Chapter 1
Introduction

1.1 Introduction

As the world's primary means of providing food for its ever-increasing population, agriculture is one of the most crucial parts of the global economy. As the world's population grows, so does its demand for food. This, in turn, creates a need for a faster and more efficient food production. There are several agricultural practices that have been developed to meet this need. One of those practices is hydroponics, which present quite a few benefits over traditional agriculture. Hydroponics offers faster growth rates, higher overall yields, and reduced water usage (Janes, 2019). However, the use of hydroponics introduces an increase in difficulty. Various environmental factors need to be carefully monitored to ensure optimal growth, such as temperature, humidity, CO_2 levels, pH levels, and nutrient PPM (Janes, 2019). Fortunately, internet of things (IoT) technology has advanced considerably, which has made it easier and cheaper to implement than ever. Utilizing a sensor array utilizing IoT technology will allow farmers to monitor the environmental conditions in real-time. Not only that, but advancements in cloud computing technology have also similarly made it easier and cheaper to implement than ever. Combining the reach provided by IoT and the flexibility provided by cloud can lead to the creation of a truly powerful solution. Thus, I propose the creation of an IoT-enabled hydroponic farm monitoring system using Arduino and cloud.

1.2 Background

Hydroponics technology is not new, the earliest recorded example of a hydroponic farm is the Hanging Gardens of Babylon. Unfortunately, the technology of the past rendered hydroponics a non-viable agricultural practice when compared to traditional agriculture. Due to advancements in technology and the need for more sustainable agricultural practices, hydroponics has seen a rise in popularity and viability (Janes, 2019).

Due to the sensitivity of hydroponics to changes in the environment, hydroponics requires careful monitoring to ensure optimal plant growth. When combined with the fact that the relevant environmental factors usually fluctuate relatively frequently, manual methods for monitoring and inspection become time-consuming and can produce inconsistencies (Prasanniya, 2023).

The emergency of IoT technology as potent tools for monitoring hydroponics in real-time allows farmers to adjust conditions as needed. One of the most popular platforms for IoT due to its versatility, low cost, and ease of use is Arduino. Cloud-based technologies like Google Cloud Platform (GCP) and Amazon Web Services (AWS) can serve as a store for data and a platform on which a support system can be deployed on, allowing unparalleled ease of access to the system, which can provide farmers with invaluable insight into the conditions of their hydroponics.

In conclusion, although difficult, advancements in technology have made hydroponics a viable and desirable alternative to traditional agriculture. Two of those technologies are IoT and cloud computing. By combining both technologies to create a hydroponic farm monitoring system, farmers can obtain invaluable information in a consistent, reliable, and timely manner. This availability can allow farmers to make quick adjustments to ensure optimal plant growth that would have otherwise been impossible to do with a meaningful degree of consistency.

1.3 Object of Study

Figure 1.1 – Just Hydroponics 1

The object of study in this case study is a hydroponic farm company called *Just Hydroponics*. It is a small to medium-sized, family-owned company located in the outskirts of Bogor City. They currently have around a dozen lots and have plans for further expansion.

Figure 1.2 – Just Hydroponics 2

1.4 Case Study Objectives

The aim of this case study is to create a systemized hydroponic farm environment monitoring solution. This solution has two main parts. The first half is an Arduino-based sensor device which collects information and sends it to the web server. The second half is a web server, which contains the database server and the web application which serves as the data storage and visualization tools.

The primary purpose of the solution is to serve as a proof of concept and technical demonstrator. It is meant to show that the proposed solution is technologically feasible. Other studies such as commercial viability and user experience is outside of the scope of this case study. This is due to constraints in both time, resources, and skill. I am the sole person working on this case study and lack the necessary expertise to conduct a meaningful study in the excluded fields.

This case study should also be able to serve as solid ground and as a steppingstone for anyone who wishes to perform further studies to perfect the solution and make it commercially viable.

On a more measurable note, the designed solution presented by this case study is hoped to have the following benefits:

1. Improved operational efficiency and effectiveness of the hydroponics environment data collection in hydroponic farms.
2. Reduce operating costs in hydroponic farms.
3. Introduce a degree of simple systemization in hydroponic farms currently operating without.
4. Vastly increased data security and integrity by storing them digitally in the cloud.
5. Increase consistency through a systemization and automation of processes that are currently done manually, subject to the interpretation of each individual.
6. Increase data availability by centralizing all data into a server, easily accessible from anywhere (with internet access).
7. Faster reaction times due to more frequent automated measurement.

1.5 Research Questions

1. How does an Arduino-based and cloud-powered hydroponic monitoring system solution perform compared to manual hand-measuring in tracking changes in a hydroponic plantation's environmental factors?
2. What are the components that would make up the proposed solution and what role would each component serve?
3. What would be required to build the components??
4. What would be the differences between the currently used manual method and the proposed solution? What are the advantages and drawbacks of each method?
5. In the state immediately following the conclusion of this case study, would the solution be ready for commercialization? Why?

1.6 Hypothesis

The hypothesis written here will directly answer the research questions with what I think *should* be the answer prior to performing the case study.

1. It depends. There will be a "crossover point" in a chart of complexity and cost effectiveness, where at a certain scale, an automated monitoring system becomes more cost effective compared to manual measurement. For a smaller hydroponic farms, this solution would be too expensive and overengineered. For a medium to large sized farm, this solution would be effective and cost efficient. Compared to manual hand-measuring, the efficiency is expected to rise exponentially the larger the farm is.

2. The proposed solution is divided into two equally important halves. The first half is the Arduino sensor device. The second half is the web server.
    a. Within the first half is the Arduino, connected to which will be 5 sensor modules collecting 6 different environmental data, and a Wi-Fi module, which serves as the data transmitter.
        i. The first component is the DS18B20 probe, which measures water temperature.
        ii. The second component is the DHT11 sensor, which measures both air temperature and air humidity.

iii. The third component is the MH-Z19B sensor, which measures CO_2 levels.
iv. The fourth component is the PH-4502C and its probe, which measures pH levels.
v. The fifth component is the DFRobot GTDS sensor, which measures TDS (Total Dissolved Solids) levels, which are presumably the nutrients inside of the hydroponic waters.
vi. The sixth and final component is the ESP-01 module, which is a Wi-Fi module. It will connect to the local internet access point and transmit data over the internet into the API.

b. Within the second half is the web server. The web server is further divided into three components.
i. The first component is the database server.
ii. The second component is the web application.
iii. The third component is the API to receive data from the Arduino.

3. Again, the assembly and building process of the solution will be largely divided into two: the Arduino sensor and the web server.
a. For the Arduino sensor:
i. Obviously, an Arduino mainboard is required, which should be an Arduino Uno.
ii. The sensor and Wi-Fi modules are also required.
iii. Jumper cables of all types (male-to-male, male-to-female, and female-to-female) are also required.
iv. Finally, a 12V DC power supply will allow the Arduino to operate independently.
b. For the web server:
i. Development of the web application and API will require the use of PHP. The application should be relatively simple and require no additional frameworks other than stock PHP 8.
ii. The database will be relational and created using MySQL.

4. I expect there to be plenty of differences, classified by advantages and drawbacks.
    a. For the advantages:
        i. Should the hydroponic farm wish to expand its business, this is without a doubt the correct way forward. As the business grows larger, the operational complexity will increase exponentially. The business operations will eventually grow to a point where it is impossible to manage without some form of assistance from a systemized farm management system.
        ii. There should be a noticeable decrease in the amount of crop write-offs and losses due to environmental anomalies.
        iii. Data will be available in near real-time.
        iv. Data is more consistent and not prone to human error.
        v. Data can be accessed from (almost) anywhere, so long as the location has internet access.
        vi. Enormous data capacity compared to manual pen and paper methods. Can be used to analyze trends.
        vii. Vastly reduced labor in the long term.

    b. As for the drawbacks:
        i. The solution is incapable of taking any actions, it can only monitor and report.
        ii. Implementing a whole system will cost a lot.
        iii. Staff members will likely need to be retrained to use the new system.
        iv. It is likely that the first few iterations of the prototype will be unreliable and require a lot of babysitting.
        v. A large short-term jump in complexity.
        vi. A systemized solution would be dependent on a stable power source.
        vii. The system is electronic, and requires maintenance, likely tedious to maintain in the short term whilst the scale is still small.

viii. Requires at least one staff member with expertise in the system's use.

5. No. To be viable for commercialization, the product must be relatively cheap, reliable, and easy to use. In its form immediately after the conclusion of this case study, it would be too expensive, unreliable, and difficult to use even for trained individuals. No doubt the solution has potential that could be developed further, but I have neither the time, skills, nor resources to continue this research.

1.7 Scope & Limitations

The research and development of this project is defined and limited to the following scope:

1. The creation of an IoT-enabled hydroponic environment monitoring device powered by Arduino.
    a. Due to the vastly increased complexity otherwise and the limited time and resources allocated to the project, the capability of the device will be strictly limited to monitoring environmental variables. Its function is strictly to provide information, it will not be able to perform any actions that can directly influence the hydroponic farm.
    b. The environmental variables stated in point (a) will be limited to six:
        i. Water Temperature
        ii. Air Temperature
        iii. Humidity
        iv. CO_2 Levels
        v. pH Levels
        vi. Nutrient PPM
    c. The device should be able to stand alone and perform its functions without a direct physical connection to a computer.

2. The creation of a web-based application to act as a terminal for the user to access the system and as a receiver for data from the Arduino.
    a. In its role as a terminal for users, the web application will have a user interface.
        i. The user interface will be able to display data collected from the Arduino.
        ii. The user interface will have a feature that allows the user to select a specific sensor to display data from.
        iii. The user interface will also have a feature that allows the user to filter data based on a range of time.
    b. In its role as a receiver for data from the Arduino, the web application will include a URL-based API for the Arduino to access.
        i. The data will be passed on via PHP's $_GET super global variable passed by URL.
        ii. The data will then be processed by the API code and be automatically inserted into the database.
    c. As a rudimentary website that features neither valuable data nor features that can be a security risk to the hydroponic farm, the security feature of the website will be limited to a simple password. Individual user accounts are deemed to be unnecessary, as the website will not feature differing access levels/tiers.
    d. Due to the simple nature of the web application, for ease of development due to the limited time and resources, it will be developed using the base PHP 8.

1.8 Case Study Outline

1.8.1 Chapter 1 – Introduction

This chapter delivers the introduction and background of the topic and solution to this case study. It also contains other information that provides an overview of this case study, including the object of study itself, the case study objectives statement, research questions, hypothesis, scope & limitations, and this case study outline.

1.8.2 Chapter 2 – Theoretical Foundation

This chapter will cover the explain theoretical concepts, principles, and literature relevant to the case study. It includes a variety of documentations and sources both academic and non-academic that serve as the conceptual basis of this case study, both for the web application and the IoT sensor. As the web application utilizes PHP for the primary application, MySQL as the database, and Apache HTTP Server as the web server, this chapter contains explanations, documentations, and sources of various comprehensiveness for all of them. As the IoT sensor utilizes Arduino and several Arduino-compatible sensors and modules, the explanations, documentations, and sources for these are likewise included.

A short literature review of existing “similar cases” to this case study will also be included near the end of this chapter.

1.8.3 Chapter 3 – Problem Analysis

This chapter covers the analysis of the problems and challenges faced byJust Hydroponics’ owners in their day-to-day operations. It discusses the limitations currently faced by the hydroponic owners and discusses in short, the possible directions that the solution can take.

By understanding the limitations of the existing system, we can ensure that the new Arduino-based automated solution is able to address the issues faced by the farm with their current processes.

1.8.4 Chapter 4 –Solution Design

This chapter describes how the devised solution can address the problems identified in the previous chapter. It describes the design of the proposed solution, including the required hardware components, software, and the integrated system architecture. Compared to the next chapter, this chapter covers the solution’s design in a high-level perspective. It envisions what the solution should look like and how it should behave. This provides a rough outline of how the solution will be designed and what it aims to accomplish.

1.8.5 Chapter 5 – Solution Implementation

This chapter describes the process of implementation and testing of the system. Compared to the previous chapter, this chapter provides a low-level view of the solution's design and how it should be implemented. It describes in detail how each component would interact with each other, and what would be required to make them work.

1.8.6 Chapter 6 – Evaluation & Discussion

This chapter provides an analysis of the performance of the system, including any room for improvement and its current limitations. It discusses challenges that I encountered during implementation and testing, the solutions I adopted to solve those challenges, and the results of my implementation and testing.

1.8.7 Chapter 7 – Conclusion & Recommendation

The final chapter summarizes the key findings of this case study, restates the research questions, and answers them based on the findings of the case study. Any recommendations for future research in this field and the practical applications of the system are also included inside of this chapter.

Chapter 2

Theoretical Foundation

2.1 Theoretical Foundation of the Hydroponic Sensor

2.1.1 Arduino Uno

Arduino is an open-source hardware and software company based in Italy that designs and manufactures easy-to-use electronics (*About Arduino*, 2021). Arduino's open-source nature allows it to be easily modified and reproduced to suit the needs of each individual project that involves Arduino. Arduino's easy-to-use nature makes it a viable platform to serve as the basis of this project's hydroponic sensor.

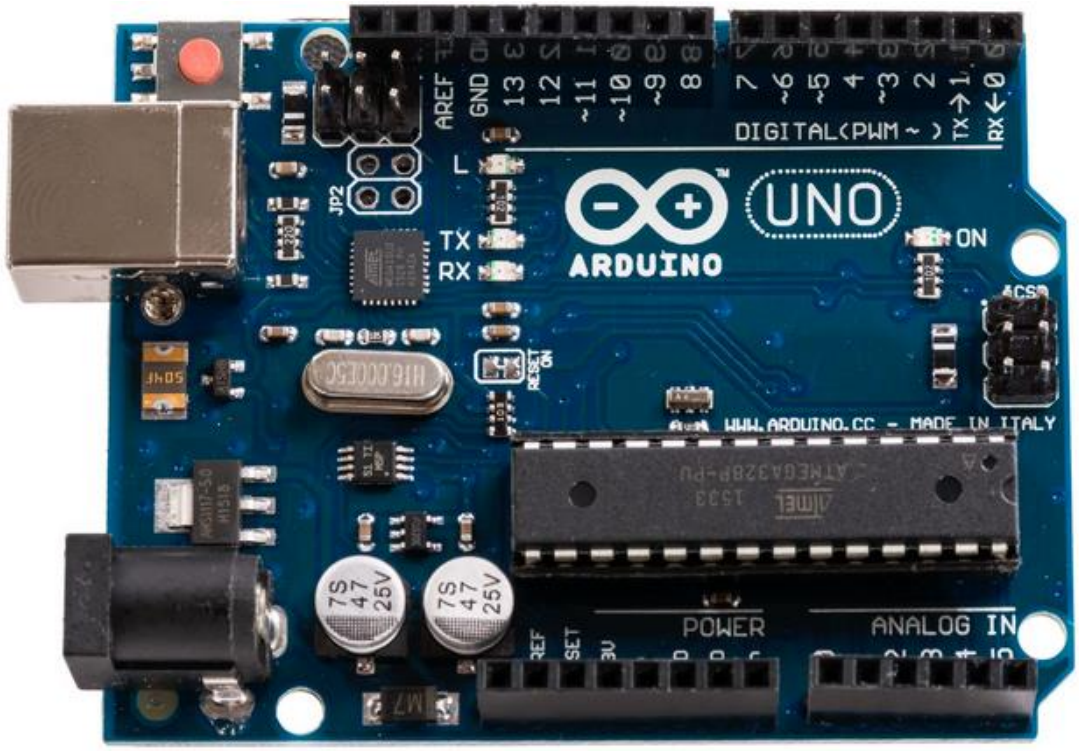


Figure 2.1 – Arduino Uno R3 16U2

2.1.1.1 Specifications

There are many types of Arduino boards available in the commercial market, each with their own microcontroller. The Arduino board chosen for this project is the Arduino Uno R3 16U2, and according to freeCodeCamp.org (2021), it has the following specifications:

1. Microcontroller: ATMEGA328P

2. Operating Voltage: 5V
3. Input Voltage (Recommended): 7-12V.
4. Input Voltage (Limits): 6-20V
5. Digital I/O Pins: 14 (6 PWM)
6. Analog Input Pins: 6
7. DC Current On I/O Pins: 40mA
8. DC Current On 3.3V Pin: 50mA
9. Flash Memory: 32KB (0.5KB Occupied by Bootloader)
10. SRAM: 2KB (ATMEGA328P)
11. EEPROM: 1KB (ATMEGA328P)
12. Clock Speed: 16 MHz

2.1.1.2 Components

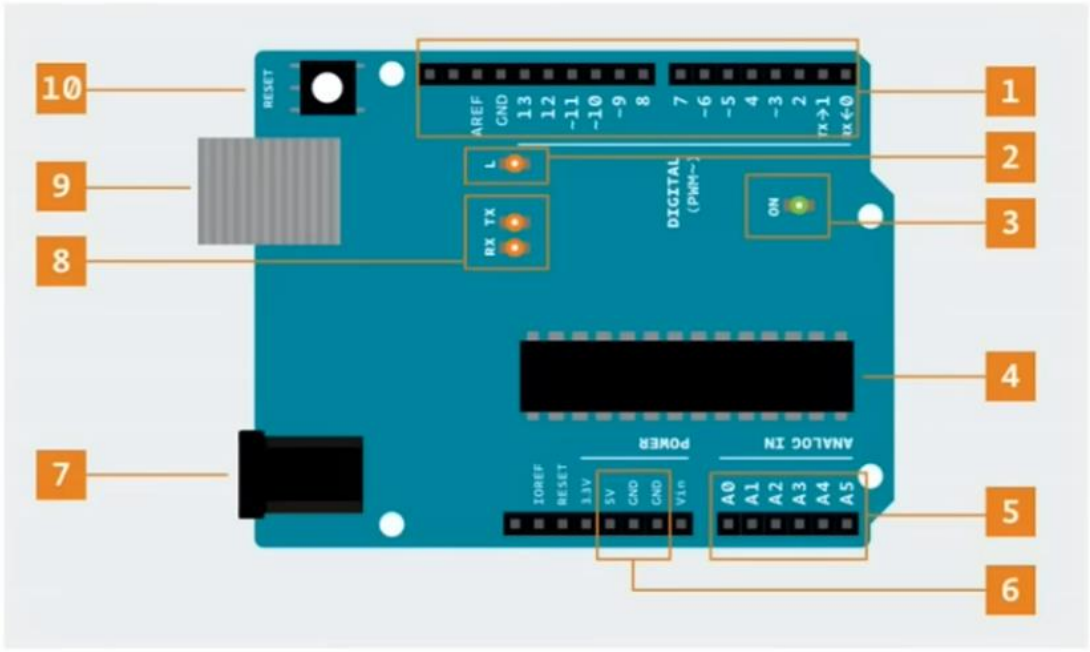


Figure 2.2 – Arduino Schematic *(freeCodeCamp.org, 2021)*

As described by freeCodeCamp.org (2021), the Arduino Uno board can be divided into these 10 typical components or component groups:

1. The first group contains the digital pins.
    a. They are numbered 0 to 13.
    b. These digital pins are usually operated using the *digitalRead()* and *digitalWrite()* functions.

    c. Some of the pins have a tilde (~) symbol, which indicate that they support PWM.
    d. The PWM-capable pins can use the *analogWrite()* function.
2. The second group contains a built-in LED.
    a. By default, this LED is connected to pin 13.
    b. When the pin is set to *HIGH*, the LED will turn on.
    c. When the pin is set to *LOW*, the LED will turn off.
3. The third group contains the Power LED. This LED indicates whether the Arduino board is powered (LED ON) or not (LED OFF).
4. The fourth group contains the ATMEGA328P microcontroller, which controls everything that happens on the Arduino board.
5. The fifth group contains the Analog Input pins.
    a. They are numbered from A0 to A5.
    b. Due to them being input pins, they are used with the *analogRead()* function.
6. The sixth group contains the built-in Power Pins.
    a. These pins on the Arduino board supports 3.3V, 5V, and grounding.
    b. These pins should not be used for high-current modules.
7. The seventh group contains the Power Connector.
    a. This connector allows the Arduino to be powered when not connected to a computer.
    b. The connector accepts 7V-12V DC.
8. The eighth group contains the RX and TX LEDs.
    a. These LEDs indicate when there is communication between the Arduino board and another device.
    b. The RX LED blinks when the Arduino board is receiving data.
    c. The TX LED blinks when the Arduino board is transmitting data.
9. The ninth group contains the USB jack.
10. The tenth group contains the Reset Button. This button resets the ATMEGA328P microcontroller but does not clear the Arduino's memory (the code that is uploaded to it).

2.1.1.3 Interfacing With the Arduino Board

The Arduino board is most typically controlled by using the Arduino IDE, which can be downloaded from Arduino's official website. At the time of this case study's writing, the Arduino IDE's latest release is version 2.0.4. The Arduino IDE uses the C++ language, though the support library used is a subset (limited version) of the standard C library due to Arduino's very small RAM capacity (Shinde, 2023).

2.1.2 Wi-Fi Module

To allow the Arduino to communicate via Wi-Fi, I used an ESP-01 Wi-Fi module, which uses the ESP-8266 chip. According to Barela (2023), the ESP-01 and the ESP-8266 family are inexpensive and powerful programmable Wi-Fi breakout boards. The cost for these modules is magnitudes lower compared to previous solutions such as the Wi-Fi Shield and Arduino Yun.

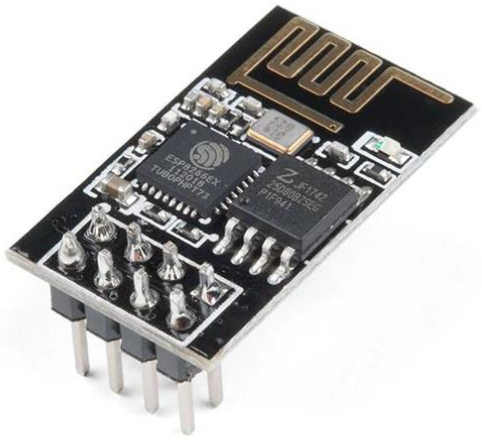


Figure 2.3 – The ESP-01 Module

2.1.2.1 Specifications

The ESP8266 chip uses 3.3V and draws over 300mA at peak draw. Using 5V on the ESP-01 module will likely irreparably damage it. The ESP8266 chip itself contains 64 KiB of instruction RAM, 96 KiB of data RAM, and 4 MiB of QIO FLASH (Ada, 2015). Unfortunately, the 3V3 port on the Arduino Uno board, which

supplies 3.3V power can only provide up to 50mA of current, which is only enough for the ESP-01 to perform operations with small amounts of data. In order to make use of the ESP-01's full potential with an Arduino Uno board, an external 3.3V power supply will be required.

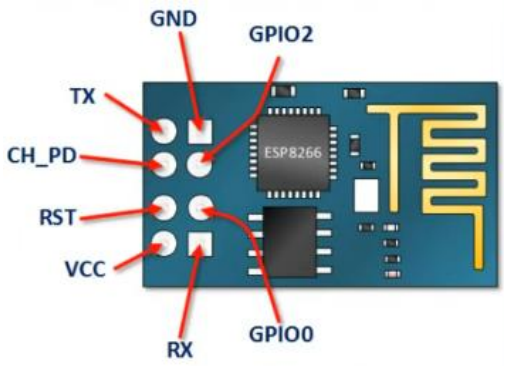


Figure 2.4 – A Schematic of the ESP-01 Module *(Electronoobs, 2019)*

The ESP-01 module has 8 pins. The figure above indicates the purpose of each of the 8 individual pins on the module. According to "ESP8266 Pinout Reference" (2019), these pins have the following functions:

1. The VCC pin is the power supply pin, which accepts 3.3V.
2. The GND pin is the ground pin.
3. GPIO0 is a general-purpose IO pin. Also used for serial programming mode.
4. GPIO2 is another general-purpose IO pin.
5. TX is the serial TX pin. It's also the GPIO1 pin.
6. RX is the serial RX pin. It's also the GPIO3 pin.
7. CH_PD is the "Active High Chip Enable" pin.
8. RST is the reset pin.

2.1.2.2 AT Commands

By default, the ESP-01's firmware is set to accept AT commands, although its capable of being *flashed* with new firmware (Bhimsen, 2022). The AT command set available to the ESP-01 is limited, but there is still quite a few of them. I will only be listing the ones relevant to this case study.

1. Get firmware version.
    a. Syntax: `AT+GMR`

   b. Expected response: Prints out the version, ended with “OK” on last line.

2. Reset module.
   a. Syntax: `AT+RST`
   b. Expected response: “OK”.
3. Set Wi-Fi mode.
   a. Syntax: `AT+CWMODE=<mode>`
   b. Parameter **mode**:
      i. **1** for station mode.
      ii. **2** for access point mode.
      iii. **3** for the “both” mode.
   c. Expected response: “OK”.
4. Connect to a Wi-Fi access point.
   a. Syntax: `AT+CWJAP=”<ssid>”,”<pass>”`
   b. Expected response: “OK”.
5. Set connection mode.
   a. Syntax: `AT+CIPMUX=<mode>`
   b. Parameter **mode**:
      i. **0** for single connection mode.
      ii. **1** for multiple connection mode.
   c. Expected response: “OK”.
6. Connect to a web server / host.
   a. Syntax: `AT+CIPSTART=0,”TCP”,”<host>”,<port>`
   b. Parameter **host**: the target web server’s URL.
   c. Parameter **port**: the target web server’s URL port.
   d. Expected response: “OK”.
7. Send packet size information.
   a. Syntax: `AT+CIPSEND=0,<length>`
   b. Parameter **length**: the length of the payload String, plus two for the carriage return and newline characters.
   c. Expected response: The character “>”, which indicates readiness to receive the payload String.
8. Close and conclude payload String, should the packet size information exceeds the actual payload size.

a. Syntax: `AT+CIPCLOSE=0`

b. Expected response: “OK”.

It is important to note that each of these commands be ended with a carriage return character (`/r`) and a newline character (`/n`) to indicate the end of the command line. Fortunately, Arduino’s `Serial::println()` function already ends with these characters. Should for whatever reason the carriage return and newline characters need to be avoided, the `Serial::print()` function does not end with them.

2.1.2.3 Operating Modes

According to Bhimsen (2022), there are at least two ways of communicating with the ESP-01. The first is “USB bypass mode”, where the Arduino Uno acts a pass-through TTL to USB adapter and the ESP-01 module communicates directly with a laptop/desktop computer. The second is through Arduino Uno’s SoftwareSerial, where the ESP-01 module communicates with the Arduino Uno’s microcontroller instead.

As the name suggests, **the first way** (USB bypass method) of communicating with the ESP-01 module involves the use of the Arduino as an “adapter” to pass data from the ESP-01’s RX & TX pin to the computer.

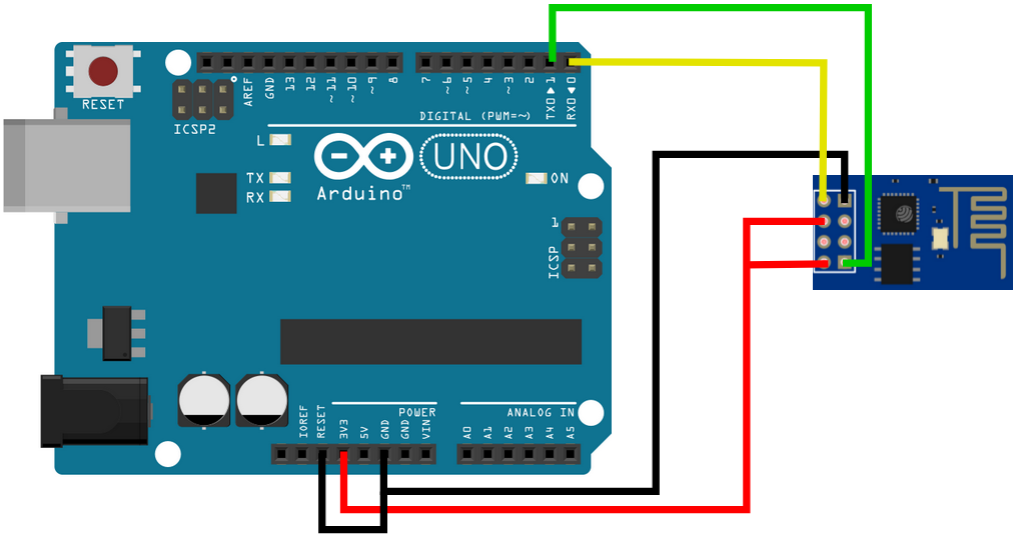


Figure 2.5 – ESP-01 in Bypass Mode

A standard power configuration for the ESP-01 module involves having the 3V3 (3.3V power) pin on the Arduino should be connected to the CH_PD and VCC pins of the ESP-01 module, and the module's GND pin should be connected to the Arduino's GND pin.

The configuration specific to "bypass mode" in the figure above refers to the wi-fi module's RX pin being connected to the hardware TX pin (pin 1) on the Arduino, and the wi-fi module's TX pin being connected to the hardware RX pin (pin 0) on the Arduino. It's also important to note that the RESET pin on the Arduino be connected to the GND pin.

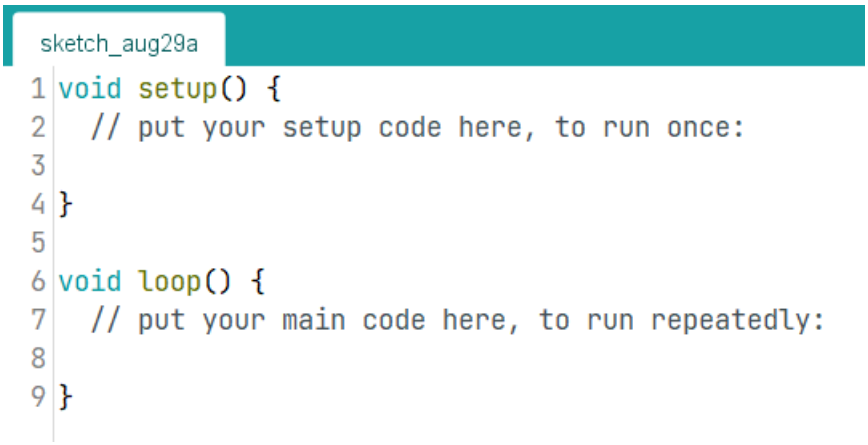


Figure 2.6 – Empty Arduino Code

After setting up the Arduino and ESP-01 module as shown in figure 2.x, connect the Arduino to your computer and upload an empty code as shown in the figure above.

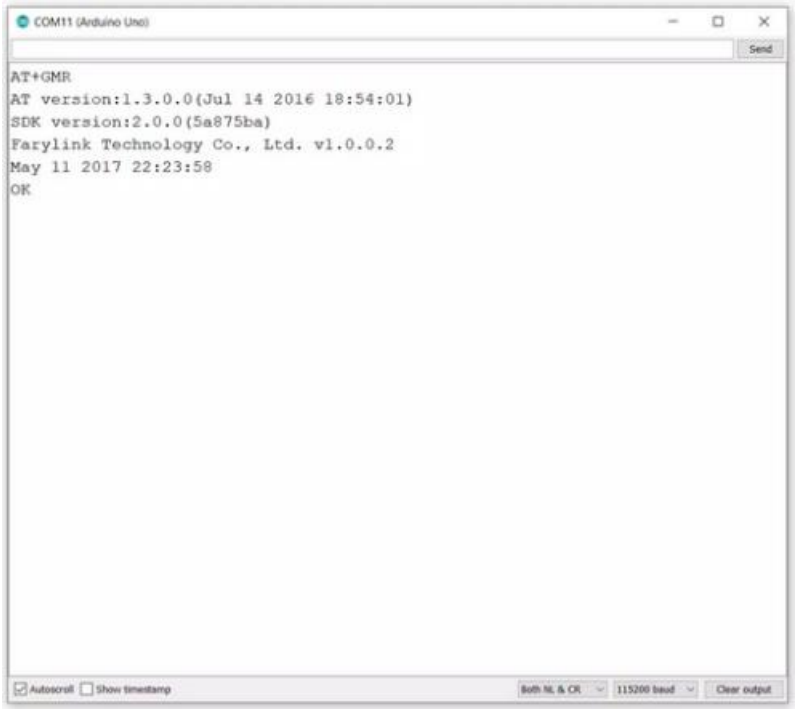


Figure 2.7 – The Arduino IDE Serial Monitor

After uploading the code, open the Arduino IDE's Serial Monitor. Before you start typing in the AT commands, make sure that the Serial Monitor is configured to append your command using **Both NL & CR**, and is set to **115200 baud**. If you opt to not use the auto append NL & CR feature, you must add the carriage return and newline characters to the command String yourself. If you do not set the baud rate to 115200, responses from the ESP-01 module might be garbled or not display at all.

As for **the second way** (SoftwareSerial method), the ESP-01 module communicates through the Arduino's microcontroller via the `SoftwareSerial` class. This method is the ideal method if we wish for the Arduino and ESP-01 module to be able to operate independently without a direct connection to a laptop or desktop computer.

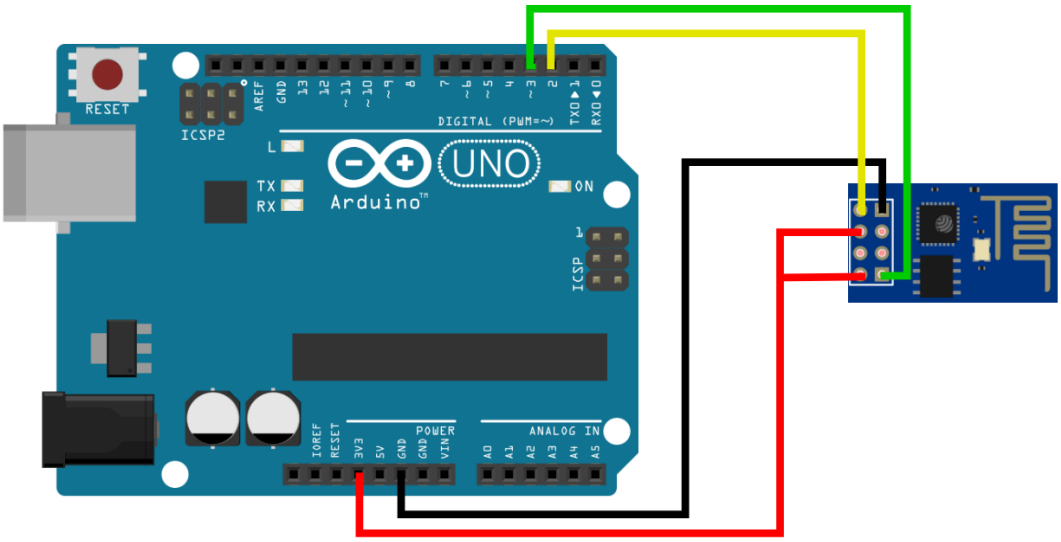


Figure 2.8 – ESP-01 in *SoftwareSerial* Mode

The configuration is largely the same as the bypass mode configuration, though with two key differences. First, the RX and TX pins should be moved to a pin other than 0 and 1. Second, the RESET-GND connection should be removed altogether.

In this specific use case, communicating with the Arduino over SoftwareSerial is unreliable, because all the sensors are rated for **9600** baud, whilst the ESP-01 module is rated for **115200** baud. This isn't a *blocking* issue, because SoftwareSerial can run in parallel of the primary *Serial.begin(9600)*. Running Serial and SoftwareSerial at different baud rates means that the communicated data between the two are garbled, hence the unreliability. Unfortunately, I know no other way that would allow the Arduino to operate independently from a dedicated computer.

To communicate with the ESP-01 module over SoftwareSerial, use the following code:

```
#include <SoftwareSerial.h>
SoftwareSerial wifi(2, 3); // RX, TX

const char* wifiName = "your_ssid_here";
const char* wifiPassword = "your_pass_here";
const char* host = "your_host_here";
const char* port = "80";
```

```
void setup() {
  Serial.begin(9600);
  wifi.begin(115200);
}

void loop() {
  /* Command Input */
  Serial.print("Command: ");
  while(Serial.available() == 0) {}
  String command = Serial.readString();
  Serial.println("User input: " + command);
  Serial.println("\n");

  /* Send Command */
  wifi.println(command);

  /* Compile Response */
  do {
    if(wifi.available()) {
      String currentResponse = wifi.readStringUntil('\n');
      Serial.println("Response: " + currentResponse);
    }
    else {
      delay(5000);
      break;
    }
  } while(true);
}
```

This code will emulate the behavior of the previous operating mode. The commands you send are always received without errors, however, the responses obtained from the ESP-01 module is garbled due to the Arduino's main Serial being run at **9600 baud** and the ESP-01's SoftwareSerial being run at **115200 baud**.

