient concentration in the hydroponic solution, ensuring that plants receive the optimal nutrients for growth.

Figure 4.15. Arduino TDS Liquid Conductivity and Quality Sensor

LCD Module 2x16: This LCD display module facilitates real-time monitoring of various parameters within the hydroponic system, offering a user-friendly interface for growers to check temperature, humidity, water level, and nutrient concentration. It shows real-time data from sensors and system status, providing an easy way to monitor critical parameters like temperature, humidity, pH, and water level, ensuring optimal conditions for plant growth.

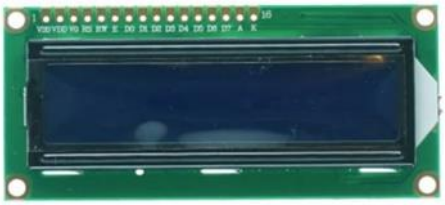

Figure 4.16. LCD Module 2x16

Arduino Board: The central controller for the entire project, the Arduino board processes data from the sensors and executes control commands. It is the brain of the system, enabling automation and integration of various components.

Figure 4.17. Arduino Board

Resistor: Essential for controlling the current in the circuit, resistors protect the components from damage by ensuring that the current remains within safe limits. They are used in various parts of the circuit, including with LEDs and sensors.

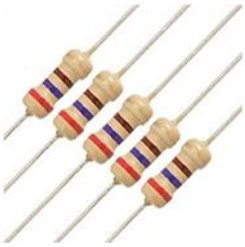

Figure 4.18. Resistor

Lightbulb: These indicator lights show the status of the system. A green light indicates that the system is on, while a red light shows that the system is off. This visual cue helps users quickly understand the operational state of the system.

Figure 4.19. Lightbulb

Aquarium Air Stone: Used to diffuse air into the nutrient solution, the air stone ensures that the water is well-oxygenated. This is crucial for the health of the plant roots in the hydroponic system, promoting better growth and preventing root rot.

Figure 4.20. Aquarium Air Stone

Plastic Tube (2 meters): These tubes are used to connect various components of the system, such as the air pump, water pump, and nutrient reservoir. They facilitate the flow of

air and water, creating a loop that delivers nutrients and aerates the plants from the bottom to the top, simulating a shower-like effect.

Figure 4.21. Plastic Tube

Transistor BC547: The existing Arduino UNO provides a 5V output. The 4-channel Relay Switch Module, on the other hand, requires a 12V input. Therefore, signals are received from the Arduino at 5V, but the switches do not operate. Hence, a transistor or op-amp is needed to act as an amplifier. In the project, it is preferred to use a transistor. The BC547 has been selected to perform the necessary amplification. Thus, the 5V output from the Arduino will be delivered to the Relay Switches as 12V.

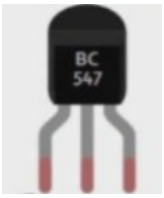

Figure 4.22. Transistor BC547

M-M and M-F Jumper Wires: M-M (Male-to-Male) and M-F (Male-to-Female) jumper wires are essential components in the development of the automated hydroponic system. These wires facilitate easy and reliable connections between different electronic components, including sensors, actuators, and the Arduino board, without the need for soldering.

Figure 4.23. M-M Jumper Wires

The Role of M-M and M-F Jumper Wires in the Automated Hydroponic System is that M-M and M-F jumper wires are used to establish connections between various parts of the hydroponic system. They enable the creation of flexible and temporary circuits, allowing for easy modifications and troubleshooting during the development and testing phases.

Figure 4.24. M-F Jumper Wires

CHAPTER FİVE IMPLEMENTATION

The tasks carried out throughout the project during the implementation phase are described under various subheadings. The codes for each task are provided under the "Appendices" title.

5.1. Work Done in First Week

In the first week of the project, the following tasks were completed: Taking output on LCD screen, Temperature measurement, Displaying temperature on LCD screen, and Wi-Fi module connection.

5.1.1. Taking Output on LCD Screen

At the initial stage, work was done to obtain output on the LCD screen. Connections were made based on the design schematic provided in the figure. After completing the connections, the address of the LCD screen is first determined using an address finder code. Once the address is found, the dimensions of the LCD, in terms of columns and rows, are correctly entered into the code. For the available LCD, the address is 0x27 with 16 columns and 2 rows. Subsequently, the necessary print function for the LCD is written to display the output on the screen.

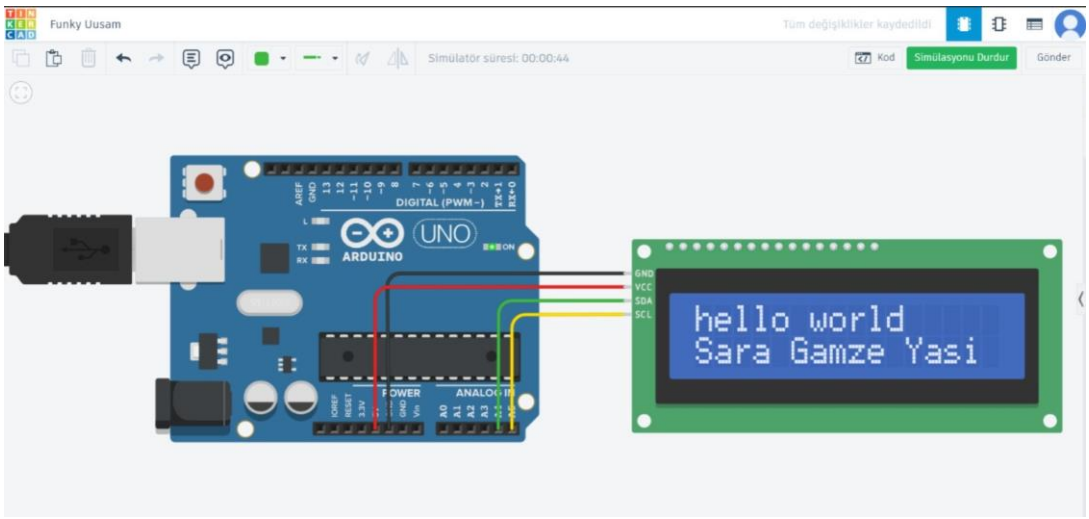

Figure 5.1. Design Schematic of LCD Screen Connection

The connections made for the implementation to obtain the first output on the LCD screen and the obtained output are shown in figure below.

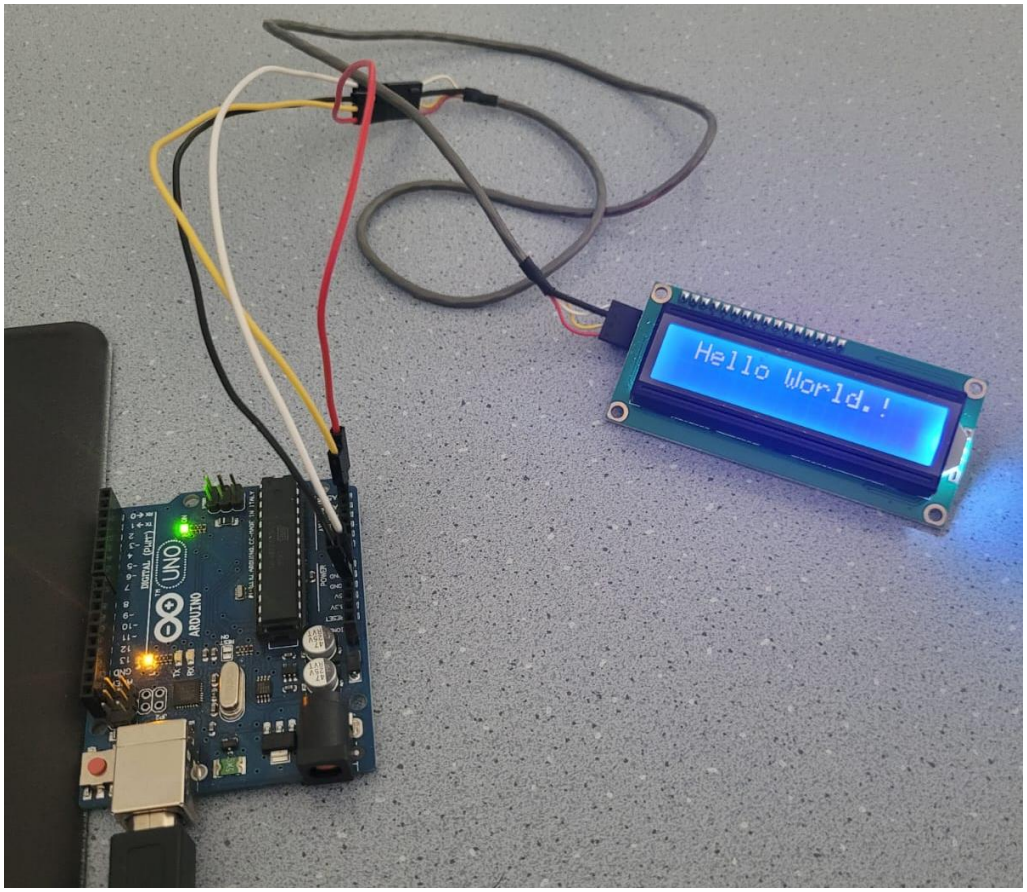

Figure 5.2. Displaying Text on LCD

5.1.2. Temperature Measurement

Temperature measurements were conducted using the DS18B20 temperature sensor. This sensor was preferred for the experiment because it is a waterproof sensor capable of operating both in air and underwater environments. The sensor has three cable connections: VCC, GND, and Data. These connections were made according to the design schematic shown in the figure below.

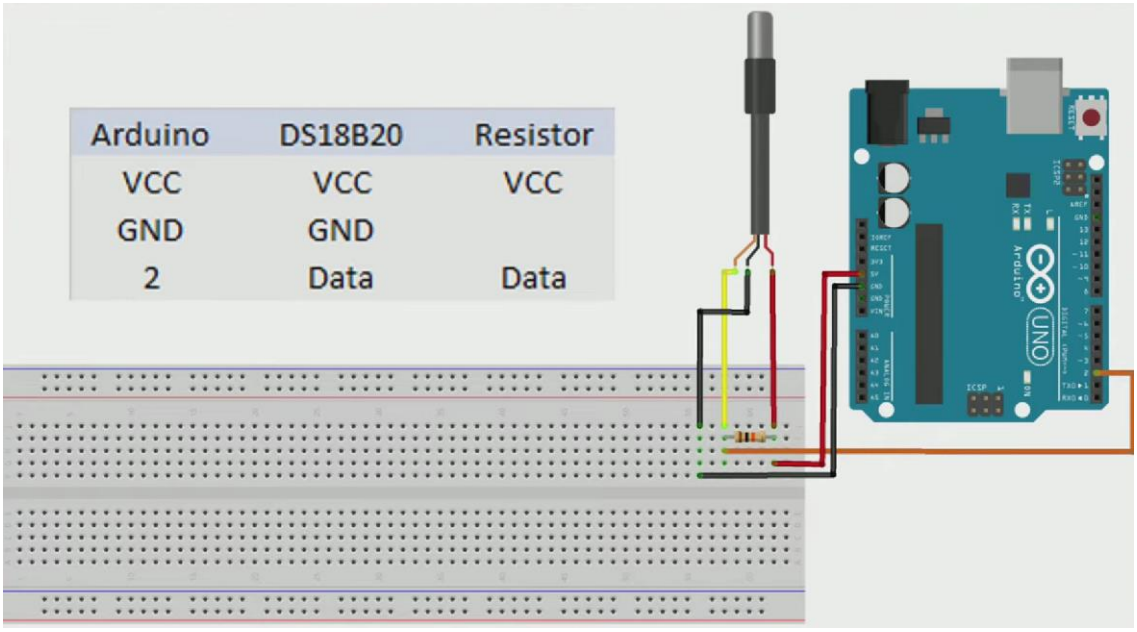

Figure 5.3. Design Schematic of Temperature Sensor Connection

The figure below illustrates the connections made for the temperature measurement application.

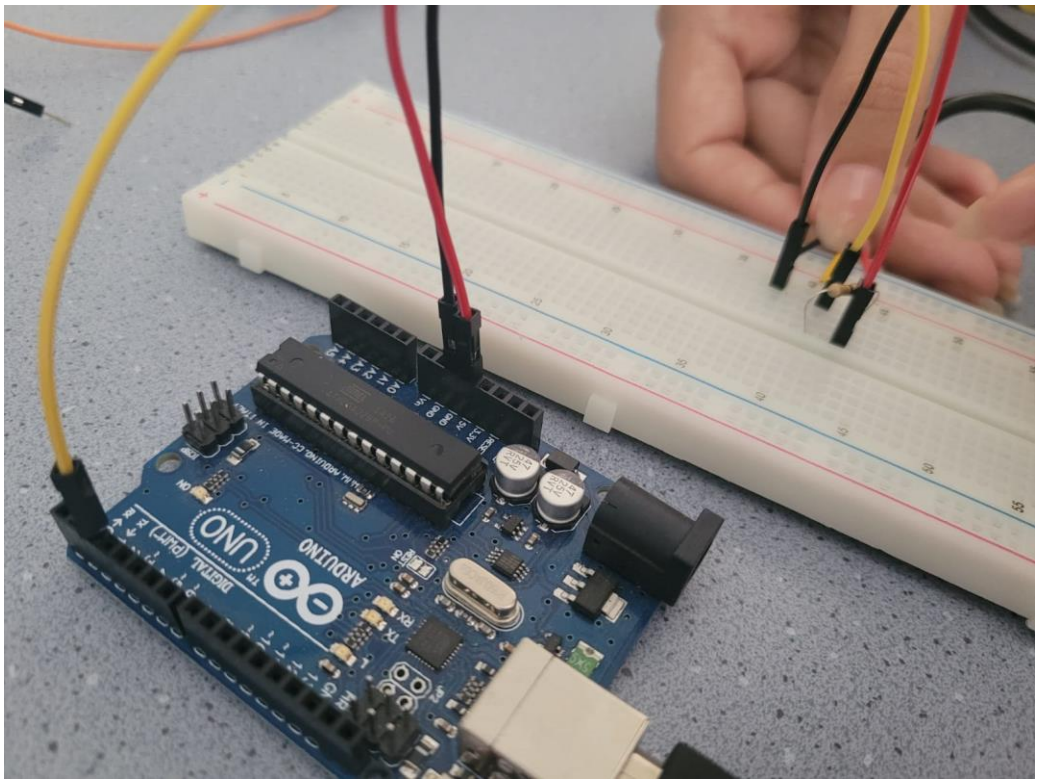

Figure 5.4. Temperature Measurement

The values obtained after measurements are presented in the figure below. Measurement results, in both Celsius and Fahrenheit, were displayed on the Serial Monitor in the Console. Data was transmitted in real-time to adapt to variable conditions.

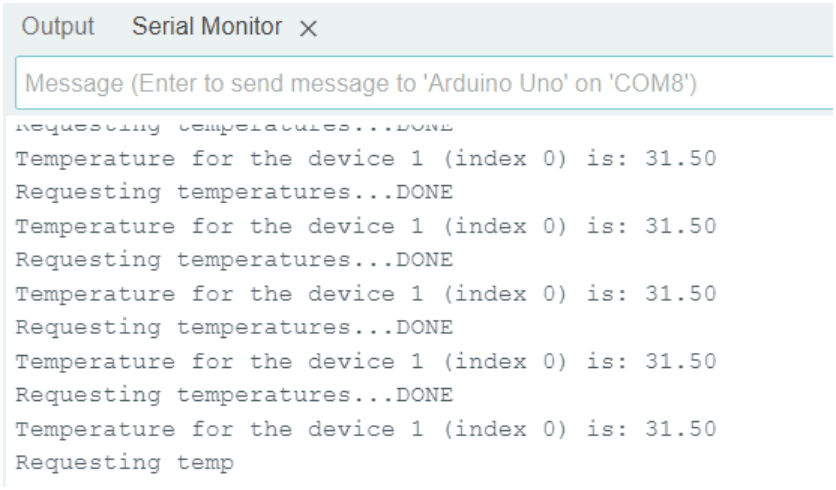

Figure 5.5. Output of Temperature Measurement

5.1.3. Displaying Temperature on LCD Screen

As the second stage of the project, data from the DS18B20 temperature sensor was not only monitored on the Console screen via Serial Monitor but also displayed to users on an LCD screen. The setup for this stage is depicted in the figure below.

