BABEȘ-BOLYAI UNIVERSITY, CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION MATHEMATICS AND COMPUTER SCIENCE IN ENGLISH

Diploma thesis

An Irrigation System for a Smart Garden

Supervisor
Lect. Dr. Coroiu Adriana Mihaela

Author
Cremene Delia-Maria

2021

UNIVERSITATEA BABEȘ-BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA MATEMATICĂ-INFORMATICĂ ÎN LIMBA ENGLEZĂ

Lucrare de licență

Un sistem de irigare pentru o gradină inteligentă

Conducător științific
Lect. Dr. Coroiu Adriana Mihaela

Absolvent
Cremene Delia-Maria

2021

Abstract

As fresh water is a vital and limited resource there is a strong need to use it wisely especially when it comes to irrigating crops which accounts for around 70% of the worldwide water consumption [15]. In addition to this, people are showing an increasing interest in growing their own food to enjoy fresh fruits and vegetables. However, most of us do not have the time to permanently take care of our crops, which results in a bad harvest since adequate watering is the key of having quality crops. While there exist many irrigation solutions for large scale agriculture, so far there are only a few suitable and affordable systems for backyard gardening and personal use.

For these reasons we want to use the Internet of Things to build an affordable smart irrigation system that could easily be scaled to water a garden bed. Our hardware prototype consists of a NodeMCU board with an ESP8266 Wi-Fi microchip to which we connect sensors and actuators. We program our system to take into account the soil moisture and only water the plants when needed, that is, when the value read from the sensor indicates that the soil is starting to dry. Furthermore, we also set up a web server where we display and plot the real time sensor readings and, although the irrigation is automated, we also want to allow the users to turn the pump on and off by simply pushing a button when they consider necessary. To help with this decision we want to display the current weather as well, so that in event of a rain forecast for example, the pump can easily be turned off by pushing a button.

We tested our proposed system for a few weeks and compared it with a scheduled irrigation program that does not take into account the soil moisture. The conclusion was that our approach does not only save water, but is also adaptable to both the different water needs each plant might have and to the environmental conditions keeping the soil moisture around the same level, all thanks to the use of sensors. Moreover, the web server allows the user to monitor and control the system remotely, hence limiting the need for physical human interaction, saving time and effort.

Contents

Chapter 1

Introduction

People seem to have an increasing interest in growing their own vegetables and fruits to enjoy healthy organic food. However, most of them do not have the time to take care of the plants that end up dying. Underwatering holds back plant growth and reduces the quality of the crops, while overwatering leaches fertilizers leading to nutrient loss and reduces the aeration of the roots of the plant impairing crop growth. On the other hand, adequate watering results in the quality of the harvest. Furthermore, fresh water is a scarce resource that needs to be used as efficiently as possible.

So far there have not been any small scale and affordable solutions for backyard gardening, which is why we tried to implement an intelligent automated plant irrigation system that minimizes water waste, reducing cost and saving resources and that can be remotely controlled via a web server, therefore limiting the need for physical human interaction and saving time and effort.

For this project we designed a system using a NodeMCU ESP8266 Wi-Fi board, a soil moisture sensor, an air humidity and temperature sensor and a water pump. We used Firebase as a NoSQL database to store real-time data read from the sensors. We also developed a web sever that allows the user to see real-time data and plots, turn the relay that controls the pump on and off and consult the weather forecast.

Most automated irrigation systems usually do not have a soil moisture sensor and the pump is scheduled to be turned on at regular intervals of time for a predefined period of time, such as for a few minutes every day. Programming such a system is based on experience and it is not adaptable to weather changes or plant needs, demanding constant adjustments. Our system proposes a different approach that only waters the plant for a small amount of time when the soil moisture drops below a lower limit, therefore attending to the plants needs, adapting to environmental conditions and also saving water.

The setup was tested thoroughly and it works as it is supposed to, monitoring the soil moisture, the air humidity and temperature levels and controlling the pump, keeping the soil moisture levels of the plant optimal.

Moreover, we have also tested for a few weeks each of the two aforementioned approaches to irrigation programs, that is scheduled irrigation and intelligent irrigation and compared the soil moisture in the two cases by plotting graphs of the data we stored in our database. Our findings confirmed the hypothesis that intelligent irrigation is both better for the plant and helps save water as you will see in the next chapters.