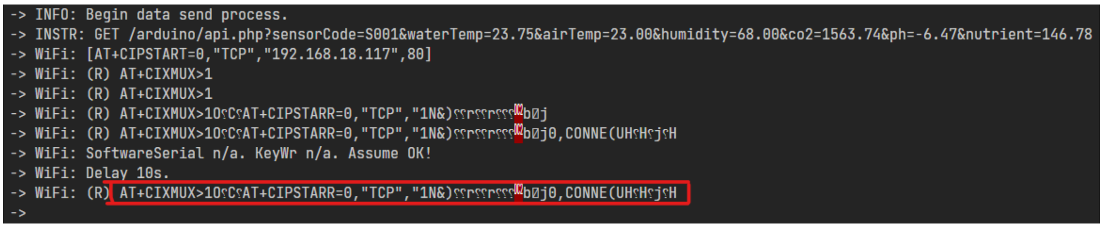


Figure 2.9 – Garbled Response From ESP-01

You can see in the figure above that the responses obtained from the ESP-01 over SoftwareSerial is unreliable. There are phrases that are “slurred, phrases that are simply incomplete, and phrases that are straight up unreadable.

Fortunately, this problem is mostly visual. My attempts to send data to my localhost database via LAN through IPv4 works fine. Unfortunately, this poses a response synchronization issue. Because the ESP-01 runs in parallel, it doesn’t automatically “synchronize” with the Uno’s main Serial. The ideal way of knowing and ensuring that the ESP-01 module has successfully received commands is by reading its response and picking out keywords such as “OK”. But with the Serial monitor reading garbled data from the ESP-01 module, this is not possible.

2.1.3 Water Temperature Sensor

To measure water temperature, I used a DS18B20 sensor. The documentation for this sensor is provided by Maxim Integrated, however, the model that I’ve bought is a third-party wired long probe, which is the one on the left on the figure below.

Figure 2.10 – Two Models of the DS18B20 Sensor

According to Maxim (2019), the DS18B20 is a digital thermometer capable of providing 9 to 12-bit Celsius temperature measurements. It’s capable of operating in either external power mode or parasitic power mode. Each DS18B20 unit has a unique 64-bit serial code, allowing multiple DS18B20s to use the same wire bus.

In more detail, according to Maxim (2019), the DS18B20 has the following features and specifications:

1. Capable of measuring temperature from -55 ºC to +125 ºC.
2. Has a ± 0.5 ºC accuracy from -10 ºC to + 85 ºC.
3. Has a programmable resolution from 9-bit to 12-bit.
4. Capable of operating in parasitic power mode, which requires only the data and GND pins.

2.1.3.1 External Power Mode

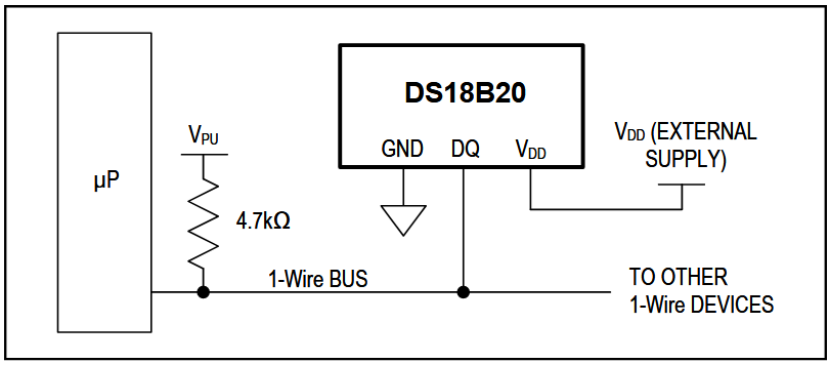


Figure 2.11 – The DS18B20 Operating in External Power Mode *(Maxim, 2019)*

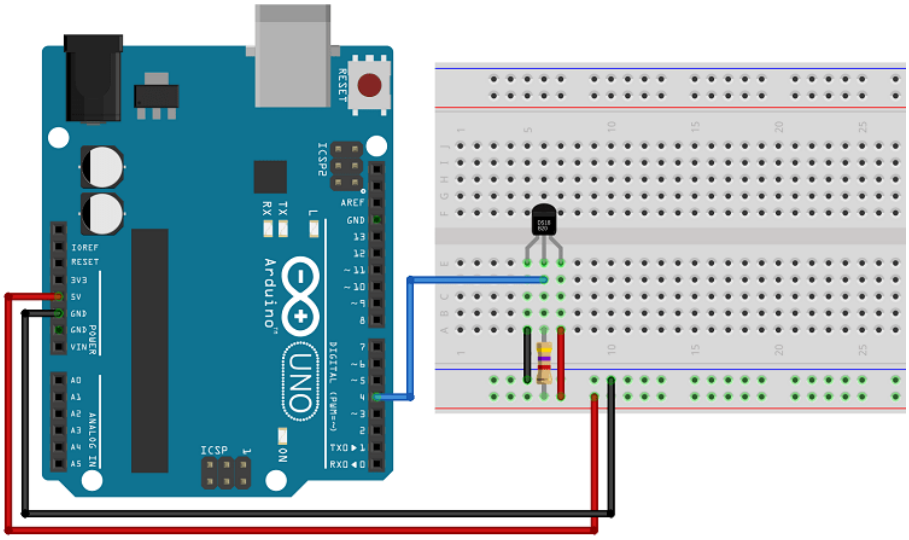


Figure 2.12 – The DS18B20 Sensor Operating in Normal Mode *(Santos, 2016)*

In external power mode, the sensor requires 3 wire connections, with power being provided by the VDD pin (Santos, 2016).

2.1.3.2 Parasitic Power Mode

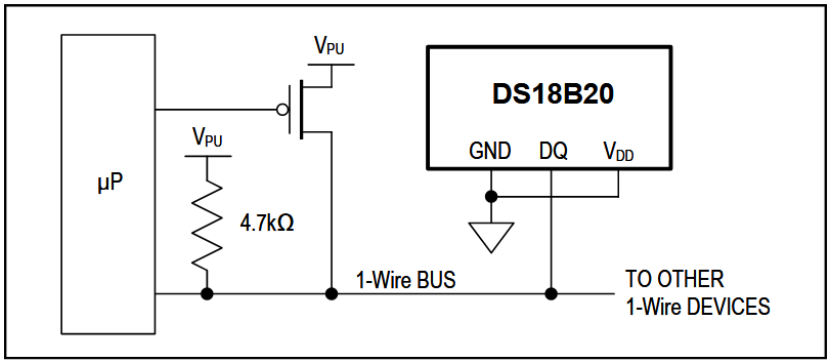


Figure 2.13 – The DS18B20 Operating in Parasite Power Mode *(Maxim, 2019)*

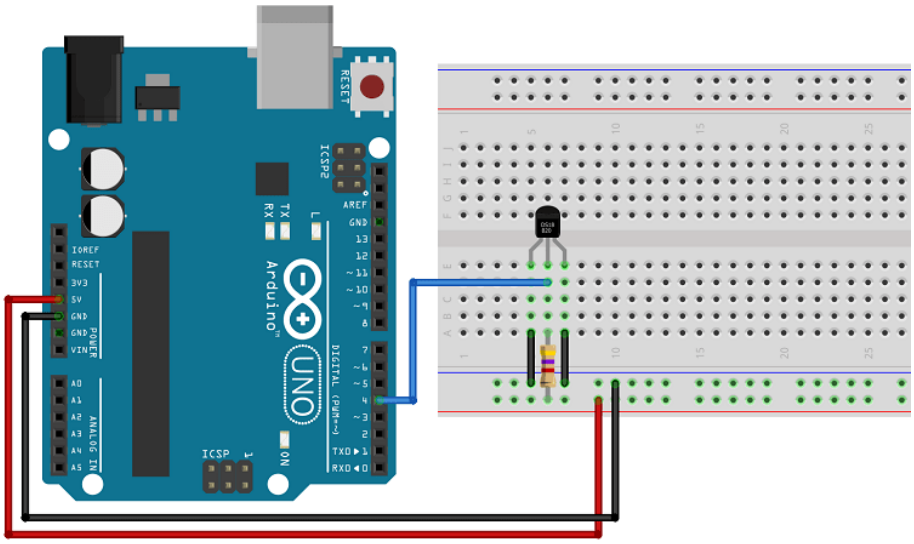


Figure 2.14 – The DS18B20 Sensor Operating in Parasitic Power Mode *(Santos, 2016)*

In parasitic power mode, because the sensor draws power from the data line, the sensor only needs two pins, the data (DQ) and ground (GND) pins (Santos, 2016).

It should be noted that the sensor's parasitic power mode has no practical use on the Arduino Uno, as the board's data pin does not have enough power to supply the sensor, hence it has to draw power from the power pin limited by a resistor.

2.1.3.3 Interfacing with the Sensor

Whichever power mode is chosen, to be able to successfully interface with the sensor, we need to install the OneWire and DallasTemperature libraries, which can be done from the "Manage Libraries" feature of the Arduino IDE (Santos, 2016).

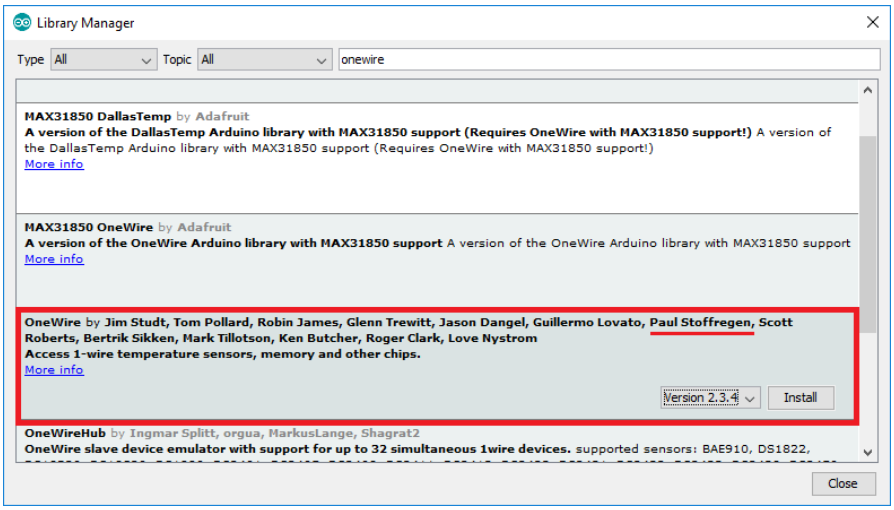


Figure 2.15 – The OneWire Library

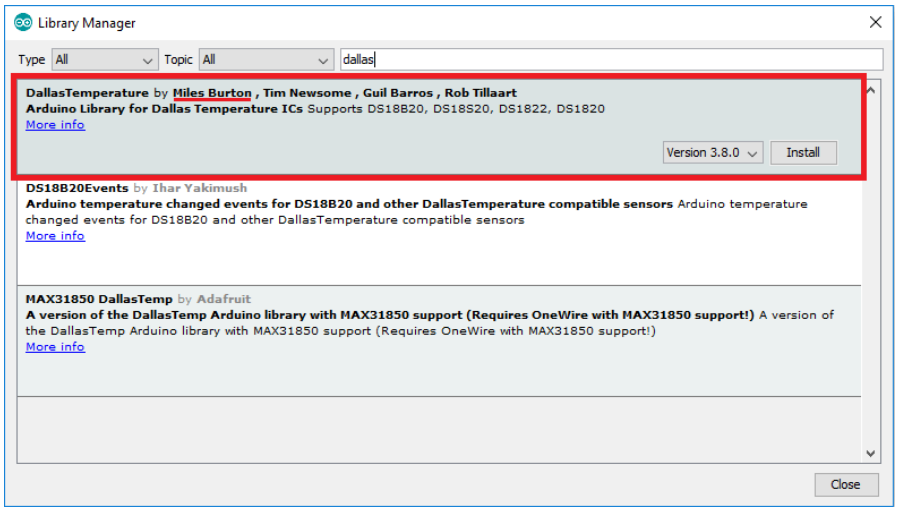


Figure 2.16 – The DallasTemperature Library

After arranging the components as shown in the figures above and installing the libraries, we can write a relatively simple code and upload it into our Arduino board to interface with the DS18B20 sensor.

```
#include “OneWire.h”
#include “DallasTemperature.h”

#define PIN_TEMPERATURE 4
OneWire temperature(PIN_TEMPERATURE);
DallasTemperature sensorTemperature(&temperature);

void setup() {
  Serial.begin(115200);
  sensorTemperature.begin();
```

```
}

void loop() {
  sensorTemperature.requestTemperatures();

  Serial.print(sensorTemperature.getTempCByIndex(0));

  delay(5000);
}
```

2.1.4 Humidity Sensor

2.1.4.1 Introduction

To measure humidity, I used a DHT11 sensor. According to Mouser Electronics (n.d.), the DHT11 is a sensor capable of measuring temperature and humidity with a calibrated digital signal output. The sensor's humidity measurement component is a resistive-type component and an its temperature measurement component is NTC. The DHT11 sensor has 4 pins, however pin 3, as is represented in the figure below, is not connected (NC).

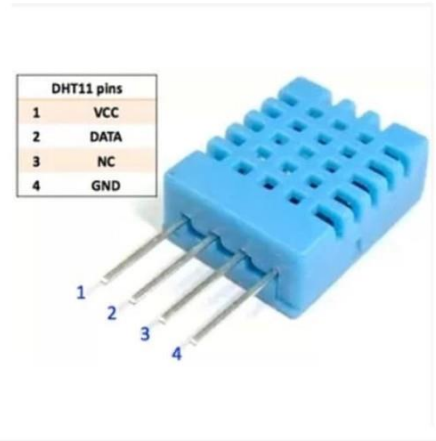


Figure 2.17 – The DHT11 Sensor *(Mouser, n.d.)*

2.1.4.2 Sensor Specifications

According to Mouser Electronics (n.d.), the DHT11 sensor has the following specifications:

1. Humidity Measurement Range: 20-90% RH
2. Humidity Accuracy: ± 5% RH
3. Temperature Measurement Range: 0-50 ºC
4. Temperature Accuracy: ± 2 ºC

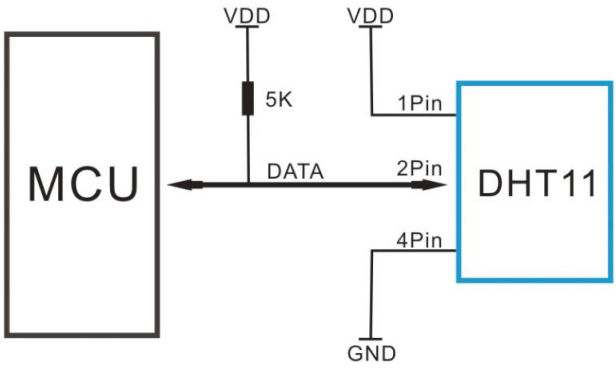


Figure 2.18 – Typical Application of the DHT11 Sensor *(Mouser, n.d.)*

The DHT11 is supplied by a 3-5.5V DC. When the sensor is powered, instructions should not be sent with less than 1 second delay to pass the unstable status (Mouser, n.d.)

If the DHT11 is operated outside of its specifications, it can result in a shift in the sensor's calibration. Chemical vapors may also interfere with the sensitive elements of the DHT11 sensor, resulting in miscalibration or permanent damage. Should these calibration shifts happen, and given that the sensor is not permanently damaged, the DHT11 can self-calibrate over-time if kept within its operating boundaries (Mouser, n.d.)

2.1.4.3 How It Works

The DHT11 sensor measures humidity (water vapor) by measuring the electrical resistance between two electrodes. Higher humidity results in lower resistance, while lower humidity results in higher resistance between the electrodes (Campbell, 2015).

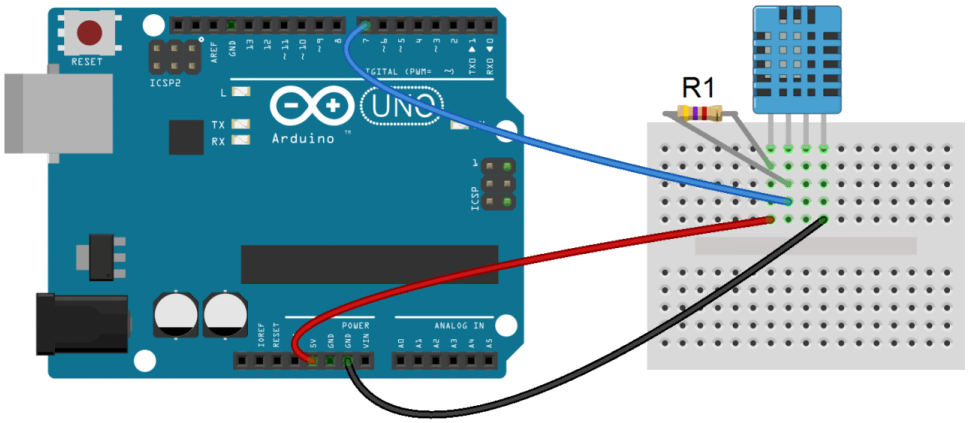


Figure 2.19 – Basic Schematic of a DHT11 Sensor in Use *(Campbell, 2015)*

To communicate and interface with the DHT11 sensor, we need to install the DHTLib library, which is available for download from a variety of websites on the internet. I obtained mine through GitHub.

2.1.4.4 The Code

After installing the DHT11 sensor according to the schematic and installing the DHTLib library, we can write and upload the relatively simple code below to interface with the DHT11 sensor.

```
#include <dht.h>

#define PIN_HUMIDITY 7
#define DHT_TYPE DHT11
DHT sensorHumidity(PIN_HUMIDITY, DHT_TYPE);

void setup() {
  Serial.begin(115200);
  sensorHumidity.begin();
}

void loop() {
  Serial.println(sensorHumidity.temperature);
  Serial.println(sensorHumidity.humidity);

  delay(1000);
}
```

2.1.5 CO_2 Sensor

To measure CO_2, I used an MH-Z19B sensor by Winsen Electronics. According to Winsen (2016), the MH-Z19B is a small-sized sensor using non-dispersive infrared (NDIR) technology to detect the presence of CO_2 in the air. It has a built-in temperature compensation, UART output, and PWM input. The MH-Z19B has a high sensitivity and resolution, low power consumption, built-in passive countermeasures against water vapor interference, and a relatively long lifespan.

Figure 2.20 – MH-Z19B Sensor *(Winsen, 2016)*

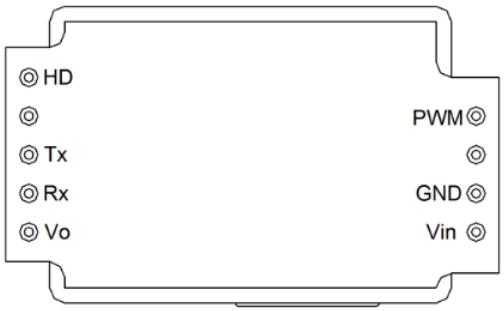


Figure 2.21 – MH-Z19B Diagram *(Winsen, 2016)*

According to Winsen (2016), as can be seen in the figure above, the MH-Z19B has nine pins, two of them are not used. The remainder 7 are:

- **Vin**, which is the power input pin.
- **GND**, which is the ground pin.
- **PWM**, which is the PWM pin.
- **Vo**, which is the analog output pin.
- **RX**, which is the UART input pin.
- **TX**, which is the UART output pin.
- **HD**, which is used for calibration purposes.

According to Winsen (2016), the MH-Z19B sensor by Zhengzhou Winsen Electronics has the following specifications:

Table 2.1 – MH-Z19B Technical Specification

Target Gas	Carbon Dioxide (CO_2)
Voltage	4.5V ~ 5.5V DC
Current (Average & Peak)	< 60mA @ 5V & 150mA @ 5V
Interface Level	3.3 V (Compatible with 5V)

Measurement Range	0 ~ 2000 ppm
	0 – 5000 ppm
Accuracy	± (50ppm + 3% measured value)
Output Signal	UART (3.3V)
	PWM
	DAC (0.4V – 2V)
Working Temperature	0 ~ 50 ^{o}C
Working Humidity	0 ~ 90% RH (No Condensation)

According to Winsen (2016), the MH-Z19B sensor by Zhengzhou Winsen Electronics has the following recommended software settings:

Table 2.1 – MH-Z19B Technical Specification

Baud Rate	9600
Data Bits	8
Stop Bits	1
Parity (Check Bits)	0 (NO)

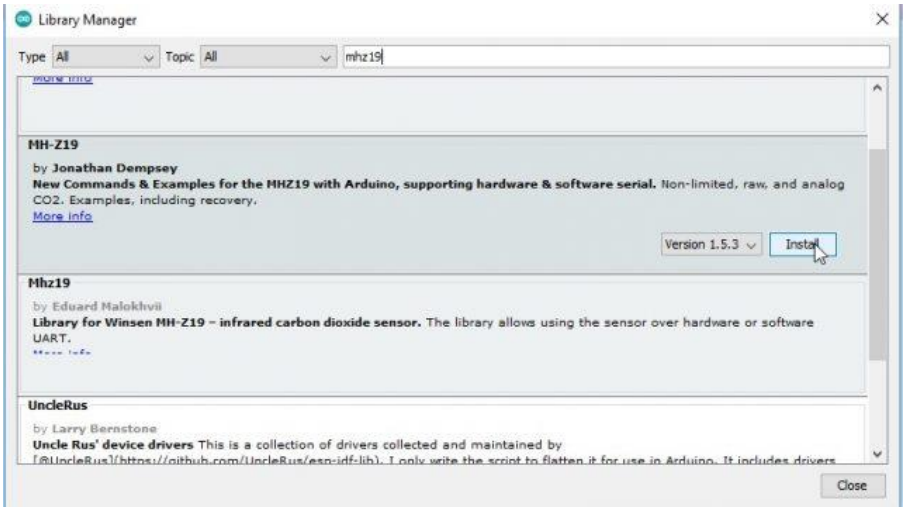
The sensor has an Automatic Baseline Correction (ABC) logic function, where the sensor will do a zero-point judgement and automatic calibration procedure every 24 hours after it is powered on. The automatic calibration has a zero-point of 400ppm. The sensor is also capable of manual and command calibration by utilizing the 0x87 and 0x88 commands. however, I've decided against using these manual methods, as my lack of expertise in this specific hardware might result in calibration inaccuracies that I might not be able to reverse (Winsen, 2016).

There are at least three ways to interface with the MH-Z19B sensor, which are digital, analog, and PWM (Winsen, 2016).

2.1.5.1 Digital Mode

The first method (digital) uses the UART (RX and TX) pins. This method requires the "MH-Z19" library. Though its theoretically possible to not use the library, it's highly impractical to do so.

To digitally interface with the MH-Z19B sensor, we need to install the “MH-Z19B” library from the Arduino IDE’s “Manage Libraries” feature.



Feature 2.22 – The MH-Z19B Library

In digital mode, the MH-Z19B sensor needs four connection points to the Arduino Uno board. The first is the Vin power pin, which needs to be connected to the 5V pin on the Arduino. The second is the GND pin, which needs to be connected to the GND pin on the Arduino. The third and fourth pins are the RX and TX pins, which needs to be connected to any of the Arduino’s digital pins.

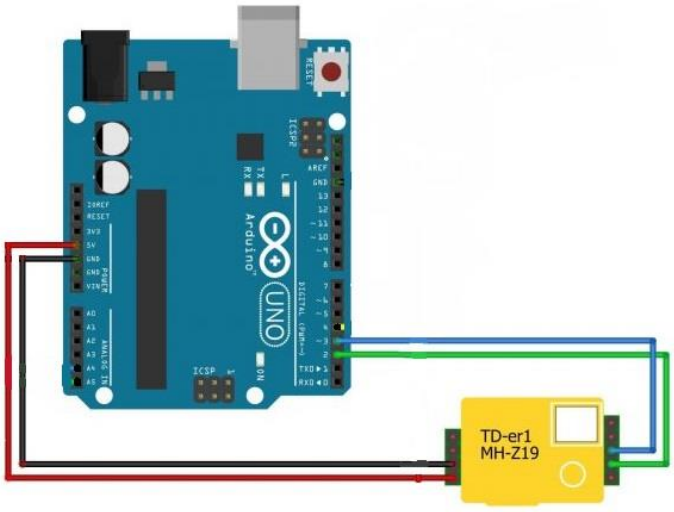


Figure 2.23 – MH-Z19B Schematic, Digital Mode *(Fahad, 2022)*

After installing the components as shown in the figure above and installing the MH-Z19B library, we can write and upload the relatively simple code below to communicate with the MH-Z19B sensor.

```
#include “Arduino.h”
#include "MHZ19.h"
#include “SoftwareSerial.h”

#define RXP 3
#define TXP 2

MHZ19 sensor;
SoftwareSerial mySerial(RXP, TXP);

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
  sensor.begin(mySerial);
  sensor.autoCalibration();
}

void loop() {
  int co2 = sensor.getCO2();
  int8_t temp = sensor.getTemperature();

  Serial.print("CO2 Levels: ");
  Serial.print(co2);
  Serial.println(" ppm");

  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println(" Celcius");

  delay(5000);
}
```

It’s important to note that the method above utilizes SoftwareSerial. While it is possible to have multiple instances of the SoftwareSerial class, only one can be used at a time, which might cause interference with other modules that requires SoftwareSerial (Arduino, 2023). Fortunately, interfacing with the MH-Z19B sensor using the analog method doesn’t have this problem.

2.1.5.2 Analog Mode

The second method (analog) uses only the analog output (Vo) pin. This method does not require the “MH-Z19” library. In analog mode, the MH-Z19B sensor only needs three connection points. The Vin and GND pins need to be connected to the 5V and GND pins on the Arduino and the Vo pin needs to be connected to any one of the Arduino’s analog pins.

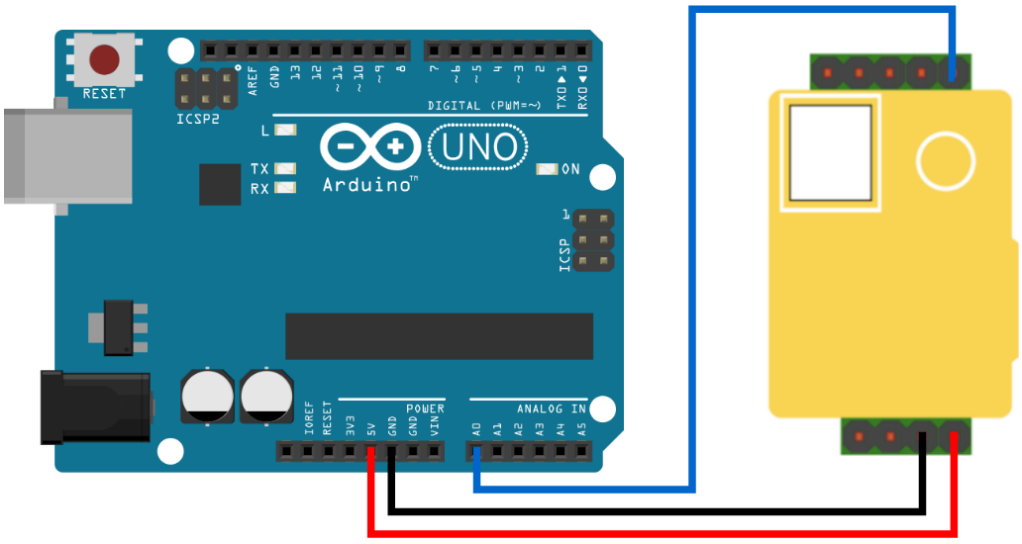


Figure 2.24 – MH-Z19 Schematic, Digital Mode

After arranging the components according to the figure and instructions above, we can upload the code below to communicate with the MH-Z19B sensor.

```
#include <Arduino.h>

#define PIN_CO2 A0

void setup() {
    Serial.begin(9600);
    pinMode(PIN_CO2, INPUT_PULLUP);
}

void loop() {
  float analogReading = analogRead(PIN_CO2);
  float co2Reading = 6s.4995 * analogReading - 590.53;

  Serial.println(co2Reading);
}
```

2.1.5.3 PWM Mode

The third method (PWM) uses only the PWM pin. This method also does not require the MH-Z19 library. Like Analog mode, the MH-Z19B sensor only needs three connection points with the Arduino. Again, the Vin and GND pins need to be connected to the 5V and GND pins on the Arduino board. The PWM pin needs to be connected to a PWM-capable digital pin on the Arduino board.

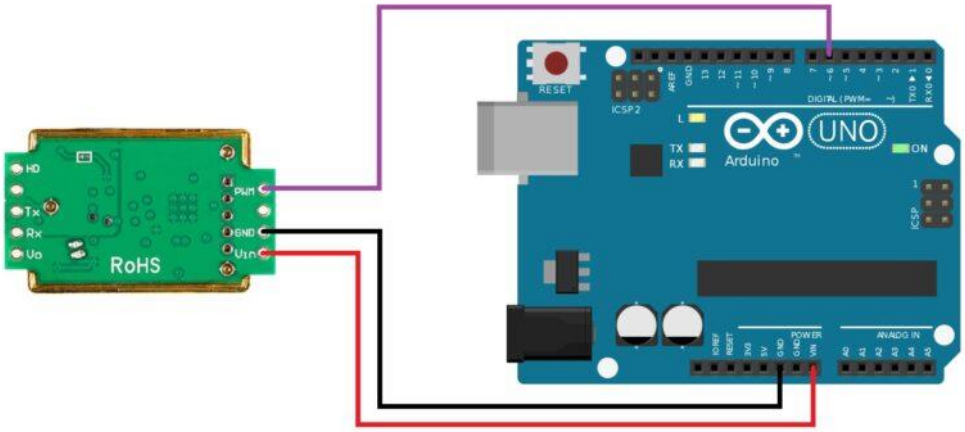


Figure 2.25 – MH-Z19 Schematic, PWM Mode *(IoTSpace, 2021)*

After arranging the components according to the figure and instructions above, we can upload the code below to communicate with the MH-Z19B sensor.

```
#define PIN_CO2 6

unsigned long pTime;

void setup() {
  pinMode(DataPin, INPUT);
  Serial.begin(152000);
}

void loop() {
  pTime = pulseIn(DataPin, HIGH, 2000000) / 1000;
  int co2ppm = 5000 * pTime / 1004.0;

  Serial.println(co2ppm);

  delay(5000);
}
```

2.1.6 pH Sensor

To measure pH levels, I used a PH-4502C sensor kit. The sensor kit comes with the PH-4502C module and a pH probe.

2.1.6.1 The Module & Specifications

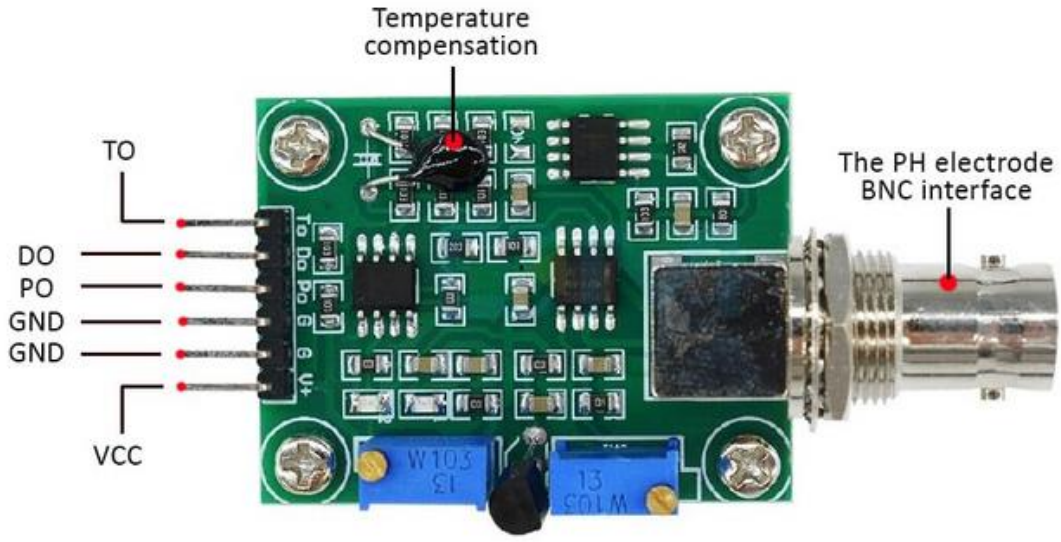


Figure 2.26 – PH-4502C Module *(CimpleO, 2020)*

According to the figure above, the PH-4502C module has six pins on its left side, each with its own functions (CimpleO, 2020).

1. The TO pin is for temperature output.
2. The DO pin is for 3.3V output for pH limit.
3. The PO pin is the analog pH output.
4. The first GND is for the pH probe.
5. The second GND is for the board itself.
6. The VCC pin is the 5V power pin.

The PH-4502C board communicates pH value via voltage. The board defaults to pH 7 at 0 V, which means it will go minus when it reads acidic an pH value, which cannot be read by Arduino (CimpleO, 2020). This should be offset so that a pH value of 0 is represented by 0V, and a pH of 14 is represented by 5V.

2.1.6.2 The Probe

Figure 2.27 – The pH Probe

The figure above is the pH probe. According to “pH Probe Architecture” (n.d.), a pH electrode measures pH levels by the voltage potential that it measures, which is a function of the acidity or alkalinity of the liquid in which it is submerged. It is important to note that pH probes are inherently unstable. They require cleaning and calibrating on a regular basis.

2.1.6.3 Setup

To setup the sensor, we need to connect 4 of the 6 available pins to the Arduino board. The PO pin should be connected to the Arduino A0 analog port. Both GND pins should be connected to the Arduino’s GND ports. The VCC pin should be connected to the Arduino’s 5V port.

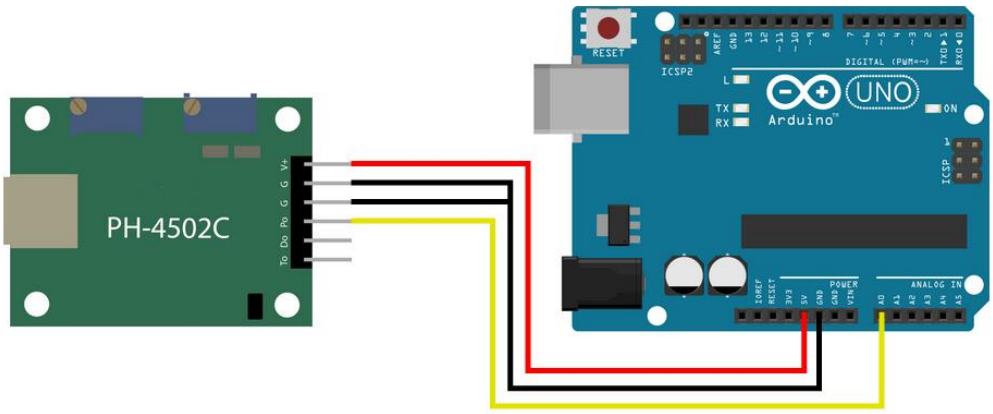


Figure 2.28 – A Schematic of the PH-4502C with Arduino

2.1.6.4 Calibration

To perform calibration, the potentiometer near the BNC connector needs to be turned to the correct offset. To set the offset, the BNC connector's inner wire needs to be shorted with its outer shell, as is shown in the figure below. This will simulate a neutral pH value of 7. After this, the value of the PO pin should be measured using a multimeter, and the offset on the potentiometer should be adjusted until it reads 2.5 V (CimpleO, 2020).

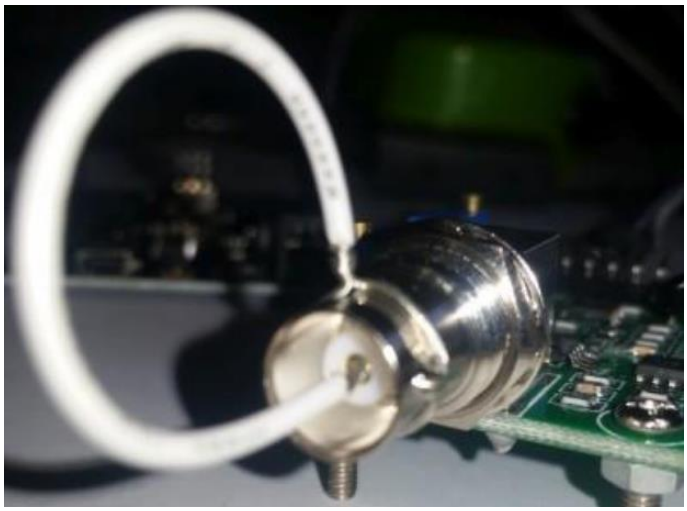


Figure 2.29 – PH-4502C Probe Offset Calibration

According to CimpleO (2020), it is important to note that a pH probe take some time to get to the correct value (at least two minutes). The readings may also fluctuate if the temperature is above 30 ºC or below 10 ºC.