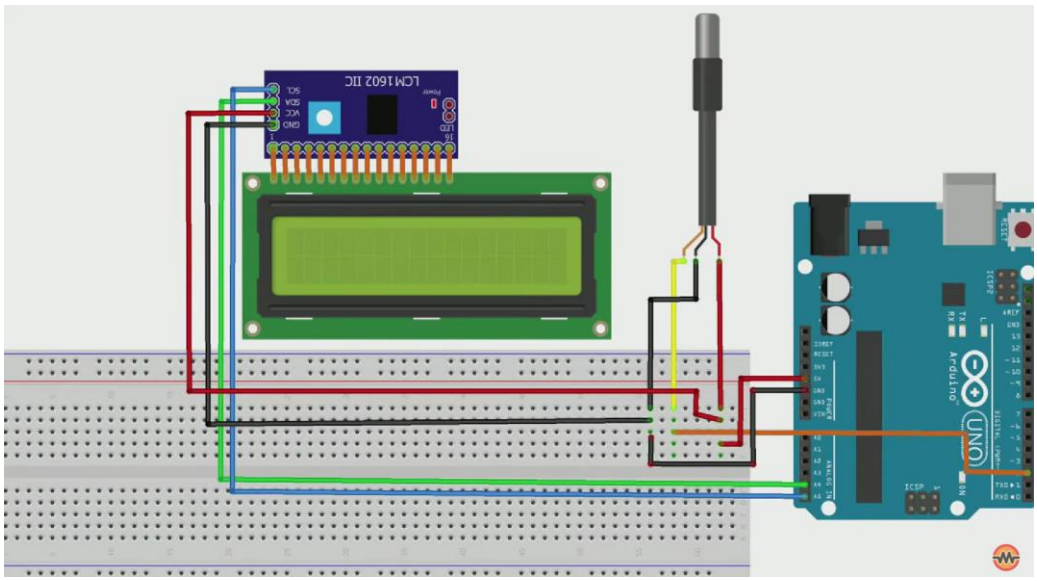

Figure 5.6. Design Schematic of Temperature Sensor and LCD Connections

The application for displaying temperature measurements from the DS18B20 temperature sensor on an LCD screen is as follows:

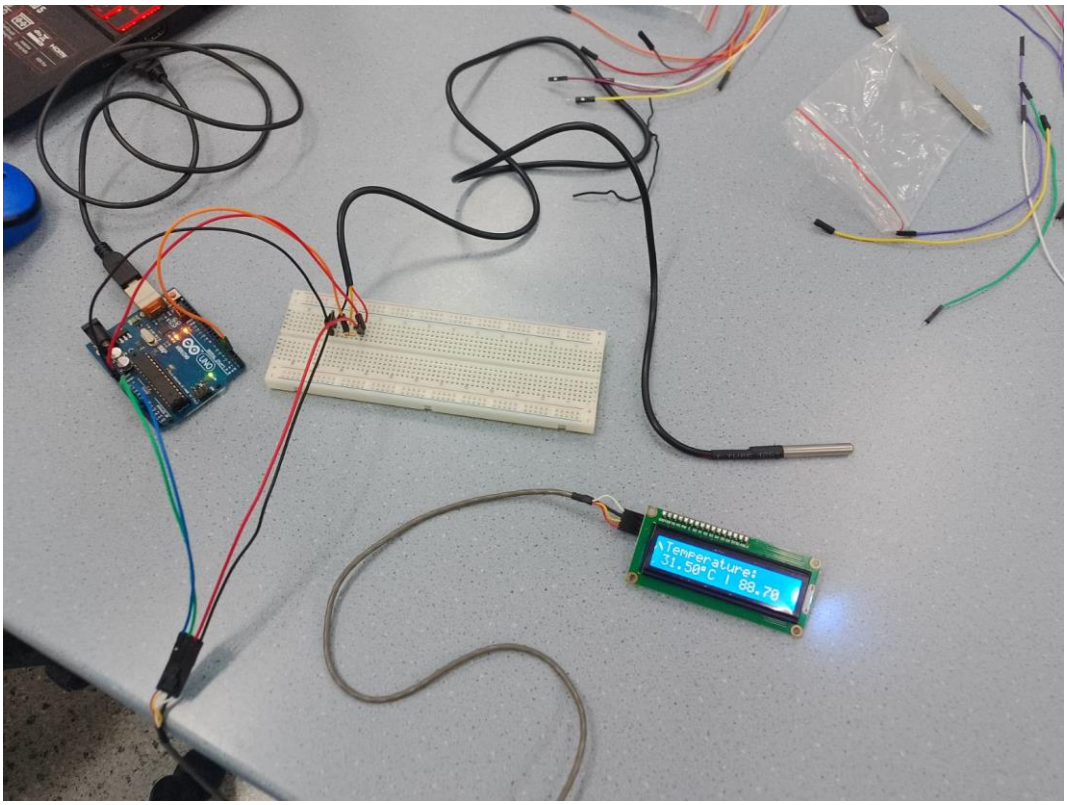

Figure 5.7. Temperature Measurement on LCD Screen

The temperature measurement results obtained in both Celsius and Fahrenheit units are as follows:

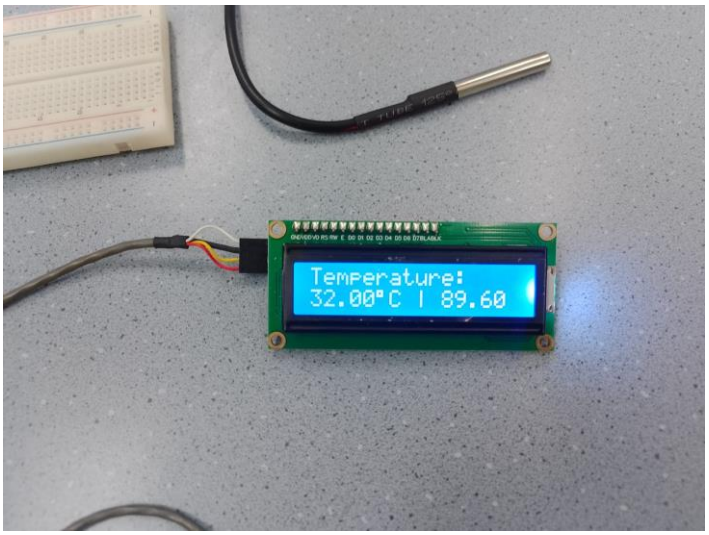

Figure 5.8. Output of Temperature Measurement on LCD Screen

5.1.4. Displaying Humidity and Temperature on LCD Screen

Humidity and temperature values are displayed using the DHT22 sensor. Due to the sensor not being waterproof, it is planned to use it solely to measure the quality of the air conditions in which the plant is located. The design schematic for the sensor connection is as follows:

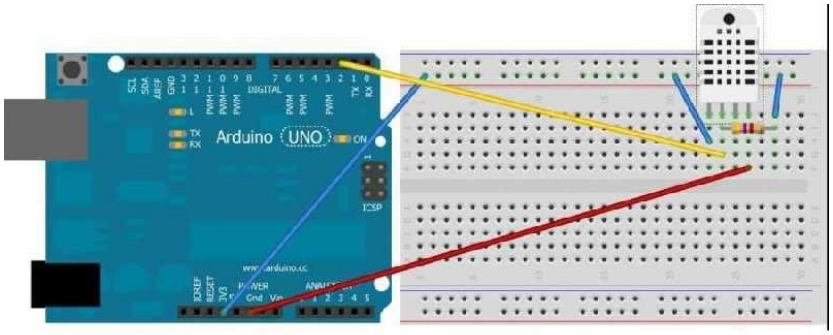

Figure 5.9. Design Schematic of Humidity and Temperature Sensor Connection

The humidity and temperature measurement results obtained as follows:

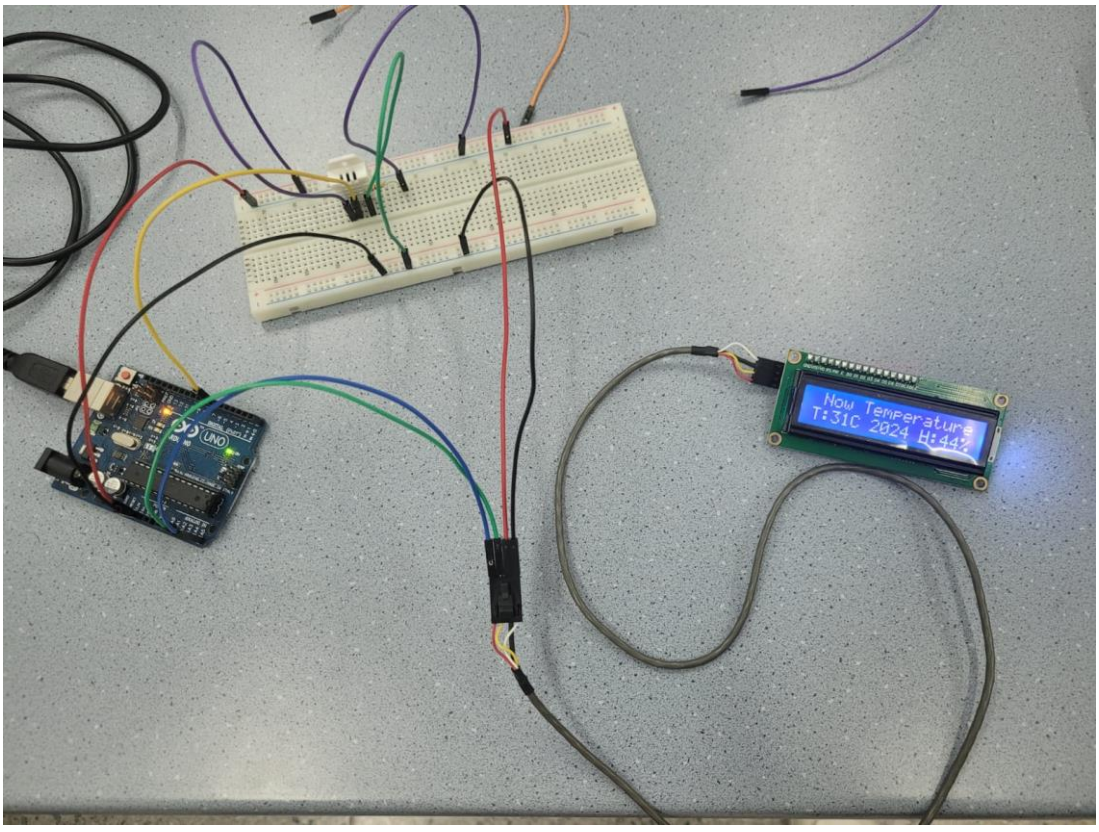

Figure 5.10. Humidity and Temperature Measurement on LCD Screen and Output

5.1.5. Wi-Fi Module Connection

The figure shows a schematic of the ports of the ESP8266 Wi-Fi Module. By using the schematic design, the standard port connections among Wi-Fi Modules with different product codes can be converted, thus resolving incompatibility issues. This way, even if the current product does not have the same pins as other products on the market, Arduino connections can be easily established.

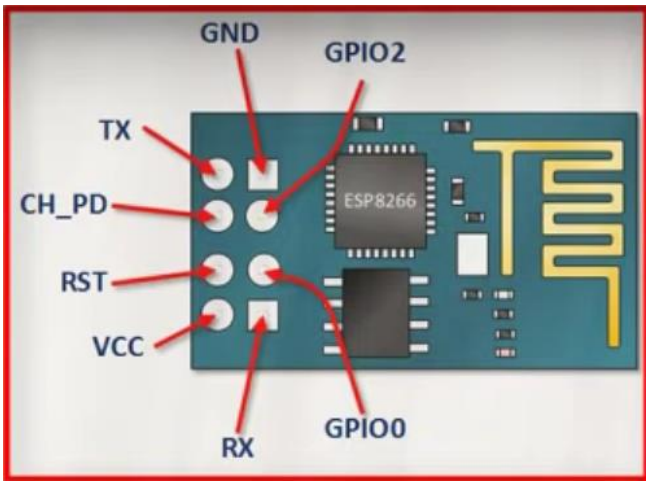

Figure 5.11. Socket Design of ESP8266

The ESP8266 Wi-Fi Module is used to connect and transfer data via the RemoteXY Android application or a web server. This allows for the design of a user-friendly GUI. The goal is to enable easy tracking and visualization of data on the user side. The application includes various elements such as text, graphics, scales, buttons, and switches. Users can control the experiment and monitor the data using these elements. The ESP8266 is designed to be used in our project to monitor characteristics such as room and water temperature, humidity, pH, TDS, and water level. The design schematic for the Wi-Fi Module connection is as follows:

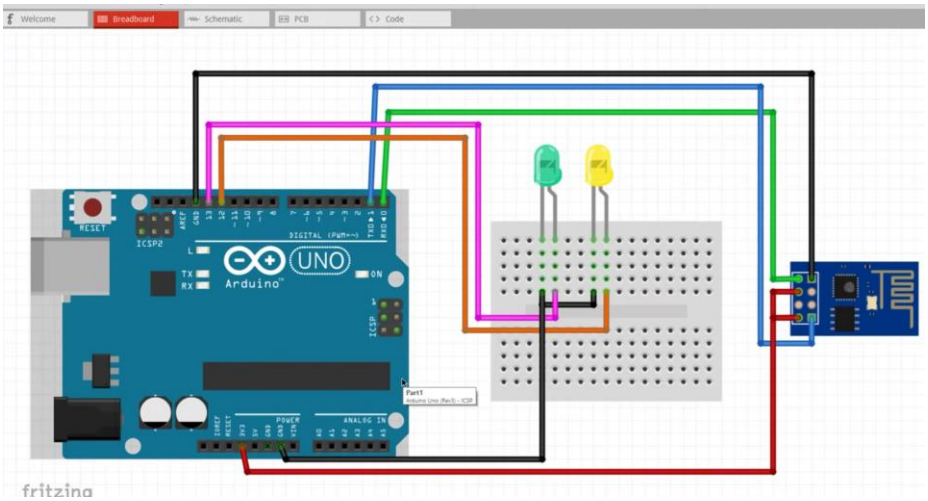

Figure 5.12. Design Schematic of Wi-Fi Module Connection

The following figure shows the application made for the ESP8266 module. In this application, each lightbulb in a circuit with 2 lightbulbs is controlled by a switch and a button. In the figure, it can be seen that the lightbulbs are in the on position when the switch and button combinations are on.

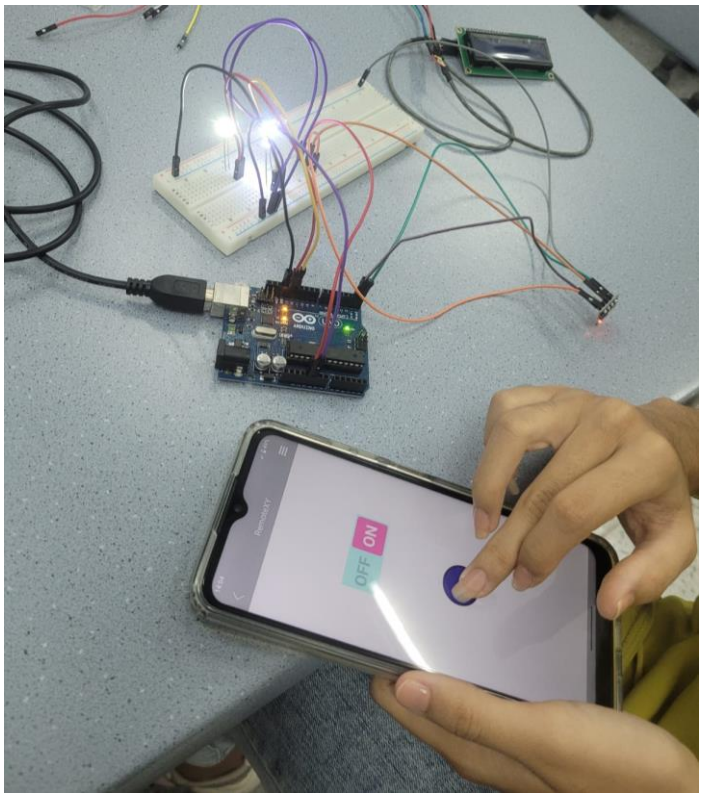

Figure 5.13. Wi-Fi Module Connection

The following figure shows the Android application GUI interface connected via RemoteXY.

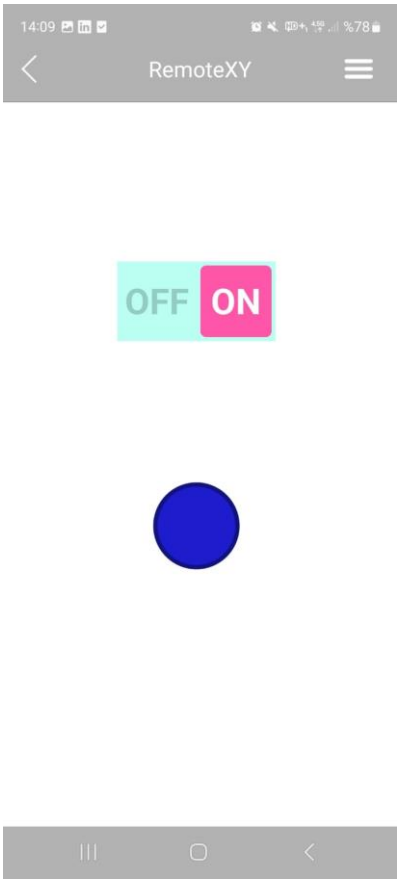

Figure 5.14. GUI on Android within RemoteXY Application

5.1.6. Wi-Fi Module Connection for Data Transmission via Web Server

A Wi-Fi Web Server connection was attempted using the ESP8266 Wi-Fi Module. Work was done on transmitting temperature values from the DS18B20 Temperature sensor to the web server. The temperature sensor data was correctly received, but it could not be transmitted due to the Web Server connection not being established.

5.2. Work Done in Second Week

In the second week of the project, experimental setup and combination of sensors were done and ensured that all the sensors work together without any problem.