1.1 Paper structure

In Chapter 2 we present a small overview of the methods used to solve the problem of intelligent irrigation until now and we describe our contribution that tries to improve those previous approaches by taking the relevant ideas from each one of them and combining them into one system. Then, in Chapter 3, we theoretically describe the hardware components and technologies used by us. Here we also go into details about our proposed system, describing both how we built the hardware prototype and how we implemented the software. In Chapter 4 we present two different approaches to the problem of saving water while obtaining healthy crops: scheduled irrigation and intelligent irrigation. Then, in the same chapter, we compare the two with an experiment to see whether our approach is better. In the end of the paper, in Chapter 5 we conclude our findings and talk about further work.

Chapter 2

Related work

Since saving water and automating the process of irrigation are topics of great importance and interest, before starting our project we found and read many articles related to this. In our work, we tried to take the most relevant concepts and methods described by the articles we are going to present below and combined them into an innovative system that not only irrigates, but also allows the user to monitor and control this process.

The article [30] made us realise the importance of designing a system that helps saving water since it is such a scarce resource and of adequate watering that dictates the quality of the harvest. It also states that there are few affordable and scalable irrigation systems for personal use in backyard gardening and emphasizes the multidisciplinarity of such projects. To solve this problem they build an easily scalable with do-it-yourself (DIY) attainability prototype that consisted of a soil moisture sensor, an air humidity and temperature sensor, water pumps, a relay, an Arduino UNO board and a NodeMCU board as a Wi-Fi module. As far as the implementation of the software went, they only stated that they stored the sensor data in a local database and build a web server that allowed saving water through data visualization, but they did not go into further details.

A similar approach based on the Internet of Things could be seen in [22] where they used a NodeMCU Wi-Fi module to which were connected a soil moisture sensor, a humidity and temperature sensor and a pressure sensor. Their aims were to reduce water consumption, project cost, labour and power consumption and to build a reliable system. Therefore, they developed an irrigation system that reduces water usage by automating the process to take into account the soil moisture. Sensor data was collected and uploaded to Firebase and ThingSpeak.com where it was displayed in the form of graphs. They also built a web sever that allowed the user to monitor real time sensor data and to control the water pump, choosing between automating the irrigation or turning the pump on or off.

An article that was thought-provoking was [1] where there was presented a comparison between a common irrigation programmer and a smart irrigation approach. Here they defined common irrigation as "irrigating at regular time intervals for predefined periods of time" and the smart irrigation as using sensors and irrigating according to the sensor data, watering the plants when the soil moisture level gets below the lower bound that indicates that the soil is dry and until reaching an upper limit that indicates that the soil is wet. They made an experiment to compare the two approaches, abstracting a garden bed with a few potted plants with different water needs and concluded that in the case of scheduled irrigation, the soil tends to dry out and then is flooded with water resulting in extreme variations in soil moisture levels that are not beneficial to plants. On the other hand, the smart irrigation program managed to maintain the soil moisture at the same level, adapting to the plant's needs and to environmental conditions.

Another interesting article [21] reinforced the fact that the market price for a small area irrigation system is quite high and that there is need for an affordable, easy to build alternative. They also emphasized the need for thoroughly testing the prototype to see if it works as intended. Their prototype consisted of a Raspberry Pi with a Wi-Pi Adapter to which were connected light and soil moisture sensors, a solenoid valve and an ultrasonic sensor that measures the water levels from the water tank. The proposed solution monitors and controls the plants based on environmental factors such as weather and soil moisture. The user can remotely control the irrigation system from their smartphone through an Android app and they are also notified via email when the water levels from the water tank are low.

In [31] it is proposed that the plants should be watered when the soil is dry and until the soil moisture reaches an upper limit that indicates that the soil is wet. They accentuate the need for trials on different soil conditions in determining the limits that suggest that the soil is dry or wet. The need for properly testing the system is emphasized here as well.

Moreover, for future work we will take into considerations the articles [23, 27] that point out that current approaches that only take into account the soil moisture might be a little simplistic since there are other factors that play an important role in the water needs of plants as well.

The article [27] proposes a fuzzy logic decision making approach that takes into consideration four sensor data: soil moisture, humidity, temperature and light. They also store data about the plant type and needs and the weather condition and geographic area.

2.1 Original contributions

All things considered, what we did was to build a reliable, scalable and affordable prototype that helps reduce water wastage and the need for physical human interaction, while also adapting to the plant's needs and environmental conditions. For this we designed a system that uses a NodeMCU Wi-Fi board to which we connected a soil moisture sensor, an air humidity and temperature sensor and a submersible water pump and tested it thoroughly to see if it behaves as expected. We wrote an intelligent irrigation program that takes into account the soil moisture values when deciding whether to water the plants or not, automating the process of irrigation. We also built a web server that allows the user to monitor real time data and plots of the sensor readings and the current weather and also to control the pump turning it on and off. On top of that, we made a little experiment similar to the one done by [1] to compare our implementation to a scheduled irrigation approach that does not take into account the soil moisture.