2.1.6.5 The Code

After arranging the parts as instructed, we can use the code below to interface and communicate with the PH-4502C module.

```
#include “Arduino.h”

#define PIN_PH A0 // ANALOG

void setup() {
  Serial.begin(115200);
}
```

```
float calcPH(float volts) {
  return ((2.5 - volt) / 0.18) + 7;
}

void loop() {
  int capture = 0;

  for(int i = 0 ; i < 10 ; ++i) {
    capture += analogRead(PIN_PH);
  }

  float volts = 5 / 1023.0 * measurings / 10;

  Serial.println(calcPH(volts));

  delay(5000);
}
```

2.1.7 PPM Sensor

2.1.7.1 Introduction

To measure PPM (parts per million), I used an Analog TDS Sensor by DFRobot. According to DFRobot (n.d.), this TDS sensor provides an analog output that is Arduino compatible. It takes a wide 3.3V ~ 5.5V input and outputs 0V ~ 2.3V analog. The TDS probe is also waterproof and can be submerged in water for extended periods of time.

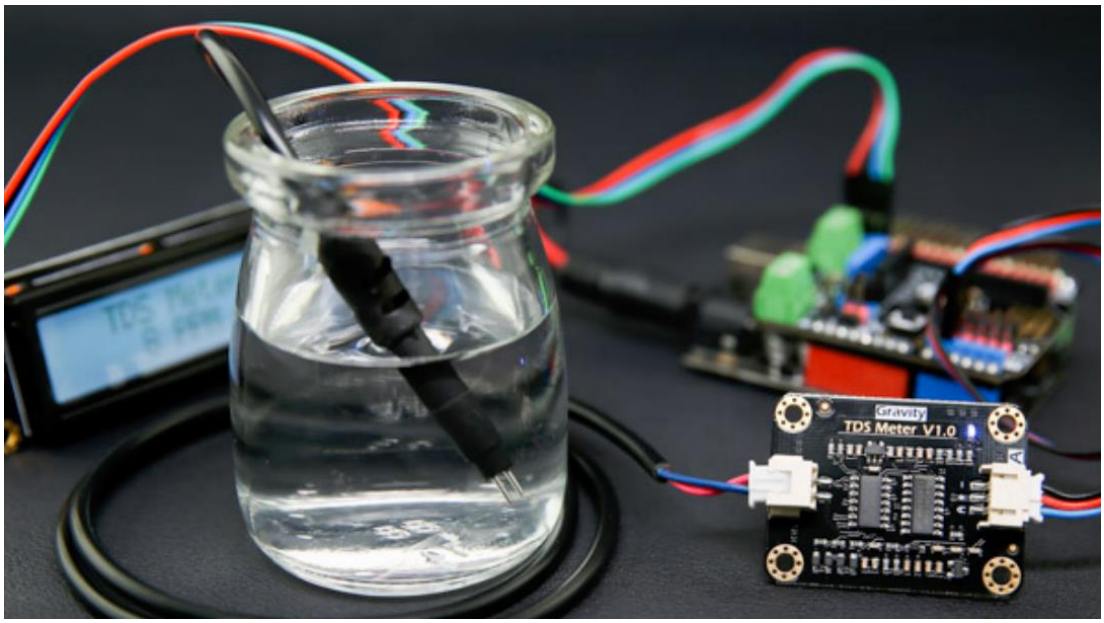


Figure 2.30 – DFRobot Analog TDS Sensor Illustration *(DFRobot, n.d.)*

2.1.7.2 Specifications

The Analog TDS Sensor kit includes two devices in a single set, which are the signal transmitter board and the TDS probe. These devices come with the following specifications (DFRobot, n.d.).

Table 2.3 – Specifications of the TDS Signal Transmitter Board *(DFRobot, n.d.)*

Input Voltage	3.3 ~ 5.5V
Output Voltage	0 ~ 2.3V
Working Current	3 ~ 6mA
Measurement Range	0 ~ 1000ppm
Measurement Accuracy	± 10% F.S. (25 ºC)
Module Interface	PH2.0-3P
Electrode Interface	XH2.54-2P

Table 2.4 – Specifications of the TDS Probe *(DFRobot, n.d.)*

Number of Needles	2
Connection Interface	XH2.54-2P
Waterproof	Yes

As is stated by the specifications above, the sensor kit's Signal Transmitter Board connects to the probe via the XH2.54-2P interface, while it communicates with the Arduino board using the PH2.0-3P interface. The details of the two interfaces can be seen in the following image.

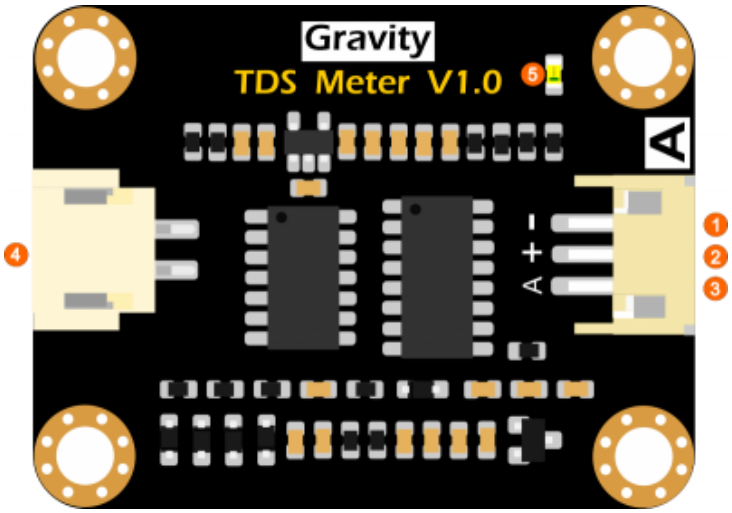


Figure 2.31 – Signal Transmitter Board *(DFRobot, n.d.)*

Shown in the figure above in numbers 1 to 3 is the PH2.0-3P connection interface, which is just a compacted version of the individual connections that make it up. As is suggested by the writings on the figure above, number 1 is the Power GND, number 2 is the Power VCC (3.3 ~ 5.5V), and number 3 is the Analog Signal Output. On the left side of the figure is the XH2.54-2P connector, which is supposed to connect to the probe. On the top right corner of the figure is number 5, which is the LED power indicator.

2.1.7.3 How It Works

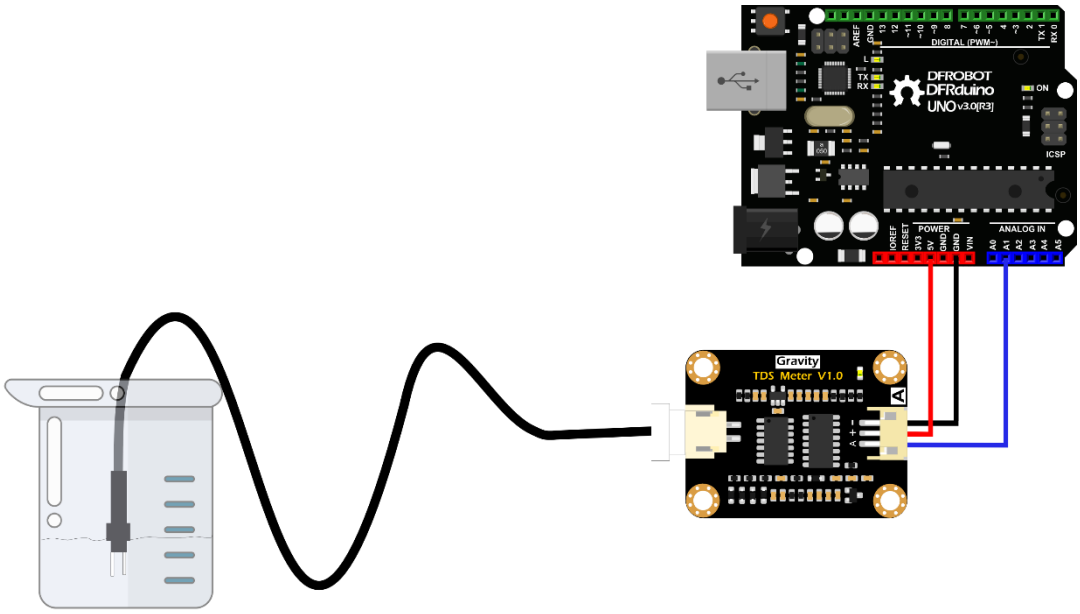


Figure 2.32 – Example Use of the Analog TDS Sensor Kit *(DFRobot, n.d.)*

Shown in the figure above is an example of how this sensor kit can be used with Arduino. The PH2.0-3P connection interface is split into three, with GND connecting to GND, VCC connecting to 5V, and Analog connecting to A0 ~ A5 on the Arduino board. The probe is also connected to the other side of the board and is dipped into the medium intended to be measured.

2.1.7.4 Calculating PPM

Before we continue, there is the very important matter of calculating the TDS PPM, which is not as simple as one might think. This sensor by DFRobot measures TDS PPM by measuring electrical conductivity. In fact, most TDS sensors are just electrical conductivity sensors with automatic conversion to PPM.

According to Fernandez (2014), electrical conductivity is measured by passing a small amount of electricity between two electrodes. Pure distilled water has no electrical conductivity, so adding solids like hydroponic nutrients increases its conductivity. Electrical conductivity is measured using a unit called siemens, though its more common to use milisiemens (mS) or microsiemens (µS).

There are many methods used to measure TDS PPM from electrical conductivity, and this usually results in confusion for people that are inexperienced in measuring them. But according to DFRobot (n.d.), their sensor uses the NaCl conversion factor which is 0.47 rounded up to 0.5. This conversion factor is multiplied by the electrical conductivity in microsiemens to produce PPM.

Even though this lack of standard is very confusing, fortunately Grotek Canada has set the standard conversion factor to use 0.5 (Fernandez, 2014).

2.1.7.5 The Code

After arranging the parts and connecting them to the Arduino as instructed in the figures above, we can use the code below to communicate with the TDS sensor.

```
#include “EEPROM.h”
#include "GravityTDS.h"

#define PIN_NUTRIENT A1
GravityTDS sensorNutrient;

void setup() {
  Serial.begin(115200);

  sensorNutrient.setPin(PIN_NUTRIENT);
  sensorNutrient.setAref(5);
  sensorNutrient.setAdcRange(1024);
  sensorNutrient.begin();

}

void loop() {
  sensorNutrient.setTemperature(waterTemperature);
  sensorNutrient.update();

  Serial.println(sensorNutrient.getTdsValue());

  delay(5000);
}
```

2.1.8 Possible Alternatives

2.1.8.1 Raspberry Pi

Figure 2.33 – The Raspberry Pi

During the conception and ideation process of this case study, I considered using a Raspberry Pi instead of Arduino Uno but decided not to. There are several reasons for why I went with Arduino instead of Raspberry Pi.

To better understand my choice, first you need to understand what a Raspberry Pi is and how it's different to an Arduino.

From my limited understanding of both platforms, Arduino is a small prototyping board with a small microcontroller, which is basically a very small and very slow computer. The amount of memory, storage, and processing power of an Arduino is very limited. An Arduino provides a flexible and very easy-to-use environment to design prototypes.

The Raspberry Pi on the other hand, though small, is a full-blown computer. Compared to the Arduino, the Raspberry Pi has a vastly faster processor, tremendously more memory, and supports a much larger storage in the form of SD cards. The Raspberry Pi can run full-blown operating systems like Linux.

For the purposes of designing the hardware of this case study, I could have gone either with Arduino or the Raspberry Pi, but I chose Arduino for several reasons:

- Arduino is much cheaper than the Pi, with a single Raspberry Pi costing over 20 times that of a single Arduino board.

- Sensor module availability is much greater with the Arduino, with almost any sensor that you can think of being available for the Arduino. Not necessarily true for the Raspberry Pi.
- Sensor modules for the Arduino are also much cheaper compared to the equivalents for Raspberry Pi.
- The capability of the Arduino Uno is sufficient for this case study. And should it be marginally insufficient, there are more powerful platforms such as the Arduino Mega, which are still much cheaper than a Raspberry Pi, which still costs 10 times as much as an Arduino Mega.
- Developing code for the Arduino is also much easier, as it has its own IDE and operating system. Developing software for the Raspberry Pi would be much more complex, as Raspberry Pis usually run Linux. Availability of software libraries also heavily favor the Arduino platform.

2.1.8.2 NodeMCU

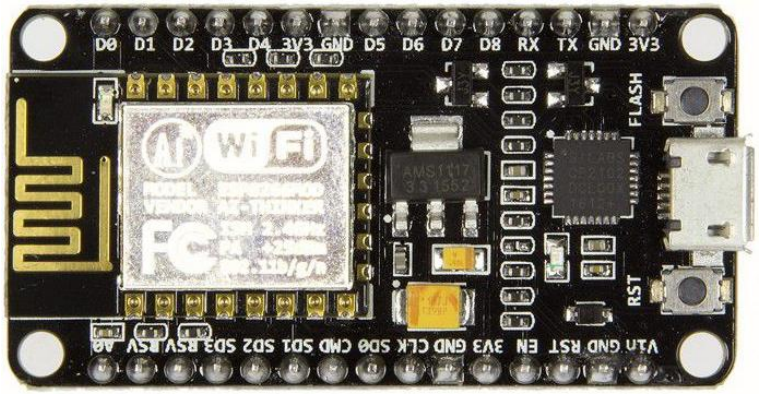


Figure 2.32 – NodeMCU Lolin

During my studies of the ESP-01 module, I briefly considered using NodeMCU instead of Arduino Uno. Not only is the NodeMCU priced similarly to the Arduino Uno R3, it is also smaller and more compact. The NodeMCU board also features a built-in ESP-8266 Wi-Fi module, which eliminates the need for the ESP-01 module. The NodeMCU also has 64KB of RAM compared to the Uno's 2KB and 4MB of flash memory compared to the Uno's 32KB ("NodeMCU ESP8266", 2020).

Ultimately, the deciding factor that led me to not use the NodeMCU was its lack of Analog ports. My use case requires at least two analog ports, whilst the NodeMCU only has one. My lack of familiarity with the NodeMCU boards also played a part in me deciding not to use them.

2.2 Theoretical Foundation of the Web Application

2.2.1 PHP

Originally known as “Personal Home Page Tools”, Hypertext Preprocessor (PHP) is a highly popular scripting language that is very commonly used in web development. It is a highly versatile and scalable, and features cross-platform compatibility, which makes it highly desirable for web developers around the globe. It has served a vital role in shaping the modern web into what it is right now (PHP, 2023).

PHP was created in 1994 by Rasmus Lerdorf as a simple set of Common Gateway Interface (CGI) binaries written in C. Over time, it evolved into a web development scripting language. The first widely used version of PHP was PHP 3. The “modern” PHP as we know it started with PHP 5, when object-oriented programming was introduced. PHP 7 improved many aspects of PHP by reducing memory consumption and increasing the syntax’s expressivity (PHP, 2023).

One of PHP’s most significant traits is its ability to communicate with a variety of database management systems such as Oracle, PostgreSQL, and MySQL, which makes it easy for developers who wish to create websites that interact with databases (PHP, 2023).

PHP is also capable of being embedded in HTML, which allows for a seamless intermixing of interface code and logic code. PHP’s robust support for web protocols like HTTP, FTP, and many others makes it an adaptable and highly versatile language, which allows PHP to seamlessly integrate with other web applications (PHP, 2023).

In addition, PHP is an already established and widely accepted programming language with a vast community of developers still actively using it. This large

number of developers contribute to PHP's development, making it evolve into ever more efficient forms. PHP's open-source nature also makes it highly accessible and cost effective to customization, making it the de facto choice for many projects large and small. Consequently, PHP has an immense amount of libraries and frameworks like Laravel and Symfony, which makes it even easier to use (PHP, 2023).

This already established and widely recognized language makes it the ideal choice for this project, especially if the management of JUST HYDROPONICS wishes to pursue this solution further. The abundance of developers with expertise in PHP makes it easy to find developers who are willing to work on this project further, far after the conclusion of this case study.

2.2.1.1 MVC Design Pattern

According to Hernandez (2021), the Model-View-Controller (MVC) design pattern is a software development pattern that is widely used for building web applications. As the name suggests, the MVC design pattern separates an application into three overarching components, which are the Model, the View, and the Controller. This design pattern ensures application code modularity, which in turn makes the code flexible, reusable, and easily maintainable.

The **Model** contains the application logic. It handles temporary data storage during runtime and data manipulation, such as file I/O, database connections, and data processing. The **View** is responsible for displaying data to the user. View files usually do not contain any application logic, it only displays data given to it by a controller. The **Controller** handles the flow of the application. Where the model is responsible for most of the heavy lifting, the controller is responsible for the "logistics" of the application, where data is exchanged between models and views, and even between multiple models (Hernandez, 2021).

The MVC design pattern has a typical and easily recognizable process and interaction flow, which usually looks like this:

1. The user performs an action on the View.
2. The action then invokes the Controller.

3. The action is then processed by the Controller, which calls the necessary Models or (other) Views.
4. The View displayed to the user is then updated accordingly, or a new one is displayed.

The MVC design pattern offers several advantages:

- **Modularity**, where various parts of the code can be developed, tested, reused, and maintained independently of each other.
- **Flexibility**, so long as the basic nature of each component (their input and output) stays the same, whatever happens inside of each can be changed.
- **Scalability**, where the design pattern introduces a modicum of rigidity that is essential in building applications that are anything other than small.

But the MVC design pattern also has its disadvantages:

- **Boilerplate Verbosity**. The MVC design pattern has rather strict rules about how the code should be written, which results in a large amount of boilerplate code. Although this ensures proper code-writing practices and makes it easy to understand for experienced developers, it significantly increases development time.
- **Complexity**. This design pattern is only suitable for medium to large sized applications. The boilerplate verbosity means that it is very inefficient and thus ill-suited for small applications.

2.2.2 HTML & CSS

Hypertext Markup Language (HTML) and Cascading Style Sheet (CSS) are the two quintessential languages in creating most modern web pages that are running today. HTML acts as the skeleton or structure of the website, which defines the layout of the website. CSS acts as the skin or appearance of the website, which defines the website's visual styling.

According to "HTML Introduction" (n.d.), HTML is a markup language whose role is to define the structure and layout of webpages. It has a set of elements that can be used to define things such as paragraphs, titles, headings, and many more.

These elements are then structured into something called a Document Object Model (DOM). The DOM's role is to render the web page itself.

HTML documents are structured into nested elements, organized into neat hierarchies. Each element is declared by using tags opened and closed using angle brackets. Each element may also have attributes, which provides additional functionality and information to the element (such as its color, positioning, alignment, etc.) ("HTML Introduction", n.d.)

In contrast, according to "CSS Introduction" (n.d.), CSS is a style sheet which allows website designers to define the visible appearance of the website, which covers things like font, color, margins, and so on. CSS allows for a more precise control over the layout, positioning, and display characteristics of a website's contents. CSS can target specific HTML elements such as "`h1`", "`p`", "`a`", etc.

CSS is designed specifically to define the presentation of HTML elements, and when combined, HTML and CSS allows for the creation of tidy, visually engaging, and user-friendly websites. Their nature of being flexible and easy to use has virtually turned it into the world's standard for frontend website development.

As the name suggests, CSS follows the cascading principle, which means that the styles defined in CSS can be inherited, overridden, or combined. CSS can also specify which style/rule takes precedence when there are multiple styles/rules applied to an element. CSS is also capable of performing "media queries", which allows website developers to style a web page according to the user's device, such as screen resolution and orientation ("CSS Introduction", n.d.).

2.2.3 JavaScript

According to Mozilla (2023), JavaScript is a dynamically typed and interpreted programming language used primarily in web development. JavaScript allows you to introduce complex, interactive, and responsive features on web pages. Along with HTML and CSS, JavaScript serves as the third technology layer that is used in the vast majority of modern websites' front facing code. Hence, knowledge of JavaScript is quintessential for websites with a user experience acceptable for modern standards.

JavaScript has many features which makes it into the programming language that we know. Some of those features are:

- JavaScript is Turing-complete, which means that it can perform any computation that can be algorithmically expressed, which makes it powerful and versatile (Binance, 2023).
- Naturally, JavaScript has all the basic features present in almost all programming languages such as variables and scope, functions, control flow structures, objects, and asynchronous programming.
- JavaScript has a C-like syntax, similar to C++, C#, and Java. This makes JavaScript's learning curve relatively easy for developers experienced in the aforementioned languages.
- JavaScript is capable of event handling, which allows it to detect user actions such as clicks and keystrokes.
- JavaScript is capable of Document Object Model (DOM) manipulation, which is basically just JavaScript's capability of manipulating static HTML code, allowing web pages to be more dynamic.

Originally, JavaScript was created for front-end website development, which revolutionized the way websites are written and enables interactive web pages. However, with the recent introduction of frameworks like Node.js, JavaScript is now also capable of server-side programming, a task traditionally handled by PHP or ASP.NET. This means that developers can create full-stack web applications using JavaScript without a dedicated server-side scripting language (Mozilla, 2023).

2.2.4 XAMPP

XAMPP is a cross-platform, open-source packaged web development environment distribution containing MariaDB, PHP, and some other modules. XAMPP's goal is to simplify the process of creating and managing a local web development environment. XAMPP includes several components, which are: Apache Web Server, MySQL, PHP, and Perl. Upon first install, all these components com pre-configured, which makes it easy for web developers to immediately begin development and testing work. I chose XAMPP not only because it's easy to use, but

also because it contains everything I need for the development and deployment testing of the web application on this case study, which are PHP, a MariaDB server, and a Web Server.

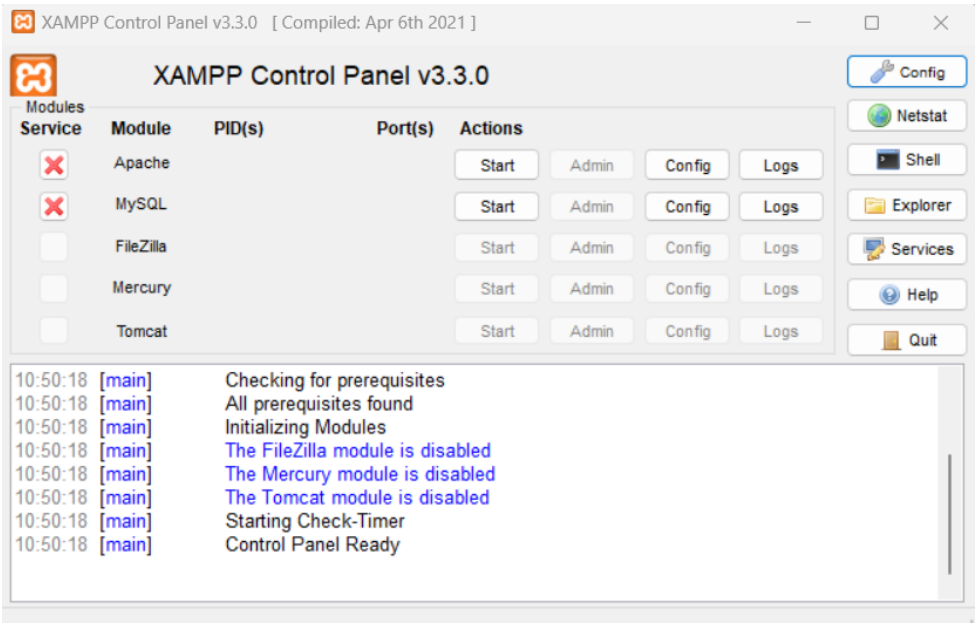


Figure 2.33 – XAMPP Control Panel

2.2.4.1 MySQL

According to MySQL (n.d.), MySQL is the most popular open-source relational database management system (RDBMS) developed by Oracle. MySQL has the following features:

- As the name suggests, MySQL follows the relational database (RD) model, where data is structured into tables with rows and columns.
- MySQL also organizes databases into *schemas*.
- Naturally, MySQL uses SQL as its query language to interact with its databases.
- MySQL supports a wide variety of data types supported by other DBMS, such as integers, floating point numbers, strings, Booleans, dates, and times.
- MySQL is relatively simple to setup and use compared to other DBMS.

I chose MySQL primarily for its simplicity, which perfectly suits this use case, and my own extensive experience with it, which I expect to tremendously increase my development speed with it compared to other DBMS.

2.2.4.2 Apache Web Server

According to the XAMPP Team (2023), the Apache Web Server is an open-source web server and application deployment platform. The Apache Web Server has the following features:

- It can be hosted locally, with a very simple one-click sever start procedure. You only need to put the web application that needs to be deployed in the *htdocs* folder.
- It is maintained primarily by the Apache Software Foundation (ASF) but has a large and active community base of users and contributors.
- Apache supports virtual hosts, which allows a single server to host multiple web applications.
- Apache is also available on all of the major operating systems such as Windows, macOS (OSX), and Linux.

The Apache Web Server serves as the test deployment platform during the development process of this case study's web application. It was chosen for its simplicity because it comes pre-packaged and pre-configured with XAMPP. It is also an acceptable web server base should this case study's results be continued to production.

2.2.4.3 phpMyAdmin

XAMPP comes pre-packaged with phpMyAdmin, which is a PHP web-based database management application for MySQL and MariaDB databases. phpMyAdmin is very useful for developers who do not yet have a dedicated database engine (XAMPP Team, 2023).

2.2.5 Amazon Web Services (AWS)

According to Amazon (2013), Amazon Web Services (AWS) is one of the most widely used cloud computing platform in the world, offering an impressive suite of over 200 services from availability zones all around the globe. Its user base extends from startups to corporations and governments.

AWS provides a wide range of services, surpassing most other cloud providers in both quality and quantity. From foundational infrastructure like computing and storage to cutting-edge technologies such as machine learning, artificial intelligence, data analytics, and the Internet of Things (IoT), the services provided by AWS are swift, efficient, and cost-effective. With these services, AWS makes it easy to migrate applications to the cloud, and even enables the creation of new diverse solutions previously unfeasible or impossible to do without a cloud platform.

AWS also boasts one of the largest and most dynamic communities, boasting millions of active customers and tens of thousands of global partners. This extensive network includes a diverse range of industries, from startups to corporations, utilizing AWS for a multitude of applications. Security is an important focus for AWS, with its core infrastructure capable of meeting the stringent security requirements of entities like global financial institutions. AWS is also known for its innovation, allowing users to leverage the latest technologies for experimentation and quick adaptation. Notably, AWS introduced serverless computing with Lambda in 2014 and developed a machine learning service called SageMaker that can be used by developers and scientists with no prior machine learning experience.

Boasting over 17 years of experience, AWS' maturity, reliability, security, and performance are only matched by other technology giants like Google, Microsoft, and Oracle. It is trusted by millions of users globally across a wide range of use cases. AWS's Region and Availability Zone model was also recommended by Gartner as the best approach for high-availability enterprise applications, further solidifying its position as a global cloud infrastructure leader.

In the scope of this project, I chose AWS for its relatively low cost, proximity of the nearest availability zone (in Jakarta), and my own personal experience with using AWS. AWS was covered during my "Cloud Technology" course and it has a 12-month free trial period, which makes it the ideal choice for the cloud computing platform for this project as I will be able to achieve much greater progress for relatively less time and resources.

2.2.5.1 Amazon RDS

According to Amazon (2023), RDS simplifies relational database management because it is scalable and cost-efficient. Unlike manual management with Amazon EC2, RDS automates tasks like backups and patching, supporting popular engines such as MySQL and PostgreSQL. It enhances security through IAM and VPC. For more control, RDS Custom provides flexibility. AWS Outposts extends managed databases on-premises. DB instances, available in various engines and classes, offer different capacities and storage options. Operating in AWS Regions with Availability Zones ensures high availability. Security groups control access, and monitoring is facilitated through Amazon CloudWatch. Amazon RDS follows a shared responsibility model, managing infrastructure while users handle query tuning and optimization.

2.2.5.2 Amazon EC2

According to Amazon (2023), EC2 provides flexible and scalable computing power, streamlining server management for dynamic capacity adjustments. Instances operate within a secure Virtual Private Cloud (VPC), each protected by a firewall. Key features include various instance types, secure login with key pairs, and diverse storage and networking capabilities.

2.3 Literature Review

This literature review was performed primarily for the purpose of providing ideas and inspiration for my own writing of this case study. It also provides theoretical contexts which supports my own case study. Combining elements from these different literatures allowed me to formulate an entirely new and different concept.

I have chosen three “literatures” below from YouTube. I chose them based on their relevance and ability to contribute to my own case study. Having these existing literatures have been a godsend, as they relieved much of the cognitive load of having to think about all the major concepts and components which make up the

solution to this case study. Having a significant amount of the components that I intend to re-create be already tried and tested by other people in these literatures was invaluable. All credit goes to each respective author.

2.3.1 "DIY Hydroponic Garden w/ Arduino and IoT"

Source: https://www.youtube.com/watch?v=Ng7qDDH9Yqk

2.3.1.1 Summary

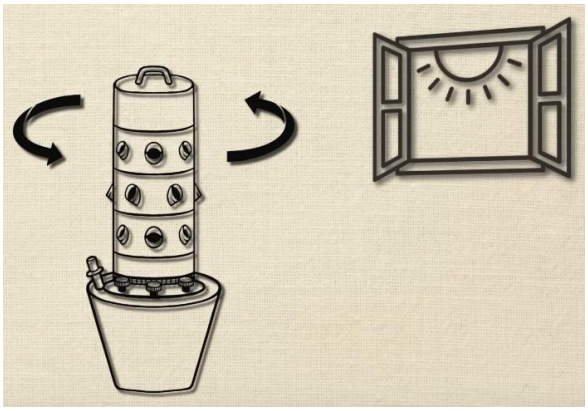


Figure 2.34 – The DIY Hydroponic Concept Sketch *(Programming Electronics Academy, 2021)*

This solution utilizes Arduino (actually a NodeMCU ESP8266 module, though an Arduino board with an external ESP8266 module also works) and a third-party IoT application called "Blynk" to develop a Hydroponic farm control system. The system is self-watering and be able to be monitored from afar via the internet. (Programming Electronics Academy, 2021).

This system utilizes a nutrient reservoir on the bottom, which can be pumped to water the crops. The system also rotates, as in a previous experiment, one side of the "Hydroponic Tower" died due to not getting enough sunlight, whilst the other bloomed. This rotation ensures all the crops get equal amounts of sunlight.

2.3.1.2 Comparison & Relevance

- This solution is purpose-built from the ground-up as an entirely new Hydroponic ecosystem. It uses its own hydroponic plantation design and is incompatible with existing hydroponic farms, which is the purpose of this case study.

- This solution is an “active” system, which has the ability directly manipulate conditions in the hydroponic farm, which is something that the solution of this case study is unable to do.
- This solution monitors only the “water level” of the hydroponic plantation and alerts the user should it get too low. It also constantly rotates the hydroponic planter “device”.
- It is fundamentally different from the solution designed in this case study. It’s primary purpose and emphasis is in automated *control*, as opposed to the case study solution’s sole purpose of automated *monitoring*.

2.3.1.3 Limitations

- It consumes a huge amount of power, as it has to *constantly* rotate the hydroponic planter device.
- It uses a third party “Blynk” IoT app, which subjects it to the application’s limitations.
- It is incompatible with the purpose of the solution built in this case study, which is to be compatible with existing yet-to-be-systemized hydroponic farms.
- It only monitors a single variable (water level).

2.3.2 “ESP8266 + Arduino + database – Control Anything from Anywhere”

Source: https://www.youtube.com/watch?v=6hpIjx8d15s&t=491s

2.3.2.1 Introduction

This solution utilizes Arduino and an ESP8266 module to control IoT-connected objects with a database. By using an internet connection, devices and sensors can be remotely controller to and from anywhere in the world, so long as it has access to internet. This is done by connecting sensors to an Arduino device, which constantly reads data from a database. We can control the sensors connected to the Arduino device by modifying data inside of the database (Electronoobs, 2019).

2.3.2.2 Comparison & Relevance

- This solution's data flow is reversed to that of the solution of this case study. Instead of reading data from the database which are sent from the sensor devices on the Arduino, this solution's Arduino device reads data from the database instead, and then performs pre-programmed acts based on the changes on the variables inside of the database.
- This solution is purely a proof-of-concept demonstration. It does not achieve anything specific other than to act as a demonstrator.
- The solution does not use any third-party applications.
- This solution has the greatest influence on the ideation process of this case study's solution.

2.3.2.3 Limitations

- As a pure demonstration prototype, it has no specific points of reference to be compared against where we can deduce limitations from.

2.3.3 "Sending data to thingspeak website using esp8266 Arduino Tutorial"

Source: https://www.youtube.com/watch?v=nMWwqcn7ofw

2.3.3.1 Introduction

This solution utilizes Arduino and an ESP8266 (ESP-01) module to a database on a website called "ThingSpeak". ThingSpeak is a specially designed website for IoT, and by using a unique API key, we can read and write data into the website (DPV Technology, 2019). Basically, ThingSpeak acts as a database emulator, that should be easier for IoT devices to connect to compared to contemporary equivalents. This solution utilizes the ESP-01 module's AT command-set to send data via GET URL to ThingSpeak.

2.3.3.2 Comparison & Relevance

- The method of data transmission used in this solution is similar to the one used in **2.3.1**, and is the most linearly comparable method to the one used in this case study's solution.

- Instead of an independent database used by this case study's solution, this solution uses a third-party web service (ThingSpeak). This approach makes it easier to set-up for inexperienced users.
- The use of the ESP-01 module's AT command-set is almost identical to the method of data transmission used by this case study's solution.

2.3.3.3 Limitations

- Because it uses a third-party web service (ThingSpeak), it is also subjected to the limitations of said service.
- It shares the limitations of this case study's solution of the Arduino Uno's incapability to provide full power to the ESP-01 module.

Chapter 3

Problem Analysis

3.1 Current Processes

3.1.1 Data Measurement & Collection

Figure 3.1 – The HI98301 TDS Meter

Currently, Just Hydroponics collects and measures data manually using hand-held tool. They primarily use the HI98301, but due to its prohibitively high cost, they only have a small number of these. They make up for the small number of these sensors with much cheaper (and much less accurate and reliable) models. They collect data once a day using these tools.

Collecting data only once a day creates its own problems. Should there be a sudden change in a hydroponic plantation's environment, by the time the staff realizes the undesirable conditions, it would be too late, and the damage would have been done. Just Hydroponics can certainly attempt to mitigate this issue by increasing data collection to several times per day, but scaling in this direction is not sustainable, as it would require a tremendous amount of man hours to meaningfully increase the frequency of data collection for relatively little gain. The amount of effort would also scale up linearly with the size of the farm, which could be changed to scale up logarithmically by introducing automation.

Even though the model illustrated above is the primary model that use, they still utilize other models. This lack of a standardized measuring tool can compromise data quality and consistency because different tools made by different manufacturers will have variations in their tolerances.

It’s important to note that with their current processes, Just Hydroponics is only capable of measuring nutrient ppm data. They do not measure other important variables such as temperature and CO_2 saturation. This creates a problem where they only see a partial “image” of their hydroponic plantation’s status.

3.1.2 Data Storage

Figure 3.2 – A Whiteboard Used for Storing Data

Currently, Just Hydroponics stores the hydroponic environmental data that they collected on whiteboards. They also have no reliable means to store historical data, as they have to wipe the whiteboards once they get full, and noting down the data into something like a notebook or data ledger is too laborious.

3.2 Problem Analysis

3.2.1 Data Vulnerability

The data being physically stored on whiteboards adds a factor of extreme data vulnerability. It does not take much to cause the valuable historical data stored on these whiteboards to be wiped, without any possibility of recovery or redundancy. All it takes is an employee wiping the whiteboards absent-mindedly or rain to fall on an unsheltered whiteboard to have the data be gone forever.

My migrating the data storage method to a cloud-based solution, data security and availability is greatly enhanced, leaving a vastly larger room for error and increasing data resilience.

3.2.2 Arduous Data Management

Storing data on whiteboards makes it extremely laborious to track all the data, making the task of data management a nightmare. Manually copying or transcribing the data onto a data ledger also increases the farm's labor workload.