5.2.1. Real Time Clock (RTC) Connection on LCD Screen

A connection between the RTC (Real-Time Clock) and Arduino was established. The current date (day/month/year) and time (hour:minute:second) were displayed on an LCD screen through the established connection. The design schematic for the Real Time Clock (RTC) connection is as follows:

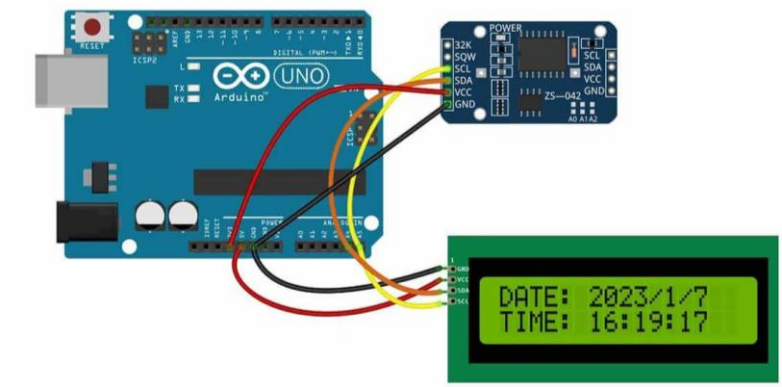

Figure 5.15. Design Schematic of RTC and LCD

The output obtained on the LCD screen for the date and time information obtained through real-time clock (RTC) for real-time measurement is as follows:

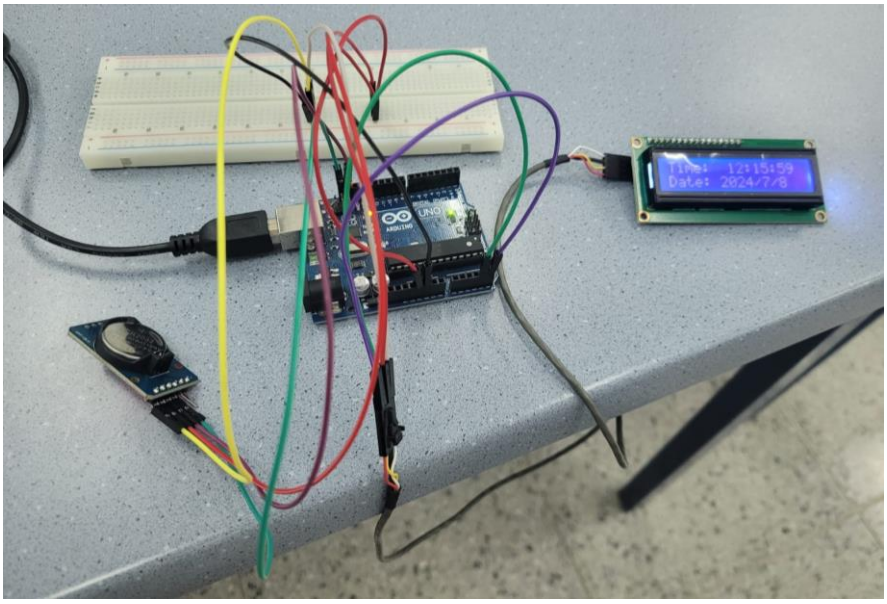

Figure 5.16. Real-Time Clock Module Output on LCD Screen

5.2.2. Relay Switch Module Connection on Lightbulbs

Sequential turning on and off of four bulbs using a 12V Relay Switch Module 4 Channel was worked on. The switches on the relay connections were activated in the correct order and duration by the executed code. However, the connection between the relay and the bulbs could not be established. That's why, COM, NO, and NC ports were researched and studied to understand well. The design schematic for the Relay Switch Module 4 Channels connection is as follows:

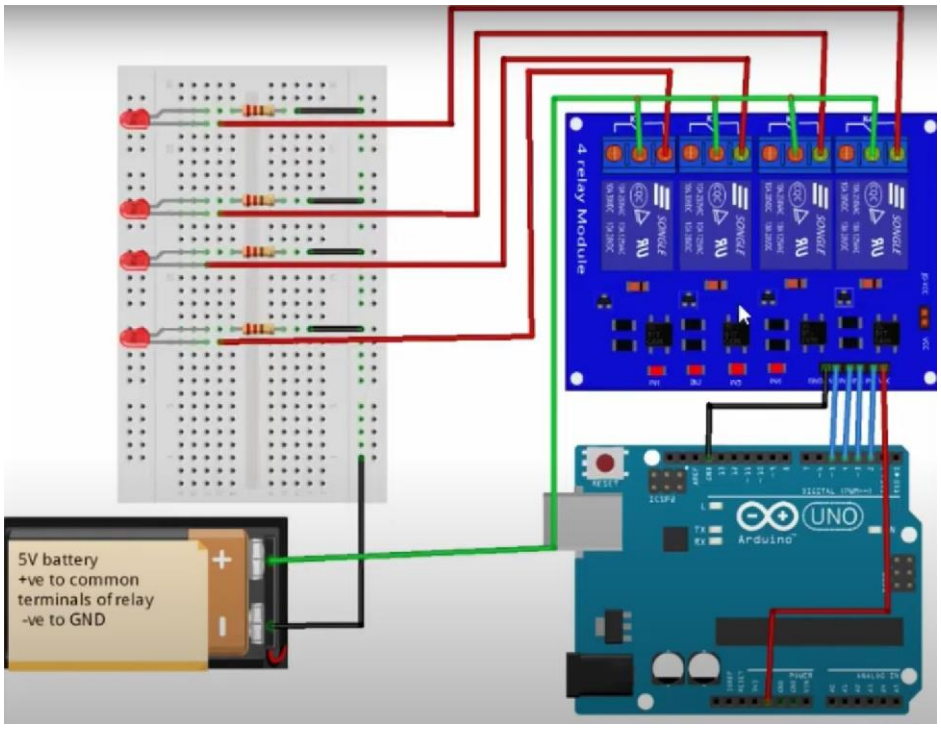

Figure 5.17. Design Schematic of Relay Switch Module with Lightbulbs

A single-channel relay switch and a single bulb design schematic were developed to understand how a single relay channel operates, after the design with a 4-channel relay switch and 4 bulbs did not yield results. The experiment was successful. The design schematic for the single relay switch and bulb is as follows:

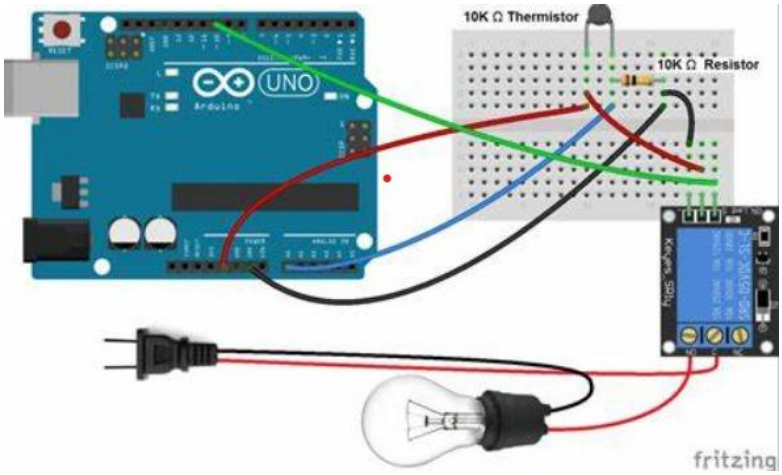

Figure 5.18. Design Schematic of Relay Module One Channel and a Lightbulb

The setup for the unsuccessful experiment with a single relay switch and a single bulb, attempted on Monday at the beginning of the week, initially did not yield results. Connections were subsequently reviewed, and further research on the matter was conducted. The design for the single relay switch and single bulb experiment conducted on Monday is as follows:

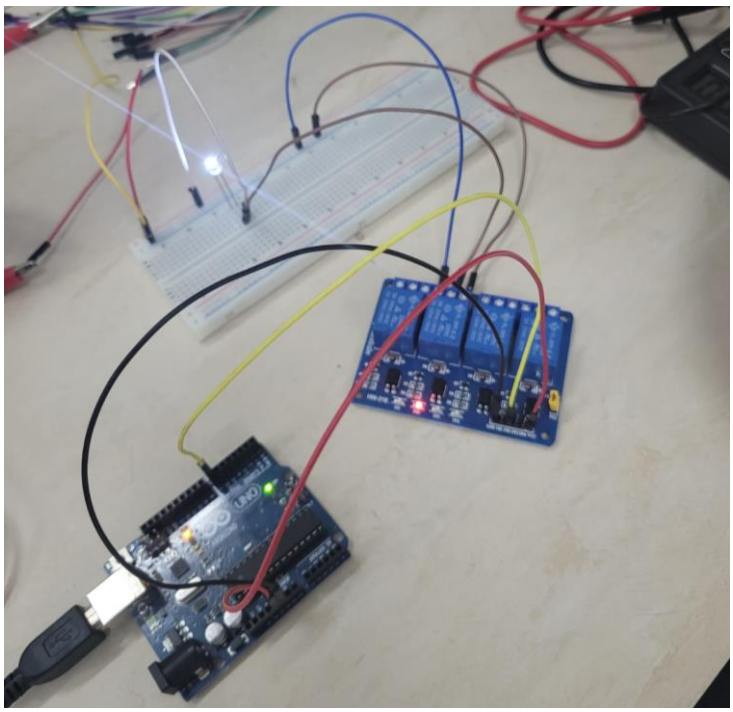

Figure 5.19. Failed Attempt of Relay Switch Module Connection with Lightbulb

Due to the failures in the experiments, the NO (Normally Open), NC (Normally Closed), and COM terminals on the switch were examined in subsequent attempts. The NO (Normally Open), NC (Normally Closed), and COM (Common) terminals on a relay switch module serve distinct functions in controlling electrical circuits. The NO terminal remains open when the relay is not activated, allowing current to flow when the relay is energized. In contrast, the NC terminal is normally closed, maintaining a circuit connection that opens when the relay is activated. The COM terminal serves as the common connection point for both NO and NC circuits, providing a central path for electrical current depending on the relay's state. Understanding these terminals is crucial for configuring relay switches to control devices effectively based on desired operational conditions. The positions of the NO (Normally Open), NC (Normally Closed), and COM (Common) terminals on the relay are as shown in the figure below:

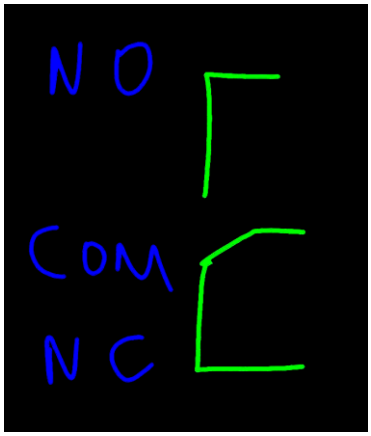

Figure 5.20. NO, COM and NC Ports of Relay Switch Module

In the following day, the bulb lighting experiment using the Relay Switch Module 4 Channel was successfully conducted. The issues encountered the previous day were identified. The module was not functioning due to the Arduino output voltage being 5V while the Relay Switch Module required a 12V input. The issue was resolved by adding a NPN BC547 transistor, which amplified the Arduino's 5V output to the 12V level needed by the Relay Switch Module. Although signals were transmitted to the relays in the previous day's experiments, the switches did not work, and the bulbs did not light up. The issue was

resolved using the same method. The NPN BC547 transistor used is as shown in the figure below:

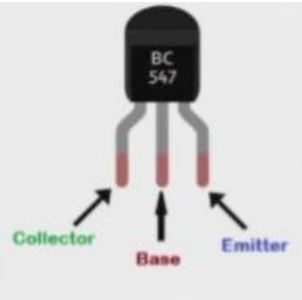

Figure 5.21. NPN BC547 Transistor

Additionally, the connections of the Relay Switch Module's VCC, GND, INPUT, and COM, NO, and NC outputs were correctly made between the Arduino's 5V, GND, and D signal control and the Power Supply's 12V and GND outputs. The Power Supply outputs were not obtained in the previous day's experiment, and changing the connection resolved the issue. The correct pairs of connections yielding successful results are as follows in the table below:

Table 5.1. Relay Switch Module Connections

Connection 1	**Connection 2**
Relay Switch Module VCC	12V Power Supply VCC
12V Power Supply VCC	Breadboard
Relay Switch Module Input2	Transistor Emitter
Relay Switch Module NO	Positive Side of Lightbulb and 220 Ω Resistor
Board Negative (Power Supply GND)	Negative Side of Lightbulb
Breadboard GND (Power Supply GND)	Relay Switch Module GND
Arduino UNO 7 Channel	Transistor Base
12V Power Supply VCC	Relay Switch COM
Power Supply GND	Breadboard negative
Breadboard GND (Negative)	Arduino GND

When examining the ports of the relay switch in the OFF state:

- *NO (Normally Open) Terminal:* Disconnected, no current flow.
- *NC (Normally Closed) Terminal:* Connected to the circuit, allowing current flow through the light bulb.
- *COM (Common) Terminal:* Provides the common connection point for the circuit.

The design and correct connections yielding successful results for the experiment conducted with a single relay switch and light bulb in the OFF state are as follows:

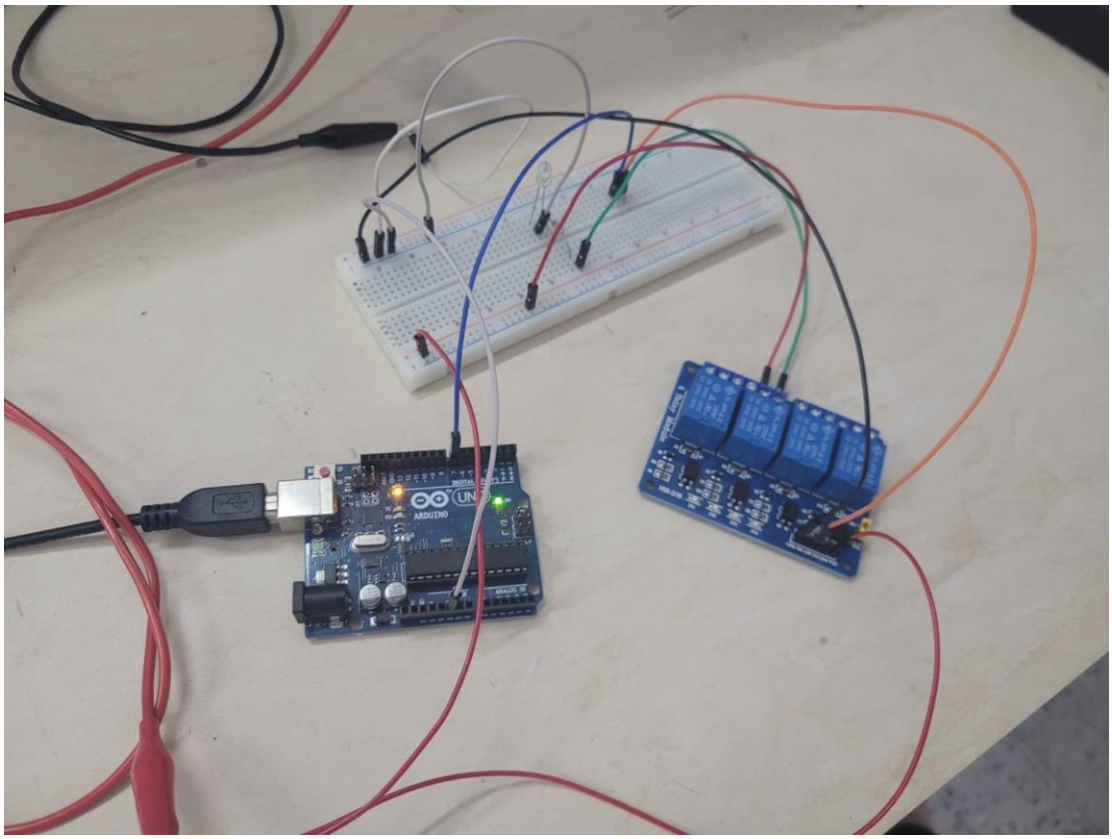

Figure 5.22. Relay Switch Module Connection with OFF State Lightbulb

When examining the ports of the relay switch in the ON state:

- *NO (Normally Open) Terminal:* Connected to the circuit, allowing current flow through the light bulb.
- *NC (Normally Closed) Terminal:* Disconnected, no current flow.
- *COM (Common) Terminal:* Provides the common connection point for the circuit.

This configuration ensures that when the relay is activated (ON state), the light bulb receives power through the NO terminal, and the NC terminal remains open, interrupting the circuit. The design and correct connections yielding successful results for the experiment conducted with a single relay switch and light bulb in the ON state are as follows:

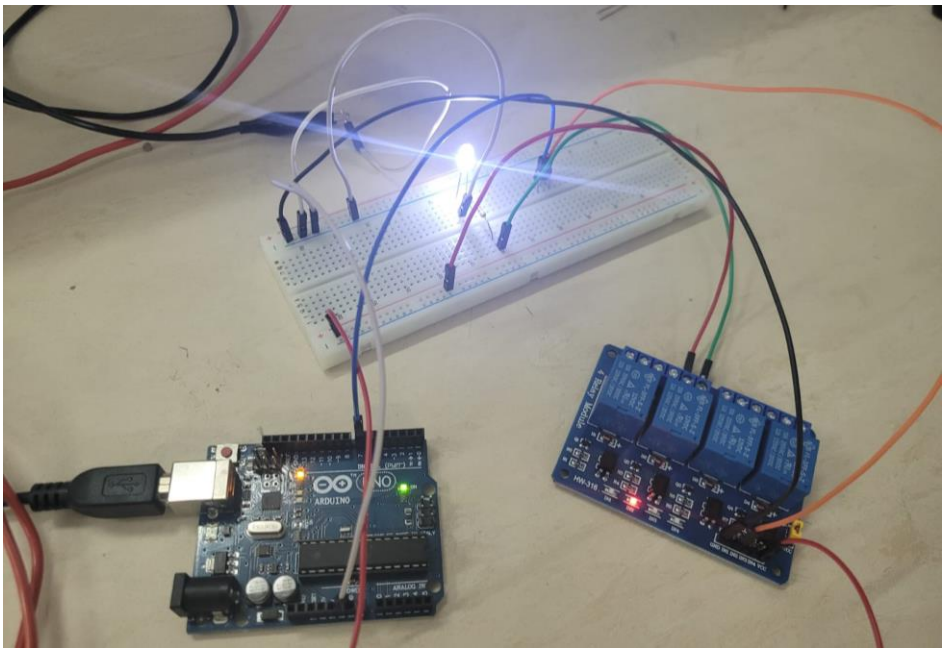

Figure 5.23. Relay Switch Module Connection with ON State Lightbulb

5.2.3. Air Pump and Air Stone Connection

The connection between the air pump and the air stone was established using a plastic tube, and it was operated in a basin. The connection between the air stone and air pump is as follows:

Figure 5.24. Air Pump and Air Stone Connection

5.2.4. Water Pump and Relay Switch Module Connection

The water pump was operated using the 12V output from the Power Supply and the Relay Switch Module, which required a 12V input. The connections made for the relay switch module with the bulb were set up on the water pump. An attempt was made to draw water from the basin for 10 seconds and transfer it for 2 seconds.