Chapter 3

System description

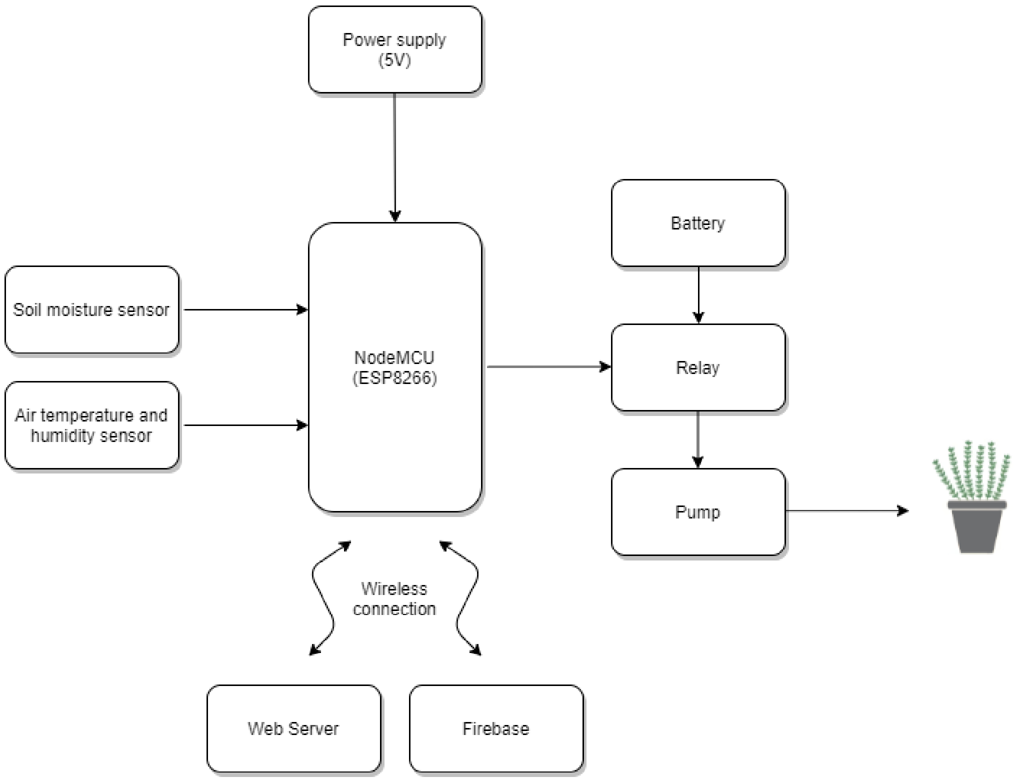


Figure 3.1: Block diagram of the system

Our system's architecture consists of a NodeMCU ESP8266 Wi-Fi board to which we connected a led, sensors and actuators, more precisely an air humidity and temperature sensor (DHT22), a capacitive soil moisture sensor and a relay that is connected to a water pump and is powered separately by a battery, as can be seen in the block diagram of the system from Figure 3.1. The board is programmed with a sketch written in Arduino IDE that allows us to read data from the sensors, connect to a wireless network and send real-time data to our database (Firebase) to be stored and also to our web server via HTTP protocols. The web server consists of HTML, CSS and JavaScript code that is stored on the board using the LittleFS filesystem plugin and it allows us to control the relay and start the water pump when necessary via two buttons and also see real time sensor data and plots and a display of the current weather using the OpenWeatherMap API.

Regarding the algorithms used, we wanted to test whether an intelligent watering program that takes soil moisture into account is better than an experience based scheduled one in terms of taking better care of the plant's needs and saving water. For the purpose of our experiment we abstracted a backyard garden with a few pots containing herbs that have different water needs, more precisely a mint plant that needs less water and a sage plant that prefers to be watered more often.

In the case of smart irrigation, each pot is monitored with a soil moisture sensor that reads data every 10 minutes and when the soil is too dry, the pump is turned on for a few seconds, then the program waits for 10 minutes for the water to distribute evenly into the soil and then measures its moisture again and waters for a few seconds if the soil moisture is still below the lower limit that indicates that the soil is drying and so on.