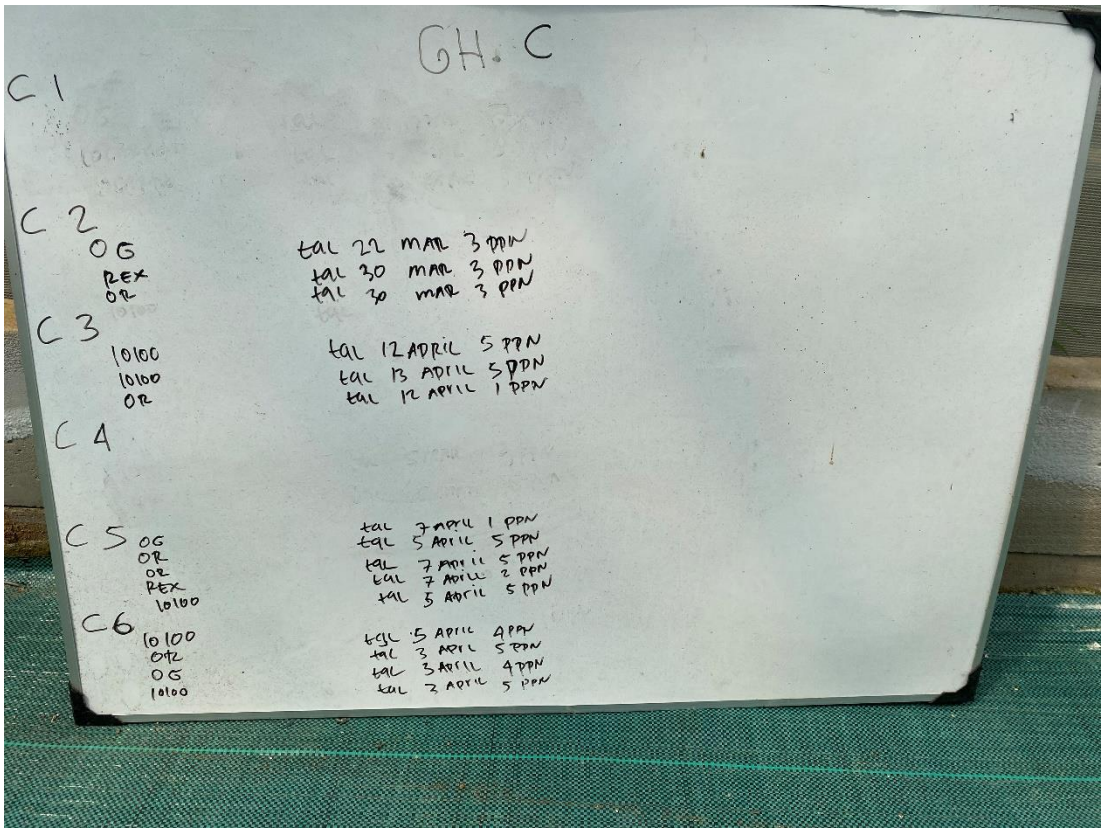


Figure 3.3 – Arduous Data Management

By having the ability to manage data on a cloud-based web application platform, the task of data management becomes much easier, allowing very few (if any at all) man-hours to be allocated to the task of data management.

3.2.3 Labor Intensive

The current process of data collection is done manually, using test kits for each individual hydroponic block. This is a labor-intensive process, requiring large

amounts of man-hours and consequently, a larger workforce. Combine this with the additional task of data management, and the amount of time taken up just to collect and maintain the data essential to the optimal operations of the hydroponic farm stacks up to a staggering amount.

Most of these processes can be automated by introducing an Arduino-based monitoring system. Because the Arduino sensors communicate with the database automatically, the farm staff only needs to make sure that the sensors are operating correctly and do the occasional maintenance. The task of data collection would be rendered completely redundant. Data maintenance becomes far easier and takes only a miniscule portion of how much time it would have taken if done manually.

3.2.4 Reduced Accuracy & Inconsistency

The manual nature of the current data collection process adds inherent inaccuracies and inconsistencies in the collected data due to the differences of how each employee might interpret the data. There is also the added factor of human error, where an employee might make a mistake during data input and transcription.

Because of the highly important nature of the data being collected, any mistakes may cause improper adjustments to the hydroponic system, causing suboptimality and imbalances that negatively impact the hydroponic system.

By using automated electronic sensors, we can reduce or outright eliminate most of the factors that are causing inaccuracies and inconsistencies. These sensors can also be calibrated to a single standard and are not affected by human error which can improve accuracy and ensures that the data collected is consistent.

3.2.5 Delayed Response

Manual data collection, though done frequently, cannot be done frequently enough to account for minute changes in the hydroponic system. The data is usually collected on a schedule, which causes a response delay should anything go wrong in the hydroponic system. These delays in response may significantly reduce the

optimality of the hydroponic system which could have been prevented by simply being notified of the problem sooner.

In periods of expected abnormalities, the data can be collected much more frequently, but this is not sustainable in the long term, as manual measuring utilizes a lot of single-use test kits and are physically laborious to do and keep track of.

On the other hand, an automated system can collect data with much higher frequency, require little to no human input, is reusable, and can even be configured to alert its users of any anomalies should it be desired. The data collection frequency can be changed by simply modifying a variable and does not incur extra costs or administrative overheads.

3.2.6 Inefficient Allocation of Financial Resources

All the problems above combined cause an inefficient allocation of financial resources, where money is being spent to maintain things that really shouldn't require maintenance and to put out fires that shouldn't have been allowed to light in the first place.

The currently difficult and laborious practices of data management and manual data collection take up manpower that could have been better spent somewhere else. The error-prone nature of processes that are done manually also take up resources to fix. Delays in response to undesirable environmental changes to the hydroponic system also take up resources to revert the unwanted changes.

All these issues are solvable by implementing an automated system. Automation lowers maintenance costs, increases reliability, vastly decreases the workload on the farm staff, are more reliable and consistent, and keep monitoring around the clock.

3.3 Empathy Map

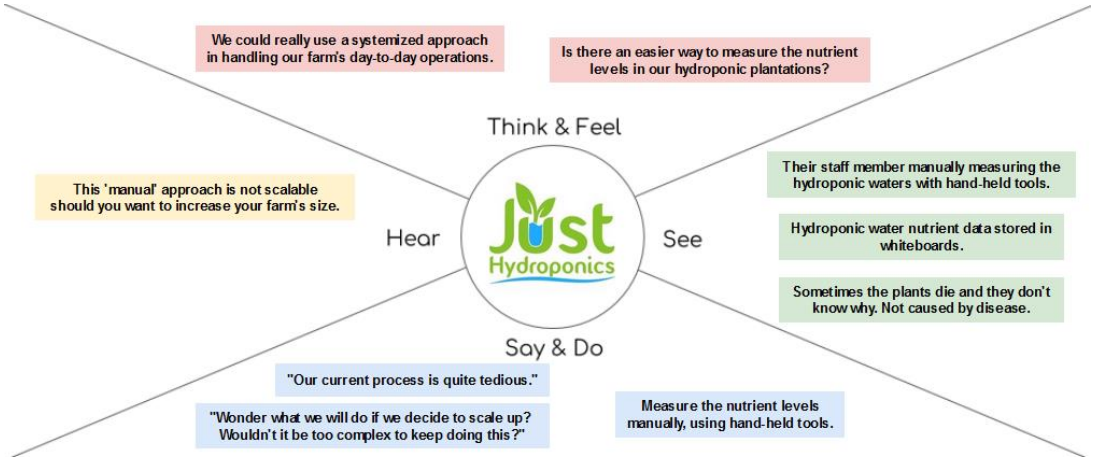


Figure 3.4 – Empathy Map

This is the empathy map, which highlights our analysis of the user (Just Hydroponics). It analyzes what the user thinks, feels, hears, sees, says, and does. From this analysis, we can deduct a "persona" of the user and tailor the solution to suit their needs.

3.4 Value Proposition Canvas

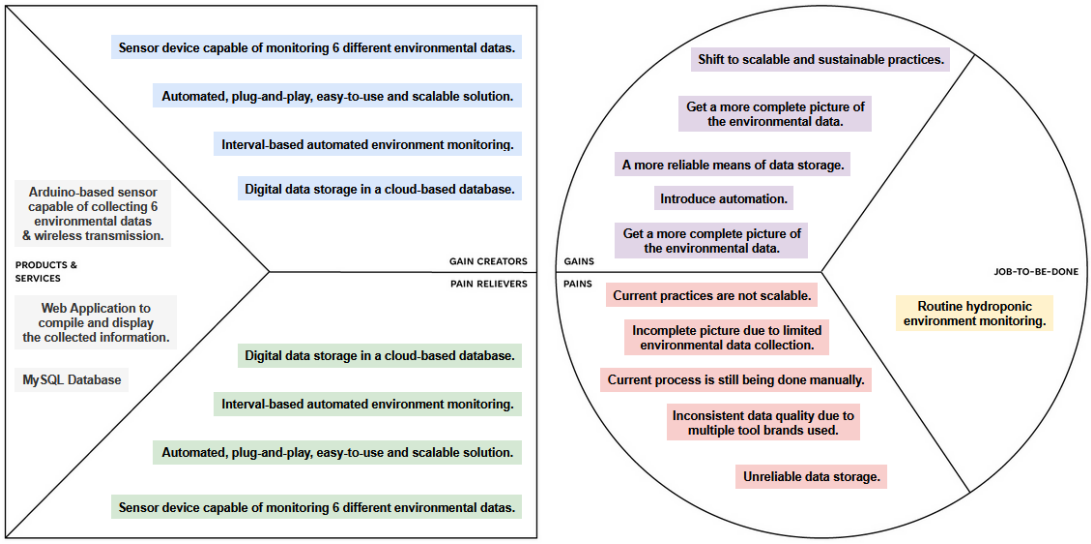


Figure 3.5 – Value Proposition Canvas

This is the Value Proposition Canvas (VPC). It highlights the pains, gains, and jobs-to-be-done of the target persona (Just Hydroponics) and how our solution can solve their pains, create their desired gains, and the products and services that are meant to do that.

The pains reflect the gains and vice versa, they are effectively one and the same, as the problems faced by Just Hydroponics consistently revolve around their lack of scalability. This is reflected in the same way on our gain creators and pain relievers, which also mirror each other.

Chapter 4

Solution Design

The solution is divided into two major components. The first component is the physical Arduino-based sensor. The second component is the web application which allows for the information collected by the sensor to be displayed to the user.

4.1 Arduino

4.1.1 Architecture Overview

The Arduino-based sensor is responsible for collecting information via an array of five sensors, which are the DS18B20, DHT11, MH-Z19B, PH-4502C, and DFRobot TDS Meter, each of them collecting temperature, humidity, CO_2, pH value, and nutrient ppm respectively.

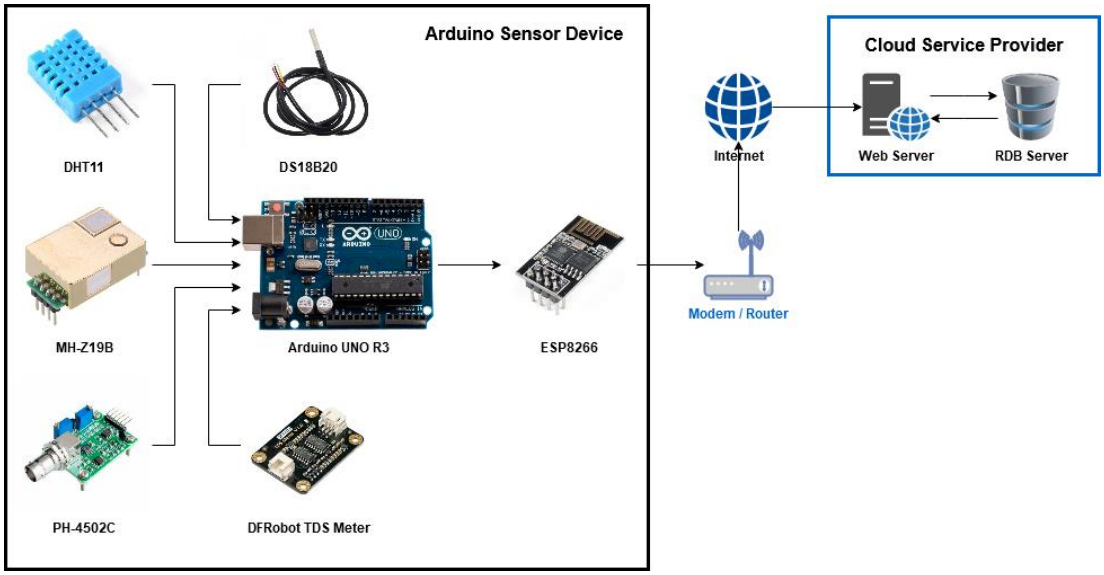


Figure 4.1 - Arduino System Architecture

The data collected by the sensors will then be compiled and formatted into a string URL which adheres to the PHP $_GET superglobal URL pattern. The data will then be sent into the internet via the ESP8266 Wi-Fi module.

This action is performed continuously with a configurable delay so long as the device is powered, and nothing goes wrong in the process cycle. This is called the “Loop Cycle”, which is explained in the point below.

4.1.2 Loop Cycle

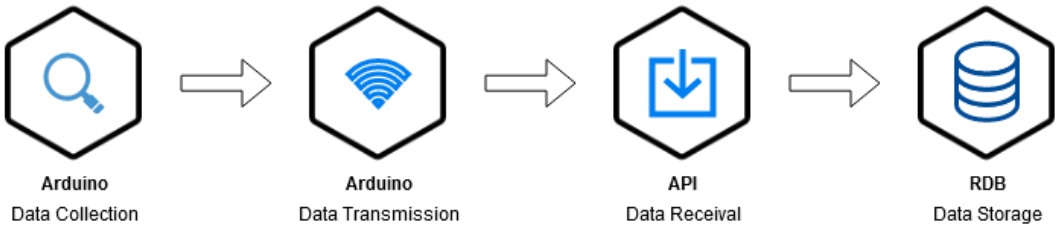


Figure 4.2 – Process Cycle

The web application contains an API, which allows it to receive data transmissions from the Arduino via internet URL. The API acts as a passive listener, which is triggered only when the Arduino attempts to connect to the server to transmit data.

4.1.3 Data Collection Frequency

The data collection frequency is **configurable**. Naturally, higher collection frequencies are better but may not be realistic if the user is unwilling to pay a premium for the increased storage space. According to General Hydroponics (2023), data should be collected at least once every day, and when in doubt of the stability of your hydroponics system, once every hour. Inferring from that source, I would advise a data collection frequency between once every half hour to once every hour.

4.1.4 Unit Cost

Table 4.1 – Unit Cost

Item	**Unit Cost**	**Total Cost**
Arduino Uno R3	Rp 120.500	Rp 120.500
Jumper Cables	Rp 8.800 (x4)	Rp 35.200
ESP-01 Wi-Fi Module	Rp 15.000	Rp 15.000
Small Breadboard	Rp 7.100	Rp 7.100
DHT11	Rp 10.800	Rp 10.800
DS18B20	Rp 9.775	Rp 9.775
Resistor 4.7k Ohm 1/2W	Rp 400 (x2)	Rp 800

MH-Z19B	Rp 574.500	Rp 574.500
DFRobot Analog TDS Sensor	Rp 227.400	Rp 227.400
PH-4502C + pH Probe	Rp 265.600	Rp 265.600
	Final Total	**Rp 1,039,275**

4.2 Web Application

4.2.1 Architecture Overview

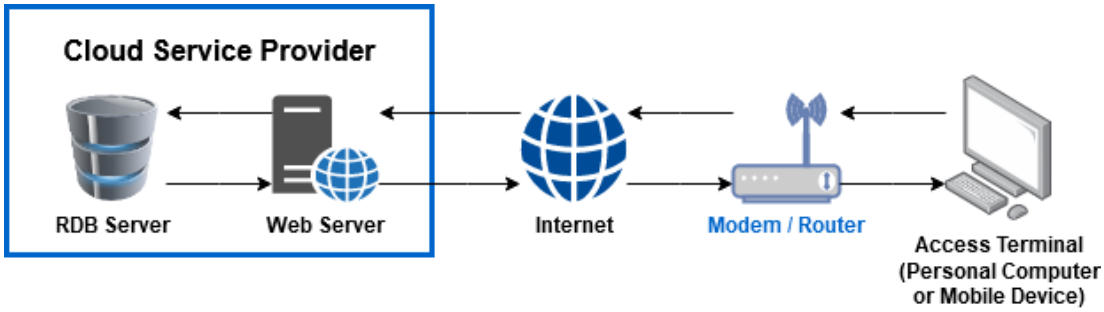


Figure 4.3 – Web Application System Architecture

The system requires an active internet connection. The internet connection used should be a basic, wireless capable connection with little to no firewall limitations or additional login requirements past the basic SSID and password combination, as the Arduino device is incapable of independently connecting to a physical LAN port or fulfilling any additional security requirements.

The simple nature of the web application should allow for it to be deployed almost anywhere. The very-low bandwidth requirements allow for high connectivity tolerances. Theoretically, it should be able to be deployed on almost any third-party cloud service provider, or it could even be hosted locally, should the local internet connection have a dynamic IP address and is capable of port-forwarding. If the cloud service provider features additional security layers, it should be disabled as in its current form, the Arduino device's design does not provide it with the capability for authentication and would likely be flagged as a spam.

4.2.2 Class Diagram

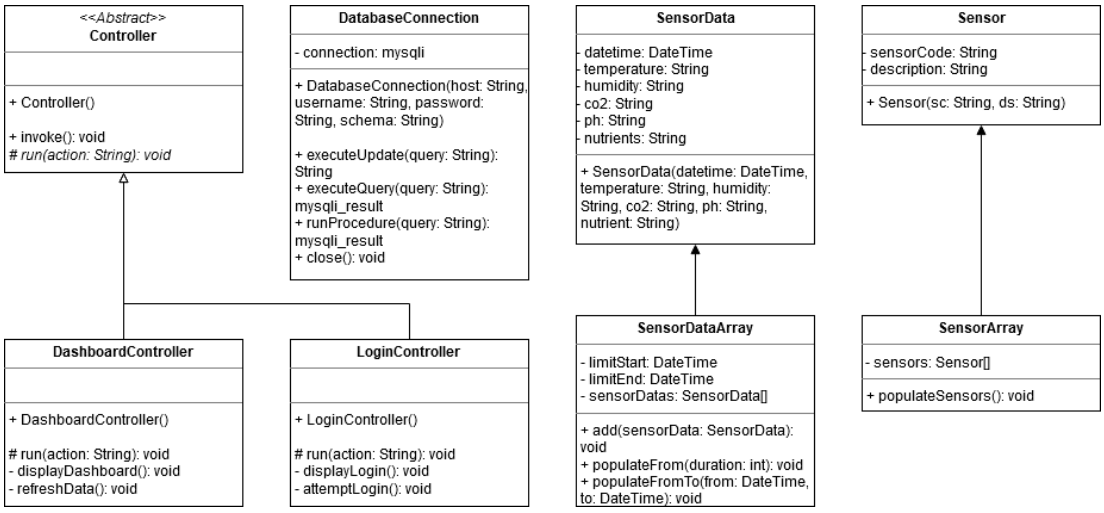


Figure 4.4 – Class Diagram

In its current implementation, the Class Diagram has **eight** classes. Due to the object-oriented nature of PHP, each of these classes serves a distinct purpose. The classes are:

- **Controller**
  - This is an abstract class. It serves as an abstraction layer for controller classes.
- **DashboardController**
  - As the name suggests, this class is a controller, which inherits the abstract *Controller* class. It controls the main page of the web application, which is the dashboard, which displays the data collected by the Arduino sensor device.
- **LoginController**
  - The login controller, which inherits the abstract *Controller* class. It controls the login page, although much of its functionality is only a placeholder, because a “real” authentication system hasn’t been made yet.
- **DatabaseConnection**
  - This class was created to handle most of the database connections and queries behind a layer of encapsulation.

  - This class also stores the connection object within its properties, so any queries can be performed by simply calling class's *executeUpdate()* or *executeQuery()* functions.

- **SensorData**
  - This class was created to classify, unify, and encapsulate the sensor data into a single object. This shortens the overall code length and improves code readability.
- **SensorDataArray**
  - This class was created specifically to retrieve an array of SensorData within a defined time frame.
  - This process would normally take approximately two dozen lines of code. By encapsulating this process behind a function within this class, overall code length is shortened, and readability is improved.
  - This class also acts as a wrapper to the SensorData class, capable of populating itself from the database.
- **Sensor**
  - This class was created to classify, unify, and encapsulate information on the sensor devices into objects.
- **SensorArray**
  - This class was created specifically to retrieve the array of Sensor, which covers all the sensors that have been registered (in the database) and used.
  - This class acts as a wrapper to the Sensor class, capable of populating itself from the database.

4.2.3 Design Pattern

The Model View Controller (MVC) design pattern is the basis of the web application's program structure. In the previous part, I explained the application's Model and Controller classes, and an auxiliary class for the database. The application has two View pages, which are *dashboard.php* and *login.php*. You'll be able to see how these two pages look like in the **User Interface** part. So, in summary, this

program has 4 Model classes, 2 Controller classes (with 1 serving as an abstraction layer), and 2 Views.

The process flow of the application starts at the *index.php* file, which invokes the router file in the *routes* subdirectory. The router file then calls upon and invokes on the desired controller. The only data passed on by the router is the controller's name itself. All other relevant data are passed through superglobals, which are either $_GET or $_POST. These superglobals will be directly read by the invoked controllers instead of passed through by the router.

After getting invoked, the controllers each have a *run()* function, which contains a switch logic, which is responsible for choosing what action the controller takes. The switch logic and the *run()* function which contains it accepts a string $action parameter, which is then passed into the switch. If no action matches the available criterias, the switch logic will perform its default action. In the case of the *DashboardController*, this default action is to display the dashboard. The $action string is passed via PHP's $_GET superglobal variable, which is then detected by the controller.

The actions performed by the controller involves the use of Model classes for processing data and performing more detailed actions. The resultant data is then displayed the View page for the user to consume and react to.

4.2.4 Use Case

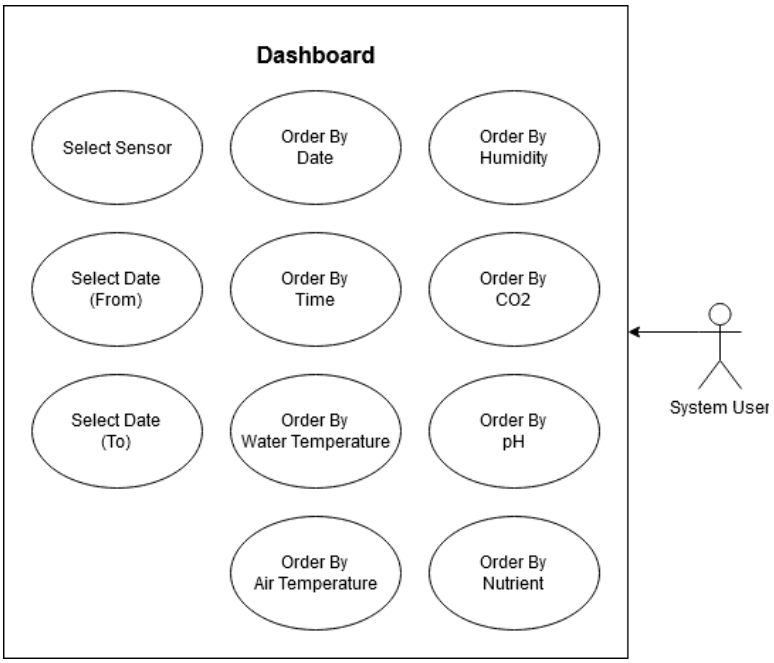


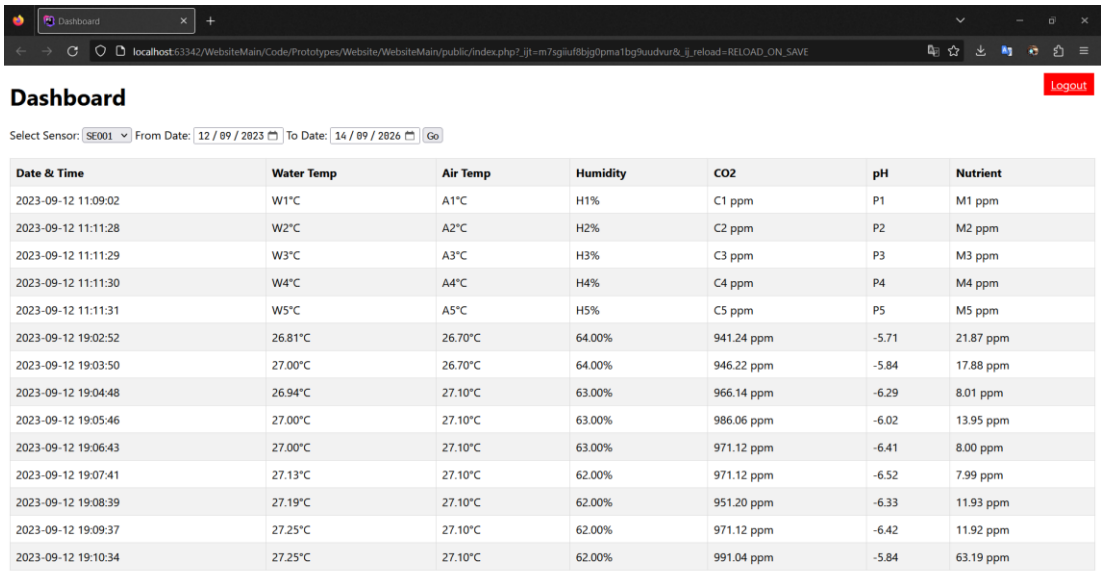
Figure 4.x – Use Case Diagram

This use case diagram depicts the interactions between the user and the system. It illustrates the range of features and possible actions available to the user. For simplicity's sake, the system's users are not differentiated. The simple nature of the system means that there is only a single access level and therefore only a single system user is depicted in the image. The system's simple nature is represented by the small number of use cases inside of the diagram above, which are:

- **Select Sensor**. This is a planned feature of the web application UI for the sake of expandability. It will allow the user to select the sensor which data they want to see. It's not used in this case study's web application as there is only one prototype sensor device.
- **Select Date (From)**. This is a feature which allows the user to display data from a selected date range. This feature allows the user to select the timeframe start date. It works in tandem with the *Select Date (To)* feature.
- **Select Date (To)**, This is a feature which allows the user to display data from a selected date range. This feature allows the user to select the timeframe end date. It works in tandem with the *Select Date (From)* feature.

- **Order by Date**. This feature allows the user to order the data entries by the date values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by Time**. This feature allows the user to order the data entries by the time values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by Water Temperature**. This feature allows the user to order the data entries by the water temperature values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by Air Temperature**. This feature allows the user to order the data entries by the air temperature values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by Humidity**. This feature allows the user to order the data entries by the humidity values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by CO_2**. This feature allows the user to order the data entries by the CO_2 values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by PH**. This feature allows the user to order the data entries by the PH values. Clicking this feature multiple times will cycle between ascending and descending.
- **Order by Nutrient**. This feature allows the user to order the data entries by the nutrient values. Clicking this feature multiple times will cycle between ascending and descending.

4.2.5 User Interface



Date & Time	Water Temp	Air Temp	Humidity	CO2	pH	Nutrient
2023-09-12 11:09:02	W1°C	A1°C	H1%	C1 ppm	P1	M1 ppm
2023-09-12 11:11:28	W2°C	A2°C	H2%	C2 ppm	P2	M2 ppm
2023-09-12 11:11:29	W3°C	A3°C	H3%	C3 ppm	P3	M3 ppm
2023-09-12 11:11:30	W4°C	A4°C	H4%	C4 ppm	P4	M4 ppm
2023-09-12 11:11:31	W5°C	A5°C	H5%	C5 ppm	P5	M5 ppm
2023-09-12 19:02:52	26.81°C	26.70°C	64.00%	941.24 ppm	-5.71	21.87 ppm
2023-09-12 19:03:50	27.00°C	26.70°C	64.00%	946.22 ppm	-5.84	17.88 ppm
2023-09-12 19:04:48	26.94°C	27.10°C	63.00%	966.14 ppm	-6.29	8.01 ppm
2023-09-12 19:05:46	27.00°C	27.10°C	63.00%	986.06 ppm	-6.02	13.95 ppm
2023-09-12 19:06:43	27.00°C	27.10°C	63.00%	971.12 ppm	-6.41	8.00 ppm
2023-09-12 19:07:41	27.13°C	27.10°C	62.00%	971.12 ppm	-6.52	7.99 ppm
2023-09-12 19:08:39	27.19°C	27.10°C	62.00%	951.20 ppm	-6.33	11.93 ppm
2023-09-12 19:09:37	27.25°C	27.10°C	62.00%	971.12 ppm	-6.42	11.92 ppm
2023-09-12 19:10:34	27.25°C	27.10°C	62.00%	991.04 ppm	-5.84	63.19 ppm

Figure 4.6 – User Interface, Dashboard

This is the initial design for the user interface. The interactions between the user and the system are reflected by the use case subchapter above. The user interface is designed to display 8 columns of data. The names of each of these columns should make self-explanatory the nature of the data that they display.

- The **Date** column displays the date of data retrieval. The date data is not retrieved from the sensor, and instead relies on the system clock of the database system. It is retrieved when the database receives data insertion instructions from the Web API. Should there be any accuracy issues, it is the database server that should come under scrutiny.
- The **Time** column displays the time of data retrieval, or more precisely, the exact time the data is received and inserted into the database. The data is retrieved in-system by the database when it receives instructions to insert data from the Web API.
- The **Air Temp** column displays the air temperature, retrieved from the DHT11 sensor.
- The **Water Temp** column displays the hydroponic water temperature, retrieved from the DS18B20 sensor.
- The **Humidity** column displays the air humidity levels, which are retrieved from the DHT11 sensor.

- The **CO_2** column displays the CO_2 levels in the air, which are retrieved from the MH-Z19B sensor.
- The **pH** column displays the pH levels in the hydroponic water, which are retrieved from the PH-4502C module along with its sensor probe.
- The **Nutrient** column displays the measured TDS (Total Dissolved Solids) levels in ppm (parts per million). This value is retrieved from the DFRobot TDS Meter sensor kit.

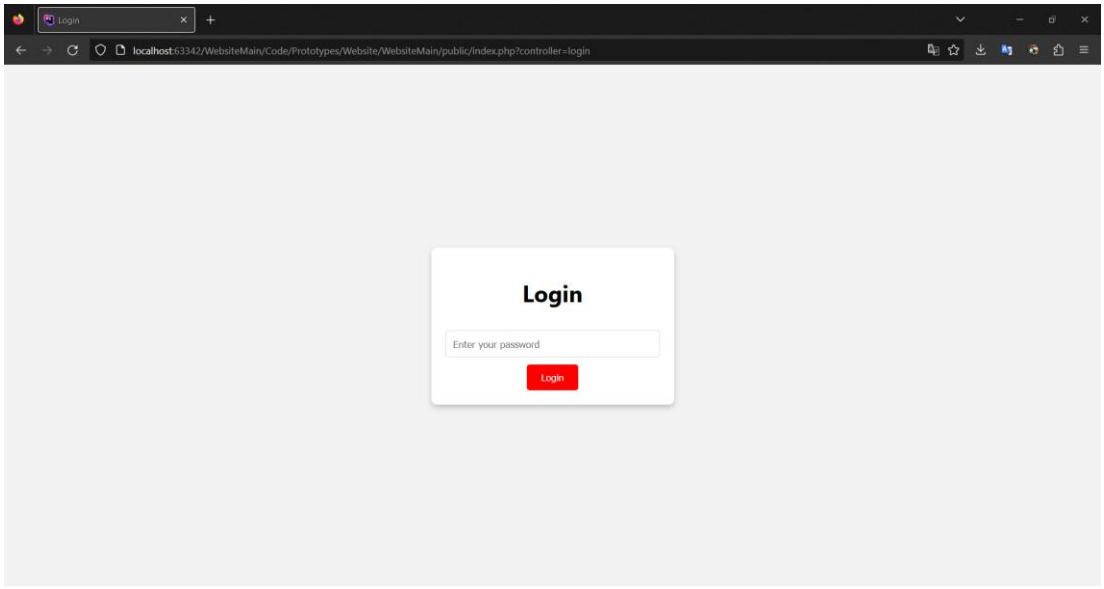


Figure 4.7 – User Interface, Login Page

This is a placeholder login page which I have made for demonstration purposes. Currently, the page is not fully functional, as no authentication system has been developed for it. In this login page, you only need to enter a password (which can be anything, so long as the field is not empty) and press login, and you will be redirected to the dashboard.

4.2.6 Database

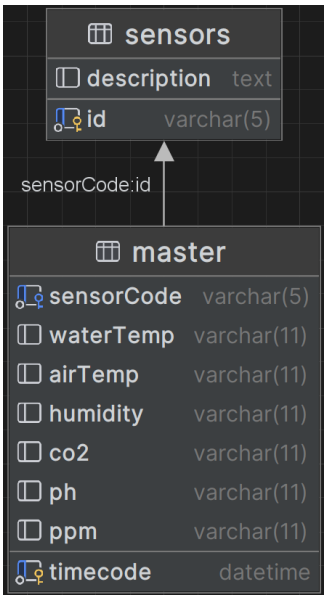


Figure 4.8 – The Database Structure

The database is a simple design. It consists of only two tables: the sensor table and the master table. The sensor table contains information on the physical sensors. The master table stores all the data collected by the sensors, it also has a foreign key that references the sensors table, to identify which data is collected by which sensor.

4.2.7 Arduino API

Figure 4.9 – Superglobal URL Pattern

The API serves as a receiver layer for data sent by the Arduino. It is integrated directly into the web application, but within a different directory (in /arduino instead of /public).

Access to the API is relatively simple. You only need to know the host, and directly access the *api.php* file within the /arduino subdirectory. The data should then be passed in PHP's standard $_GET superglobal format, with '`variable=value`' and each separated by the '`&`' sign.

4.3 Cloud Deployment

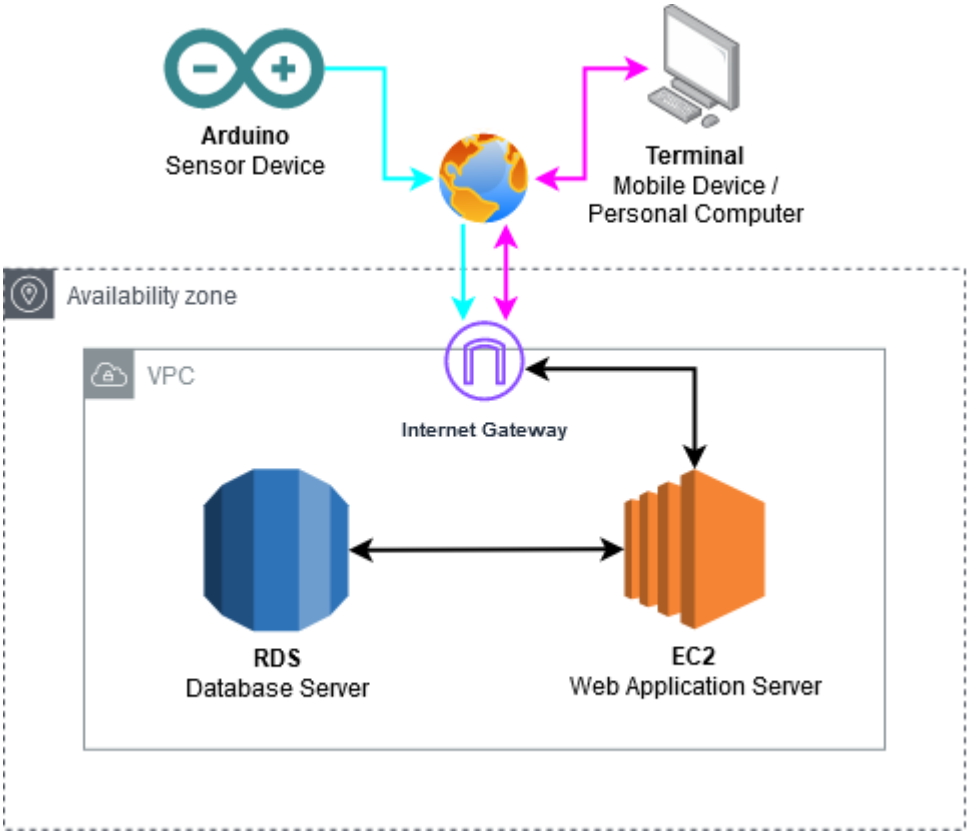


Figure 4.10 – AWS Cloud Architecture

The figure above shows the implementation plan of this solution into AWS. The diagram is relatively simple, which suits our use case and situation perfectly.

Depicted in the diagram, the “system” involves two separate connections over the internet from the Arduino and whatever internet-enabled personal device will be used to access the web application. The connection from the Arduino, which is highlighted in blue, is a one-way connection. The Arduino device will only be able to transmit data. The connection from and to the *terminal*, highlighted in purple, will be two ways, as it will make a request to the server and the server will send back the requested data.

As depicted in the diagram, under “normal” operating conditions, the database server should have absolutely no contact with either the user or the sensor device. Any connections and communication with the database will be handled the by web application. However, for the sake of simplicity (and at the expense of security), *public internet access* on the database server is enabled (which means there are no firewalls blocking access to it and you can connect to it directly if you have

the correct username and password) to make it easier for me to debug the database. In order to achieve this, the RDS database is placed in the same VPC as the EC2 server and (as aforementioned) have its public accessibility enabled, which allows me to connect to and manage the database directly from tools such as DataGrip, DBeaver, and MySQL Workbench.

4.4 Testing Plan

Due to the relatively simple and small scale of this case study, the most efficient and effective method of testing is **manual testing**. Manual testing will be used for both the Arduino device and the web application. Manual testing is chosen because automated testing would be inefficient and a waste of time on such a small-scale web application.