The connection of the relay switch and water pump is based on the previously used design schematic for the relay switch and lightbulb connection. Accordingly, the Water Pump and Relay Switch Module Connection is as follows:

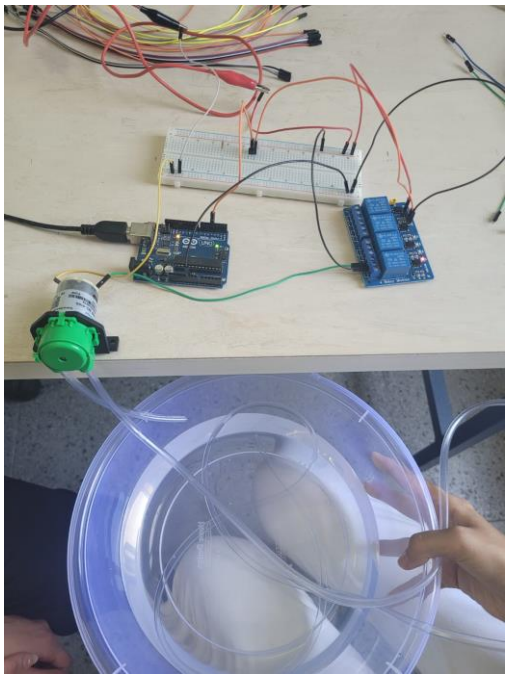

Figure 5.25. Water Pump and Relay Switch Module Connection

5.2.5. Wi-Fi Module Connection on Web Server

The ESP8266 Webserver operation was attempted, but it was unsuccessful due to library errors. It was determined that the errors occurred because the Arduino Uno used did not support Wi-Fi.

5.2.6. TDS and Water Quality Sensor Connection on LCD Screen

TDS (Total Dissolved Solids) and water quality were measured with a sensor, and the measurement results in ppm were displayed on an LCD screen. The design schematic for the TDS and LCD screen connection is as follows:

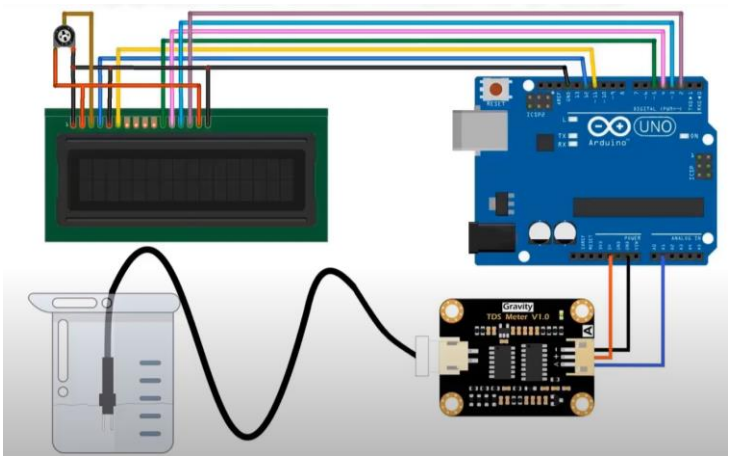

Figure 5.26. Design Schematic of TDS Sensor with LCD Screen Connection

The connections made according to the design schematic of the TDS and water quality sensors, as well as the output obtained on the LCD screen, are as follows:

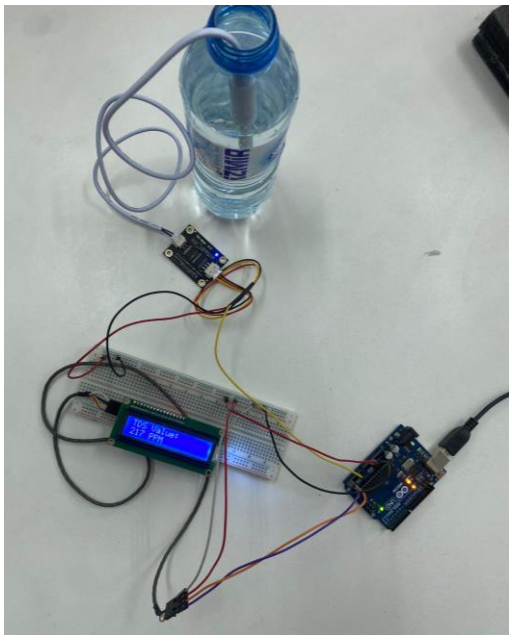

Figure 5.27. TDS Output on LCD Screen

It was observed that the TDS sensor, when immersed in water, measured a value of 221 ppm, which dropped to 0 ppm after the sensor was removed from the water. The observation result on the console, obtained through the Serial Monitor, is shown in the figure below:

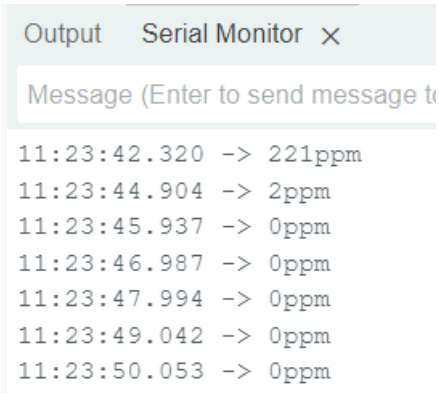

Figure 5.28. TDS Output on Serial Monitor Console

5.2.7. pH Sensor Connection

The datasheet for the pH sensor was reviewed. The measurement was postponed due to the mandatory 8-hour soaking preparation in special water before use. The design schematic of the pH sensor is as follows:

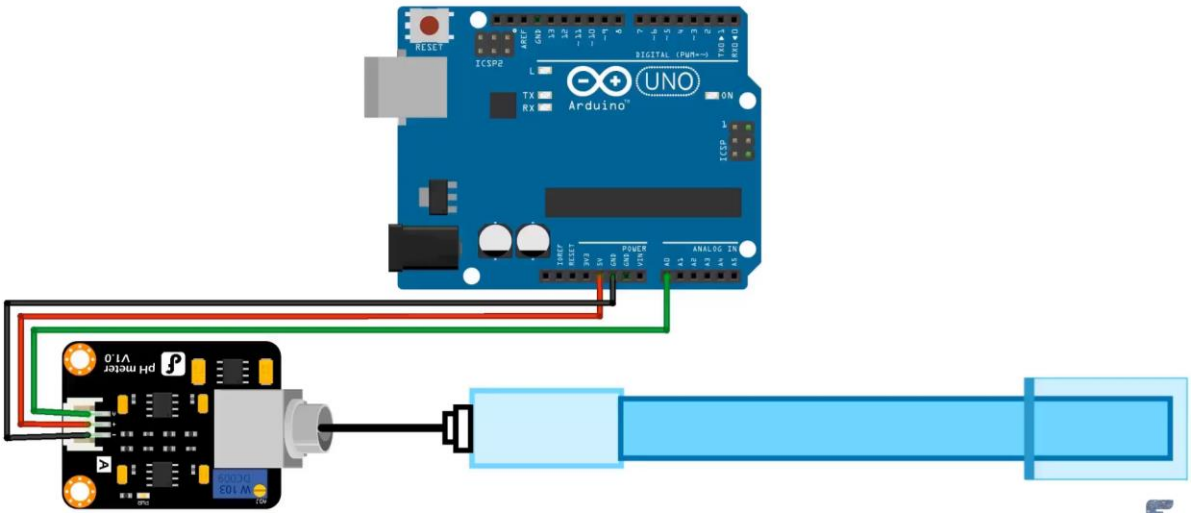

Figure 5.29. Design Schematic of pH Sensor

When the datasheet of the pH sensor was examined, it was stated that the sensor should be soaked in a KCl (Potassium Chloride) solution with a pH of 7, which is neutral, for 8 hours before use, either after a long period or when first opened. Therefore, DEU Chemistry Department was visited to get expert advice on the sensor and to prepare it for use. With the contributions of Dr. Taşkın Mumcu, the pH sensor was soaked in distilled water and calibrated to make it ready for use. It was determined that the values observed with the pH sensor in distilled water and 0.1 M NaOH test solution were accurate.

After ensuring that the pH sensor was ready for use, the laboratory was visited for testing. The pH sensor was connected according to the design schematic and operated with the written code. It was observed that the sensor was working correctly. The connection based on the design schematic of the pH sensor is as follows:

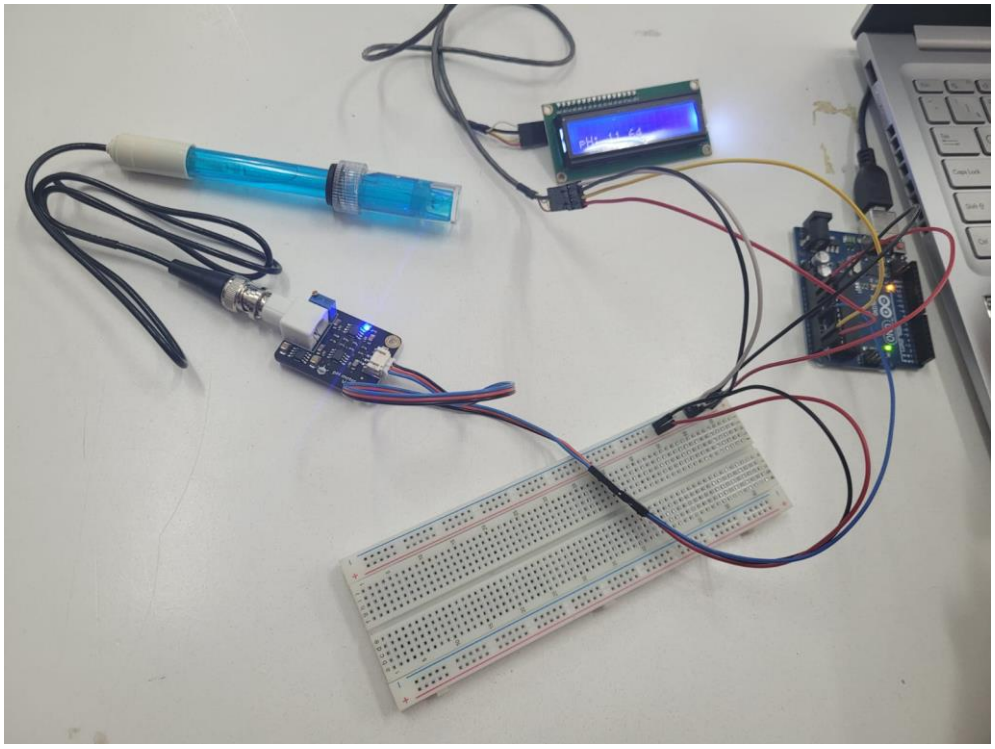

Figure 5.30. pH Sensor Output on LCD Screen

5.2.8. Experimental Setup

The focus was then placed on the hardware part of the design and the experimental setup was started. Holes were drilled for the placement of the sensors. A connection was established between the air stone and the air pump through the drilled hole for the air stone.

Necessary connections were made on the board through the drilled holes for the LCD, TDS, pH, temperature, humidity, and RTC sensor modules. The LCD was placed in a box to give it a television appearance, enhancing its visual appeal. The display of the LCD screen is as follows:

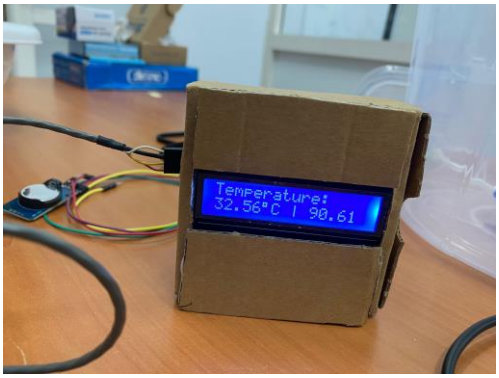

Figure 5.31. LCD Screen Output

Later, the focus shifted to the software part of the design. The codes were merged into a single code by adding the sensors to the board one by one. At the end of the design, it was ensured that the sensors worked harmoniously with each other. The values obtained from the sensors were displayed on the LCD screen in the following order: Time and Date, TDS, pH, water temperature, weather temperature, and humidity. The progress recorded regarding the design of the experimental setup is as follows:

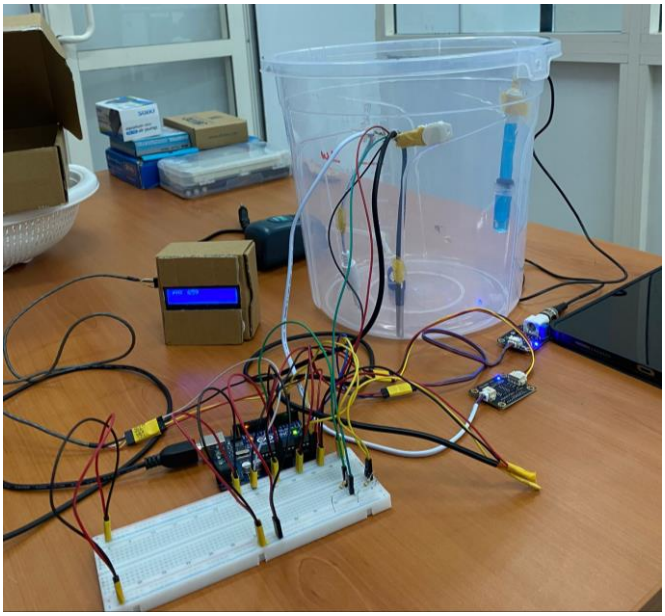

Figure 5.32. Experimental Setup Design

5.2.9. Wi-Fi Connection via RemoteXY

Wi-Fi connection has been done using a different library, "SoftwareSerial.h." Designs were carried out on RemoteXY; however, the work was canceled due to the free version not supporting more than 5 elements. The user interface, created through RemoteXY and consisting of a total of three pages, is shown in the figure below. After the application is activated, real-time sensor data will be reflected in place of the words "text" on the figure.

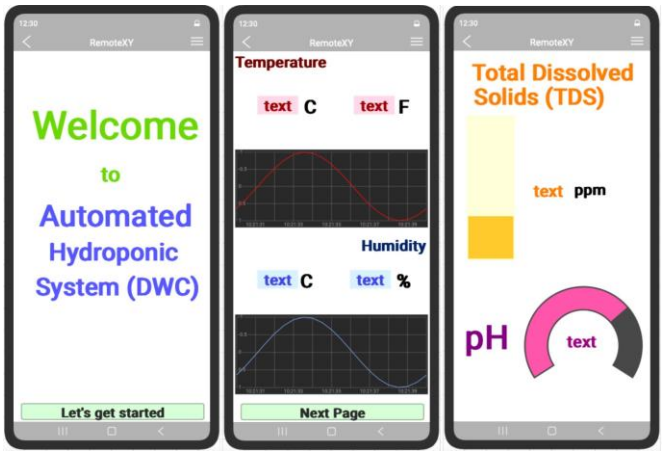

Figure 5.33. RemoteXY Application User Interface

However, as previously mentioned, the application does not support more than 5 widgets, so it has provided a single-page display for 20 seconds. After that, a purchase screen appears, allowing the display right for only one user. The figure regarding the situation is shown below.

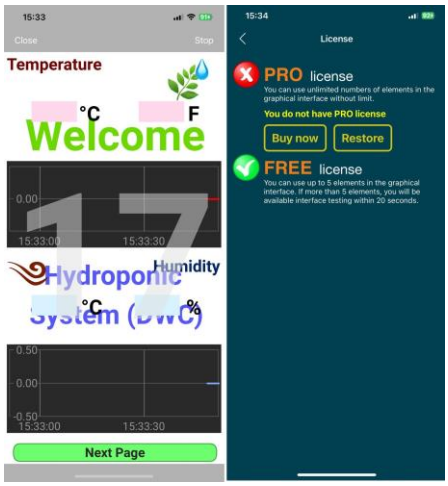

Figure 5.34. Limited Display and License Screen

5.3. Work Done in Third Week

In the third week of the project, hardware elements were added to the system and ensured that it works appropriately. Wi-Fi module was integrated into the system to track the specific characteristics indicated earlier.