3.1 Hardware description

3.1.1 Internet of Things

By now, we cannot imagine a world without the Internet. As of January 2021, according to Statista [32], almost 4.66 billion people were active Internet users, which represents 59.5 percent of the global population. While more and more people will gain access to the Internet, there has also been an increase in the number of machines and smart objects that communicate, dialogue, compute and communicate using the Internet. This innovation is possible by embedding electronics into everyday physical objects, making them "smart" and integrating them seamlessly within the global infrastructure. From a conceptual point of view, the IoT (Internet of Things) is built on three pillars, that is, the ability of smart objects to: *be identifiable, to communicate* and *to interact.*

The IoT vision consists of shifting from an Internet used for interconnecting end-user devices to an Internet used for interconnecting physical objects that communicate with each other and/or with humans in order to offer a given service. The challenge is to rethink anew some of the conventional approaches used in networking, computing and service provisioning/management and to develop technologies and solutions to enable such a vision [26].

A key aspect of a smart object is its ability to sense physical phenomena (e.g. temperature, light, humidity etc.) and translate them into a stream of information data, providing information on the current context/environment or to trigger actions having an effect on the physical realm (actuators). From a conceptual standpoint, IoT is about entities acting as providers and/or consumers of massive amounts of data and information related to the physical world (rather than communication). In order to turn the data into useful information it is necessary provide data with adequate and standardized formats, using well-defined languages and formats. This will enable IoT applications to support automated reasoning which will be useful for adopting these technologies on a wide scale.

The IoT technologies can provide advantages over current solutions in many sectors such as: environmental monitoring, smart cities, smart business/inventory and product management, smart homes/buildings management, healthcare, security and surveillance. In the field of smart homes/buildings some advantages offered by adopting advanced IoT technologies may be reducing the consumption of resources associated to buildings (water, electricity) as well as improving the satisfaction of humans. Again, the key role is played by sensors (light, temperature etc.) which are used to both monitor resource consumption as well as to proactively detect the user's needs and expectations and take decisions (e.g. switch on/off lightning, heating, cooling etc.).

The possibility of seamlessly merging the physical world and the virtual world through the multitude of embedded devices opens up new exciting directions for research and business. Moreover, the IoT vision provides many opportunities to users. From a user standpoint, the IoT enables a large number of always responsive services that answer to the user's needs, offer support in everyday activities and are also able to autonomously react to a wide range of different situations in order to minimize human interaction.

For our project we used the Internet of Things to build a small scale prototype of an irrigation system that is embedded with sensors, actuators and software allowing us to connect and exchange real-time data over the Internet. The circuit diagram is shown in Figure 3.10.

The components of the circuit are:

3.1.2 NodeMCU ESP8266

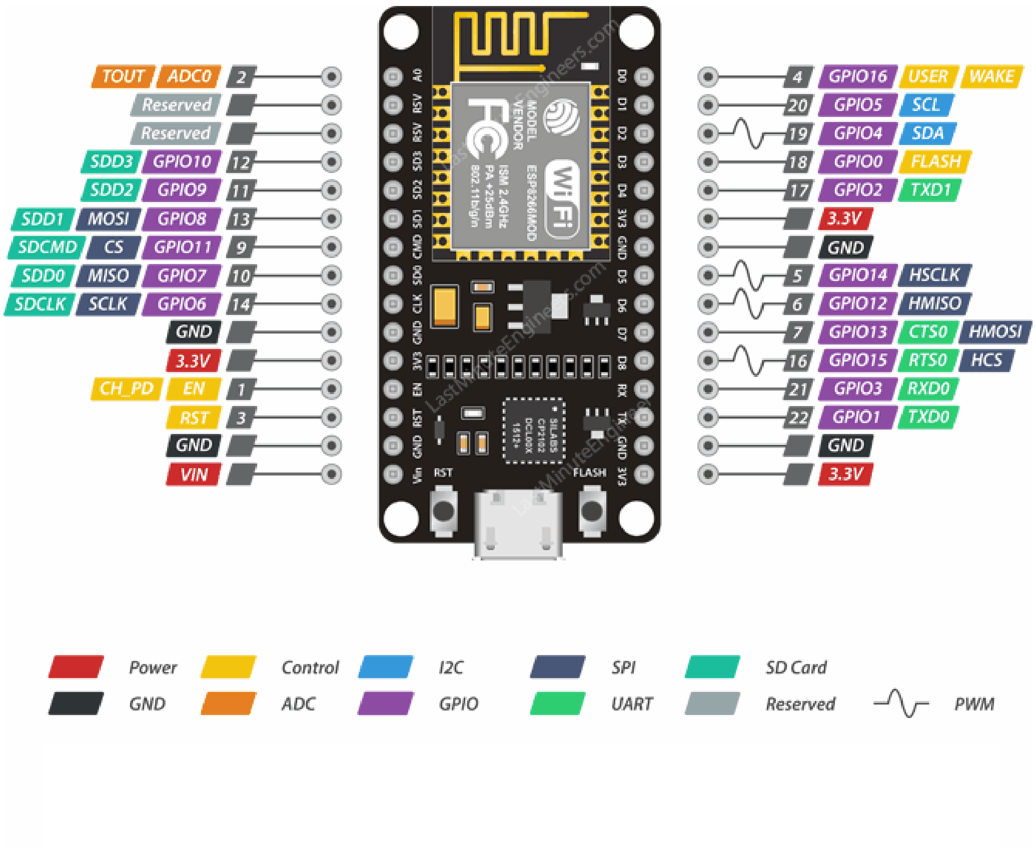
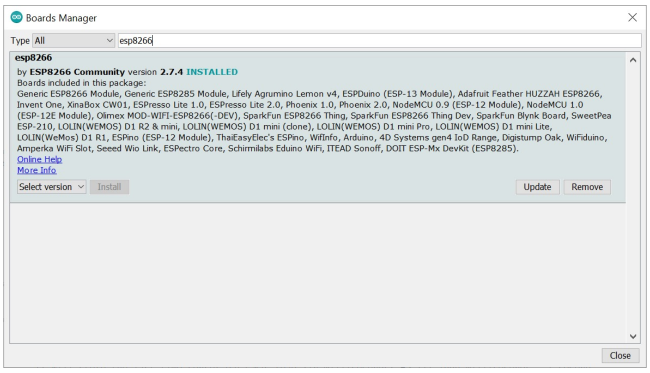