For the Arduino device, the manual testing would be divided into two, which are unit testing and integration testing. Due to the Arduino device having multiple modules, each module will be independently tested during **unit testing**. Then, after the final version of the Arduino has been built according to the final architecture schematics, an **integration testing** will be carried out. This is done because whilst the sensor devices themselves might work individually; this may not be the case when all of them are combined.

For the Web Application, the manual testing will similarly be divided into **unit testing** and **integration testing**. The web application is largely divided into two components, which are the API, and the web application interface. The database itself will not be tested, as there is little point in testing a simple two-table database with little to no rules and constraints. Unit testing for the API and the web application interface will of course involve the database, though the database itself is not the test subject. The API requires the database to insert data into, and the web application interface requires the database to read data from.

At the end, there will be an **end-to-end testing**. This part will be divided into three parts, which are: (1) **basic acceptance testing**, (2) **multi-device testing**, and (3) **overall reliability testing**. These tests must be completed in sequential order, and each stage must succeed before moving onto the next. In basic acceptance testing, we will simply combine all the solution's components and test to see if it works or not.

After the solution passes basic acceptance testing, it will need to pass through multi-device testing, where at least two devices will simultaneously be periodically send data into the database through the API. Finally, if it passes the second test, it will have to go through the overall reliability testing, where at least two devices will be left powered on and operational for long periods of time (10+ hours). Then I will measure the frequency and severity of deviancies from the desired data collection interval. If the interval pattern closely resembles the desired pattern, then the test will be a success. If the interval pattern deviates frequently and by large margins, then the test will be a failure.

Chapter 5

Solution Implementation

5.1 Arduino

5.1.1 Architecture

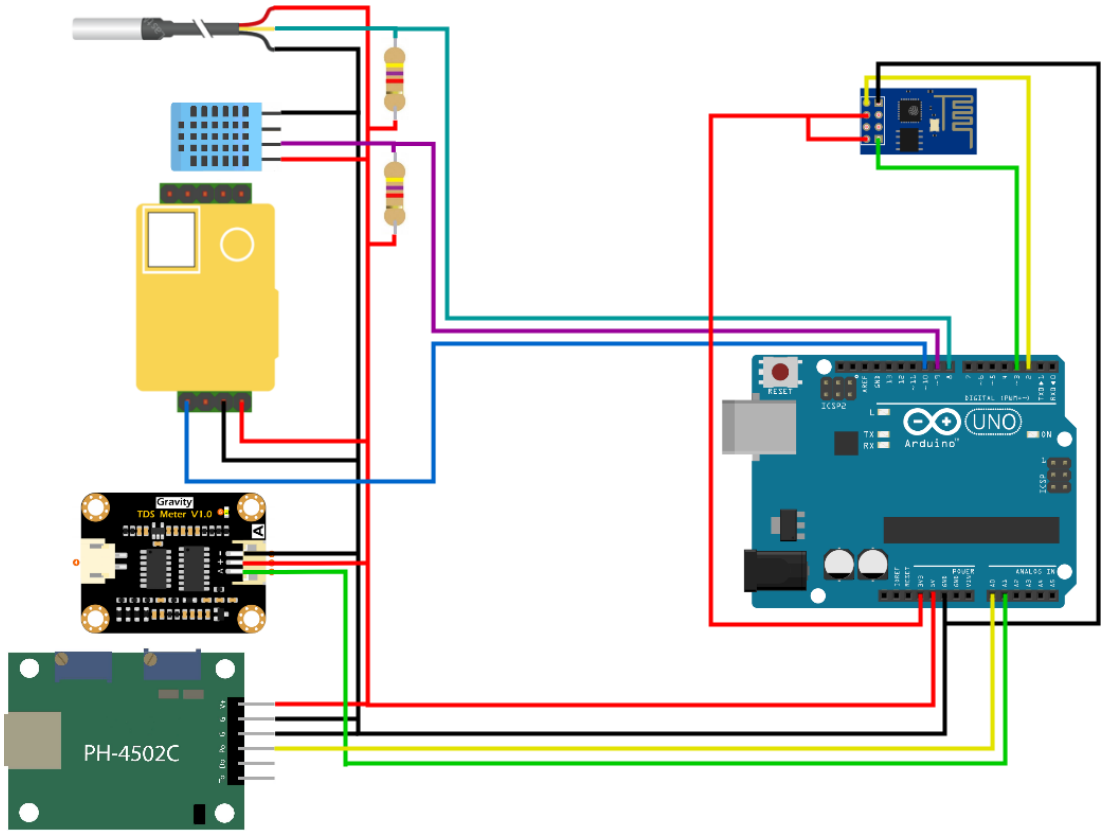


Figure 5.x – Schematic of the Arduino Prototype

The figure above represents the Arduino's architecture. The real design would be slightly more complicated and less elegant due to the need for a breadboard. The details are as follows:

- The 3.3V power supply is used to power only the Wi-Fi module.
- The 5V power supply is used to power all five sensor modules.
- Data collected from the pH module (PH-4502C) is collected via the Analog port A0.
- Important to note that the PH-4502C has two ground pins. Both should be connected to the Arduino's ground pin.
- Data collected from the TDS module (Gravity TDS Meter) is collected via the Analog port A1.

- Data collected from the water temperature probe (DS18B20) is collected via digital port 8.
- The data port for the DS18B20 needs to be provided with power from the 5V line, limited with a 4.7k Ohm resistor.
- Data collected from the air temperature and humidity sensor (DHT11) is collected via digital port 9.
- The data port for the DHT11 needs to be provided with power from the 5V line, limited with a 4.7k Ohm resistor.
- Data collected from the CO_2 module (MH-Z19B) is collected via PWM (~) digital port 10.
- The Wi-Fi module (ESP-01)'s RX and TX ports should be connected to the Arduino's digital pins 2 and 3. Do not use digital pins 0 and 1 as these are reserved for hardware serial.

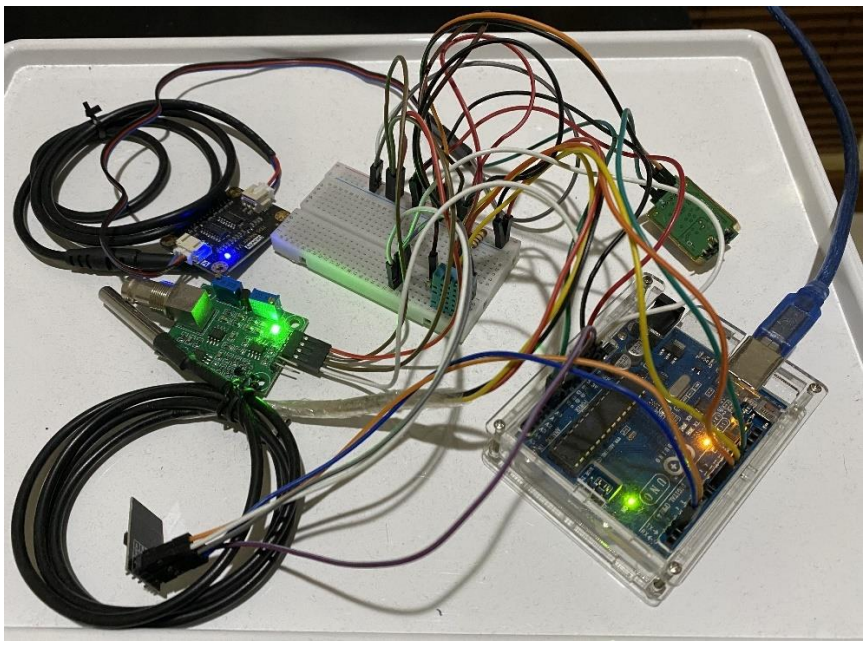


Figure 5.2 – The Arduino Prototype

The image above is the prototype device that I have assembled, according to the architecture diagram. As a prototype, the design is not refined and obviously not ready for commercial use. It serves primarily as a proof of concept, that this design can work.

5.1.2 Code Structure

5.1.2.1 File Structure

Figure 5.3 – The File Structure

For the sake of readability and maintainability, I have structured the Arduino code into 7 separate files. Generally, they are divided by their roles, and are structured as follows:

1. ArduinoMain
    a. This file contains the "main" code, where the `void setup()` and `void loop()` functions are located. This file also contains the code responsible for importing libraries. The code in this file acts as the main "executor" of the program, which calls functions from the other files.
2. 00_WiFi
    a. This file contains code relevant to the operations of the Wi-Fi module (ESP-01).
    b. It defines the module's serial pins, instantiates the SoftwareSerial object to represent the Wi-Fi module, and also contains code responsible for setting up the module, and sending commands to the module.
3. 01_WaterTemperature
    a. This file contains code relevant to the operations of the water temperature probe.
    b. It defines the module's single serial data pin and instantiates the OneWire and DallasTemperature objects crucial to the operations of this sensor module.
    c. It also contains the code for setting up the sensor module, and to retrieve water temperature data from the probe module.
4. 02_Humidity
    a. This file contains code relevant to the operations of the humidity sensor module.

   b. It defines the modules single serial data pin and the DHT type (‘DHT11’) and instantiates the DHT object crucial to the operations of this sensor module.
   c. It also contains the code for setting up the sensor module, and to retrieve either air temperature or humidity data from the sensor module.
5. 03_CO2
   a. This file contains code relevant to the operations of the CO2 sensor module.
   b. It defines the module’s single serial data pin. It also defines the sensor’s floating-point constant to control the sensor’s range (5000.0).
   c. It also contains the code for setting up the module, and to retrieve CO2 data from the sensor module.
6. 04_PH
   a. This file contains code relevant to the operations of the PH sensor module.
   b. It defines the module’s single analog data pin.
   c. This sensor does not require any setup.
   d. It also contains the code for retrieving PH data from the module.
7. 05_Nutrient
   a. This file contains code relevant to the operations of the TDS meter sensor module.
   b. It defines the module’s single analog data pin. It also instantiates the sensor’s GravityTDS object, which is crucial to the operations of the sensor module.
   c. It also contains the code for setting up the module and to retrieve nutrient TDS data from the sensor module.

5.1.2.2 Program Logic & Loop

Upon clicking the “Upload” button, the Arduino IDE first compiles the code from all the files. The file structure that we’ve made does not matter at all, as after compilation, the code is then combined.

The first code to be executed are the definitions outside of the functions such as `#define`. Then the program enters the `void setup()` function, which then proceeds to call the different setup functions in the other code files, which then sets up all the modules and make them ready to use. After that, the program enters the `void loop()` function, where the program will loop endlessly unless instructed to stop or the device is powered down. Within the loop function, It will continuously (in sequence) call each sensor module to retrieve data, and then call on the Wi-Fi module to transmit the collected data via $_GET URL.

5.1.3 The Code

5.1.3.1 Main

The main code starts with importing the libraries. It also defines the sensor's unique sensor code. After that, it enters the setup function block, where the Serial begins, and then calls the setup functions for each module. There are several delays placed strategically to ensure the software's smooth operation.

Finally, the software then enters the loop function block, where the environmental variables are retrieved by calling their respective functions and then transmitted by calling the Wi-Fi module's send data function.

At the end of each loop iteration there is a delay. This is the delay that the user should configure to set the delay between data collections. Due to the fact that SoftwareSerial runs in parallel, the actual delay will vary. It will go faster should the Wi-Fi module happen to connect and send data quickly and vice versa. Therefor, it is unrealistic to expect the data to be collected and sent at perfect intervals. The user should instead have a target median interval instead, and the actual intervals should not be far off the targeted median.

```
#include "Arduino.h"
#include <SoftwareSerial.h>
#include "OneWire.h"
#include "DallasTemperature.h"
#include "DHT.h"
#include "MHZ19.h"
```

```
#include "EEPROM.h"
#include "GravityTDS.h"

#define SENSOR_CODE "SE001"

void setup() {
  Serial.begin(9600);
  delay(500);

  setupWifi();
  delay(500);

  setupWaterTemperature();
  delay(100);

  setupHumidity();
  delay(100);

  setupCO2();
  delay(100);

  setupPH();
  delay(100);

  setupNutrient();
  delay(100);
}

void loop() {
  delay(834);
  float waterTemp = getWaterTemp();

  delay(834);
  float airTemp = getAirTemp();

  delay(833);
```

```
  float humidity = getHumidity();

  delay(833);
  float co2 = getCO2();

  delay(833);
  float ph = getPH();

  delay(833);
  float nutrient = getNutrient(waterTemp);

  Serial.println("\n=-=-=-=-=\n");

  Serial.println("INFO: Begin data send process.");
  String instructions =
  (String) "GET /arduino/api.php?"
  + "sensorCode=" + SENSOR_CODE + "&"
  + "waterTemp=" + waterTemp + "&"
  + "airTemp=" + airTemp + "&"
  + "humidity=" + humidity + "&"
  + "co2=" + co2 + "&"
  + "ph=" + ph + "&"
  + "nutrient=" + nutrient;

  sendSensorData(instructions, (instructions.length() + 2));

  delay(5000);
}
```

5.1.3.2 Wi-Fi (ESP-01)

The Wi-Fi code starts with defining the module's RX and TX pins, and then it instantiates the SoftwareSerial object, where the values of the RX and TX pins are passed into it. The wifi name, password, host, and port constants are also declared here.

After that, a setup function is declared. In this function, the begin function is called on the SoftwareSerial, with a baud rate of 115200. After that, a series of command are sent to the module to reset the module, set it to station mode, connect it to Wi-Fi, and finally set the module to multiple connections mode.

The `sendSensorData()` function is then declared. This function is called on every iteration of the main code's `loop()` function. Inside of this function, a series of commands are sent to the Wi-Fi module. First, the module is instructed to connect to the server. Second, the module is given the size of the package to be transmitted. The package size is the instruction string's length, added by two to compensate for the endline and carriage return characters. Third, the package length is sent as a command to the module. Finally, as a failsafe, the module is informed that the instruction string has completed. Normally, this should not be necessary, as if you send the correct package size, the module should automatically recognize that the instruction string is complete. If the package size is too small, the data will not be sent. If the package size is too big, you will need to manually close it as is done in the fourth instruction.

Finally, the `runCommand()` function is declared. The function takes two string parameters. The first parameter is the command, and the second parameter is the expected response. When the command is invoked, it will send the command to the Wi-Fi module by calling `println()` on the Wi-Fi module's SoftwareSerial object. After that, there is a logic loop to try and find the expected keyword. Should the keyword be found, it will break out of the loop immediately. If a keyword is not found, then it will wait for 10 seconds before breaking out of the loop.

Unfortunately, because the program's Serial is running at 9600 baud, and the Wi-Fi module is running at 115200 baud, the responses from the SoftwareSerial given to the program's main Serial is garbled. This means that looking for the correct keyword in the response is a completely unreliable method, which is why I introduced the 10 second delay between instructions instead. The 10 second delay assumes that everything is working normally, which raises reliability concerns. Unfortunately, I have yet to find a better method to handle this problem.

```
#define PIN_WIFI_RX 2
#define PIN_WIFI_TX 3
```

```
SoftwareSerial wifi(PIN_WIFI_RX, PIN_WIFI_TX);

String wifiName = "MySSID";
String wifiPassword = "MyPassword";
String host = "My.Host.Here.Yes";
String port = "MyPort";

void setupWifi() {
  Serial.println("SETUP: WiFi...");
  wifi.begin(115200);

  delay(1000);

  Serial.println("WiFi: [AT+RST]");
  runCommand("AT+RST", "OK"); // Reset module.
  Serial.println("WiFi: [AT+CWMODE=1]");
  runCommand("AT+CWMODE=1", "OK"); // Set to station mode.
  Serial.println("WiFi: [AT+CWJAP=\"" + wifiName + "\",\"" +
wifiPassword + "\"]");
  runCommand("AT+CWJAP=\"" + wifiName + "\",\"" + wifiPassword +
"\"", "OK"); // Connect to WiFi.
  Serial.println("WiFi: [AT+CIPMUX=1]");
  runCommand("AT+CIPMUX=1", "OK"); // Set to multi-connection mode.

  Serial.println("WiFi: Ready!");
}

void sendSensorData(String inst, int len) {
  Serial.print("INSTR: ");
  Serial.print(inst);
  Serial.println();

  Serial.println("WiFi: [AT+CIPSTART=0,\"TCP\",\"" + host + "\"," +
port + "]");
  runCommand("AT+CIPSTART=0,\"TCP\",\"" + host + "\"," + port,
```

```
"OK");              // AT+CIPSTART=0,"TCP","192.168.18.117",80

  Serial.println("WiFi: [AT+CIPSEND=0," + (String) len + "]");
  runCommand("AT+CIPSEND=0," + (String) len, ">");
// AT+CIPSEND=0,102

  Serial.println("WiFi: SENDING PACKAGE!");
  runCommand(inst, "");
// GET
/arduino/api.php?sensorCode=1&waterTemp=ARD&airTemp=ARD&humidity=AR
D&co2=ARD&ph=ARD&nutrient=ARD

  Serial.println("WiFi: [AT+CIPCLOSE=0]");
  runCommand("AT+CIPCLOSE=0", "");

  delay(1000);
}

void runCommand(String command, String expectedResponse) {
  /* Send Command */
  wifi.println(command);

  /* Compile Response */
  boolean keyFound = false;
  String finalResponse = "";
  do {
    if(wifi.available()) {
      String currentResponse = wifi.readStringUntil('\n');
      currentResponse.trim();
      int nli = currentResponse.indexOf("\n");
      if(nli > 0) currentResponse.remove(nli);
      finalResponse += currentResponse;
//      Serial.println("WiFi: Current response: " +
currentResponse);

      if(finalResponse.indexOf(expectedResponse) > 0) {
```

```
        keyFound = true;
        Serial.println("WiFi: Keyword found!");
      }
    }
    else {
      Serial.println("WiFi: SoftwareSerial n/a. Keyword n/a. Assume
OK!\nWiFi: Delay 10s.");
      delay(10000);
      keyFound = true;
    }
    Serial.println("WiFi: (R) " + finalResponse);
    delay(200);
  } while(!keyFound);

  Serial.println("\n");
}
```

5.1.3.3 Water Temperature (DS18B20)

The water temperature sensor code begins with a pin slot definition. After that, OneWire and DallasTemperature objects are instantiated. The OneWire object takes the pin slot integer as parameter. The DallasTemperature object takes the OneWire object as parameter. After that, a setup function is declared for the sensor's setup process, and a data retrieval function is also declared to obtain data from the module.

```
#define PIN_WATERTEMP 8
OneWire temperature(PIN_WATERTEMP);
DallasTemperature sensorTemperature(&temperature);

void setupWaterTemperature() {
  Serial.print("Sensor: WT... ");

  sensorTemperature.begin();
  delay(50);
```

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

float getWaterTemp() {
  sensorTemperature.requestTemperatures();
  return sensorTemperature.getTempCByIndex(0);
}
```

5.1.3.4 Air Temperature & Humidity (DHT11)

The air temperature and humidity sensor code begin with a pin slot definition and a DHT type definition. After that, a DHT object which accepts the pin slot and DHT type as parameters are instantiated. Then, a setup function is declared for the sensor's setup process, and two data retrieval functions are also declared to obtain data from the module, one for air temperature and one for humidity.

```
#define PIN_HUMIDITY 9
#define DHT_TYPE DHT11
DHT sensorHumidity(PIN_HUMIDITY, DHT_TYPE);

void setupHumidity() {
  Serial.print("Sensor: Humidity... ");

  sensorHumidity.begin();

  Serial.println("Ready!");
}

float getAirTemp() {
  return sensorHumidity.readTemperature();
}

float getHumidity() {
  return sensorHumidity.readHumidity();
}
```

5.1.3.5 CO_2 (MH-Z19B)

The CO_2 sensor code begins with a pin slot definition. After that, a constant which defines the sensor's range is declared. Then, a setup function is declared for the sensor's setup process, and a data retrieval function is also declared. Due to the sensor's data being retrieved through PWM, returning a CO_2 ppm number requires the use of an algorithm, which is retrieved from https://iotspace.dev/arduino-co2-sensor-mh-z19-beispiel-und-sketch/. You can read more about this in the second chapter.

```
#define PIN_CO2 10

float sensorRange = 5000.0;

void setupCO2() {
  Serial.print("Sensor: CO2... ");

  pinMode(PIN_CO2, INPUT);

  Serial.println("Ready!");
}

float getCO2() {
  unsigned long pwmFreq = pulseIn(PIN_CO2, HIGH, 2000000) / 1000;
  float pulseRate = pwmFreq / 1004.0;
  float co2ppm = sensorRange * pulseRate;

  return co2ppm;
}
```

5.1.3.6 pH (PH-4502C)

The pH sensor code begins with an analog pin slot definition. After that, a data retrieval function is declared to obtain data from the module. Due to the analog method of data collection, the pH value must be converted from voltage. Additional information on this is on the second chapter. This sensor does not require a setup.

```
#define PIN_PH A0 // ANALOG

// ADC is 1024
// SAMPLES is 10

void setupPH() {
  // NO SETUP NECESSARY.
  Serial.println("Sensor: pH... Ready!");
}

float getPH() {
  int measurings = 0;

  for(int i = 0 ; i < 10 ; ++i) {
    measurings += analogRead(PIN_PH);
    delay(10);
  }

  float voltage = 5 / 1023.0 * measurings / 10;

  return calcPH(voltage);
}

float calcPH(float voltage) {
  return 7 + ((2.5 - voltage) / 0.18);
}
```

5.1.3.7 Nutrient (TDS Meter)

The nutrient TDS sensor code begins with an analog pin slot definition. After that a GravityTDS object is instantiated. After that, a setup function is declared for the sensor's setup process, and a data retrieval function is also declared to obtain data from the module. Calculating an accurate TDS value requires that the water temperature be known. If it is not known, then the accuracy of the obtained value

will be reduced. Hence, the water temperature value which has previously been retrieved by the water temperature probe is passed onto the TDS sensor module.

```
#define PIN_NUTRIENT A1 // ANALOG
GravityTDS sensorNutrient;

// ADC is 1024
// AREF is 5

void setupNutrient() {
  Serial.print("Sensor: Nutrient... ");

  sensorNutrient.setPin(PIN_NUTRIENT);
  sensorNutrient.setAref(5);
  sensorNutrient.setAdcRange(1024);
  sensorNutrient.begin();

  Serial.println("Ready!");
}

float getNutrient(float waterTemperature) {
  sensorNutrient.setTemperature(waterTemperature);
  sensorNutrient.update();

  return sensorNutrient.getTdsValue();
}
```

5.2 Web Application

5.2.1 Application Code Structure

5.2.1.1 File Structure

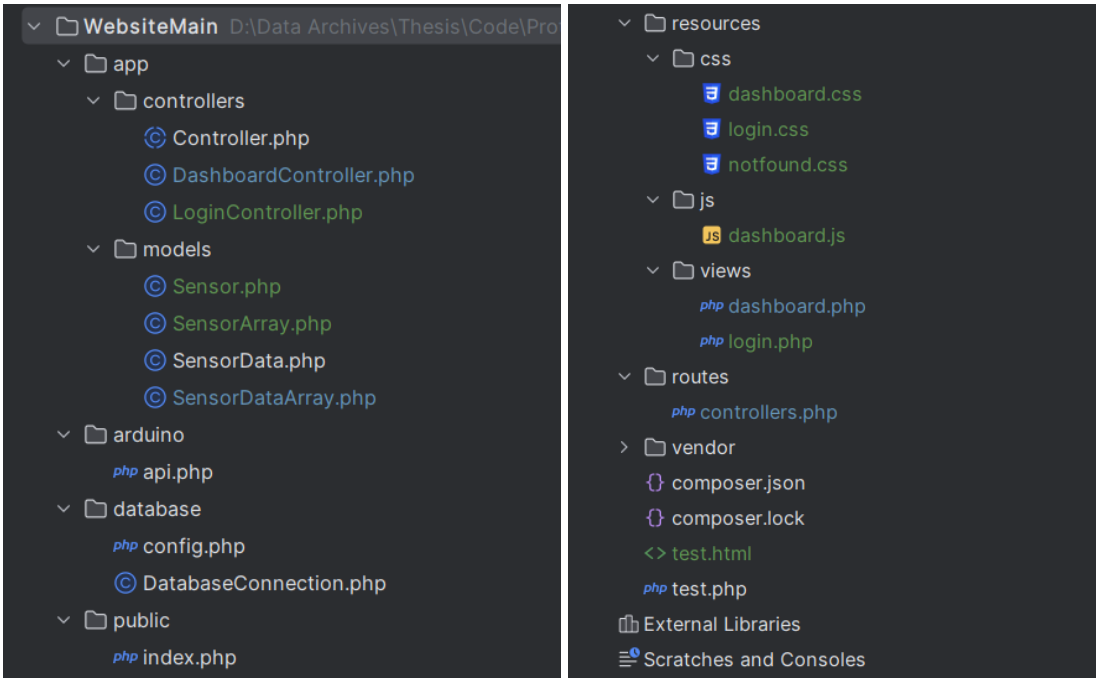


Figure 5.4 – The Web Application's File Structure

The figure above is a screenshot of the file structure taken directly from the IDE which I used to develop the web application (PHPStorm). The web application's file and folder structure was inspired by Laravel.

Within the **app** folder is located most of the application's logic, though I took some small liberties by putting the class responsible for handling database connections within the database folder. In the app folder, you can find the **controllers** and **models** folders. As their name suggests, these contain the application's controllers and models respectively. There are two controllers (DashboardController and LoginController), which adheres to the design pattern of the abstract Controller. There are four model classes, which are Sensor, SensorArray, SensorData, and SensorDataArray. SensorArray and SensorDataArray are wrapper model classes which are wrappers for the Sensor and SensorData model classes.

Within the **arduino** folder is located the API, which is responsible for accepting data sent by the Arduino device via $_GET URL.

Within the **database** folder lies the database configuration file and the DatabaseConnection class file. In retrospect, the DatabaseConnection class file should have been placed in the **app** folder, but changing it now is not worth the effort.

Within the **public** folder lies the index file, which is the application's "landing page" or starting location. The index is responsible for calling the router.

Within the **resources** folder lies the view pages and their supporting elements. The view pages themselves are located in the **views** subfolder. The **js** subfolder contains the view-pages' JavaScript code. The view-pages' CSS files are located inside of the **css** subfolder.

The **routes** folder contains the router (*controllers.php*) file. This file is responsible for the controller routing, which is responsible for deciding which controllers should be active and hence what pages are displayed.

The **vendor** folder is largely unused. It contains files for composer, which is not used, as the server is being hosted locally using Apache on XAMPP.

5.2.1.2 Program Logic

```php
<?php

require_once __DIR__.'/../routes/controllers.php';

$controller = check();
$controller->invoke();
```

Figure 5.5 – The Index File

The program logically starts in the *index.php* file within the *public* subdirectory, where it imports the routing *controllers.php* file, and obtains the controller to-be-invoked. After the controller object has been attached to the *$controller* variable, the controller is then invoked by calling the *invoke()* function.

```php
function check(): Controller {
    if(isset($_GET['controller'])) {
        $controller = $_GET['controller'];
    }
    else if(isset($_POST['controller'])) {
        $controller = $_POST['controller'];
    }
    else {
        $controller = 'default';
    }

    setcookie('session_controller', $controller, 0);
    return load($controller);
}
```

Figure 5.6 – Router, Check Function

```php
function load($controller) {
    switch($controller) {
        case 'dashboard':
            require_once __DIR__.'/../app/controllers/DashboardController.php';
            return new DashboardController();
        case 'login':
            require_once __DIR__.'/../app/controllers/LoginController.php';
            return new LoginController();
        default:
            require_once __DIR__.'/../app/controllers/DashboardController.php';
            return new DashboardController();
    }
}
```

Figure 5. – Router, Load Function.

Within the router (*controllers.php*) file are two functions used for the application's routing. The primary function in this router file is the *load()* function, which is responsible for loading a controller depending on the passed parameter. However, this function is not called directly by the index file. That job falls to the *check()* function, through which the *load()* function is routed. The *check()* function acts as a "wrapper function" for the *load()* function, which adds controller retention functionality through the use of cookies, which allows the website to remember pages and not reset to the main page at every action / click.

```php
abstract class Controller {
    no usages  2 overrides  Naughtless
    public abstract function __construct();

    1 usage  Naughtless
    public function invoke(): void
    {
        if(isset($_GET['action'])){
            $this->run($_GET['action']);
        }
        else if(isset($_POST['action'])) {
            $this->run($_POST['action']);
        }
        else {
            $this->run( action: null);
        }
    }

    3 usages  2 overrides  Naughtless
    protected abstract function run($action);
}
```

Figure 5.8 – Controllers Abstraction Layer

The controllers have an abstraction layer in the form of the abstract *Controller* class. Each controller has two primary functions, which are inherited from this abstract class, which are *invoke()* and *run()*. The *invoke()* function is generally unchanging, and usually directly inherited from this abstract class. The *run()* function is different for every controller and is therefor not implemented in the abstraction layer.

```php
protected function run($action) : void {
    switch($action) {
        case 'filterRefresh':
            $sensor = $_GET['sensor'];
            $from = new DateTime( datetime: $_GET['from-date'].' 00:00:00');
            $to = new DateTime( datetime: $_GET['to-date'].' 23:59:59');
            $this->displayDashboard($sensor, $from, $to);
            break;
        case 'logout':
            header( header: "location: index.php?controller=login");
            exit;
        default:
            $this->displayDashboard( sensor: null,  from: null,  to: null);
            break;
    }
}
```

Figure 5.9 – The DashboardController's run() Function

Although each controller has a different implementation of their *run()* function, they share and follow a common design pattern. They all contain a *switch* and the list of possible actions that can be performed by the controller.

Almost all the application's dataflow between the view pages, controllers, and models are done via GET or POST superglobals. POST superglobals are passed exclusively by forms and GET superglobals can be passed by either forms or directly through URL header manipulation.

5.2.2 Database & API

5.2.2.1 Database Structure

The database is a simple design, with only two tables. They are built to the specifications explained in chapter **4.2.6 Database**.

	timecode	sensorCode	waterTemp	airTemp	humidity	co2	ph	ppm
8	2023-09-12 19:02:52	SE001	26.81	26.70	64.00	941.24	-5.71	21.87
9	2023-09-12 19:03:50	SE001	27.00	26.70	64.00	946.22	-5.84	17.88
10	2023-09-12 19:04:48	SE001	26.94	27.10	63.00	966.14	-6.29	8.01
11	2023-09-12 19:05:46	SE001	27.00	27.10	63.00	986.06	-6.02	13.95
12	2023-09-12 19:06:10	SE002	69.42	69.42	69.42	69.42	69.42	69.42
13	2023-09-12 19:06:43	SE001	27.00	27.10	63.00	971.12	-6.41	8.00
14	2023-09-12 19:07:06	SE002	69.42	69.42	69.42	69.42	69.42	69.42
15	2023-09-12 19:07:41	SE001	27.13	27.10	62.00	971.12	-6.52	7.99
16	2023-09-12 19:08:02	SE002	69.42	69.42	69.42	69.42	69.42	69.42
17	2023-09-12 19:08:39	SE001	27.19	27.10	62.00	951.20	-6.33	11.93
18	2023-09-12 19:09:37	SE001	27.25	27.10	62.00	971.12	-6.42	11.92
19	2023-09-12 19:09:41	SE002	69.42	69.42	69.42	69.42	69.42	69.42
20	2023-09-12 19:10:34	SE001	27.25	27.10	62.00	991.04	-5.84	63.19

Figure 5.10 – Database Master Table

Created according to specifications, the master table has 8 columns which store data on the timecode, sensor code, water temperature, air temperature, humidity, co2, pH, and nutrient. As the name suggests, the master table is the primary data table which stores data collected from the Arduino sensor device.

	id	description
1	SAMPL	Placeholder sensor.
2	SE001	Sensor 1
3	SE002	Sensor 2

Figure 5.11 – Database Sensors Table

Also created according to specifications, the sensors table is much simpler than the master table. It stores information on sensors. Each sensor has a sensor code, and information regarding them are stored in this table in the *description* column.

5.2.2.2 API Logic

The API code is written in the *api.php* file, which is located inside of the */arduino* subfolder of the web application. The API accepts 7 variables, which are sensor code, water temperature, air temperature, humidity, CO_2 levels, pH levels, and nutrient density levels.