5.3.1. Water Level Sensor Connection

The water level sensor was run on the written code. The water level value and the four statuses, "empty," "low," "medium," and "high," were visualized on the LCD screen. The water level sensor was tested on the hardware along with other sensors, and its output was shown on the LCD screen. The code for the water level sensor was added to the overall code. The design schematic for the water level and LCD screen connection is as follows:

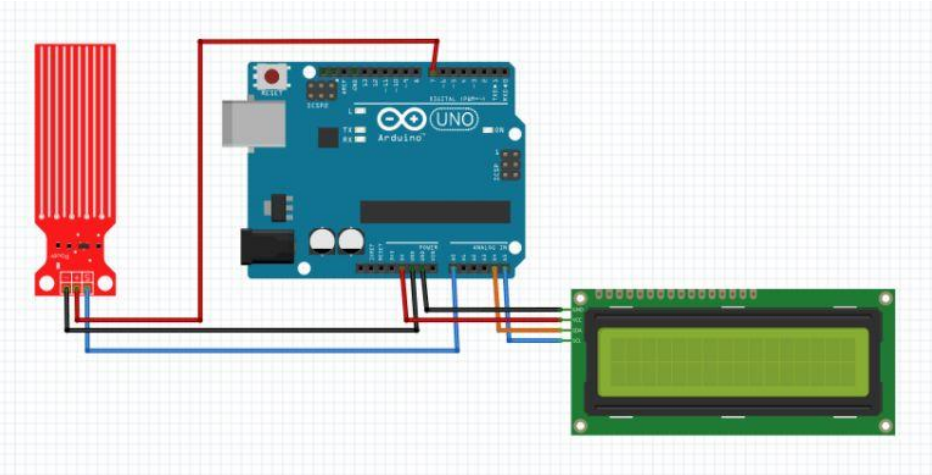

Figure 5.35. Design Schematic of Water Level Sensor and LCD Connections

The output regarding the water level sensor on the LCD screen is as follows. In the sample application, the water level sensor was kept outside of the water, resulting in an output of 0 mm and "EMPTY" in terms of level.

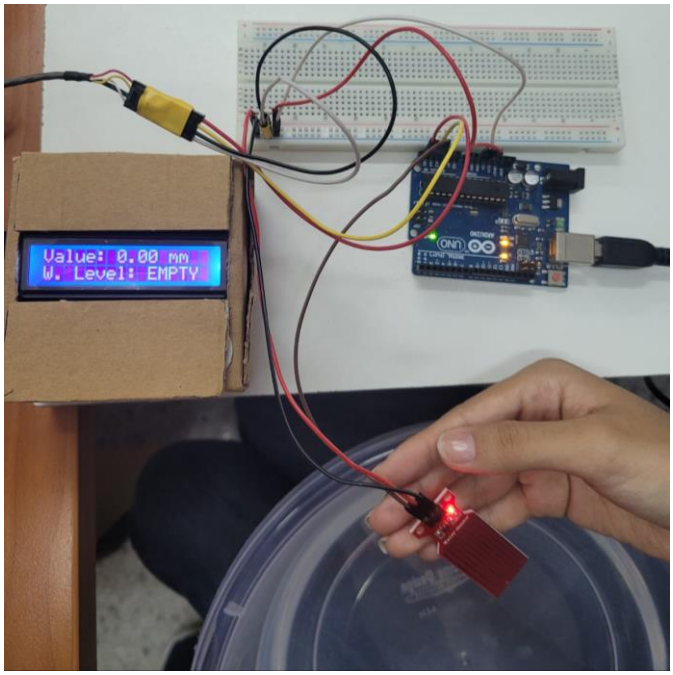

Figure 5.36. Water Level Sensor Output on LCD Screen

Based on the sensor value obtained from the water level measurement, if the water level is 0, the output is "EMPTY". For values up to 350, the output is "LOW"; for values up to 510, the output is "MEDIUM"; and for values above 510, the output is "HIGH". In the experiment conducted to verify the sensor's accuracy, 4 different states were examined. The mentioned outputs are shown in the figure below:

Figure 5.37. Four States of Water Level

5.3.2. Building the Hardware Side of the System

The pumice stones were washed and cleaned of dust, then added to the Automated Hydroponic System hardware. The visual regarding the cleaning of pumice stones is shown below:

Figure 5.38. Cleaning Pumice Stones

The renewed water pump was tested. It was observed that it provided stronger water suction and transmission thanks to the silicone tube. For visual improvements, the box in which the sensors and cables would be housed was painted. The hardware setup where the plant will be placed was created in an orderly manner. The arrangement made in the experimental setup is shown in the image below:

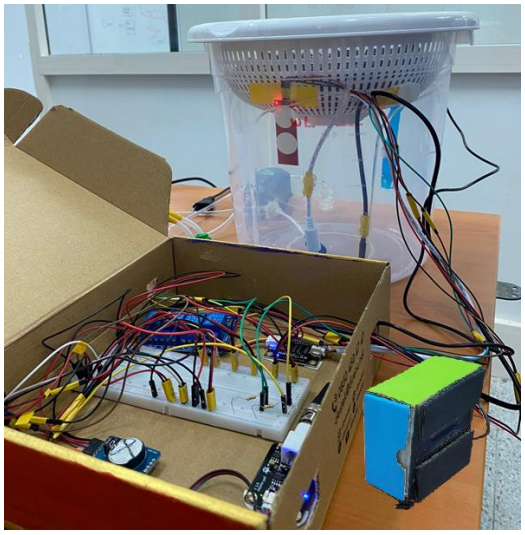

Figure 5.39. Organized Experimental Setup

The cables and sensors were fixed and hidden inside a box. Water and fertilizer reservoirs to be connected to the water pump were created, and visual improvements were made. As previously mentioned, to create a more organized environment for the experiment, boxes to contain the experimental setup were painted. The visual improvements made on the experimental setup are shown below:

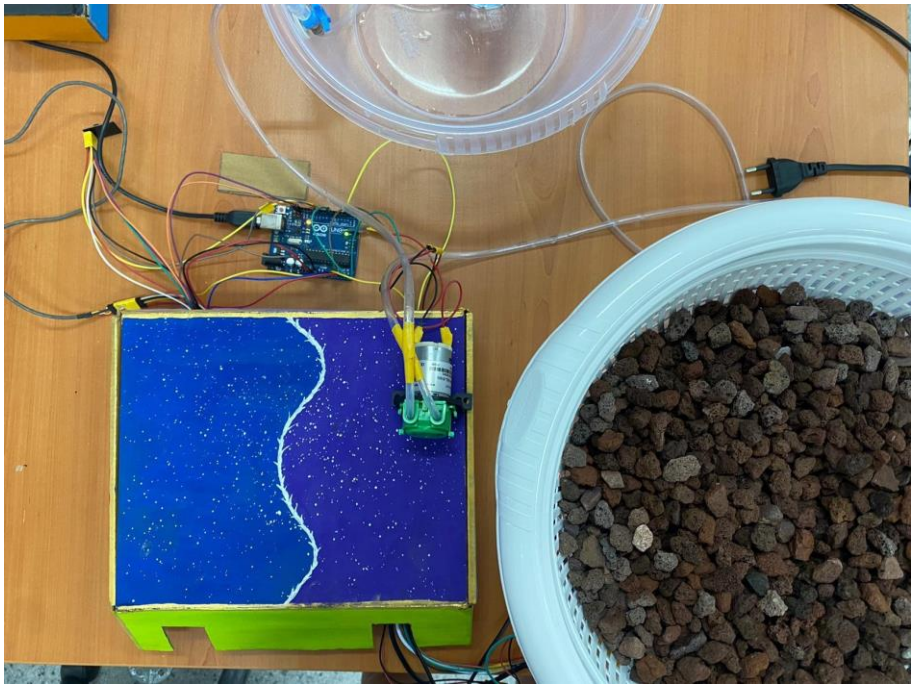

Figure 5.40. Visual Improvements on Design

5.3.3. Building a Web Site

A website for the webserver to be used was created. HTML, CSS, and JavaScript codes were written for the created website. HTML was used for the skeleton of the website, while visual enhancements were provided through CSS. Functional features on the site were coded with JavaScript to ensure functionality. A logo and photos representing the group were added to the site. Graphical areas designed to visualize the data from the sensors were prepared.

A domain was obtained from the Infinity Free site for the website to be used for the transmission of data from the sensors. The web address used was chosen as www.ahs-ygs-

deu-ceng.000.pe/?i=1 The figure regarding the mentioned details of the website is shown below:

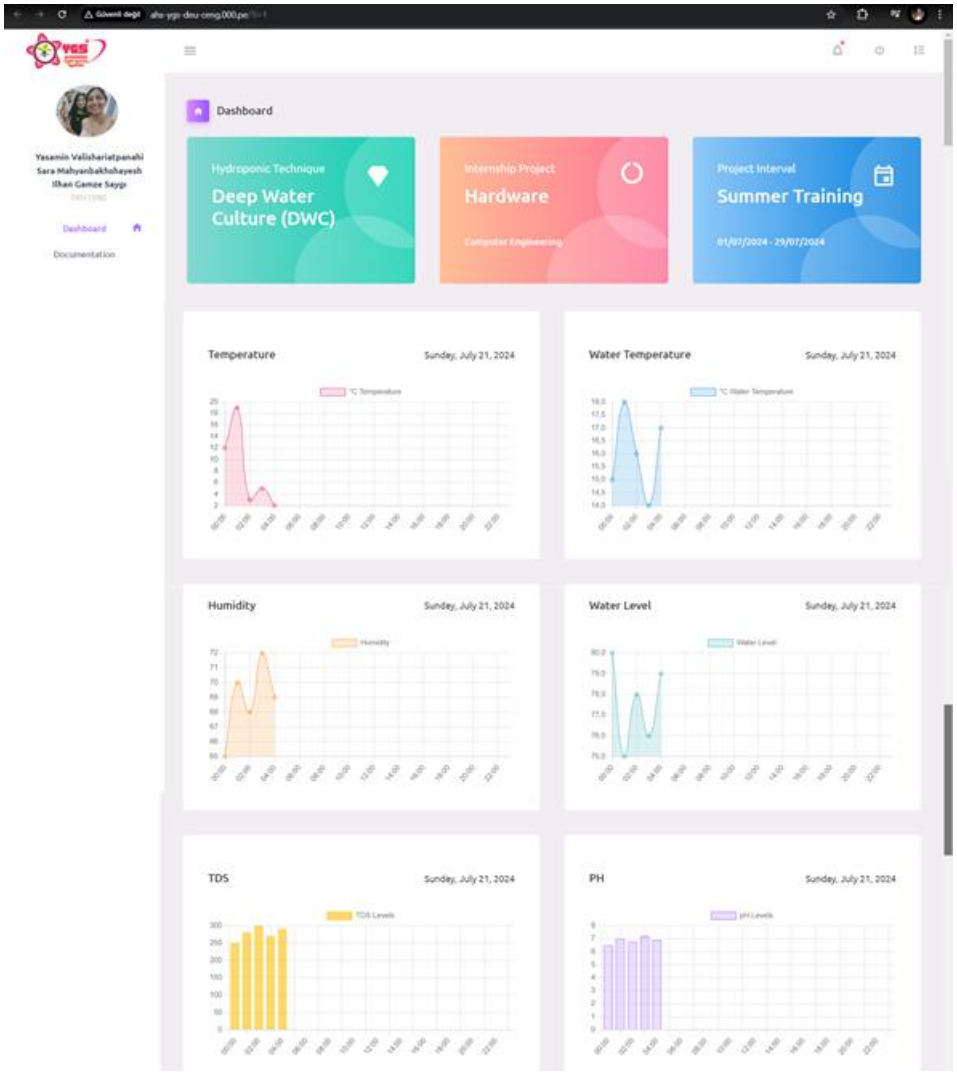

Figure 5.41. Website Design

Additionally, indicators showing the offline and online status of the sensors were added to the main screen. Furthermore, a calendar displaying the current date is also included. The figure regarding the mentioned details is shown below:

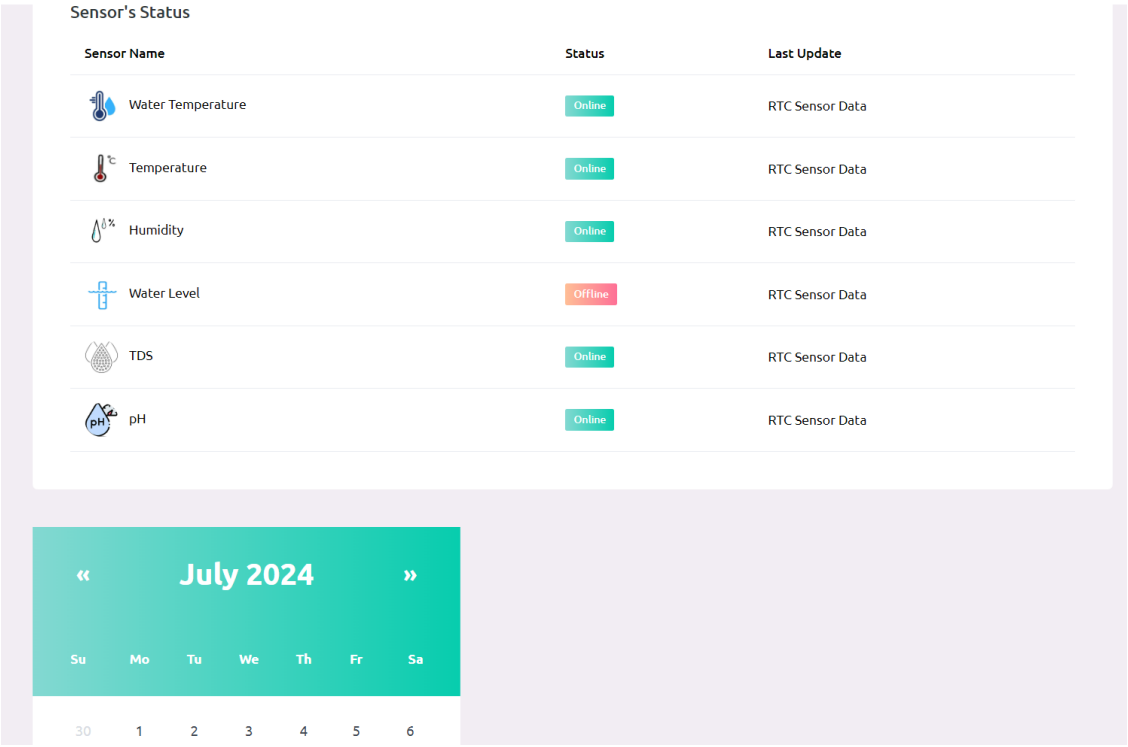

Figure 5.42. Sensor's Status and Calendar

5.3.4. Data Transmission on Web Server

During research conducted to overcome the connection errors in the ESP8266 Wi-Fi Module, the ThingSpeak site was discovered. To connect to ThingSpeak, a modification was made in the design schematic to place the TX and RX wires of the ESP8266 in different locations compared to the classic connection. Initially, the Temperature and Humidity data from the DHT22 were sent to the web server. Additionally, to overcome library issues related to the ESP8266, the Arduino version was downgraded from 2.3.2 to 1.8.19. The design schematic used for the connections is shown below:

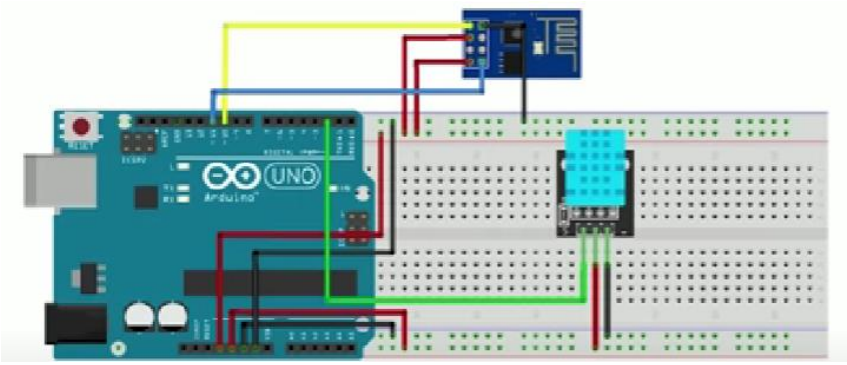

Figure 5.43. Design Schematic of ESP8266 and DHT22 Connection

The connections made between the Arduino UNO and the ESP8266 are detailed in the table below. The CH-PD acts as the enable pin, while TX is the pin for sending data, and RX is the pin for receiving data.

Table 5.2. Arduino UNO and ESP8266 Connections

Arduino UNO	**ESP8266 Wi-Fi Module**
GND	GND
3.3V VCC	VCC
3.3V VCC	CH-PD (EN)
10	TX (data sent)
11	RX (data received)

In the experiment using the IP address and GET API commands of the ThingSpeak site, the Temperature and Humidity data from the DHT22 were transmitted to the fields and then to the ThingSpeak site. Thus, it was understood that there was no problem with data transmission on the web server with the ESP8266 module, the problem was with connecting through localhost in the previous codes, and that the module needed to be assigned an IP address. The web server was set up using only the "SoftwareSerial.h" library, thereby overcoming potential library issues. The visualization of the data transmitted from the sensors on ThingSpeak is shown in the figure below:

Figure 5.44. ThingSpeak Web Server

Data transmission through the ESP8266 Wi-Fi module was ensured to be sent and visualized to the ThingSpeak web server on a minute-per-time-period basis for each sensor. In the initial phase, 4 hours of data were collected flawlessly. The connection with the ThingSpeak Web Server and the tracking of data transmission from the sensors were monitored via the Serial Monitor. It can be seen from the output that the data is being

transmitted on a minute-by-minute basis. The output of the Serial Monitor is shown in the figure below:

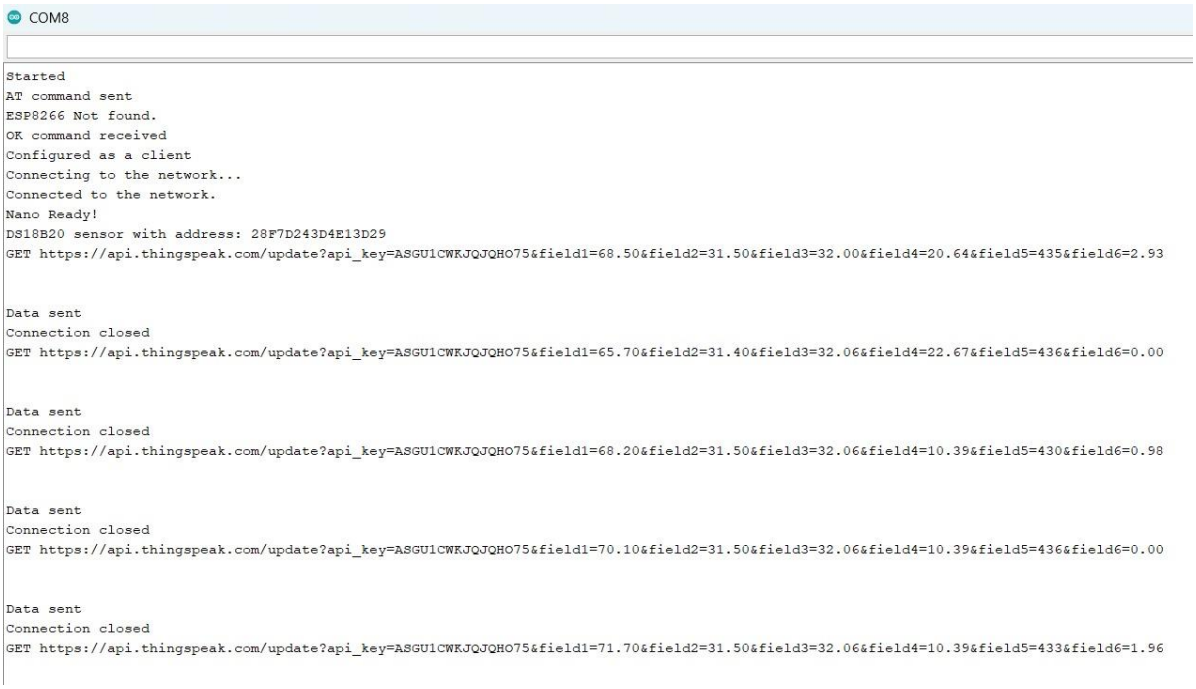

Figure 5.45. ThingSpeak Connection on Serial Monitor

While displaying on the LCD screen with the combined code for the web server and sensors, a "Low memory available, stability problems may occur." warning was encountered on the compilation screen. Following this warning, issues like garbage values on the LCD screen and disconnections and reconnections leading to an infinity loop were observed. It is thought that the problem is related to the amount of RAM available on the current Arduino UNO. To overcome the problem, efforts will be made to shorten the code and variable names. The screenshot regarding the warning encountered is shown below:

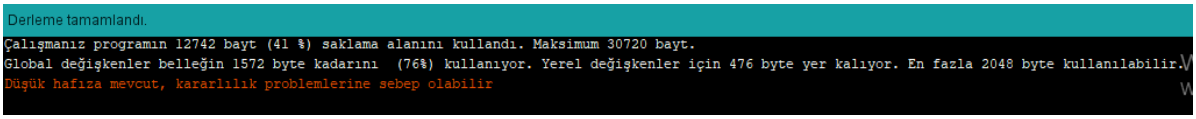

Figure 5.46. Low Memory Warning

5.4. Work to be Done in Fourth Week

In the fourth week of the project, web server was completed with a user-friendly GUI design. The experimental setup was completed and the data which comes from sensors were collected. Testing and debugging phases were maintained. The situation of the plant in the system was tracked and monitored.

5.4.1. Building a Web Site on Glitch.me

A database was created on the local host using Node.js and Express.js through the ESP8266 Wi-Fi module. To transition from the local host to the website, a domain was acquired, and a website was created using free hosting provided by Glitch.me.

Real-time data from the sensors was attempted to be transferred to the website hosted on Glitch.me, but it was not successful. To transfer data to the website through Glitch.me, research was conducted on the API, but the sensor data could not be transferred to the website set up on Glitch.me due to the backend knowledge required by the API. The screenshot of the website connected to Glitch.me is shown in the figure below:

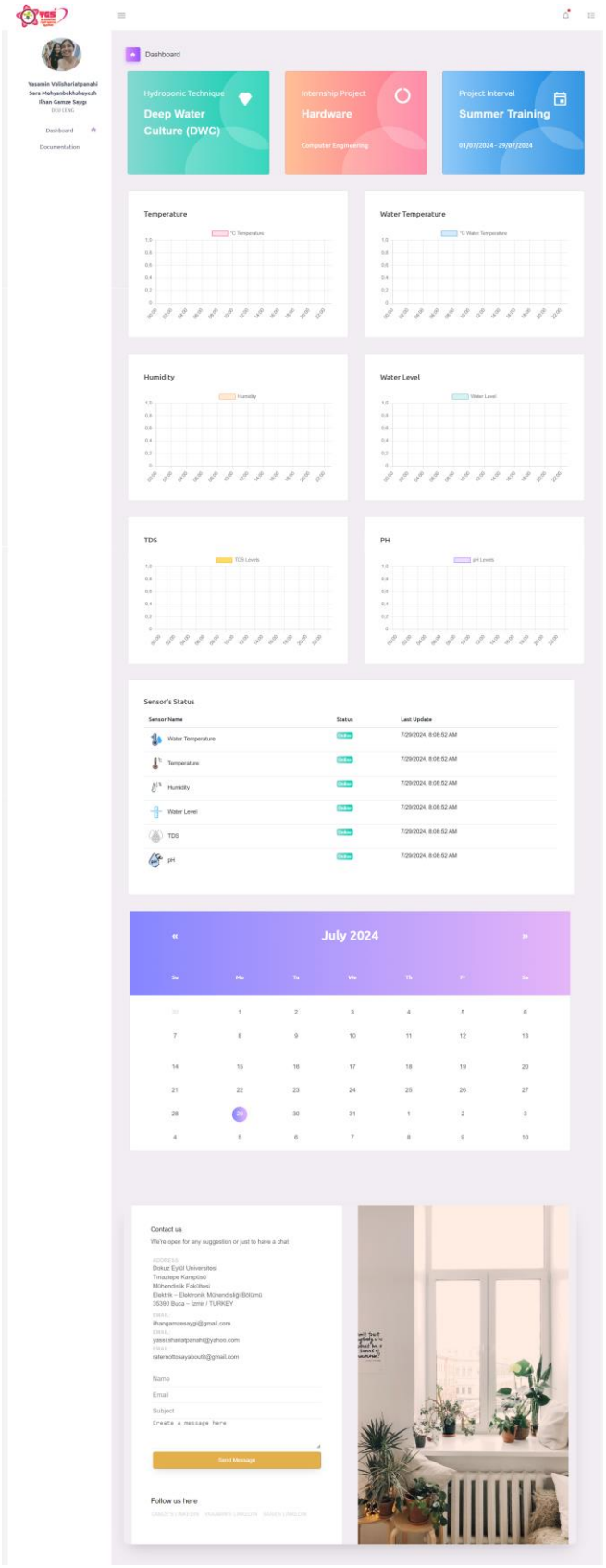

Figure 5.47. Website Connected to Glitch.me

5.4.2. Building a Web Site on ThingSpeak

The sensor data that was previously sent to ThingSpeak was successfully retrieved and transferred to the website hosted on the domain obtained through Infinity. By doing so, the integration of the automated hydroponic system with the website was completed, allowing users to view real-time updates seamlessly.

During the development process, a low memory warning was encountered, which posed a challenge to the stability and performance of the system. This issue was effectively resolved by optimizing the code. Specific measures included commenting out certain non-essential parts of the code, such as the lines responsible for printing to the Serial Monitor and shortening variable names to reduce memory usage. These adjustments proved sufficient to eliminate the warning and improve the system's efficiency.

Following these modifications, the web server connection was successfully established. The website achieved full functionality, becoming entirely accessible to users. This meant that users could now visit the website to monitor the hydroponic data being collected and displayed in real time.

The accompanying figure below provides a visual representation of the website when it was connected to ThingSpeak. This screenshot highlights the successful data retrieval and display, showcasing the seamless integration and functionality of the entire system.

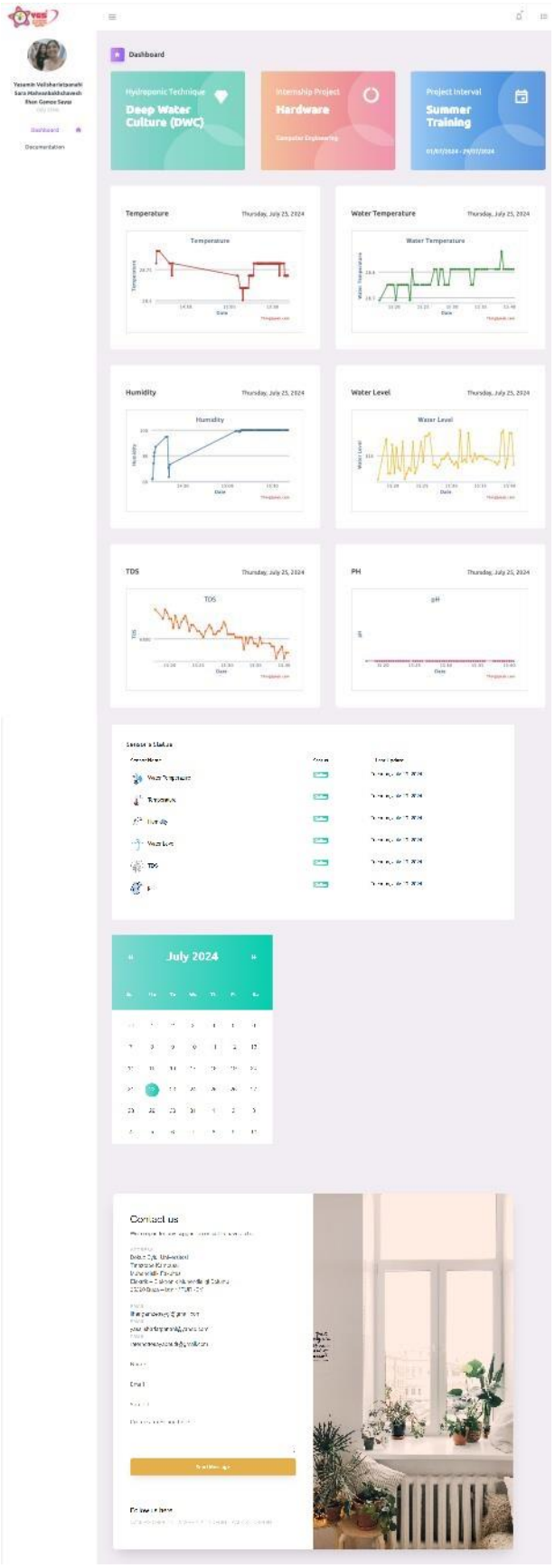

Figure 5.48. Website Connected to ThingSpeak

5.4.3. Completion of Experimental Setup

The plant to be placed in the setup was washed and positioned onto the experimental setup. Due to the pH sensor outputting 0, an investigation was conducted. It was determined through research that the value displayed on the screen was the voltage value, not the pH. Based on the sensor's datasheet and the 420 V output it provides in pure water with a known pH of 7, a regression equation for converting voltage to pH was attempted to be derived. The datasheet of the pH electrode characteristics shows the conversion from voltage to mV, as depicted in the figure below:

Table 5.3. pH Electrode Characteristics Datasheet

VOLTAGE (mV)	pH value	VOLTAGE (mV)	pH value
414.12	0.00	-414.12	14.00
354.96	1.00	-354.96	13.00
295.80	2.00	-295.80	12.00
236.64	3.00	-236.64	11.00
177.48	4.00	-177.48	10.00
118.32	5.00	-118.32	9.00
59.16	6.00	-59.16	8.00
0.00	7.00	0.00	7.00

Fertilizer was included in the system. Following the usage instructions, a teaspoon of fertilizer was added to a prepared PET bottle containing half a liter of water. Visual improvements were made to the fertilizer PET bottle. To allow the flow of either fertilizer-enhanced water or tap water into the system, a valve head was used, enabling the selection of water or fertilizer feed as needed. The experimental setup was completed. The setup was connected to the power supply and ensured to be operational. Sensor data was collected on the web server. The final state of the experimental setup is shown in the figure below:

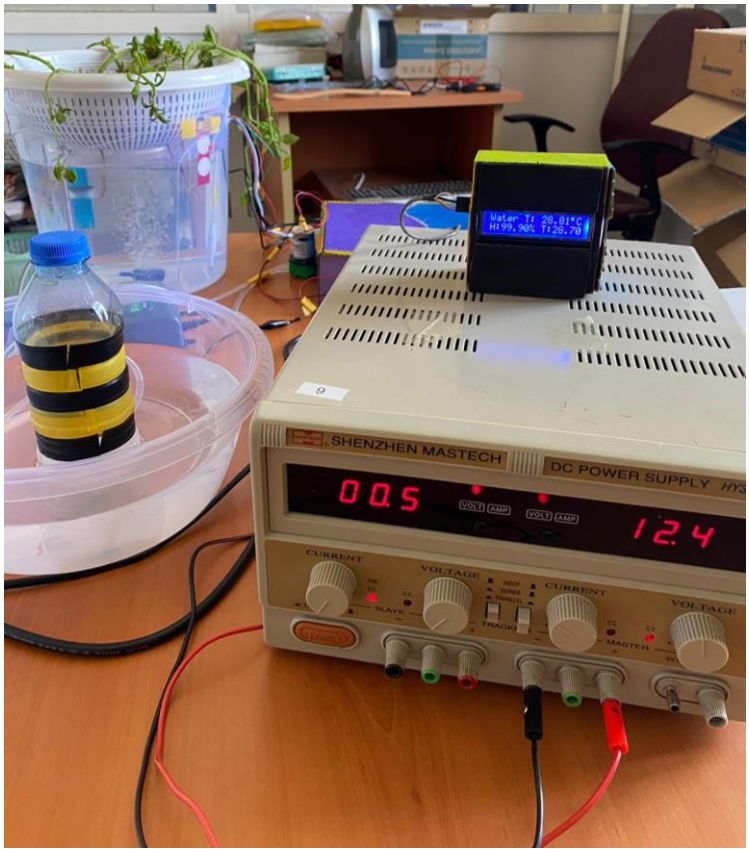

Figure 5.49. Experimental Setup Final State

CHAPTER SİX CONCLUSION

The completion of our automated hydroponic system (DWC) project marks a significant milestone in our journey towards mastering the intricacies of modern home gardening techniques. Over the span of a month, our dedicated team of three has collaborated intensively, resulting in a robust and efficient system designed to promote healthy plant growth through innovative automation.

Throughout this project, we have successfully integrated various components, including sensors, pumps, and controllers, to create a seamless and self-sustaining hydroponic environment. The system's ability to monitor and regulate essential parameters such as temperature, humidity, and nutrient levels ensures optimal conditions for plant growth, reflecting the precision and reliability of our design.

Our work on this project has not only been technically rewarding but also immensely enjoyable. We have navigated through numerous challenges, from integrating hardware with software to fine-tuning the system for peak performance. These experiences have significantly enhanced our problem-solving skills, technical knowledge, and teamwork abilities. Each obstacle we encountered served as a valuable learning opportunity, reinforcing our understanding of automated systems and hydroponic gardening.

The project's successful execution has equipped us with a diverse set of skills, including programming, system integration, and real-time data analysis. Moreover, our ability to work collaboratively under tight deadlines has been a testament to our dedication and passion for the project. This experience has undoubtedly prepared us for future endeavors in home gardening and automation.

In conclusion, our automated hydroponic system project has been a resounding success, yielding both a functional gardening solution and a wealth of knowledge and skills. As we move forward, we are excited to apply what we have learned to enhance our home gardening practices, confident in our ability to innovate and excel in creating efficient and sustainable gardening solutions.

REFERENCES

[1] Tronics Lk. *Arduino LCD I2C Tutorial*. https://www.youtube.com/channel/UCYJa3gs8q49-N3TLm-7ygUw?sub_confirmation=1, Last Accessed Time: 3 Jul 2024.

[2] Brainy-Bits. *DHT11 Temperature & Humidity sensor with Arduino – Tutorial*. (9 Jan 2015). https://www.youtube.com/watch?v=OogldLc9uYc&ab_channel=Brainy-Bits, Last Accessed Time: 4 Jul 2024.

[3] milliohm. *How to use DS18B20 Temperature sensor with arduino.* (13 Feb 2021). https://www.youtube.com/watch?v=Y1__vmkr8-g&ab_channel=miliohm, Last Accessed Time: 4 Jul 2024.

[4] Virginia Tech. (n.d.). Hydroponic Production of Edible Crops: Deep Water Culture (DWC) Systems. Retrieved from https://ext.vt.edu/content/pubs_ext_vt_edu/en/SPES/spes-464/spes-464.html

[5] Iswanto, Prisma Megantoro, & Alfian Ma'arif. (2020). Nutrient Film Technique for Automatic Hydroponic System Based on Arduino. *In 2nd International Conference on Industrial Electrical and Electronics (ICIEE).* Retrieved from https://www.researchgate.net/publication/347425278_Nutrient_Film_Technique_for_Automatic_Hydroponic_System_Based_on_Arduino

[6] Muhammad Daud, Vandi Handika & Andik Bintoro. (2018). Design and Realization of Fuzzy Logic Control for Ebb and Flow Hydroponic System. *In International Journal of Scientific & Technology Research.* Retrieved from https://www.researchgate.net/publication/327869009_Design_And_Realization_Of_Fuzzy_Logic_Control_For_Ebb_And_Flow_Hydroponic_System

[7] Awhile Ani & Prakash Gopalakirishnan. (2020). Automated Hydroponic Drip Irrigation Using Big Data. *Proceedings of the Second International Conference on Inventive Research in Computing Applications.* Retrieved from https://www.researchgate.net/publication/344051655_Automated_Hydroponic_Drip_Irrigation_Using_Big_Data

[8] Astronaut A. *Arduino Tutorial : Controling 4 relay module for use with a 5 volt water pump.* (9 Dec 2018). https://www.youtube.com/watch?v=Z0SZ-jzu_q8&ab_channel=AstronautA , Last Accessed Time: 9 Jul 2024.