```php
<?php
# Retrieve data.
$sensorCode = $_GET['sensorCode'] ?? null;
$waterTemp = $_GET['waterTemp'] ?? null;
$airTemp = $_GET['airTemp'] ?? null;
$humidity = $_GET['humidity'] ?? null;
$co2 = $_GET['co2'] ?? null;
$ph = $_GET['ph'] ?? null;
$nutrient = $_GET['nutrient'] ??  null;

# Check data.
if(is_null($waterTemp) and is_null($airTemp) and is_null($humidity) and is_null($co2) and is_null($ph) and is_null($nutrient)) {
    die("WARNING - No data inserted!");
}

# Insert data.
require_once __DIR__.'/../database/DatabaseConnection.php';
require_once __DIR__.'/../database/config.php';
$connection = new DatabaseConnection( host: DB_HOST, username: DB_USERNAME, password: DB_PASSWORD, schema: DB_SCHEMA);
$query = "
INSERT INTO master
VALUES(
        CURTIME(),
        "."'"."$sensorCode"."'".","
        "."'"."$waterTemp"."'".","
        "."'"."$airTemp"."'".","
        "."'"."$humidity"."'".","
        "."'"."$co2"."'".","
        "."'"."$ph"."'".","
        "."'"."$nutrient"."'"."
        );
        ";
$connection->executeUpdate($query);
$connection->close();
```

Figure 5.12– The API Code

When invoked or accessed, the API will attempt to retrieve data from the seven $_GET superglobal variables. Should any of the variables unspecified they will default to null. Then, the API will check if **all** the variables are null. If all the variables are null, the API will reject the data insertion attempt and invokes the native *die()* function.

Should the insertion pass the **all null** check, the data insertion process will begin. First it will import the *DatabaseConnection* class and the database configuration files. Then it will instantiate a new instance of *DatabaseConnection*. Then it will build the query string from the collected variables. Finally, the API will call the *executeUpdate()* function, which sends the SQL command passed onto it as parameter into the database (whose information was defined during instantiation of the *DatabaseConnection* object). Last but not least, the connection will be closed by calling the *close()* function.

5.2.3 Cloud Deployment

In this subchapter, I will be writing a cloud deployment implementation guide using AWS. Options and features that are not directly addressed in this guide should be left to their default value.

5.2.3.1 Database (RDS)

Step 1: Navigate to RDS

The first step is to login to your AWS Console Dashboard. After that, navigate to the RDS Dashboard to proceed with the database creation process.

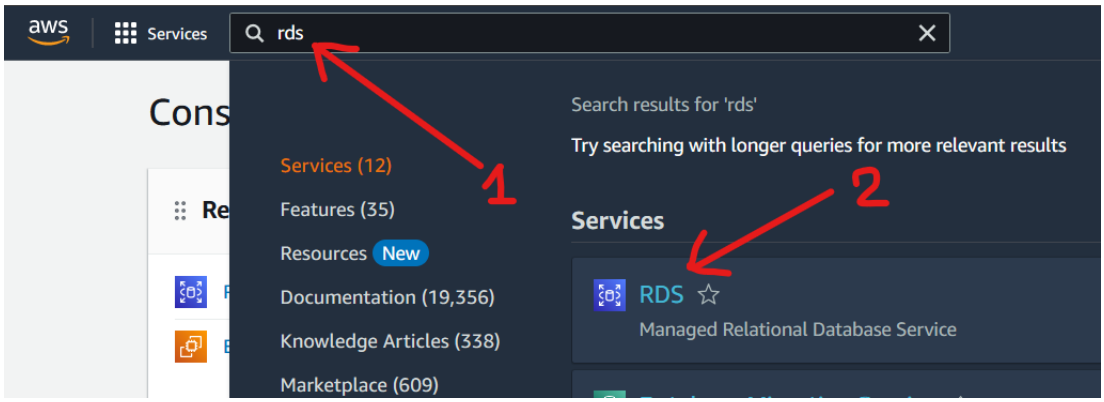


Figure 5.x – Deployment S1/A

Type in “rds” in the search bar and click on **RDS** on the search results. This should take you to the RDS dashboard.

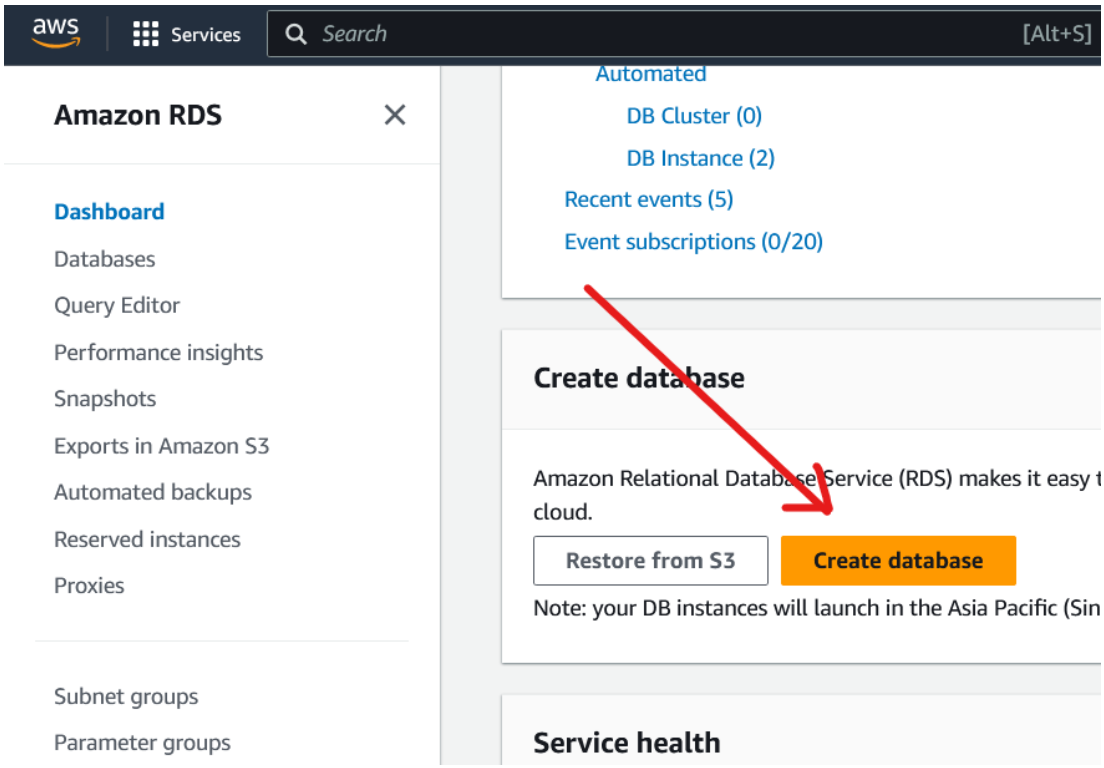


Figure 5.14 – Deployment S1/B

After arriving on the dashboard, scroll down and look for the **Create database** button. Click this button to open the database creation page.

Step 2: Create Database

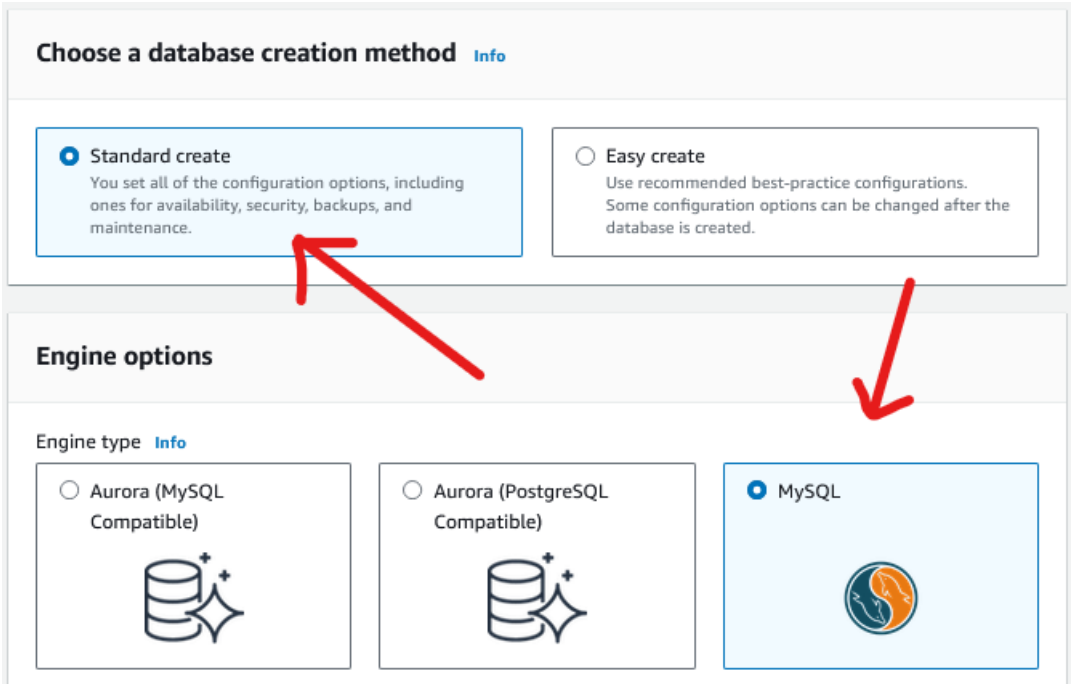


Figure 5.15 – Deployment S2/A

On the database creation screen, choose the **Standard create** method, and choose **MySQL** as the database engine.

Figure 5.16 – Deployment S2/B

It's very important that you make sure that you select the **Free tier** template, otherwise you will be charged a steep price for the *Production* or *Dev/Test* instances, as they are not included in the AWS free tier.

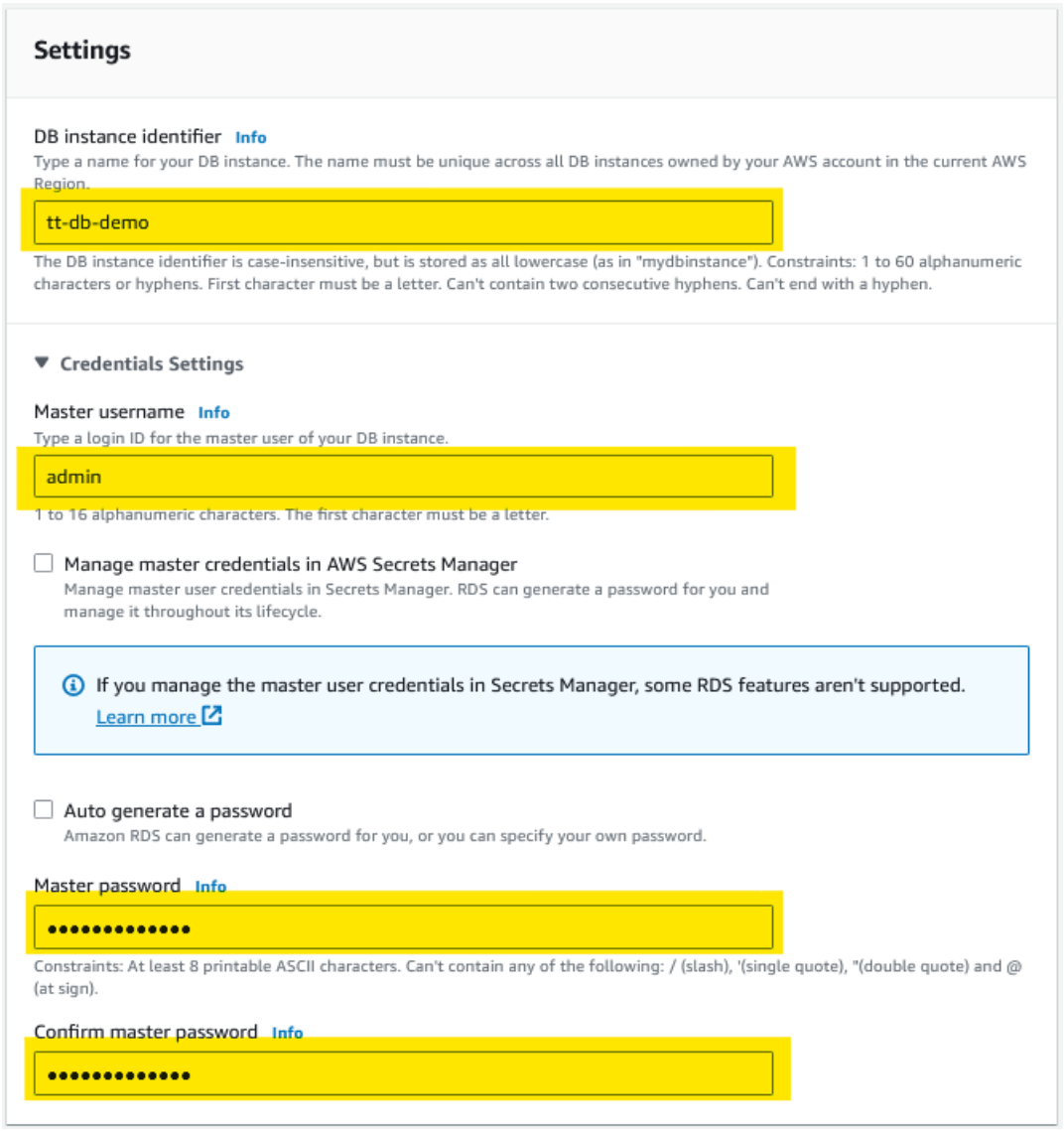


Figure 5.17 – Deployment S2/C

Then, choose a name for your database in the **DB instance identifier** field. The **Master username** should already be set to “admin” by default. You can change this, should you wish, but keep in mind that you will have to adapt any future references to “admin” in this deployment guide. After that, choose a **Master password**.

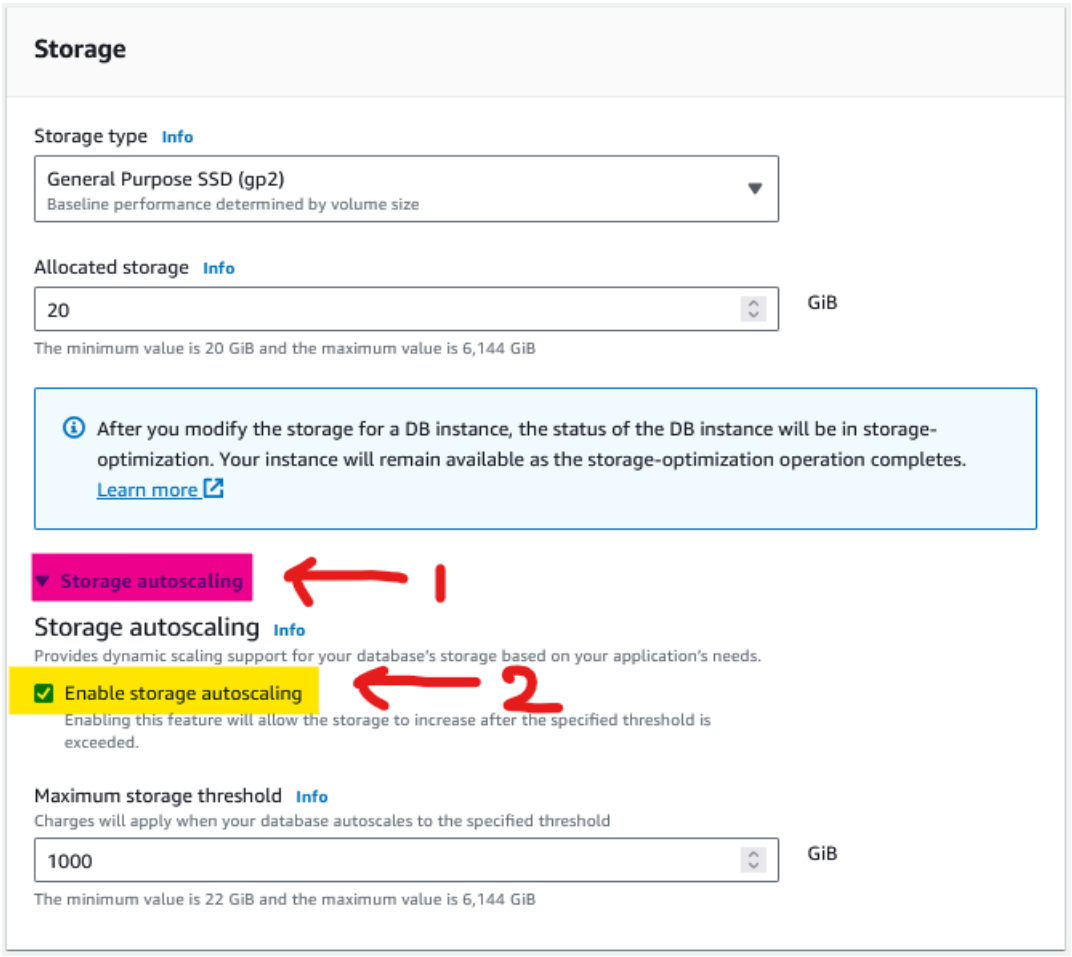


Figure 5.18 – Deployment S2/D

For storage, the **Allocated storage** should default to 20 GiB. You are safe to leave this value to its default. Then, expand a subsection called **Storage autoscaling** and make sure to check the **Enable storage autoscaling** option. The maximum threshold can be left to the default value.

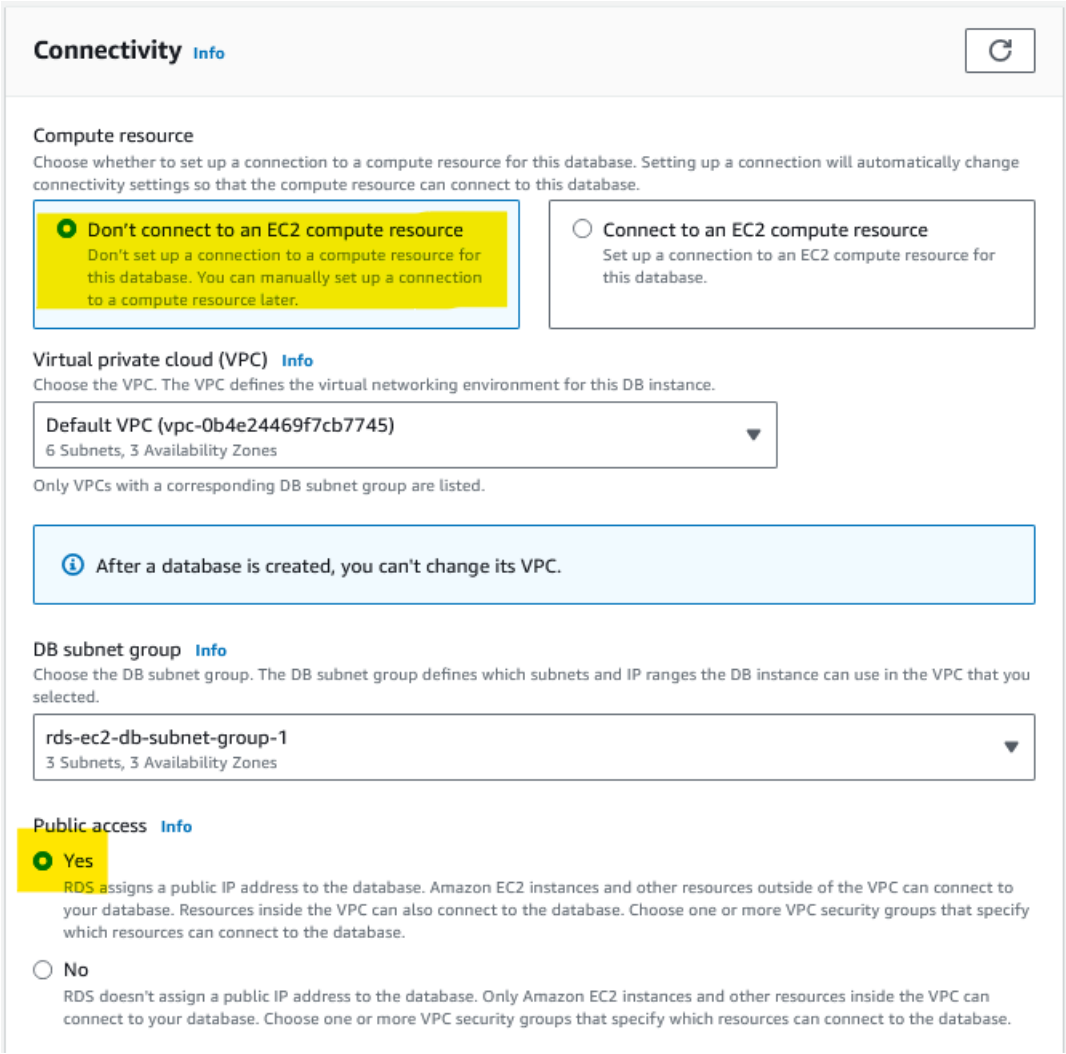


Figure 5.19 – Deployment S2/E

As we do not have an EC2 instance running yet, choose the **Don't connect to an EC2 compute resource** option. Leave the **VPC** to default. Make sure you enable **Public access** (set it to "Yes").

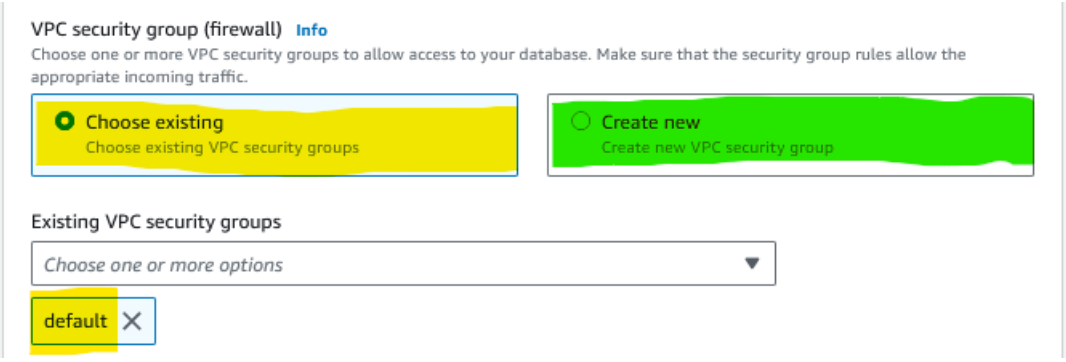


Figure 5.20 – Deployment S2/F

For the **VPC security group**, you may **Choose existing** and pick the "default" configured VPC or create a new one. If you choose to create a new one, the

additional configuration doesn't matter for now, as we will be addressing that in the next step.

Step 3: Configure Firewall (Security Group)

After the database creation process is completed, you will need to configure the database server's firewalls to allow connections from external sources (your computer).

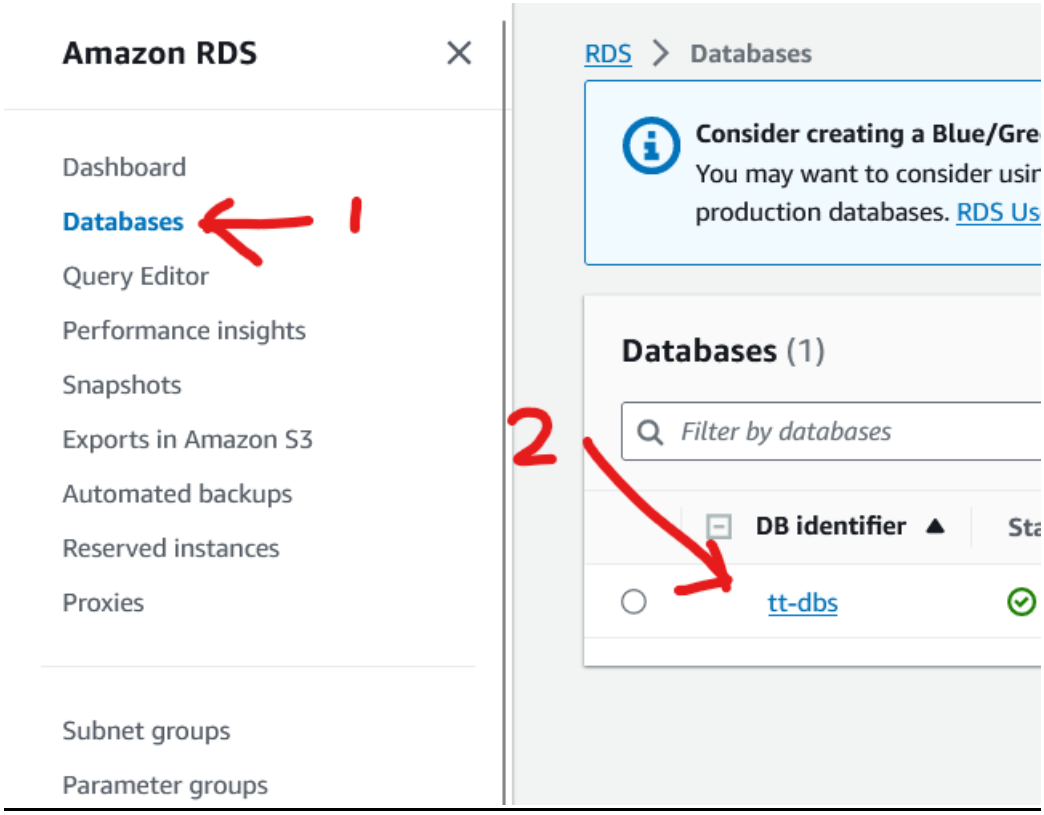


Figure 5.21 – Deployment S3/A

After your database has completed its creation cycle, click on the side-panel, and click on **Databases**. Then click on the created **DB Identifier**, which should be whatever you named your database in the previous step.

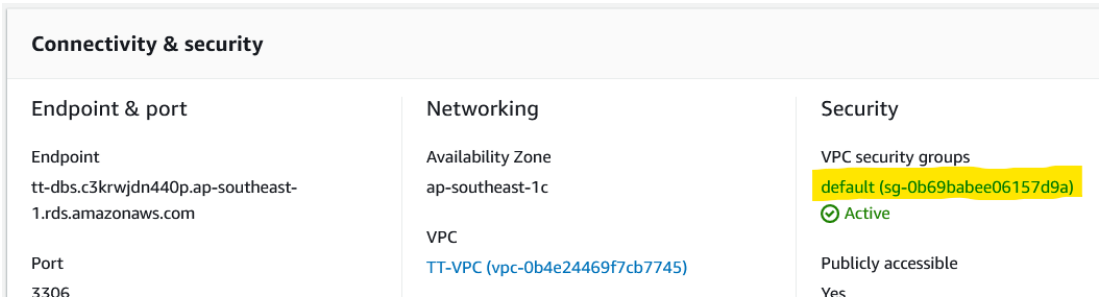


Figure 5.22 – Deployment S3/B

You will then be redirected to the instance dashboard. Scroll down, click the **Connectivity & Security** tab if it isn't already displayed, then click on the **VPC security group**.

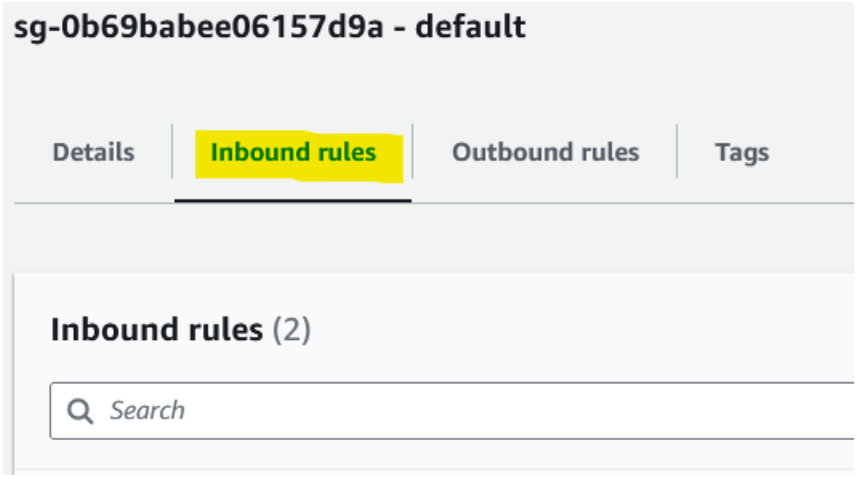


Figure 5.23 – Deployment S3/C

Clicking on the button will redirect you to the security group's dashboard. Scroll down and select the **Inbound rules** tab.

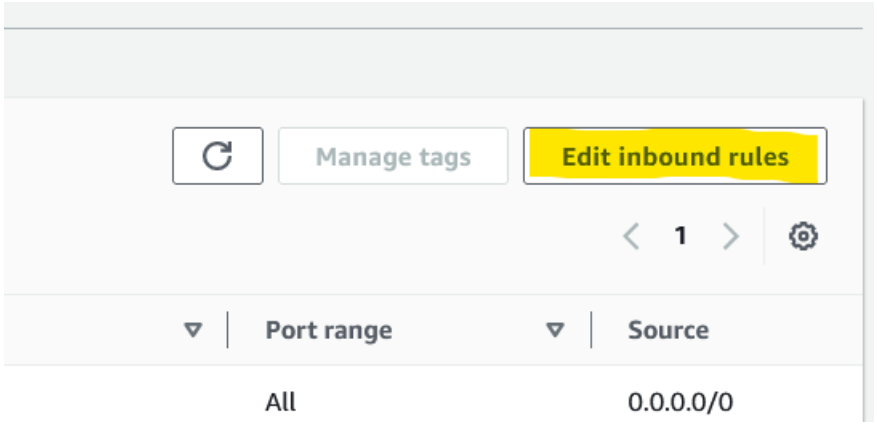


Figure 5.24 – Deployment S3/D

Then look towards the right side of the screen and click on the **Edit inbound rules** button.

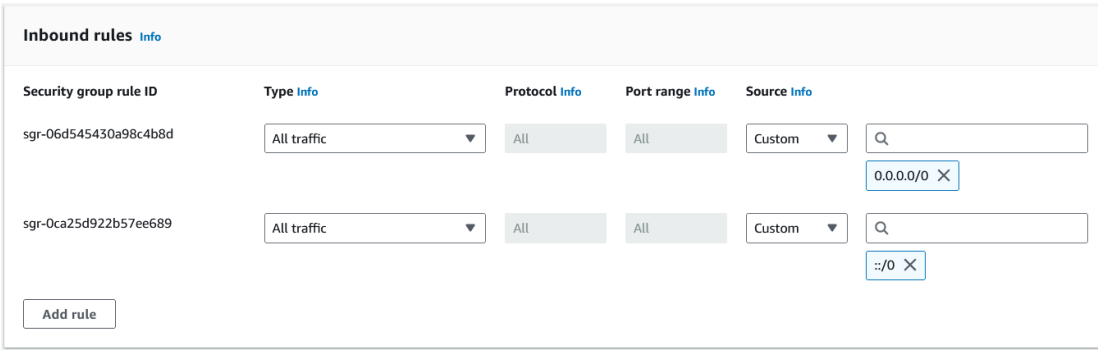


Figure 5.25 – Deployment S3/E

Add two security group rules as displayed in the figure above. The traffic type should be **All traffic**. The source should be set to **Any IPv4** and **Any IPv6**, which will then set the field to *0.0.0.0/0* and *::/0* respectively. Then save the changes.

Step 4: Configure Internet Gateway (VPC Route Table)

After configuring the firewall, in order to connect to the database, you still need to configure an internet gateway in the VPC's route table.

Figure 5.26 – Deployment S4/A

Return to your database instance dashboard, scroll down and choose the **Connectivity & security** tab if it isn't selected already and find **VPC**. Then click on it.

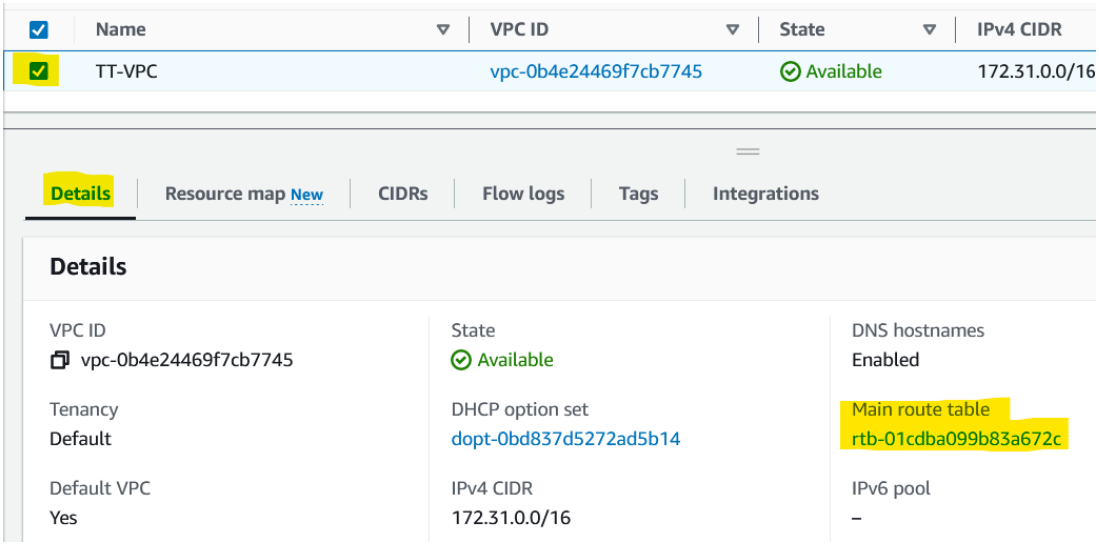


Figure 5.27 – Deployment S4/B

You should then be navigated to the VPC dashboard. Check the box to select the VPC. Select the **Details** tab, and then find the **Main route table** and click on it.

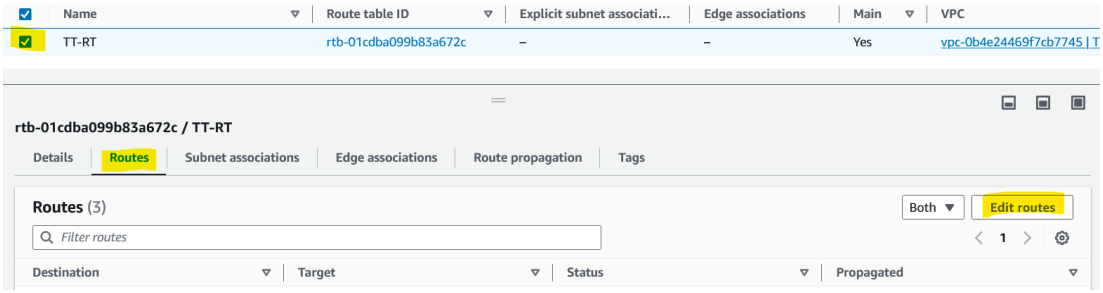


Figure 5.28 – Deployment S4/C

You should arrive on the route table dashboard. Again, check the box to select the route table and select the **Routes** tab on the lower pane. Then click on the **Edit routes** button.

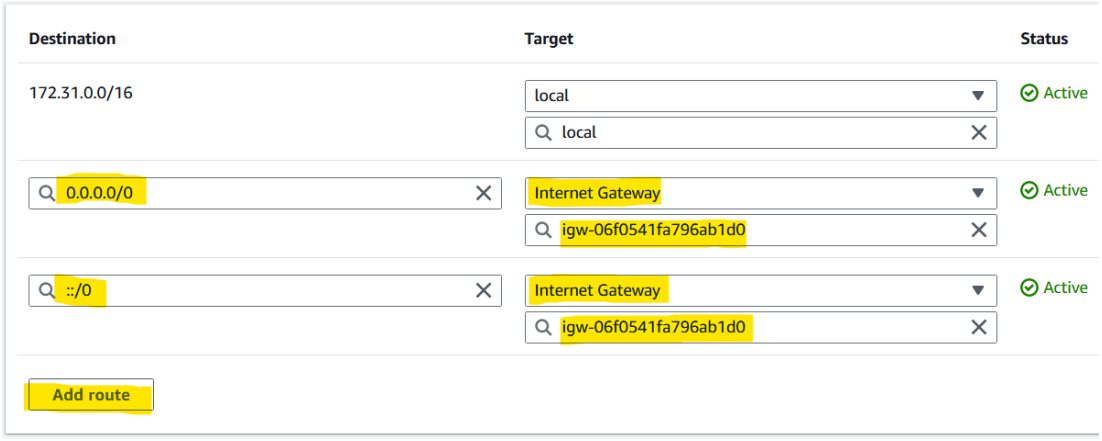


Figure 5.29 – Deployment S4/D

Add two routes, both targeting **Internet Gateway**. Then choose the available internet gateway (**igw-XXXXX**). Set the destination for the internet gateways to *0.0.0.0/0* and *::/0* respectively to open connections for IPv4 and IPv6. After that **save changes**.

Step 5: Connect to Database

After the setup is done, we can try to connect to the database using a database management tool of your choice, I used DataGrip by Jetbrains.

5.2.3.2 Web Application (EC2)

Step 1: Navigate to EC2

First, you need to navigate to the EC2 dashboard. This can be done in the exact same way as it was done for RDS, except look for “ec2” instead.

Step 2: Launch Instance

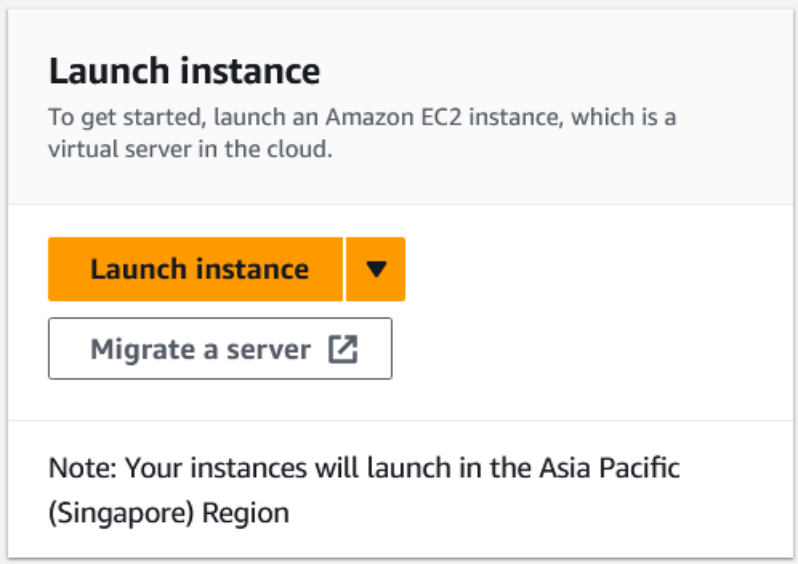


Figure 5.30 – EC2 Deployment S2/A

Upon arrival on the EC2 Dashboard, scroll down, find, and click on the **Launch instance** button. This will take you to the instance launching page, where we can configure the EC2 instance before finalizing the instance launch. As before,

any parameters and fields that are not specified in this step-by-step guide should be left to their default values.

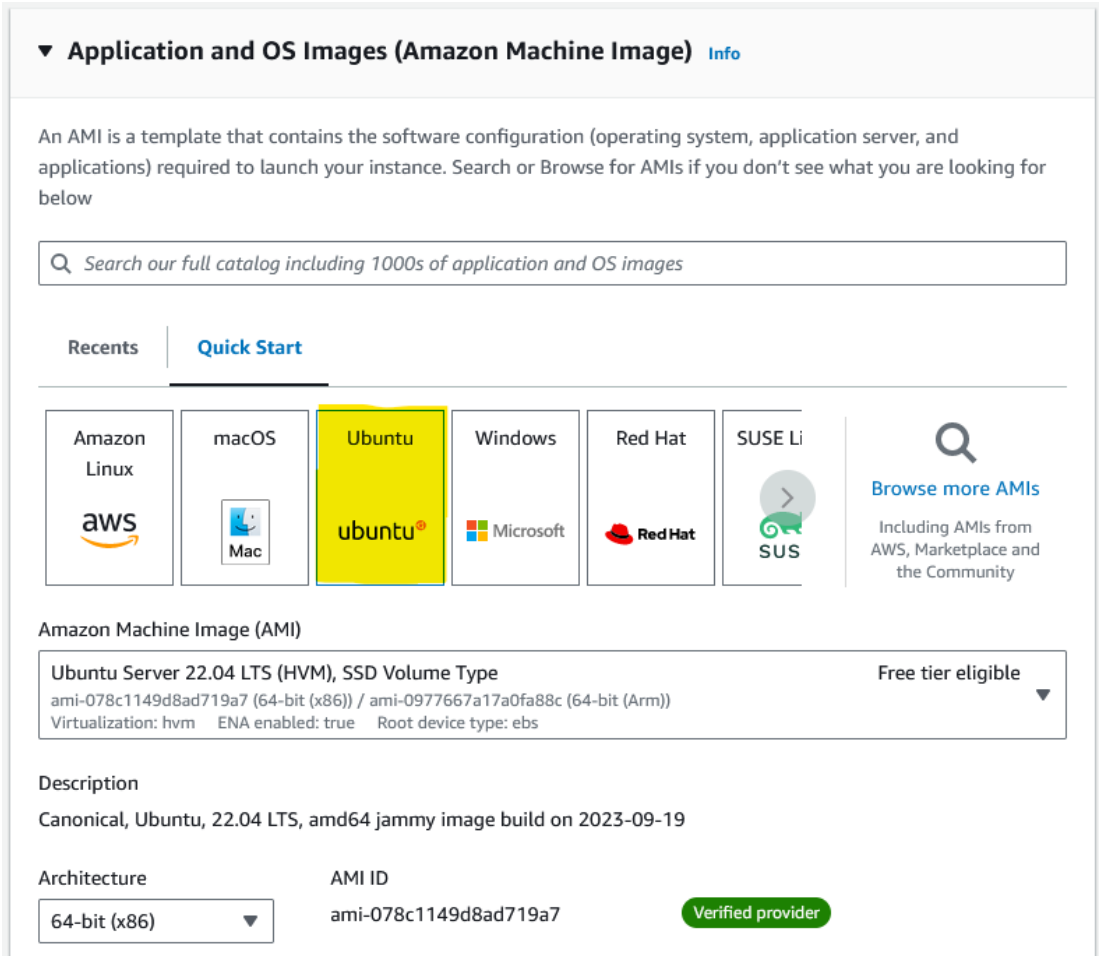


Figure 5.31 – EC2 Deployment S2/B

Give the instance a name and select Ubuntu. On the **Amazon Machine Image (AMI)** section, make sure you choose an LTS version. Choose a **64-bit (x86)** architecture.

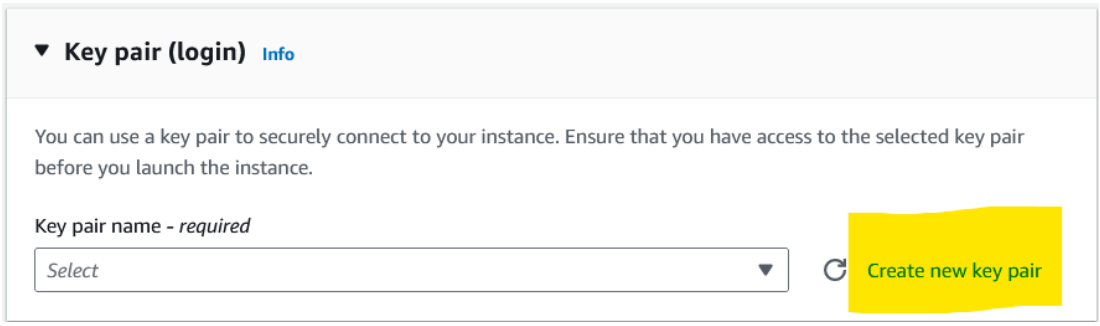


Figure 5.32 – EC2 Deployment S2/C

Leave the instance type to the default value (usually **t2.micro**). Then make sure to generate a new key pair. Click on the **Create new key pair** button

highlighted in the figure above. Or, if you already have a key pair generated, you may use that instead.

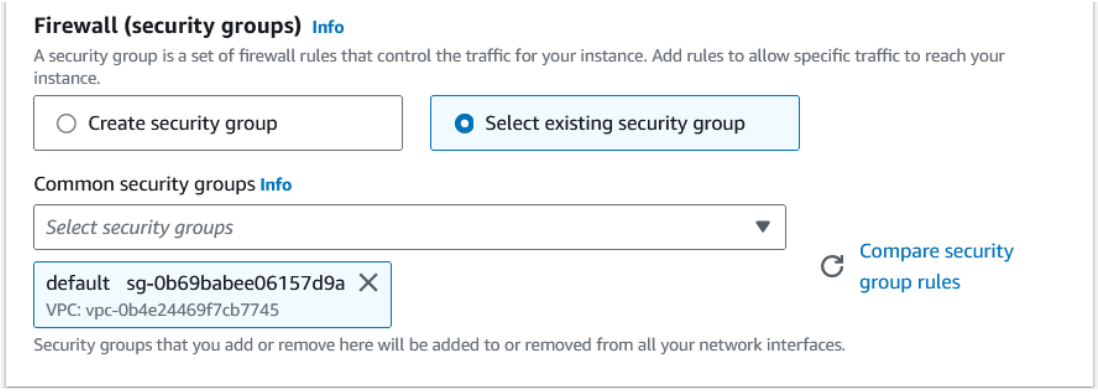


Figure 5.33 – EC2 Deployment S2/D

For the **Firewall (security groups)** setting, use the security group that we had just configured for RDS. In my case this is the **default** security group.

Everything else can be left at default. After you are done, scroll all the way down and click on the **Launch instance** button on the bottom right corner of the form.

Please note that if you created your own VPC during the RDS instance creation, you might have to repeat **Step 4** on the RDS instance creation guide to open an internet gateway on the VPC, otherwise the EC2 instance will use the default VPC (same as the RDS instance) and need no further configuration.

Step 3: Server Configuration

Finally, we can begin the server configuration process. We can begin by connecting to the Ubuntu EC2 server via SSH. The **Terminal** or **Command Prompt** on Windows 11 should be capable of SSH.

- First, open Terminal or Command Prompt *as administrator*.
- Then type in `ssh -i “<put_key_pair_here>.pem” ubuntu@<host>`
- Replace <***put_key_pair_here***> with your key-pair file name and <***host***> with your **Public IPv4 DNS**, which can be found at your EC2 instance summary page.
- After you successfully gained access to the server via SSH, type in `sudo apt update`

- Then, install Apache2 using `sudo apt install apache2 -y`
- Then, install MySQL using `sudo apt install php php-mysql -y`
- Then, you can start the server service using `sudo service apache2 start`

After this, you can paste down the **Public IPv4 DNS** into your web browser's URL bar and access the placeholder Apache page. You might have to **remove 'https://'** to make the website work.

After confirming that the web server is up and running, we can proceed to try and replace the placeholder files with our local server files using an FTP client like FileZilla. But before we do that, because the web server's files are stored in /var/www/html, which is a directory owned by the root user, FileZilla will not have write access to that directory by default. In order to solve this, enter this command into the SSH console: sudo chown ubuntu /var/www/html. Now FileZilla (which logs-in using the ubuntu user) should have read and write access to the directory.

Chapter 6
Evaluation & Discussion

6.1 Evaluation

6.1.1 Methodology

The methodology used for evaluating the results of this case study is **Case Study Evaluation**. A Case Study Evaluation is an evaluation methodology that is focused on identifying common themes and patterns to allow better understanding of the challenges and successes experienced by the solution ("Case Study Evaluation Approach", n.d.). The Case Study Evaluation methodology is chosen because it perfectly suits this case study's needs to analyze what went right and what went wrong during the solution's implementation.

It is important to note that because this thesis is a case study, most of the way this thesis is structured adheres to the structure of a Case Study Evaluation. However, for reasons of practicality, this case study's adherence to the formal structure of a Case Study Evaluation is not rigidly enforced.

This formal "Evaluation" chapter will summarize the results of the case study's solution's implementation.

6.1.2 Arduino

6.1.2.1 Unit Testing

The **test condition** for the unit testing of each of these modules are identical to the ones in the **theoretical foundation chapter**. The passing criteria for all the sensor modules are simply to successfully display environmental data 10 times in a row. For the Wi-Fi module, the passing criteria is to successfully insert data into the database 10 times in a row.

- **Wi-Fi Module**
  Result: **Pass**