[9] How To Electronics. *TDS Meter using TDS & Temperature Sensor with Arduino || Measure Water Quality in PPM.* (9 Mar 2020). https://www.youtube.com/watch?v=ofZ7D8lVsXM&ab_channel=HowToElectronics, Last Accessed Time: 10 Jul 2024.

[10] SriTu Hobby. *WATER LEVEL SENSOR with Arduino UNO | How to use WATER LEVEL SENSOR ARDUINO [Code & Circuit Diagram].* (28 Oct 2020). https://www.youtube.com/watch?v=GnD-hRnWFLA&ab_channel=SriTuHobby, Last Accessed Time: 16 Jul 2024.

[11] Mucit Pilot. *Arduino ile ESP8266 Kullanımı ve İnternet Erişimi-1 (İnternet Üzerinden Veri Çekme).* (16 Sep 2020). https://www.youtube.com/watch?v=eAg4ZJ4b84Q&ab_channel=MucitPilot, Last Accessed Time: 19 Jul 2024.

[12] Robotistan. *Arduino ESP8266 Kullanımı - Thingspeak Nasıl Kullanılır?.* (23 Dec 2018). https://www.youtube.com/watch?v=6B6WFahLkMo&ab_channel=Robotistan, Last Accessed Time: 19 Jul 2024.

[13] Massimo Banzi and David Cuartielles (2005). *Arduino* (Version 1.8.19) [Computer Software]. arduino.cc/en/Main/Software, Last Accessed Time: 19 Jul 2024.

[14] MathWorks (2010). *ThingSpeak* (Version 3) [Computer Software]. thingspeak.com, Last Accessed Time: 19 Jul 2024.

[15] DFRobot. *pH Electrode Characteristics.* (No date specified). https://wiki.dfrobot.com/PH_meter_SKU__SEN0161_, Last Accessed Time: 24 Jul 2024.

APPENDICES

APPENDIX – 1: I2C ADDRESS FINDER

I2C_Address_Finder.ino

```
#include <Wire.h>
void setup() {
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}
void loop() {
  byte error, address;
  int Devices;
  Serial.println("Scanning...");
Devices = 0;
  for (address = 1; address < 127; address++ ) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
      if (error == 0) {
      Serial.print("I2C device found at address 0x");
        if (address<16)
          Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      Devices++;
      }
      else if (error==4) {
        Serial.print("Unknown error at address 0x");
        if (address<16)
          Serial.print("0");
        Serial.println(address,HEX);
      }
  }
  if (Devices == 0) Serial.println("No I2C devices found\n");
  else Serial.println("done\n");
  delay(5000);
}
```

APPENDIX – 2: I2C DISPLAYING TEXT ON LCD

I2C_LCD_Code.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// set the LCD address to 0x27 for a 16 cha
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  // initialize the lcd
  lcd.begin();
  // Turn on the LCD screen backlight
  lcd.backlight();
}

void loop()
{
  lcd.setCursor(1, 0);
  lcd.print("Hello World.!");
  delay(3000);
  lcd.clear();

  lcd.setCursor(2, 0);
  lcd.print("This is a LCD ");
  lcd.setCursor(2, 1);
  lcd.print("Screen Test");
  delay(3000);
  lcd.clear();
}
```

APPENDIX – 3: DISPLAYING TEMPERATURE ON LCD - PART 1

TempetureLCD.ino

```
1   // Include the libraries we need
2   #include <OneWire.h>
3   #include <DallasTemperature.h>
4
5   #include <Wire.h>
6   #include <LiquidCrystal_I2C.h>
7
8   LiquidCrystal_I2C lcd(0x27, 16, 2); //The LCD address and size. You
9
10  // Data wire is plugged into port 2 on the Arduino
11  #define ONE_WIRE_BUS 2 //pin for sensor
12
13  // Setup a oneWire instance to communicate with any OneWire devices
14  OneWire oneWire(ONE_WIRE_BUS);
15
16  // Pass our oneWire reference to Dallas Temperature.
17  DallasTemperature sensors(&oneWire);
18
19  /*
20      The setup function. We only start the sensors here
21  */
22  void setup(void)
23  {
24    // start serial port
25    Serial.begin(9600);
26    Serial.println("Dallas Temperature IC Control Library Demo");
27
28    // Start up the library
29    sensors.begin();
30    lcd.begin();
31    // Print a message to the LCD.
32    lcd.backlight();
33    lcd.setCursor(0, 0);
```

APPENDIX – 4: DISPLAYING TEMPERATURE ON LCD - PART 2

TempetureLCD.ino

```
33    lcd.print("DS18B20 TEST!");
34    lcd.setCursor(0, 1);
35    lcd.print("by miliohm.com");
36    delay(2000);
37    lcd.clear();
38  }
39  /*
40      Main function, get and show the temperature
41  */
42  void loop(void) {
43    // call sensors.requestTemperatures() to issue a global temperat
44    Serial.print("Requesting temperatures...");
45    sensors.requestTemperatures(); // Send the command to get temper
46    Serial.println("DONE");
47    // After we got the temperatures, we can print them here.
48    // We use the function ByIndex, and as an example get the temper
49    float tempC = sensors.getTempCByIndex(0);
50    if (tempC != DEVICE_DISCONNECTED_C) { // Check if reading was su
51      Serial.print("Temperature for the device 1 (index 0) is: ");
52      Serial.println(tempC);
53      lcd.setCursor(0, 0);
54      lcd.print("Temperature:");
55      lcd.setCursor(0, 1);
56      lcd.print(tempC);
57      lcd.print((char)223);
58      lcd.print("C");
59      lcd.print(" | ");
60      lcd.print(DallasTemperature::toFahrenheit(tempC));
61      lcd.print(" F");
62    } else {
63      Serial.println("Error: Could not read temperature data");
64    }
65  }
```

APPENDIX – 5: DISPLAYING HUMIDITY AND TEMPERATURE ON LCD – PART 1

HumidityLCD.ino

```
1   //* How to use the DHT-22 sensor with Arduino
2   //   Temperature and humidity sensor and
3   //   I2C LCD1602
4    //  SDA --> A4
5    //  SCL --> A5
6    //DHT 22 out-->D7
7
8   //Libraries
9   #include <DHT.h>;
10  //I2C LCD:
11  #include <LiquidCrystal_I2C.h>
12  #include <Wire.h>
13
14  LiquidCrystal_I2C lcd(0x27,16,2); // set the LCD address to 0x2
15
16  //Constants
17  #define DHTPIN 7     // what pin we're connected to
18  #define DHTTYPE DHT22   // DHT 22
19  DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor for normal
20
21  //Variables
22  //int chk;
23  int h;  //Stores humidity value
24  int t; //Stores temperature value
25
26  void setup() {
27      Serial.begin(9600);
28      Serial.println("Temperature and Humidity Sensor Test");
29      dht.begin();
30      lcd.begin(); //initialize the lcd
31      lcd.backlight(); //open the backlight
32  }
```

APPENDIX – 6: DISPLAYING HUMIDITY AND TEMPERATURE ON LCD – PART 2

HumidityLCD.ino

```
33
34  void loop() {
35      //Read data and store it to variables h (humidit
36      // Reading temperature or humidity takes about 2
37      h = dht.readHumidity();
38      t = dht.readTemperature();
39
40      //Print temp and humidity values to serial monit
41      Serial.print("Humidity: ");
42      Serial.print(h);
43      Serial.print(" %, Temp: ");
44      Serial.print(t);
45      Serial.println(" ° Celsius");
46  // set the cursor to (0,0):
47  // print from 0 to 9:
48      lcd.setCursor(0, 0);
49      lcd.println(" Now Temperature ");
50
51      lcd.setCursor(0, 1);
52      lcd.print("T:");
53      lcd.print(t);
54      lcd.print("C");
55
56      lcd.setCursor(6, 1);
57      lcd.println("2024 ");
58
59      lcd.setCursor(11, 1);
60      lcd.print("H:");
61      lcd.print(h);
62      lcd.print("%");
63
64    delay(1000); //Delay 1 sec.
65  }
```

APPENDIX – 7: WI-FI MODULE – PART 1

WifiModule.ino

```
1   ////////////////////////////////////////////
2   //        RemoteXY include library          //
3   ////////////////////////////////////////////
4
5   // you can enable debug logging to Serial at 115200
6   //#define REMOTEXY__DEBUGLOG
7
8   // RemoteXY select connection mode and include library
9   #define REMOTEXY_MODE__ESP8266_HARDSERIAL_POINT
10
11  // RemoteXY connection settings
12  #define REMOTEXY_SERIAL Serial
13  #define REMOTEXY_SERIAL_SPEED 115200
14  #define REMOTEXY_WIFI_SSID "RemoteXY"
15  #define REMOTEXY_WIFI_PASSWORD "123456789"
16  #define REMOTEXY_SERVER_PORT 6377
17
18  #include <RemoteXY.h>
19
20  // RemoteXY GUI configuration
21  #pragma pack(push, 1)
22  uint8_t RemoteXY_CONF[] =   // 45 bytes
23    { 255,2,0,0,0,38,0,17,0,0,0,31,1,106,200,1,1,2,0,2,
24    30,44,44,22,0,248,21,31,31,79,78,0,79,70,70,0,1,40,105,24,
25    24,0,6,31,0 };
26
27  // this structure defines all the variables and events of your
28  struct {
29      // input variables
30    uint8_t switch_01; // =1 if switch ON and =0 if OFF
31    uint8_t button_01; // =1 if button pressed, else =0
32
33      // other variable
```

APPENDIX – 8: WI-FI MODULE – PART 2

WifiModule.ino

```
    // input variables
  uint8_t switch_01; // =1 if switch ON and =0 if OFF
  uint8_t button_01; // =1 if button pressed, else =0

    // other variable
  uint8_t connect_flag;  // =1 if wire connected, else =0

} RemoteXY;
#pragma pack(pop)

//////////////////////////////////////////////
//           END RemoteXY include          //
//////////////////////////////////////////////

#define PIN_SWITCH_01 13
#define PIN_BUTTON_01 12

void setup() {
  RemoteXY_Init ();
  pinMode (PIN_SWITCH_01, OUTPUT);
  pinMode (PIN_BUTTON_01, OUTPUT);
  // TODO you setup code
}

void loop()
{
  RemoteXY_Handler ();
  digitalWrite(PIN_SWITCH_01, (RemoteXY.switch_01==0)?LOW:HIGH);
  digitalWrite(PIN_BUTTON_01, (RemoteXY.button_01==0)?LOW:HIGH);
  // TODO you loop code
  // use the RemoteXY structure for data transfer
  // do not call delay(), use instead RemoteXY_delay()
}
```

APPENDIX – 9: REAL TIME CLOCK MODULE – PART 1

realtimeclock.ino

```
#include <Wire.h>
#include <RTClib.h>
#include <LiquidCrystal_I2C.h>

RTC_DS3231 myRTC;
LiquidCrystal_I2C lcd(0x27, 16, 2); // Set the LCD I2C address to 0x27 for

void setup() {
  Serial.begin(57600);
  Wire.begin();
  delay(500);
  Serial.println("Nano Ready!");

  if (!myRTC.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (myRTC.lostPower()) {
    Serial.println("RTC lost power, let's set the time!");
    // Comment out below lines once the time is set for the first time
    myRTC.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  lcd.init();
  lcd.backlight();
}
void loop() {
  DateTime now = myRTC.now();

  lcd.setCursor(0, 0);
  lcd.print("Time:  ");
  lcd.print(now.hour(), DEC);
  lcd.print(':');
  lcd.print(now.minute(), DEC);
  lcd.print(':');
  lcd.print(now.second(), DEC);

```

APPENDIX – 10: REAL TIME CLOCK MODULE – PART 2

realtimeclock.ino

```
  Serial.println("Nano Ready!");

  if (!myRTC.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (myRTC.lostPower()) {
    Serial.println("RTC lost power, let's set the time!");
    // Comment out below lines once the time is set for the first
    myRTC.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  lcd.init();
  lcd.backlight();
}
void loop() {
  DateTime now = myRTC.now();

  lcd.setCursor(0, 0);
  lcd.print("Time:  ");
  lcd.print(now.hour(), DEC);
  lcd.print(':');
  lcd.print(now.minute(), DEC);
  lcd.print(':');
  lcd.print(now.second(), DEC);

  lcd.setCursor(0, 1);
  lcd.print("Date: ");
  lcd.print(now.year(), DEC);
  lcd.print('/');
  lcd.print(now.month(), DEC);
  lcd.print('/');
  lcd.print(now.day(), DEC);

  delay(1000);
}
```

APPENDIX – 11: RELAY SWITCH MODULE FOUR CHANNELS WITH LIGHTBULBS

RelayModuleLightbulb.ino

```
1   int ch = 4; // number of relaysint
2   relay[] = {2,3,4,5}; // pin number of relays
3   int wait = 2000;
4   int i = 0;
5
6   void setup() {
7     for (int i = 0; i < ch; i++) {
8     pinMode(relay[i], OUTPUT); // set relay pins as output
9     digitalWrite(relay[i], HIGH); // turn relay off
10    }
11  }
12
13  void loop() {
14    for (int i = 0; i < ch; i++) {
15      digitalWrite(relay[i], LOW); // turn relay ON
16      delay(wait);
17    }
18
19    for (int i = 0; i < ch; i++) {
20      digitalWrite(relay[i], HIGH); // turn relay OFF
21      delay(wait);
22    }
23  }
```

APPENDIX – 12: RELAY SWITCH MODULE ONE CHANNEL WITH A LIGHTBULB

RelayModuleOnlyOneLightbulb.ino

```
1   int relayPin = 7;
2   void setup() {
3     // put your setup code here, to run once:
4     pinMode(relayPin, OUTPUT);
5   }
6
7   void loop() {
8     // put your main code here, to run repeatedly:
9     digitalWrite(relayPin, HIGH);
10    delay(10000);
11    digitalWrite(relayPin, LOW);
12    delay(10000);
13  }
```

APPENDIX – 14: WATER PUMP WITH RELAY SWITCH MODULE

WaterPump.ino

```
byte pump = 4;

void setup() {
  Serial.begin(9600);
  while (!Serial);
  pinMode(pump, OUTPUT); // variant low/high
}

void loop() {
  digitalWrite(pump, HIGH); // pump deactivated
  delay(2000); // pushing period time
  digitalWrite(pump, LOW); // pump activated
  delay(10000); // pulling period time
}
```

APPENDIX – 15: TOTAL DISSOLVED SOLIDS MODULE ON LCD SCREEN

TDS_LCD.ino

```
#include <EEPROM.h>
#include "GravityTDS.h"
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
// set the LCD address to 0x27 for a 16 chars and 2 line displa

#define TdsSensorPin A1
GravityTDS gravityTds;

float temperature = 25, tdsValue = 0;

void setup() {
    Serial.begin(115200);
    lcd.begin(); // initialize the lcd
    lcd.backlight();  // Turn on the LCD screen backlight
    gravityTds.setPin(TdsSensorPin);
    gravityTds.setAref(5.0);  //reference voltage on ADC, defau
    gravityTds.setAdcRange(1024);  //1024 for 10bit ADC;4096 fo
    gravityTds.begin();  //initialization
}

void loop() {
    //temperature = readTemperature();  //add your temperature
    gravityTds.setTemperature(temperature);  // set the tempera
    gravityTds.update();  //sample and calculate
    tdsValue = gravityTds.getTdsValue();  // then get the value
    Serial.print(tdsValue,0);
    Serial.println("ppm");
    lcd.setCursor(0, 0);
    lcd.print("TDS Value:");
    lcd.setCursor(0, 1);
    lcd.print(tdsValue,0);
    lcd.print(" PPM");
    delay(1000);
    lcd.clear();
}
```

APPENDIX – 16: PH SENSOR ON LCD SCREEN

pH.ino

```
#include <LiquidCrystal_I2C.h> // LCD Library
LiquidCrystal_I2C lcd(0x27, 16, 2); // Set the LCD address to 0x27 for a

const int PH_PIN = A0; // Analog pin connected to the pH sensor

void setup() {
  Serial.begin(9600); // Initialize serial communication
  lcd.init(); // Initialize the lcd
  lcd.backlight();  // Turn on the LCD screen backlight
  lcd.setCursor(0, 0);
  lcd.clear();
}

void loop() {
  int phValue = analogRead(PH_PIN); // Read the value from the pH sensor
  Serial.print("pH Value: ");
  Serial.println(phValue); // Print pH value to the serial monitor
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("pH: ");
  lcd.print(pH_value);
  delay(1000); // Wait for 1 second
}
```

APPENDIX – 17: WATER LEVEL SENSOR ON LCD SCREEN

WaterLevelLCD.ino

```
#include <LiquidCrystal_I2C.h>

#define WATER_LEVEL_SENSOR_PIN A3

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(9600);
  lcd.begin();
  lcd.backlight();
  lcd.clear();
}

void loop() {
  int value = analogRead(WATER_LEVEL_SENSOR_PIN);
  lcd.setCursor(0, 0);
  lcd.print("Value: ");
  float waterLevel_mm = value * (500.0 / 1023.0); // milimeters
  lcd.print(waterLevel_mm);
  lcd.print(" mm   ");
  Serial.println(value);
  lcd.setCursor(0, 1);
  lcd.print("W. Level: ");


  if (value == 0) {
    lcd.print("EMPTY ");
  } else if (value > 1 && value < 350) {
    lcd.print("LOW   ");
  } else if (value > 350 && value < 510) {
    lcd.print("MEDIUM");
  } else if (value > 510){
    lcd.print("HIGH  ");
  }
}
```

APPENDIX – 18: WEB SERVER ON THINGSPEAK – PART 1

Robotistan

```
#include <SoftwareSerial.h>
#include <DHT.h>

String agAdi = "Virus";                    //Ağımızın adını
String agSifresi = "ifdbcgtj9.9";              //Ağımızın s

int rxPin = 10;
int txPin = 11;

String ip = "184.106.153.149";
float sicaklik, nem;

#define DHTPIN 7
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

SoftwareSerial esp(rxPin, txPin);

void setup() {

  Serial.begin(9600);  //Seri port ile haberleşmemizi ba
  Serial.println("Started");
  dht.begin();
  esp.begin(115200);
  esp.println("AT");
  Serial.println("AT Yollandı");
  while(!esp.find("OK")){
    esp.println("AT");
    Serial.println("ESP8266 Bulunamadı.");
  }
  Serial.println("OK Komutu Alındı");
  esp.println("AT+CWMODE=1");
```

APPENDIX – 19: WEB SERVER ON THINGSPEAK – PART 2

Robotistan

```
  esp.println("AT+CWJAP=\""+agAdi+"\",\""+agSifresi+"\"");     //Agımıza baglanıyo
  while(!esp.find("OK"));                                      //Ağa bağlanana kad
  Serial.println("Aga Baglandi.");
  delay(1000);
}
void loop() {
  esp.println("AT+CIPSTART=\"TCP\",\""+ip+"\",80");           //Thingspeak'e bağl
  if(esp.find("Error")){                                       //Bağlantı hatası k
    Serial.println("AT+CIPSTART Error");
  }

  sicaklik = (float)dht.readTemperature();
  nem = (float)dht.readHumidity();
  String veri = "GET https://api.thingspeak.com/update?api_key=ASGU1CWKJQJQHO75";
  veri += "&field1=";
  //Serial.println("Sıcaklık: " + sicaklik);
  veri += String(sicaklik);
  veri += "&field2=";
  //Serial.println("Nem: " + nem);
  veri += String(nem);                                         //Göndereceğimizne
  veri += "\r\n\r\n";
  esp.print("AT+CIPSEND=");                                    //ESP'ye göndereceğ
  esp.println(veri.length()+2);
  delay(2000);
  if(esp.find(">")){                                           //ESP8266 hazır old
    esp.print(veri);                                           //Veriyi gönderiyor
    Serial.println(veri);
    Serial.println("Veri gonderildi.");
    delay(1000);
  }
  Serial.println("Baglantı Kapatildi.");
  esp.println("AT+CIPCLOSE");                                //Bağlantıyı kapatıy
  delay(1000);                                                 //Yeni veri gönderim
}
```

APPENDIX – 19: AUTOMATED HYDROPONIC SYSTEM – PART 1

AutomatedHydroponicSystemThingspeak

```
// ******** DEFINE SOFTWARE SERIAL *********** //
#include <SoftwareSerial.h>                                    // The SoftwareSeria

String networkName = "Virus";                                  // The network name
String networkPassword = "ifdbcgtj9.9";                        // The network passw

int rxPin = 10;                                                // ESP8266 RX pin
int txPin = 11;                                                // ESP8266 TX pin

String ip = "184.106.153.149";                                 // Thingspeak IP add

// ********************** //

// ******** DEFINE LIBRARIES *********** //
#include <EEPROM.h>
#include "GravityTDS.h" // TDS Library
#include <LiquidCrystal_I2C.h> // LCD Library

#include <OneWire.h> // DS18B20 Temperature Library
#include <DallasTemperature.h> // DS18B20 Temperature Library
#include <Wire.h> // DS18B20 Temperature and DHT22 Library

#include <DHT.h> // DHT22 Humidity and Temperature Library

#include <RTClib.h> // Real time Clock Library

// ********************** //

LiquidCrystal_I2C lcd(0x27, 16, 2); // Set the LCD address to 0x27 for a 16 chars

// CONSTANTS
#define TdsSensorPin A1 // Define the pin for the TDS sensor
#define ONE_WIRE_BUS 2 // Define the data pin for the OneWire bus, used for DS18B2
#define DHTPIN 7 // Define the digital pin for the DHT22 sensor
```

APPENDIX – 20: AUTOMATED HYDROPONIC SYSTEM – PART 2

AutomatedHydroponicSystemThingspeak

```
#define DHTTYPE DHT22 // Define the type of DHT sensor being used (DHT22)
#define PH_PIN A2 // Define the analog pin for the pH sensor
#define WATER_LEVEL_SENSOR_PIN A3 // Define the analog pin for the water level sensor

// VARIABLES
float temperature = 25; // Variable to store temperature value, initialized to 25 degrees Celsius
float tds = 0; // Variable to store TDS value, initialized to 0
byte pump = 4; // Define the pin for the water pump
float h; // Variable to store humidity value from the DHT22 sensor
float t; // Variable to store temperature value from the DHT22 sensor
float thresholdWL = 305;
float wL = thresholdWL;

// OBJECTS
GravityTDS gravityTds; // Create an instance of the GravityTDS class for TDS sensor operations
OneWire oneWire(ONE_WIRE_BUS); // Create a OneWire instance to communicate with any OneWire device
DallasTemperature sensors(&oneWire); // Create a DallasTemperature instance for the DS18B20 sensor
DeviceAddress tempSensorAddress;
DHT dht(DHTPIN, DHTTYPE); // Create a DHT instance for the DHT22 sensor with specified pin and typ
RTC_DS3231 myRTC;// Create a RTC instance for the RTC module

SoftwareSerial esp(rxPin, txPin);                          // The serial communication pin sett

void setup() {
  // ******** SOFTWARE SERIAL *********** //
  Serial.begin(9600);                                      //Seri port ile haberleşmemizi başla
  Serial.println("Started");
  dht.begin();
  esp.begin(115200);                                       // Serial communication with the ESP
  esp.println("AT");                                       // The module is controlled with the
  Serial.println("AT command sent");
  while (!esp.find("OK")) {                                // Waiting is done until the module
    esp.println("AT");
    Serial.println("ESP8266 Not found.");
  }
```

APPENDIX – 21: AUTOMATED HYDROPONIC SYSTEM – PART 3

AutomatedHydroponicSystemThingspeak

```
Serial.println("OK command received");
esp.println("AT+CWMODE=1");                                    // The ESP8266 modul
while (!esp.find("OK")) {                                      // Waiting is done u
  esp.println("AT+CWMODE=1");
  Serial.println("Setting is being applied...");
}
Serial.println("Configured as a client");
Serial.println("Connecting to the network...");
esp.println("AT+CWJAP=\"" + networkName + "\",\"" + networkPassword + "\""); //
while (!esp.find("OK"));                                       // Waiting is done u
Serial.println("Connected to the network.");
delay(1000);
// ******** SOFTWARE SERIAL *********** //

// ***** REAL TIME CLOCK READING ***** //
Wire.begin();
delay(500);

// Start When Real Time Clock is found and ready
Serial.println("Nano Ready!");
if (!myRTC.begin()) {
  Serial.println("Couldn't find RTC");
  while (1);
}
if (myRTC.lostPower()) {
  Serial.println("RTC lost power, let's set the time!");
  // Comment out below lines once the time is set for the first time
  myRTC.adjust(DateTime (F(__DATE__), F(__TIME__)));
}
// ***** REAL TIME CLOCK READING ***** //

// ***** LCD SETUP ***** //
lcd.begin(16, 2); // Initialize the lcd
lcd.backlight();  // Turn on the LCD screen backlight
lcd.clear();
```

APPENDIX – 22: AUTOMATED HYDROPONIC SYSTEM – PART 4

AutomatedHydroponicSystemThingspeak

```
  // ***** LCD SETUP ***** //

  // ***** TDS SENSOR READING ***** //
  gravityTds.setPin(TdsSensorPin);
  gravityTds.setAref(5.0);  // Reference voltage on ADC, default 5.0V on Arduino UNO
  gravityTds.setAdcRange(1024);  // 1024 for 10bit ADC;4096 for 12bit ADC
  gravityTds.begin();  // Initialization
  // ***** TDS SENSOR READING ***** //

  // ***** RELAY SWITCH AND WATER PUMP ACTIVATOR ***** //
  while (!Serial);
  pinMode(pump, OUTPUT); // variant low/high
  // ***** RELAY SWITCH AND WATER PUMP ACTIVATOR ***** //

  // ***** DS18B20 TEMPERATURE SENSOR READING ***** //
  sensors.begin(); // Start up the library
  // ***** DS18B20 TEMPERATURE SENSOR READING ***** //

  // ***** DHT HUMIDITY AND TEMPERATURE SENSOR READING ***** //
  Serial.begin(9600);
  dht.begin();
  // ***** DHT HUMIDITY AND TEMPERATURE SENSOR READING ***** //
}

void loop() {
  // ******** SOFTWARE SERIAL ********** //
  esp.println("AT+CIPSTART=\"TCP\",\"" + ip + "\",80");       // Connection to Thingspeak is being
  if (esp.find("Error")) {                                    // Connection error is checked
    Serial.println("AT+CIPSTART Error");
  }
  // ******** SOFTWARE SERIAL ********** //

  // ***** REAL TIME CLOCK READING ***** //
  DateTime now = myRTC.now();
```

APPENDIX – 23: AUTOMATED HYDROPONIC SYSTEM – PART 5

AutomatedHydroponicSystemThingspeak

```
// ***** REAL TIME CLOCK READING ***** //

// ***** PH SENSOR READING ***** //
int voltage = analogRead(PH_PIN);
// Calculate the pH value using the regression equation
double pH = (-0.0145 * voltage) + 7.38;
// ***** PH SENSOR READING ***** //

// ***** TDS SENSOR READING ***** //
//temperature = readTemperature();  // Add your temperature sensor and read it
//gravityTds.setTemperature(temperature);  // Set the temperature and execute te
gravityTds.update();  // Sample and calculate
tds = gravityTds.getTdsValue();  // Then get the value
// ***** TDS SENSOR READING ***** //

// **** WATER LEVEL SENSOR READING ***** //
int value = analogRead(WATER_LEVEL_SENSOR_PIN);
wL = value * (500.0 / 1023.0); // milimeters
// **** WATER LEVEL SENSOR READING ***** //

digitalWrite(pump, HIGH); // Pump deactivated

// ***** RELAY SWITCH AND WATER PUMP ACTIVATOR ***** //
while (wL < thresholdWL) {
  if (wL < thresholdWL) {
    digitalWrite(pump, LOW); // Pump activated

    // **** WATER LEVEL SENSOR READING ***** //
    int value = analogRead(WATER_LEVEL_SENSOR_PIN);
    wL = value * (500.0 / 1023.0); // milimeters
    // **** WATER LEVEL SENSOR READING ***** //

  } else {
    digitalWrite(pump, HIGH); // Pump deactivated
  }
```

APPENDIX – 24: AUTOMATED HYDROPONIC SYSTEM – PART 6

AutomatedHydroponicSystemThingspeak

```
}
if (wL > thresholdWL){
  digitalWrite(pump, HIGH); // Pump deactivated
}
// ***** RELAY SWITCH AND WATER PUMP ACTIVATOR ***** //

// **** DS18B20 TEMPERATURE SENSOR READING ***** //
// Call sensors.requestTemperatures() to issue a global temperature
sensors.requestTemperatures(); // Send the command to get temperatures
float tempC = sensors.getTempCByIndex(0); // Get the temperature from the first sensor
// **** DS18B20 TEMPERATURE SENSOR READING ***** //

// **** DHT HUMIDITY AND TEMPERATURE SENSOR READING ***** //
// Read data and store it to variables h (humidity) and t (temperature)
// Reading temperature or humidity takes about 2
h = (float) dht.readHumidity();
t = (float) dht.readTemperature();
// *** DHT HUMIDITY AND TEMPERATURE SENSOR READING ***** //

// Display values on the LCD alternately one by one
for (int i = 0; i < 4; i++) {
  lcd.clear();
  if (i == 0) {
    // Print the time and date values
    lcd.setCursor(0, 0);
    lcd.print("Time: ");           //Serial.print("Time: ");
    lcd.print(now.hour(), DEC);    //Serial.print(now.hour(), DEC);
    lcd.print(':');                //Serial.print(':');
    lcd.print(now.minute(), DEC);  //Serial.print(now.minute(), DEC);
    lcd.print(':');                //Serial.print(':');
    lcd.print(now.second(), DEC);  //Serial.println(now.second(),DEC);
    lcd.setCursor(0, 1);
    lcd.print("Date: ");           //Serial.print("Date: ");
    lcd.print(now.year(), DEC);    //Serial.print(now.year(), DEC);
    lcd.print('/');                //Serial.print('/');
```

APPENDIX – 25: AUTOMATED HYDROPONIC SYSTEM – PART 7

AutomatedHydroponicSystemThingspeak

```
    lcd.print(now.month(), DEC);    //Serial.print(now.month(), DEC);
    lcd.print('/');                 //Serial.print('/');
    lcd.print(now.day(), DEC);      //Serial.println(now.day(), DEC);
    delay(5000);
  }
  else if (i == 1) {
    // Print the TDS value
    lcd.setCursor(0, 0);
    lcd.print("TDS: ");             //Serial.print("TDS: ");
    lcd.print(tds);                 //Serial.print(tds);
    lcd.print(" ppm       ");      //Serial.println(" ppm");

    // Print the pH values
    lcd.setCursor(0, 1);
    lcd.print("pH: ");              //Serial.print("pH Value: ");
    lcd.print(pH);                  //Serial.println(pH);
    delay(5000); // Wait for 2 seconds
  }
  else if (i == 2) {
    if (tempC != DEVICE_DISCONNECTED_C) // Check if reading was successful
    {
      // Print the temperature values
      lcd.setCursor(0, 0);
      lcd.print("Water T: ");       //Serial.print("Water Temperature: ");
      //lcd.setCursor(9, 0);
      lcd.print(tempC);             //Serial.print(tempC);
      lcd.print((char)223);         //Serial.print((char)223);
      lcd.print("C");               //Serial.print("C");
      //lcd.print(" | ");             //Serial.print(" | ");
      //lcd.print(DallasTemperature::toFahrenheit(tempC)); //Serial.print(DallasTempe
      //lcd.print("F");               //Serial.println("F");

      lcd.setCursor(0, 1);
      lcd.print("H:");               //Serial.print(" | Hum:");
      lcd.print(h);                  //Serial.print(h);
```

APPENDIX – 26: AUTOMATED HYDROPONIC SYSTEM – PART 8

AutomatedHydroponicSystemThingspeak

```
        lcd.print("%");                //Serial.println("%");

        lcd.setCursor(9, 1);
        lcd.print("T:");               //Serial.print("T:");
        lcd.print(t);                  //Serial.print(t);
        lcd.print((char)223);          //Serial.print((char)223);
        lcd.print("C");                //Serial.println("C");
        delay(5000);
      }
    }
    else if (i == 3) {
      // Print the water level value
      lcd.setCursor(0, 0);
      lcd.print("Value: ");          //Serial.print("Water Level Value: ");
      lcd.print(wL);                 //Serial.println(wL);
      lcd.print(" mm    ");          //Serial.print(" mm");

      lcd.setCursor(0, 1);
      lcd.print("W. Level: ");       //Serial.print("W. Level: ");

      if (value == 0) {
        lcd.print("EMPTY ");         //Serial.println("EMPTY ");
      } else if (value > 1 && value < 350) {
        lcd.print("LOW   ");         //Serial.println("LOW   ");
      } else if (value > 350 && value < 510) {
        lcd.print("MEDIUM");         //Serial.println("MEDIUM");
      } else if (value > 510) {
        lcd.print("HIGH  ");         //Serial.println("HIGH  ");
      }
    }
}
// ******** SOFTWARE SERIAL *********** //
String data = "GET https://api.thingspeak.com/update?api_key=ASGU1CWKJQJQHO75";   /
//The temperature variable we will send
data += "&field1=";
```

APPENDIX – 27: AUTOMATED HYDROPONIC SYSTEM – PART 9

AutomatedHydroponicSystemThingspeak

```
        lcd.print("HIGH  ");        //Serial.println("HIGH  ");
      }
    }
  }
  // ******** SOFTWARE SERIAL *********** //
  String data = "GET https://api.thingspeak.com/update?api_key=ASGU1CWKJQJQHO75";
  //The temperature variable we will send
  data += "&field1=";
  data += String(h);                                    // The temperature vari
  data += "&field2=";
  data += String(t);                                    // The humidity variabl
  data += "&field3=";
  data += String(tempC);                                // The water temperatur
  data += "&field4=";
  data += String(tds);                                  // The TDS variable to
  data += "&field5=";
  data += String(pH);                                   // The pH level variabl
  data += "&field6=";
  data += String(wL);                                   // The water level vari
  data += "\r\n\r\n";
  esp.print("AT+CIPSEND=");                             // The length of the da
  esp.println(data.length() + 2);
  delay(2000);
  if (esp.find(">")) {                                  // Commands are execute
    esp.print(data);                                    // The data is being se
    Serial.println(data);
    Serial.println("Data sent");
    delay(1000);
  }
  Serial.println("Connection closed");
  esp.println("AT+CIPCLOSE");                          // The connection is bei
  delay(1000);                                          // A 1-minute wait is ob
  // ******** SOFTWARE SERIAL *********** //

}
```