```
22:19:08.950 -> SETUP: WiFi...
22:19:09.937 -> WiFi: [AT+RST]
22:19:09.984 -> WiFi: SoftwareSerial n/a. KeyWr n/a. Assume OK!
22:19:10.030 -> WiFi: Delay 10s.
22:19:20.013 -> WiFi: (R)
22:19:20.200 ->
22:19:20.200 ->
22:19:20.200 -> WiFi: [AT+CWMODE=1]
22:19:20.200 -> WiFi: (R) AT+RST
22:19:20.388 -> WiFi: (R) AT+RST
22:19:20.578 -> WiFi: Keyword found!
22:19:20.626 -> WiFi: (R) AT+RSTOK
22:19:20.815 ->
22:19:20.815 ->
22:19:20.815 -> WiFi: [AT+CWJAP="Helios","jasondavid"]
22:19:20.863 -> WiFi: (R) WIFI DR§§§b§(UH§
22:19:21.998 -> WiFi: (R) WIFI DR§§§b§(UH§ets J`n  8 2013,rst cause:2,AT+CWS§T§AT+CWJP§□B§§§,WIGI DISCO
22:19:22.234 -> WiFi: SoftwareSerial n/a. KeyWr n/a. Assume OK!
22:19:22.281 -> WiFi: Delay 10s.
22:19:32.245 -> WiFi: (R) WIFI DR§§§b§(UH§ets J`n  8 2013,rst cause:2,AT+CWS§T§AT+CWJP§□B§§§,WIGI DISCO
22:19:32.481 ->
22:19:32.481 ->
22:19:32.481 -> WiFi: [AT+CIPMUX=1]
22:19:32.529 -> WiFi: (R) WIFI CONNECVED
22:19:32.671 -> WiFi: (R) WIFI CONNECVEDWIFI GOT IP
22:19:32.906 -> WiFi: (R) WIFI CONNECVEDWIFI GOT IP
22:19:33.096 -> WiFi: Keyword found!
22:19:33.096 -> WiFi: (R) WIFI CONNECVEDWIFI GOT IPOK
22:19:33.284 ->
22:19:33.284 ->
22:19:33.284 -> WiFi: Ready!
22:19:33.798 ->
22:19:33.798 -> =-=-=-=-=
```

Figure 6.1 – Wi-Fi Module Unit Test Setup (Serial Monitor)

```
22:19:33.798 -> INFO: Begin data send process.
22:19:33.846 -> INSTR: GET /arduino/api.php?sensorCode=SE002&waterTemp=69.42&airTemp=69.42&humidity=69.42&co2=69.42&ph=69.42&nutrient=69.42
22:19:33.988 -> WiFi: [AT+CIPSTART=0,"TCP","192.168.18.117",80]
22:19:34.036 -> WiFi: (R) AT+CIXMUX=1
22:19:34.178 -> WiFi: (R) AT+CIXMUX=1
22:19:34.367 -> WiFi: (R) AT+CIXMUX=1o§c§AT+CIPSTARTOb□§5A§b□§jq§§§r§§r§§§□c□j
22:19:34.602 -> WiFi: (R) AT+CIXMUX=1o§c§AT+CIPSTARTOb□§5A§b□§jq§§§r§§r§§§□c□j0,CONNECT
22:19:34.790 -> WiFi: (R) AT+CIXMUX=1o§c§AT+CIPSTARTOb□§5A§b□§jq§§§r§§r§§§□c□j0,CONNECT
22:19:36.020 -> WiFi: Keyword found!
22:19:36.020 -> WiFi: (R) AT+CIXMUX=1o§c§AT+CIPSTARTOb□§5A§b□§jq§§§r§§r§§§□c□j0,CONNECTOK
22:19:36.256 ->
22:19:36.256 ->
22:19:36.256 -> WiFi: [AT+CIPSEND=0,118]
22:19:36.256 -> WiFi: SoftwareSerial n/a. KeyWr n/a. Assume OK!
22:19:36.349 -> WiFi: Delay 10s.
22:19:46.304 -> WiFi: (R)
22:19:46.492 ->
22:19:46.492 ->
22:19:46.492 -> WiFi: SENDING PACKAGE!
22:19:46.539 -> WiFi: (R) AT+CIPSDND=0,118
22:19:46.727 -> WiFi: (R) AT+CIPSDND=0,118OK
22:19:46.917 -> WiFi: (R) AT+CIPSDND=0,118OK>
22:19:47.106 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5
22:19:47.293 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5
22:19:47.526 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5SQ§□z-5
22:19:47.713 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5SQ§□z-5
22:19:48.936 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5SQ§□z-5+IXD,0,230:69.2269.4269.4269
22:19:49.124 -> WiFi: SoftwareSerial n/a. KeyWr n/a. Assume OK!
22:19:49.170 -> WiFi: Delay 10s.
22:19:59.154 -> WiFi: (R) AT+CIPSDND=0,118OK>Recv 1L□□§s§5SQ§□z-5+IXD,0,230:69.2269.4269.4269
22:19:59.390 ->
22:19:59.390 ->
22:19:59.390 -> WiFi: [AT+CIPCLOSE=0]
22:19:59.390 -> WiFi: SoftwareSerial n/a. KeyWr n/a. Assume OK!
22:19:59.438 -> WiFi: Delay 10s.
22:20:09.413 -> WiFi: (R)
```

Figure 6.2 – Wi-Fi Module Unit Test Results (Serial Monitor)

52	2023-10-26 22:19:46	SE002	69.42	69.42	69.42	69.42	69.42	69.42
53	2023-10-26 22:20:31	SE002	69.42	69.42	69.42	69.42	69.42	69.42
54	2023-10-26 22:21:28	SE002	69.42	69.42	69.42	69.42	69.42	69.42
55	2023-10-26 22:22:24	SE002	69.42	69.42	69.42	69.42	69.42	69.42
56	2023-10-26 22:23:09	SE002	69.42	69.42	69.42	69.42	69.42	69.42
57	2023-10-26 22:23:55	SE002	69.42	69.42	69.42	69.42	69.42	69.42
58	2023-10-26 22:24:51	SE002	69.42	69.42	69.42	69.42	69.42	69.42
59	2023-10-26 22:25:47	SE002	69.42	69.42	69.42	69.42	69.42	69.42
60	2023-10-26 22:26:32	SE002	69.42	69.42	69.42	69.42	69.42	69.42
61	2023-10-26 22:27:29	SE002	69.42	69.42	69.42	69.42	69.42	69.42

Figure 6.3 – Wi-Fi Module Unit Test Results (Database)

The Wi-Fi module successfully performed insertion into the database 10 times in a row using AT commands. It is important to note that it's not a complete success. Because the ESP-01 module runs at 115200 baud and the main Serial runs at 9600 baud, the message passed from the ESP-01 module's SoftwareSerial to the Arduino's main Serial is garbled. This makes it so that sequencing and timing commands by reading the ESP-01 module's response completely unusable. I had to fall back to using a generous delay of 10 seconds between each command to give the ESP-01 module time to process them, and with the assumption that the commands would work as theory dictates.

The Serial Monitor figure only represents a single entry, because putting all 10 images here would take too much space.

- **Water Temperature Module**

  Result: **Pass**

```
21:17:16.555 -> [1]   WT: 22.25°C
21:17:23.520 -> [1]   WT: 22.13°C
21:17:30.493 -> [1]   WT: 22.25°C
21:17:37.508 -> [1]   WT: 22.25°C
21:17:44.476 -> [1]   WT: 22.25°C
21:17:51.428 -> [1]   WT: 22.25°C
21:17:58.387 -> [1]   WT: 22.06°C
21:18:05.366 -> [1]   WT: 22.31°C
21:18:12.335 -> [1]   WT: 22.25°C
21:18:19.305 -> [1]   WT: 22.31°C
```

Figure 6.4 – Water Temperature Sensor Unit Test Results

The sensor module performed as expected. It successfully retrieved environmental data 10 times in a row. The temperature probe is much less responsive when used outside of water, which is to be expected, as its primary purpose is to measure water temperature.

- **Humidity Module**

  Result: **Pass**

```
21:20:18.898 -> [2A]  AT: 22.20°C
21:20:18.898 -> [2B] HUM: 57.00%
21:20:25.868 -> [2A]  AT: 22.20°C
21:20:25.868 -> [2B] HUM: 57.00%
21:20:32.837 -> [2A]  AT: 22.20°C
21:20:32.884 -> [2B] HUM: 57.00%
21:20:39.801 -> [2A]  AT: 22.20°C
21:20:39.848 -> [2B] HUM: 57.00%
21:20:46.800 -> [2A]  AT: 22.20°C
21:20:46.800 -> [2B] HUM: 57.00%
21:20:53.785 -> [2A]  AT: 22.20°C
21:20:53.785 -> [2B] HUM: 57.00%
21:21:00.743 -> [2A]  AT: 22.20°C
21:21:00.790 -> [2B] HUM: 57.00%
21:21:07.739 -> [2A]  AT: 22.20°C
21:21:07.739 -> [2B] HUM: 57.00%
21:21:14.703 -> [2A]  AT: 22.20°C
21:21:14.703 -> [2B] HUM: 57.00%
21:21:21.652 -> [2A]  AT: 22.20°C
21:21:21.698 -> [2B] HUM: 57.00%
```

Figure 6.5 – Humidity Sensor Unit Test Results

The sensor module performed as expected. It successfully retrieved environmental data 10 times in a row. Both for air temperature and humidity.

- **CO_2 Module**

  Result: **Pass**

```
21:22:57.347 -> [3]  CO2: 1922.31ppm
21:23:04.310 -> [3]  CO2: 1937.25ppm
21:23:11.274 -> [3]  CO2: 1932.27ppm
21:23:18.253 -> [3]  CO2: 1922.31ppm
21:23:25.211 -> [3]  CO2: 1907.37ppm
21:23:32.208 -> [3]  CO2: 1902.39ppm
21:23:39.149 -> [3]  CO2: 1882.47ppm
21:23:46.155 -> [3]  CO2: 1892.43ppm
21:23:53.113 -> [3]  CO2: 1892.43ppm
21:24:00.081 -> [3]  CO2: 1892.43ppm
```

Figure 6.6 – CO_2 Sensor Unit Test Results

The sensor module performed as expected. It successfully retrieved environmental data 10 times in a row. It’s important to note that the sensor

requires manual physical calibration, though it's ability to collect information is completely unhindered.

- **pH Module**

  Result: **Pass**

```
21:33:48.907 -> [3]  CO2: 6.98ppm
21:33:55.883 -> [3]  CO2: 6.99ppm
21:34:02.852 -> [3]  CO2: 7.02ppm
21:34:09.828 -> [3]  CO2: 7.06ppm
21:34:16.808 -> [3]  CO2: 7.00ppm
21:34:23.791 -> [3]  CO2: 7.01ppm
21:34:30.776 -> [3]  CO2: 6.96ppm
21:34:37.714 -> [3]  CO2: 7.06ppm
21:34:44.697 -> [3]  CO2: 6.95ppm
21:34:51.682 -> [3]  CO2: 6.99ppm
```

Figure 6.7 – pH Sensor Unit Test Results

The sensor module performed as expected. It successfully retrieved environmental data 10 times in a row. Though it must be noted that due to the analog input, the voltages are skewed if the pH probe is not connected to the PH-4502C module's BNC connection point.

- **TDS Module**

  Result: **Pass**

```
21:45:03.421 -> [5]  TDS: 19.65ppm
21:45:10.387 -> [5]  TDS: 19.65ppm
21:45:17.331 -> [5]  TDS: 19.65ppm
21:45:24.333 -> [5]  TDS: 19.68ppm
21:45:31.306 -> [5]  TDS: 19.65ppm
21:45:38.259 -> [5]  TDS: 19.65ppm
21:45:45.216 -> [5]  TDS: 21.78ppm
21:45:52.215 -> [5]  TDS: 21.81ppm
21:45:59.169 -> [5]  TDS: 19.71ppm
21:46:06.131 -> [5]  TDS: 19.65ppm
```

Figure 6.8 – TDS Sensor Unit Test Results

The sensor module performed as expected. It successfully retrieved environmental data 10 times in a row. Though it is important to note that the TDS value is not 0 when the electrodes are dried and held up in the air. Theoretically it should be 0, as air is not conductive. Minor discrepancy.

6.1.2.2 Integration Testing

The conditions for integration testing are similar to the conditions used for unit testing, with small modifications to the code to allow the modules to work with each other.

Result: **Pass**

```
21:51:56.861 -> INFO: Begin data send process.
21:51:56.861 -> INSTR: GET /arduino/api.php?sensorCode=SE001&waterTemp=22.25&airTemp=21.80&humidity=57.00&co2=2211.16&ph=-6.58&nutrient=130.70
21:51:57.003 -> WiFi: [AT+CIPSTART=0,"TCP","192.168.18.117",80]
21:51:57.050 -> WiFi: (R) AT+CIPMUX=1
21:51:57.240 -> WiFi: (R) AT+CIPMUX=1
21:51:57.427 -> WiFi: (R) AT+CIPMUX=1OSCSAT+CIPSTART=0,"TCP","192.168.18.117",80
21:51:58.645 -> WiFi: (R) AT+CIPMUX=1OSCSAT+CIPSTART=0,"TCP","192.168.18.117",800,CSBS(UHSHSjS
21:51:58.831 -> WiFi: SoftwareSerial n/a. Keyword n/a. Assume OK!
21:51:58.878 -> WiFi: Delay 10s.
21:52:08.877 -> WiFi: (R) AT+CIPMUX=1OSCSAT+CIPSTART=0,"TCP","192.168.18.117",800,CSBS(UHSHSjS
21:52:09.065 ->
21:52:09.065 ->
21:52:09.065 -> WiFi: [AT+CIPSEND=0,121]
21:52:09.111 -> WiFi: SoftwareSerial n/a. Keyword n/a. Assume OK!
21:52:09.159 -> WiFi: Delay 10s.
21:52:19.102 -> WiFi: (R)
21:52:19.336 ->
21:52:19.336 ->
21:52:19.336 -> WiFi: SENDING PACKAGE!
21:52:19.336 -> WiFi: (R) AT+CIPSQSTSbSSSj
21:52:19.522 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOS
21:52:19.710 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOSRfcv 121 bytes
21:52:19.944 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOSRfcv 121 bytes
21:52:20.129 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOSRfcv 121 bytesSEND OK
21:52:21.351 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOSRfcv 121 bytesSEND OK+IPD,0,236:22.2521.8057.
21:52:21.542 -> WiFi: SoftwareSerial n/a. Keyword n/a. Assume OK!
21:52:21.590 -> WiFi: Delay 10s.
21:52:31.570 -> WiFi: (R) AT+CIPSQSTSbSSSjcSjSHSOSRfcv 121 bytesSEND OK+IPD,0,236:22.2521.8057.
21:52:31.760 ->
21:52:31.760 ->
21:52:31.807 -> WiFi: [AT+CIPCLOSE=0]
21:52:31.807 -> WiFi: SoftwareSerial n/a. Keyword n/a. Assume OK!
21:52:31.855 -> WiFi: Delay 10s.
21:52:41.820 -> WiFi: (R)
21:52:42.007 ->
21:52:42.007 ->
21:52:54.615 -> =-=-=-=-=
```

Figure 6.9 – Integration Test Results (Serial Monitor)

2023-10-26 21:52:19	SE001	22.25	21.80	57.00	2211.16	-6.58	130.70
2023-10-26 21:53:17	SE001	22.31	21.80	57.00	2251.00	-6.49	19.65
2023-10-26 21:54:15	SE001	22.38	21.80	57.00	2181.27	-4.17	21.78
2023-10-26 21:55:12	SE001	22.31	21.80	57.00	2131.47	-6.53	19.65
2023-10-26 21:56:10	SE001	22.44	21.80	57.00	2141.43	-6.56	19.60
2023-10-26 21:57:08	SE001	22.31	21.80	57.00	2246.02	-6.50	19.65
2023-10-26 21:58:06	SE001	22.31	21.80	57.00	2196.22	-6.57	19.65
2023-10-26 21:59:03	SE001	22.56	21.80	57.00	2255.98	-6.54	19.55
2023-10-26 21:59:50	SE001	22.44	21.80	57.00	2280.88	-6.57	19.60
2023-10-26 22:00:48	SE001	22.44	21.80	57.00	2255.98	-6.55	19.60

Figure 6.10 – Integration Test Results (Database)

When all the sensor modules and the Wi-Fi module are combined, the Arduino prototype worked as expected. It successfully inserted the environmental data retrieved by the sensor modules 10 times in a row into the database. The figures above does not show all 10 insertions because it would be too big and long.

6.1.3 Web Application

6.1.3.1 Unit Testing

Unit testing will be conducted manually on a case-by-case basis. I will perform as many test cases that I can think of, but I need to state a disclaimer that as a single person, I lack the perspective and resources of a team, so you as the reader might be able to think of a few test cases that didn't come to me during the writing of this case study.

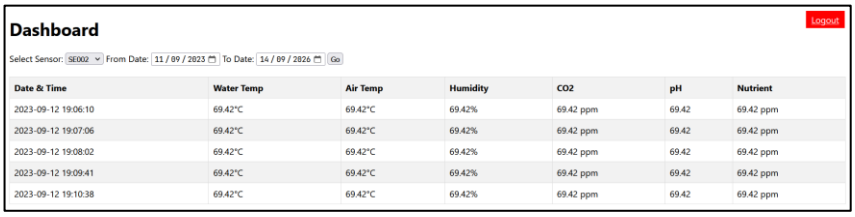
- **Website**
  - ❖ Case 1: Display data from the database.
    - Data retrieved from the database and displayed as expected.
    - 

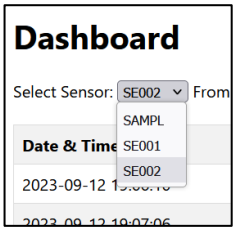
      Figure 6.11 – Dashboard Display
  - ❖ Case 2: Sensor selection.
    - List of sensors successfully retrieved from the database. The website successfully retrieves per-sensor data according to the selected sensor.
    - 

      Figure 6.12 – Sensor Selection
  - ❖ Case 3: Data table columns sorting.
    - Alphabetical (A to Z and Z to A) order sorting behaving as expected.

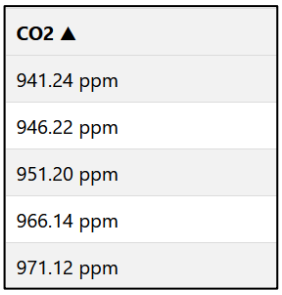
- 

Figure 6.13 – Column Sorting

❖ Case 4: Logout button.

- Logout button behaving as expected. Clicking it will navigate to the placeholder login page.

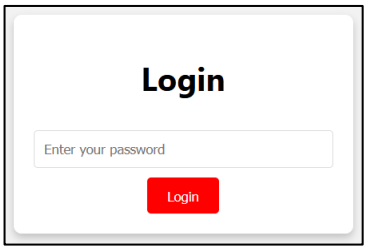
- 

Figure 6.14 – Login Box

❖ Case 5: Applying time filters.

- Time filters applied successfully and behaving as expected.

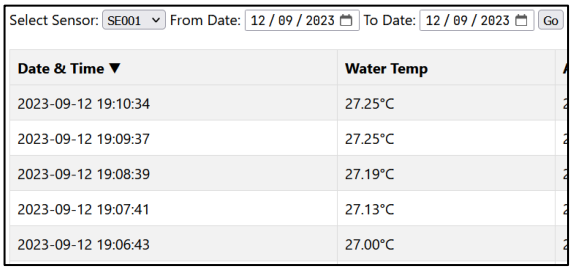
- 

Figure 6.15 – Time Filters

❖ Case 6: Functionality to remember time filters.

- Stored filter memory behavior **exceeds** expectations. The website remembers selections through server shutdown cycles due to the filter data being stored client-side.

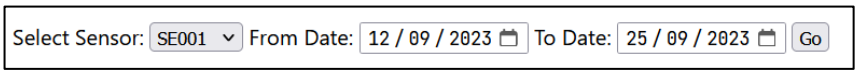
- 

Figure 6.16 – Remember Functionality

- **Database & API**
  - ❖ Case 1: Normal data insert through API.
    - The database and API behaved as expected. Data inserted successfully.
    - 

timecode	sensorCode	waterTemp	airTemp	humidity	co2	ph	ppm
2023-09-25 14:19:01	SAMPL	E	X	A	M	P	LE

Figure 6.17 – DB Inserted

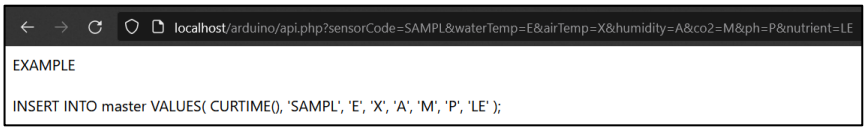
    - 

Figure 6.18 – DB Insertion Instructions

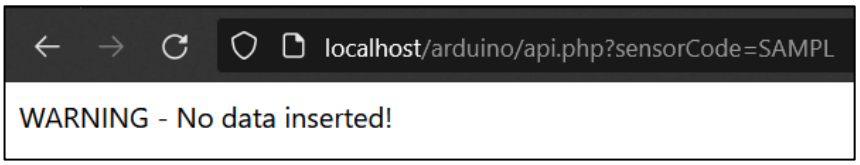
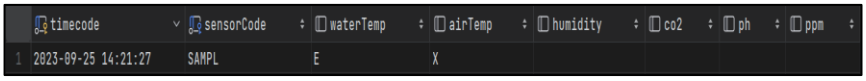
  - ❖ Case 2: All null data insert through API.
    - The database and API behaved as expected. “Data” is rejected.
    - 

Figure 6.19 – Null Insert Test

  - ❖ Case 3: Partial null data insert through API.
    - The database and API behaved as expected. The inserted data is accepted, and unspecified variables defaulted to null.n
    - 

	timecode	sensorCode	waterTemp	airTemp	humidity	co2	ph	ppm
1	2023-09-25 14:21:27	SAMPL	E	X				

Figure 6.20 – Partial Null Insert Test

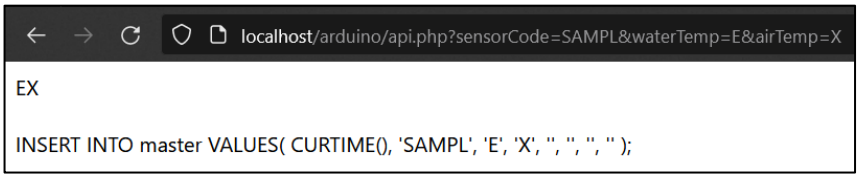
    - 

Figure 6.21 – Partial Null Insert Instructions

6.1.3.2 Integration Testing

The integration testing for the web application alone is very simple. As the goal of the test is to evaluate the functionality of the web application’s dashboard, Arduino API, and database, this test builds on top of the already-performed unit tests.

The final thing to test here is to see if the data inserted during unit testing is displayed successfully on the dashboard.

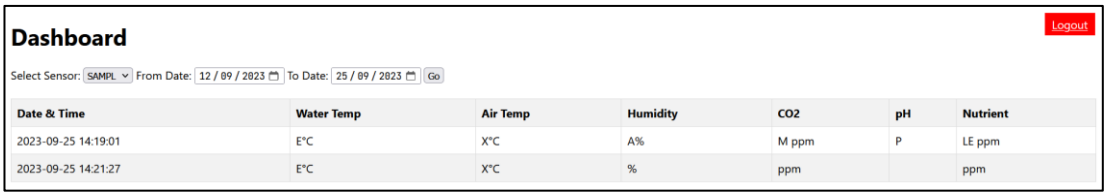


Figure 6.22 – Dashboard Displaying Inserted Test Data

As expected, the dashboard displays the data inserted during unit testing flawlessly. It successfully displays the data inserted during the “normal insertion” and “partial null insertion” tests.

6.1.4 End-to-End

6.1.4.1 Basic Acceptance Testing

As the name suggests, the basic acceptance testing is quite basic. The goal of this test is to evaluate whether the whole system can achieve basic end-to-end functionality. The test will be done manually.

In this test, there will be three key evaluation points. The first evaluation point is running the Arduino device attached to my laptop to view the Serial Monitor. I will be evaluating the whether the Arduino sensor device is behaving as expected via the debug response messages on the Serial Monitor. The second evaluation point is whether the data has been successfully inserted into the database. The third and final evaluation point is to see if the data appears in the web application’s dashboard.

- **Evaluation Point 1: Serial Monitor**

Figure 6.23 - Evaluation Point 1A

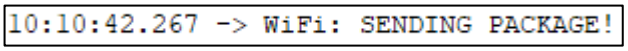


Figure 6.24 – Evaluation Point 1B

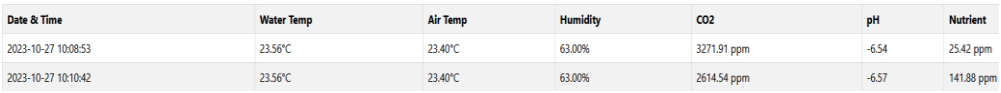
The Arduino sensor device is sending data via Wi-Fi to the database server on **10:10:42.267** according to the IDE's Serial Monitor.

- **Evaluation Point 2: Database**

Figure 6.25 - Evaluation Point 2

The data is confirmed to have been inserted into the database on timecode **2023-10-27 10:10:42**.

- **Evaluation Point 3: Dashboard**



Date & Time	Water Temp	Air Temp	Humidity	CO2	pH	Nutrient
2023-10-27 10:08:53	23.56°C	23.40°C	63.00%	3271.91 ppm	-6.54	25.42 ppm
2023-10-27 10:10:42	23.56°C	23.40°C	63.00%	2614.54 ppm	-6.57	141.88 ppm

Figure 6.26 - Evaluation Point 3

The data inside of the database is directly represented and displayed on the dashboard. The timecode is **2023-10-27 10:10:42**.

6.1.4.2 Reliability Testing

This test is performed over a relatively long period of time (~10 hours). In this test, the Arduino device and web server will be kept running for the duration of the test. After the test's conclusion, the data collection interval will be analyzed for discrepancies. From the result of this analysis, we will be able to infer the reliability of the system. To isolate the device's reliability and to eliminate external factors, the test is done on local network (localhost) instead of on cloud provider servers. Performing the test on local network ensures that any anomalies or irregularities caused by third party factors (such as the cloud service provider) is eliminated.

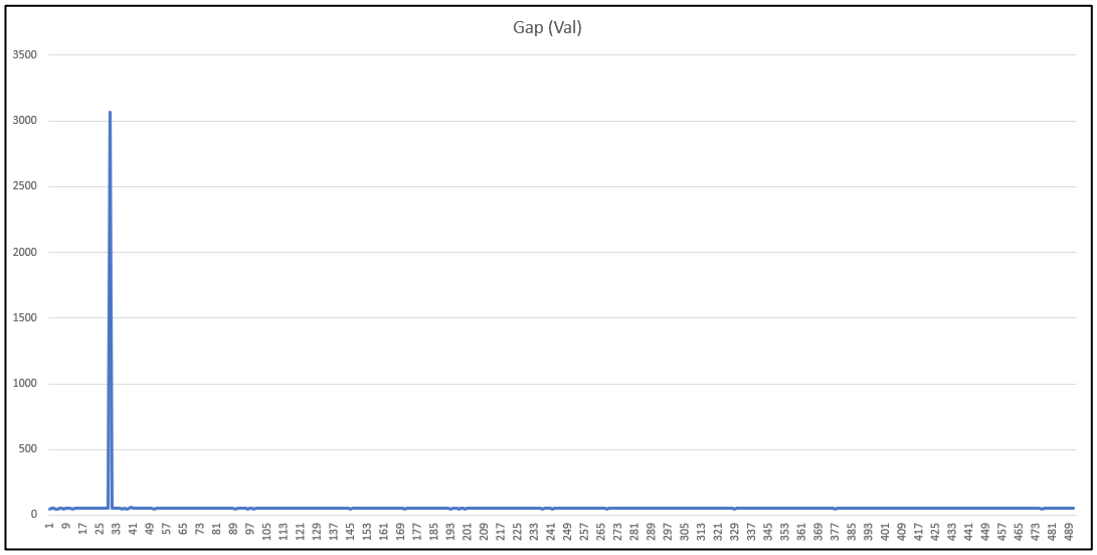


Figure 6.27 – Visualization of the Reliability Testing Results

1	Timecode	Gap	Gap (Val)
27	26/10/2023 22:42	00:56	54.44444
28	26/10/2023 22:43	00:56	54.44444
29	26/10/2023 22:44	00:57	55.41667
30	26/10/2023 22:45	00:56	54.44444
31	26/10/2023 22:46	52:33	3065.417
32	26/10/2023 23:38	00:57	55.41667
33	26/10/2023 23:39	00:57	55.41667
34	26/10/2023 23:40	00:56	54.44444
35	26/10/2023 23:41	00:57	55.41667
36	26/10/2023 23:42	00:56	54.44444

Figure 6.28 – The Visualized Gap

The reliability testing was performed from **26/10/2023 10:19:46 PM** until **27/10/2023 6:45:21 AM**, which amounted to 8 hours, 25 minutes, and 35 seconds. In this timeframe, there was 492 successful insertion cycles with only a single anomaly between the 30th and 31st insertions. Under normal operating conditions, the gap between insertions is around ~56 seconds, but in the anomaly, the delay between insertions was 52 minutes and 33 seconds. Within that gap there should have been ~60 insertion cycles.

The overall reliability rating can be calculated by dividing **what is** by **what should have**, which means that the reliability is ~ 492 / (492+60), which amounts to ~89.13%.

6.1.4.3 Reliability Testing (Cloud)

Like the previous test, this test is performed over a long period of time (~24 hours). This test is identical to the previous one, but the test will be conducted on the cloud instead, and over a longer period of time (around 3 times the length). The Arduino device and web server will be kept running for the duration of the test. After the test's conclusion, the same analysis which was performed on the previous test will be done.

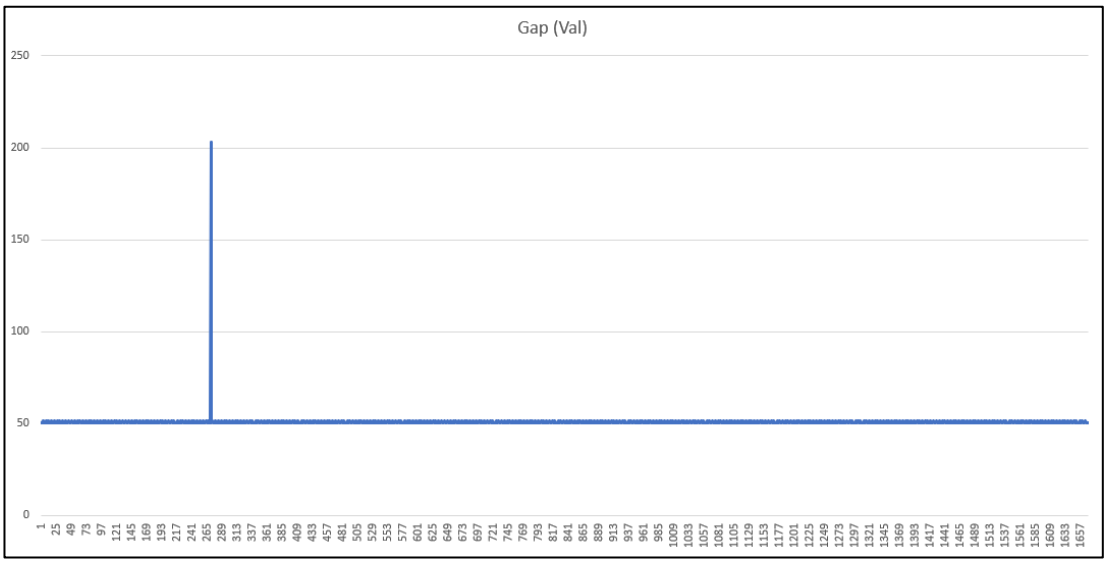


Figure 6.29 - Visualization of the Reliability Testing Results (Cloud)

270	28/11/2023 17:45	00:52	50.55556
271	28/11/2023 17:46	00:52	50.55556
272	28/11/2023 17:47	00:52	50.55556
273	28/11/2023 17:47	03:29	203.1944
274	28/11/2023 17:51	00:52	50.55556
275	28/11/2023 17:52	00:53	51.52778
276	28/11/2023 17:53	00:52	50.55556
277	28/11/2023 17:54	00:52	50.55556

Figure 6.30 – The Visualized Gap (Cloud)

The cloud reliability testing was performed from **28/11/2023 13:52:04 PM UTC** until **29/11/2023 14:07:18 PM UTC**, which amounted to **24 hours, 15 minutes, and 14 seconds**. In this timeframe, there was 1670 successful insertion cycles with only a single anomaly between the 272nd and 273rd insertions. Under normal operating conditions, the gap between insertions is around ~52-53 seconds, but in the anomaly, the delay between insertions was 3 minutes and 29 seconds. Within that gap there should have been ~4 insertion cycles.

The overall reliability rating can be calculated by dividing **what is** by **what should have**, which means that the reliability is ~ 1670 / (1670+4), which amounts to ~99.76%.

6.1.4.4 Multi-Device Testing

In this test, I will evaluate the API's capability to accept connections (Data insertion) from multiple devices. This test is very simple. It will evaluate whether anything in the system behaves abnormally when data is being inserted from two devices simultaneously for 10 connection (data insertion) cycles.

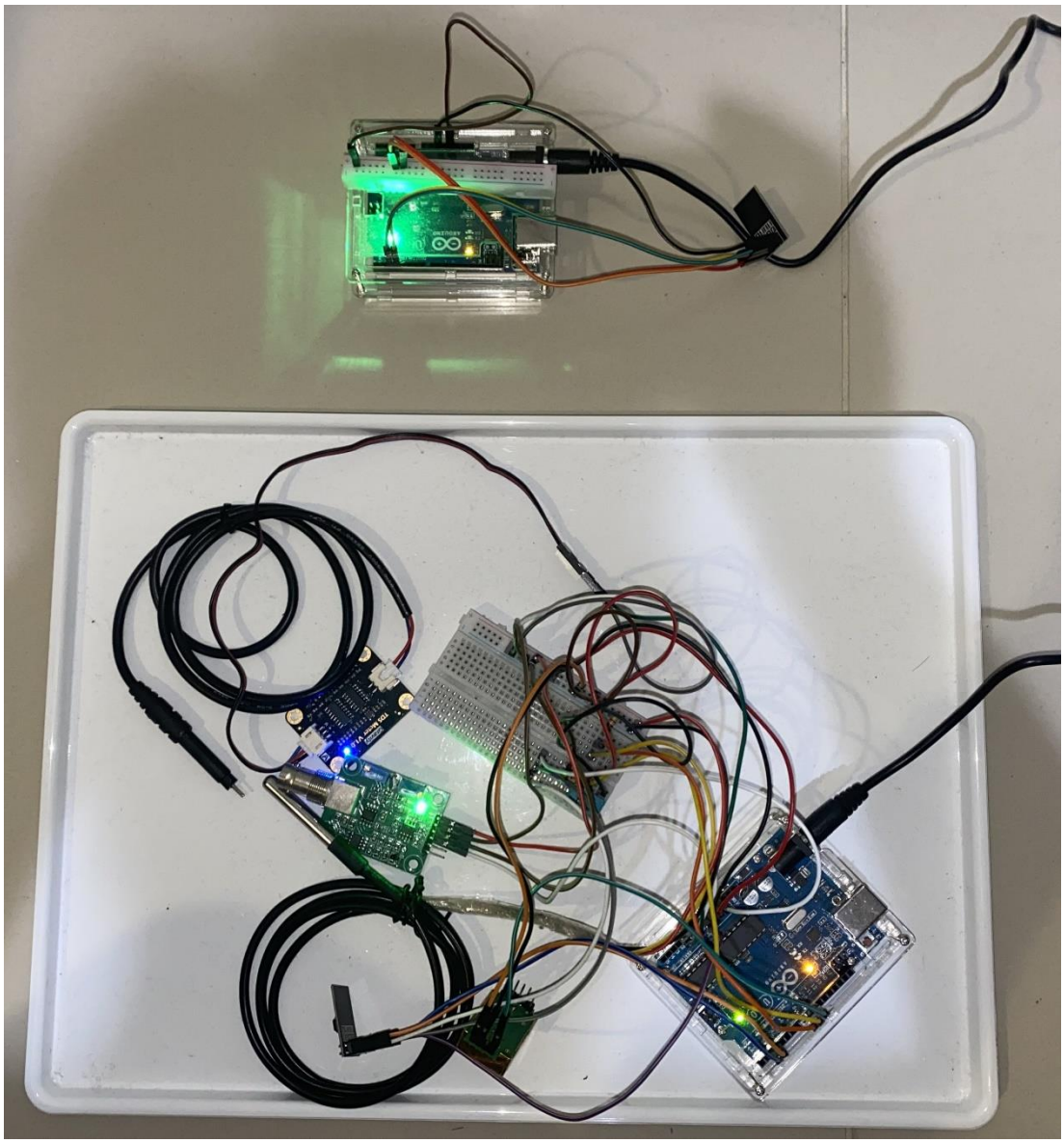


Figure 6.31 – The Main Prototype & The Emulator

For the purposes of this test, I have made a second Arduino contraption but without any of the sensor modules. The second contraption is designed to emulate a second sensor device and is therefore equipped only with a Wi-Fi (ESP-01) module. All the data that the emulator device inserts will be ’69.42’.

11	2023-09-12 19:05:46	SE001	27.00	27.10	63.00	986.06	-6.02	13.95
12	2023-09-12 19:06:10	SE002	69.42	69.42	69.42	69.42	69.42	69.42
13	2023-09-12 19:06:43	SE001	27.00	27.10	63.00	971.12	-6.41	8.00
14	2023-09-12 19:07:06	SE002	69.42	69.42	69.42	69.42	69.42	69.42
15	2023-09-12 19:07:41	SE001	27.13	27.10	62.00	971.12	-6.52	7.99
16	2023-09-12 19:08:02	SE002	69.42	69.42	69.42	69.42	69.42	69.42
17	2023-09-12 19:08:39	SE001	27.19	27.10	62.00	951.20	-6.33	11.93
18	2023-09-12 19:09:37	SE001	27.25	27.10	62.00	971.12	-6.42	11.92
19	2023-09-12 19:09:41	SE002	69.42	69.42	69.42	69.42	69.42	69.42
20	2023-09-12 19:10:34	SE001	27.25	27.10	62.00	991.04	-5.84	63.19
21	2023-09-12 19:10:38	SE002	69.42	69.42	69.42	69.42	69.42	69.42

Figure 6.32 – Database Results for MDT

As can be seen in the figure above, the data insertion process has proceeded normally. Both devices inserted data into the database without issue.

6.1.5 Research Questions & Hypotheses Evaluation

1) Research Question 1
    a. **Question**: How does an Arduino-based and cloud-powered hydroponic monitoring system solution perform compared to manual hand-measuring in tracking changes in a hydroponic plantation's environmental factors?
    b. **Result**:
        i. Hypothesis is only partially accurate.
        ii. In the final state following the conclusion of this case study's implementation, the device is still not ready for commercialization. In its prototype stage, it is unfit for any kind of hydroponic work as exposed jumper cables do not mix well with water. Though if a water tight packaging were to be developed for it, it would be ready for use, but would still require the farm to consult a skilled professional to maintain it, as it is still not user friendly and requires code modification.
2) Research Question 2
    a. **Question**: What are the components that would make up the proposed solution and what role would each component serve?
    b. **Result**:
        i. Hypothesis is accurate.
        ii. The hypothesis is correct, though now reviewing it in hindsight, it seems to be missing the fact that a "packaging" would be needed if the device were to be effective working anywhere close to water (hydroponics).
3) Research Question 3
    a. **Question**: What would be required to build the components??
    b. **Result**:
        i. Hypothesis is only partially accurate.

ii. During the writing of the hypothesis, it didn't come to mind that there would be a variety of tools required to build the components. The architectural design process required the use of Draw.io. Creating and uploading the code for the Arduino requires the Arduino IDE. Creating the web application and database also required development environments like PHPStorm and DataGrip.

iii. But the hypothesis is correct on the fact that the web application can be built on purely PHP alone without any frameworks.

4) Research Question 4

a. **Question**: What would be the differences between the currently used manual method and the proposed solution? What are the advantages and drawbacks of each method?

b. **Result**:

i. Hypothesis is only partially accurate.

ii. Advantages:

1. Yes, indeed automation is the way forward.
2. It is not possible to measure the solution's impact on reducing harvest write-offs, as that would take too much time (way beyond this thesis's writing period).
3. Indeed, data is available in real-time.
4. Yes, the data collected is more consistent, though not necessarily more accurate, as the sensors specifications mention that they require frequent calibration.
5. Correct, data is accessible from anywhere that has an internet access. Accessible even from mobile devices.
6. Impossible to confirm, as time range for this goes beyond the scope of the thesis writing period.

iii. Disadvantages:

1. Correct.
2. Incorrect. Though the unit cost per device is considerable, is likely still cheaper than the 1-month

wage of a single employee. Materially far more economical.

3. Correct. System has a rather difficult learning curve.
4. Correct. Current iteration (a prototype) not yet suitable for commercial use.
5. Correct.
6. Correct. Though can be solved by the use batteries, though the internet modem and wireless access points would also require battery power.
7. Unknown. Likely correct. Testing period not lengthy enough to require maintenance.
8. Partially correct. Person responsible for maintenance can be employed part-time or on a per-case contract basis instead to save costs.

5) Research Question 5
    a. **Question**: In the state immediately following the conclusion of this case study, would the solution be ready for commercialization? Why?
    b. **Result**:
        i. Hypothesis is only partially correct.
        ii. Yes, the product is not commercially viable.
        iii. No, the product is not too expensive even in the prototype stage. It will get even cheaper when mass production starts due to economies of scale.
        iv. Yes, the product is currently unreliable.
        v. No, the product is difficult to use for untrained individuals, but for trained individuals, it is very easy to use.

6.2 Discussion

There are a few known issues that are not covered by the tests within the evaluation subchapter. These are not evaluated because there is simply nothing that can be done to solve them during this case study. These problems are unsolvable primarily due to time constraints and a lack in the required expertise. Though it is

important to note them down as future studies in this topic might find a way to solve them. They are as follows:

6.2.1 Not Ready for Commercialization

The implemented solution is not ready for commercialization because it is too difficult to use for end consumers. At its current state, its only acceptable as a technical demonstrator. Additional development is required to make it commercially viable.

6.2.2 Range Extension

Dr. Michael Siek expressed his interest in expanding the Arduino sensor device's capabilities by making it able to communicate with each other and act similarly to a "mesh" network. Unfortunately, the approach suggested by Dr. Michael is physically not possible with the current prototype.

The Wi-Fi module on the Arduino, which is an ESP-01, is capable of drawing 300mA @ 3.3V at peak draw, whilst the Arduino's 3.3V power supply pin is only capable of supplying 50mA at maximum. Configuring the ESP-01 module to communicate with *each other* is most certainly going to increase the power draw beyond what the Arduino is capable of supplying.

There is also the issue of the Arduino Uno's limited flash storage and memory. Even with the relative simplicity of the current configuration, where the Arduino only needs to communicate directly with the wireless access point, its memory and flash storage utilization is over ~60%. It's important to note that due to this approach being written off as impossible, I haven't done any in-depth research into this topic, so I cannot confirm whether the increase in code complexity and length will go beyond the Arduino Uno's capabilities. But the limited amount of (undocumented) research that I have done suggests that the Arduino Uno will not be capable of bearing the increased code complexity.

It is also important to note that even if both of the issues stated above somehow magically disappear, I still do not know if the ESP-01 module is even

capable of communicating with each other. I did not research this topic further due to the approach associated with it being written off (as stated above, not enough power) and the rather tight time constraints.

Fortunately, there is a solution to the problem of range extension. There are Wi-Fi extending mesh routers commercially available for purchase. Products such as TP-Link's Deco are already known for their reliability, simplicity, and relative affordability. These products are produced for the mass market, which means that they will both cheaper and more polished due to economy of scale. Utilizing these products to extend the Wi-Fi range itself rather than extend the Arduino's reach is the way to go.

6.2.3 Wi-Fi Reliability

As can be seen from the test results in the subchapters above, the ESP-01 Wi-Fi module itself is quite reliable. Unfortunately, should the Wi-Fi connection be dropped for any reason, the device will have to be manually (physically) reset by hitting the Arduino's reset button or unplugging and re-plugging the power cord.

Currently, I can not find a way to increase reliability on this factor. I can't set up an automated system to detect if the Wi-Fi disconnects because the response strings returned by the Arduino's *SoftwareSerial* is garbled and unreliable. I also can't set the Wi-Fi module to reset (AT+RST) and re-connect to the wireless access point on every loop iteration because this effectively and statistically lowers reliability instead, as the device sometimes fails to connect.

6.2.4 Data Collection Frequency

I've stated in the solution design chapter that the data collection frequency should be at least once every 30 minutes to once every hour. If the database can be configured to either periodically delete old data to conserve storage space or overwrite existing old data when the storage gets full. Of course, the decision on what to do falls to the customer or user. Deciding on the best course of action when it comes what to do with the hydroponic data is outside of the scope of this case study.

6.2.5 Documentation Scarcity

During the creation and execution of this case study, I have made several discoveries which was rather arduous to do, as the documentation was either incomplete or completely nonexistent. I hope that the information I collected during the writing of this case study may be of use as a reference material in future research into this field.

Documentation of the ESP8266 Wi-Fi module is sparse and are often unreliable. Information on how to make an Arduino device communicate directly with an ESP8266 (ESP-01) module was almost entirely non-existent. What instructions exist were either poorly explained, or was only verbally explained in scattered tutorial videos.

And though widely used, documentation and discussion on the use of the *SoftwareSerial* class in the Arduino IDE is also unreliable with almost no citations, leading me to have to experiment to determine how to properly use it. The nature of the Arduino IDE itself is also not made apparent, being buried deep inside documentation. The perceived “shallow” nature of the language used by Arduino also does not do anything to help it, and the community around Arduino is not a particularly strong one, especially compared to those like Java, C++, and C#.

I’ve written down whatever knowledge I’ve compiled during the process of writing this case study within the second chapter, which covers the theoretical foundation necessary to this case study.

6.2.6 UI Design Simplicity

The UI design is overtly simple. Though this may be a positive point for serious businesses who don’t care much for fancy and heavy graphics, this may turn off some customers and give an improper “product image”. I have made some suggestions for the UI design which can be found in this case study’s appendices.

Chapter 7
Conclusion & Recommendation

7.1 Conclusion

Just Hydroponics currently relies on manual data collection using a set of hand-held measuring tools. This infrequent data collection poses challenges in responding to sudden environmental changes, making it difficult to maintain the optimal conditions for their hydroponic plantations. The absence of standardized measuring tools further jeopardizes data quality, leading to an incomplete picture of the condition of their plantations.

Moreover, the use of whiteboards for data storage is precarious due to data vulnerability. The whiteboards can be easily wiped, leading to potential data loss, while the manual transcription of data is labor-intensive. The current processes are not only time-consuming but also prone to errors, causing inaccuracies and inconsistencies in data.

This manual approach to data collection and management results in a labor-intensive and inefficient allocation of resources, with an inadequate response to environmental changes and delayed troubleshooting. These challenges hinder the scalability and profitability of Just Hydroponics.

To address these problems, an automated system, based on Arduino sensors and a cloud-based web application, is proposed. The transition to an automated approach offers solutions to data vulnerability, data management inefficiencies, labor intensity, reduced accuracy, delayed responses, and inefficient allocation of financial resources.

This case study has managed to successfully produce a working prototype of such a system, albeit only as a proof of concept that is currently commercially unviable. However, it proved that an Arduino-based hydroponic sensor is not only possible to develop, but quite viable. Unfortunately, the limited resources and skillsets (mine alone) allocated to this case study means that commercialization is far

outside the scope of this case study. To make the product commercially viable, further development is required.

If a lone individual (me) with limited resources and a very tight time constraint can develop this solution, a company with significantly more resources can complete the development of this product and introduce it to the market.

7.2 Recommendation

Given the nature of this case study's goal being to produce a product proof-of-concept, there are a lot of exciting ways to improve the product. Implementing some or all these recommendations should put you on the right path to making the product ready for commercialization, though this list of recommendations is not exhaustive of possible ways to improve the product. These recommendations are as follows:

- Casing Design
    - In its current state, the prototype is quite literally sensors stitched together with jumper cables. It is not ready for use in any environment, especially not a hot and humid hydroponic greenhouse environment. A casing would need to be designed for it to make it resistant to environmental factors and disruptions that are harmful to electronic components.
- Independent Power Supply for the Wi-Fi Module
    - Currently, the power provided by the 3.3V pin on the Arduino board is insufficient for the Wi-Fi module's full capabilities. Though the power is sufficient for the module's current use case, a lackluster power supply would harm the Wi-Fi module's reliability should it somehow draw more power than is available.
- Economic Viability Study
    - In its current iteration, the device's unit cost is probably on the higher side. An economic viability study would allow you to find ways to lower the unit cost and make it more attractive for consumers.
- Parallel Databases

- In its current implementation, the system relies heavily on a stable Wi-Fi connection. Which means that the inherently unstable Wi-Fi on a remote hydroponic farm would cause problems to the system’s reliability. This can be solved by implementing a parallel database system, where a smaller local database can be used to temporarily store data during periods of unstable internet connection, and then synchronize with the primary cloud-based database.

References

About Arduino. (2021, September 15). Arduino. https://www.arduino.cc/en/about

Ada, L. (2015). *Adafruit HUZZAH ESP8266 Breakout*. Adafruit. https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/overview

Amazon. (2023). *What is AWS?* Amazon AWS. https://aws.amazon.com/what-is-aws/

Amazon. (2023). *What is Amazon Relational Database Service (Amazon RDS)?* Amazon AWS. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

Amazon. (2023). *What is Amazon EC2?* Amazon AWS. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Arduino. (2023, June 26). *SoftwareSerial Library*. Arduino Documentation. https://docs.arduino.cc/learn/built-in-libraries/software-serial

Barela, A. (2023). *ESP8266 Temperature / Humidity Webserver*. Adafruit. https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/overview

Bhimsen. (2022, September 9). *ESP8266 AT Commands List and Working Explained.* Electronics Fun. https://electronics-fun.com/esp8266-at-commands/

Binance. (2023). *Turing Complete*. Binance Academy. https://academy.binance.com/en/glossary/turing-complete

Bootstrap – The most popular HTML, CSS, and JS library in the world. (n.d.). Bootstrap. Retrieved April 17, 2023, from https://getbootstrap.com/

Campbell, S. (2015, October 1). *How to Set Up the DHT11 Humidity Sensor on an Arduino*. Circuit Basics. https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/

Case Study Evaluation Approach. (n.d.). EvalCommunity. Retrieved September 26, 2023, from https://www.evalcommunity.com/career-center/case-study-evaluation-approach/

CimpleO, (2020, April 23). *Arduino pH-meter using PH-4502C*. CimpleO. https://cimpleo.com/blog/simple-arduino-ph-meter/

CSS Introduction. (n.d.). W3Schools. Retrieved April 17, 2023, from https://www.w3schools.com/css/css_intro.asp

DFRobot. (n.d.). *SEN0244 Gravity Analog TDS Sensor Meter for Arduino*. DFRobot Open-Source Hardware Electronics and Kits. Retrieved April 16, 2023, from https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244

DPV Technology. (2019, September 29). *Sending data to thingspeak website using esp8266 Arduino Tutorial* [Video]. YouTube. URL https://www.youtube.com/watch?v=nMWwqcn7ofw

Electronoobs. (2019, August 4). *ESP8266 + Arduino + database – Control Anything from Anywhere* [Video]. YouTube. URL https://www.youtube.com/watch?v=6hpIjx8d15s&t=491s

ESP8266 Pinout Reference: Which GPIO pins should you use? (2019, May 6). Random Nerd Tutorials. https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

Fahad, E. (2022, June 24). *MH-Z19B NDIR CO_2 Sensor with Arduino*. Electronic Clinic. https://www.electroniclinic.com/mh-z19b-ndir-co2-sensor-with-arduino-mhz19b/

Fernandez, E. (2014, February 20). Hydroponic Nutrients TDS, PPMs and EC Explained! [Video]. YouTube. https://www.youtube.com/watch?v=uI9D-ONNdHg

freeCodeCamp.org. (2021, June 8). Arduino Course for Beginners – Open-Source sElectronics Platform [Video]. YouTube. https://www.youtube.com/watch?v=zJ-LqeX_fLU

General Hydroponics. (2023). *How Often Should I Check My pH Level?* General Hydroponics FAQs. https://generalhydroponics.com/faqs/how-often-should-i-check-my-ph-level/

Google. (2023). *Google Cloud Documentation*. Google Cloud. https://cloud.google.com/docs

Hernandez, R. D. (2021, April 19). *The Model View Controller Pattern – MVC Architecture and Frameworks Explained*. freeCodeCamp. https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

HTML Introduction. (n.d.). W3Schools. Retrieved April 17, 2023, from https://www.w3schools.com/html/html_intro.asp

IoTSpace. (2021, September 17). *Arduino CO_2 Sensor – MH-Z19 Beispiel und Sketch*. IoTSpace. https://iotspace.dev/arduino-co2-sensor-mh-z19-beispiel-und-sketch/

Janes, L. (2019, July 30). *Top 15 Reasons Why You Should Grow Vegetables In A Hydroponic Garden*. VH Hydroponics. https://www.vhhydroponics.com/hydroponic-garden/

MariaDB. (2023). *MariaDB Server: The open-source relational database*. MariaDB. https://mariadb.org/

Maxim Integrated. (2019). *DS18B20 – Programmable Resolution 1-Wire Digital Thermometer*. Maxim Integrated. https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf

Mouser Electronics. (n.d.). *DHT11 Humidity & Temperature Sensor*. Mouser Electronics. Retrieved April 16, 2023, from https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Mozilla. (2023, September 16). *What is JavaScript?* MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

MySQL. (n.d.). *MySQL 8.0 Reference Manual*. Dev MySQL. https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

NodeMCU ESP8266. (2020, April 22). Components 101. https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet

PHP. (2023). *History of PHP and Related Projects*. PHP. https://www.php.net/manual/en/history.php

pH Probe Architecture. (n.d.). Digital Analysis Corporation. Retrieved 4 September 2023, from https://www.digital-analysis.com/TArticles/pH_Probe.html

Prasanniya. (2023, March 28). *A Closer Look at the Latest Technologies in Hydroponics*. Hydroponic Way. https://hydroponicway.com/latest-technologies-in-hydroponics/

Programming Electronics Academy. (2021, April 11). *DIY Hydroponic Garden w/Arduinoand IoT* [Video]. URL https://www.youtube.com/watch?v=Ng7qDDH9Yqk

Santos, R. (2016, August 24). *Guide for DS18B20 Temperature Sensor with Arduino*. Random Nerd Tutorials. https://randomnerdtutorials.com/guide-for-ds18b20-temperature-sensor-with-arduino/

Shinde, S. (2023, January 25). *What are the Key Pros and Cons of the Arduino Programming Language?* Emeritus. https://emeritus.org/blog/coding-arduino-programming-language/

Winsen. (2016). *Intelligent Infrared CO_2 Module (Model: MH-Z19B)*. Zhengzhou Winsen Electronics Technology Co., Ltd. https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf

XAMPP Team. (2023). *Windows Frequently Asked Questions*. Apache Friends. https://www.apachefriends.org/faq_windows.html

Curriculum Vitae

Name	:	Jason Alexander Tan
Place, Day of Birth	:	Jakarta, 5 November 2001
Sex	:	Male
Address	:	Jl. Tajuk Rencana H-117 Cipinang Muara, Jatinegara, Jakarta Timur 13420
Telephone	:	(+62) 858 8888 9900
Education and Training	:	

Binus University International **July 2020 - Present**
Undergraduate, Business Information Systems Jakarta, Indonesia

SMAK 1 BPK Penabur Jakarta **July 2017 – Jun 2020**
High School Diploma, School of Science Jakarta, Indonesia

Work History :

PT. Bank DBS Indonesia **February 2023 - Present**
Business Platform Management (Internship) Jakarta, Indonesia

NHS England **February 2022 – September 2022**
Project Management (Internship, Remote) Jakarta, Indonesia

Appendices

Appendix 1: UI Design Suggestions

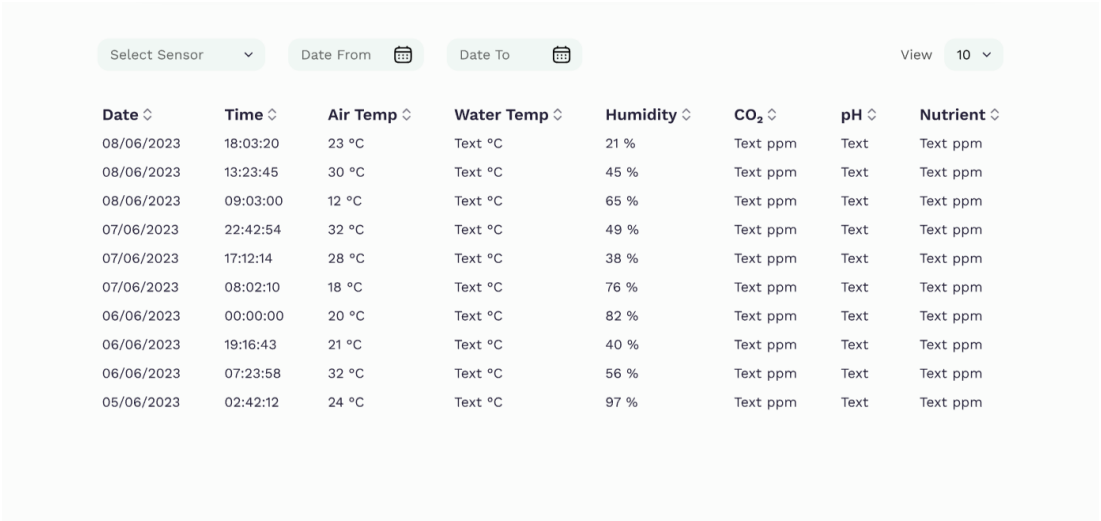


Figure A1.1 – UI Suggestion 1

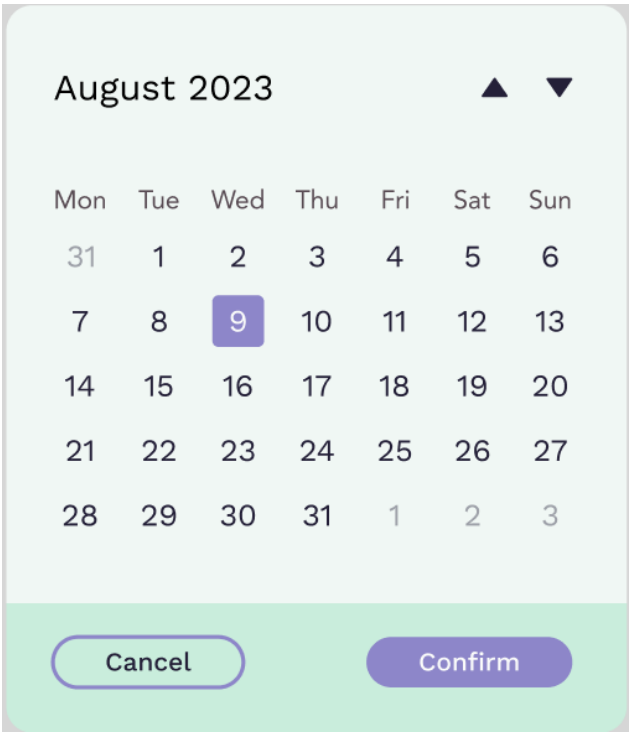


Figure A1.2 – UI Suggestion 2

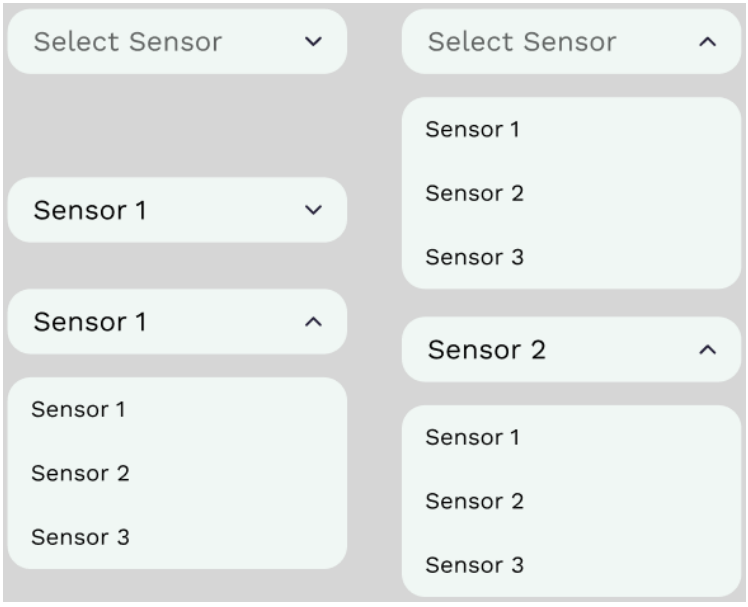


Figure A1.3 – UI Suggestion 3

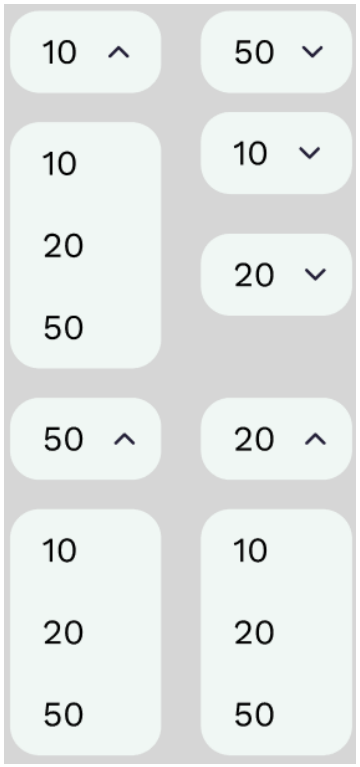


Figure A1.4 – UI Suggestion 4

Appendix 2: Results of Interview with Just Hydroponics

(Interview Results)

Date : Sunday, 1st of January 2023.

Time : ~11:00 AM

Location : Just Hydroponics, Outskirts of Bogor City
Jl. Pasir Bogor, Cipelang, Cijeruk, Bogor, Jawa Barat 16740

Interviewee : Anton & Ina (Owners of Just Hydroponics)

**Note: This interview is not a 100% accurate & direct reference to what was said during the interview. For the sake of readability, the written transcript has been modified, rewritten, translated to English, and had some unnecessary dialogue which covers some banter, finances, other people's comments and the business's customer base removed.*

[Begin Interview]

Jason : Hi Auntie Ina, so I have this case upcoming case study thesis that I will need to start writing soon. I've been looking for ideas and your hydroponic farm seems like it would be a great object of study for my thesis. Do you mind if I use it and ask you some questions?

Ina : Hi Jason! Sure, of course. We've actually had someone else, I forgot who it was, who asked us the very same thing. He also wanted to use my farm for their thesis.

Jason : Do you mind giving me a tour of the place?

Ina : Yes sure! Everyone else can also come! We're planning to give you all a tour anyways, and at the end you can get your pick at some of our cancelled orders, on the house!

[*Tour* – Aunt Ina showed me (and quite a number of people from my extended family) her hydroponic farm, accompanied by her husband, Uncle Anton.]

Jason : Oh, so these whiteboards are where you write down the information about the [hydroponic] trays?

Ina : Yes, we measure the "PPM" of the hydroponic waters and write them down here for reference.

Jason : How do you measure the PPM levels?

Ina : Ah, I don't know too much about that, Uncle Anton usually handles this. <Calls uncle Anton over.>

Anton : Ah yes, we use handheld test kits, think I have them somewhere over here… Here it is. So you just dip this end into the water and… <explains how the tool works>.

Jason : So you only measure the "PPM"?

Anton : Yes, it's the "nutrient density". I don't really understand how the science behind it works, but I follow guides.

Jason : How about other things like CO_2 and temperature?

Anton : That's why we built this greenhouse around it. I don't know exactly how much CO_2 there is and is needed, but this greenhouse works wonders.

Jason : And you store the nutrient density notes on these whiteboards? What if they get wet, or accidentally wiped?

Anton : Well of course we will try not to let that happen, but what's the worst that could happen if it does get wiped? It's not like the crops are just going to die, it's still going to grow normally, it's just that we won't know exactly what's going on, until we measure it again that is, which is pretty simple. Plus, these greenhouses don't allow rainwater in, so the whiteboards won't get wet.

Jason : But what if I can come up with a way to store these notes like on a database or something?

Anton : It would be interesting, but I'm getting old, might be too much for me to learn.

Jason : Shouldn't be a problem, I should be able to make it very simple to use. You do have Wi-Fi here right?

Anton : Yes, but it's rather slow, and can be unreliable at times.

Jason : Hmm, its good that there is Wi-Fi, but the unreliability could be a problem…

[*Tour* – We moved to the far end of the farm complex, to the undeveloped lands around it.]

Ina : Well, our lands are quite big as you can see.

Jason : Wow, that's quite large, I wonder how much all of this costs.

Ina : Well, it's remote, with difficult access to almost everything really, so it's not too expensive. This whole entire land costs around IDR 2 bio.

Jason : Wow, that's really cheap for a land this massive, especially compared to Jakarta.

Anton : Yeah, that's the point, we plan to expand because demand is on the rise.

Jason : Wouldn't it get more and more tedious to individually monitor and care for all these greenhouses the more you expand?

Anton : Well yes, but that's why we have staff tending to the crops. Of course, it will eventually get difficult, but that's still far in the future, might even be a decade ahead, so we'll think about that when it comes to it.

Jason : I might be able to do something about that with my case study.

Ina : Really? We'll it'll be interesting to see what you can come up with, maybe we'll even pay you hahaha.

Jason : Yeah…, maybe…, haha…

Anton : Well, do you have any other questions?

Jason : No, I think that's it for now… thanks a lot for all of this. I know I must be bothering you.

Ina : Oh no not at all…, <proceeded to shift the conversation to about my university life.>

[End Interview]

Appendix 3: Additional Attachments

I have included the following attachments onto the final submission of this thesis:

- Arduino High-Level Architecture Diagram
- Web Application High-Level Architecture Diagram
- Class Diagram
- Empathy Map
- Process Cycle Diagram
- Use Case Diagram
- Value Proposition Canvas
- Arduino “Final” Schematic
- Schematic of ESP-01 in Bypass Mode
- Schematic of ESP-01 in SoftwareSerial Mode
- Schematic of the MH-Z19B in Analog Mode
- Schematic of the PH-4502C

Any tables, figures, or diagrams not attached in this appendix should be available directly in this report in high-resolution. Please zoom-in to view.

Appendix 4: Database Seed File

I will also be attaching the **.SQL** *seed* file, which was generated via *mysqldump* from my own local test database.

Appendix 5: Web Application Code, PHP Project Folder

I am attaching the web application’s source code in the form of the PHP project folder as an appendix for inspection.

Appendix 6: Hydroponic Sensor Code, Arduino Project Folder

I am also attaching the Hydroponic Sensor device’s source code in the form of the Arduino project folder.