Figure 3.2: Pinout of a NodeMCU ESP8266 microcontroller [24]

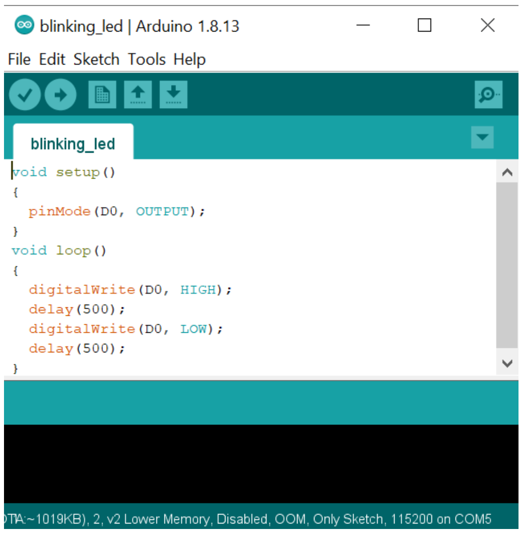
The different components of our prototype are connected with a NodeMCU that is a low-cost open source IoT platform. An IoT platform bridges the gap between device sensors and data network. The firmware runs on the ESP8266 Wi-Fi which is a low-cost Wi-Fi microchip with a full TCP/IP stack. This module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections. It uses many open source projects, such as lua-cjson and SPIFFS [36]. The ESP8266 NodeMCU has a total of 30 pins that can be seen in Figure 3.2.

To test the NodeMCU we first installed the latest Arduino IDE version and then added to it the ESP8266 Package [11] and installed it going to Tools> Boards>Boards Manager and typing "esp8266". Since the D0 pin on the board is connected to the on-board blue LED and can be programmed, we uploaded a simple sketch that makes the LED blink. All this can be seen in Figure 3.3.



esp8266
by **ESP8266 Community** version **2.7.4** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, Lifely Agrumino Lemon v4, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini (clone), LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, LOLIN(WeMos) D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmilabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).

((a)) Adding the ESP8266 add-on for Arduino that supports the NodeMCU board



```
void setup()
{
  pinMode(D0, OUTPUT);
}
void loop()
{
  digitalWrite(D0, HIGH);
  delay(500);
  digitalWrite(D0, LOW);
  delay(500);
}
```

```
TA:~1019KB), 2, v2 Lower Memory, Disabled, OOM, Only Sketch, 115200 on COM5
```

((b)) A sketch that makes the on-board LED blinky

Figure 3.3: Testing the NodeMCU board

3.1.3 Capacitive soil moisture sensor

Soil moisture is the content of the water present in the soil. It can be influenced by precipitations, temperature, humidity and so on. Having an appropriate soil moisture is a key detail in optimal plant development. For an intelligent irrigation this might be the most important aspect to take into consideration. There are two popular sensors that are used in similar projects: resistive and capacitive soil moisture sensors. Both of them operate on either 3.3V or 5V, making them easy to pair with our board, but there are also some major differences that should be taken into consideration before settling for one of them as can be seen in Table 3.1.

Table 3.1: Main differences between the soil moisture sensors

Resistive soil moisture sensor	**Capacitive soil moisture sensor**
• measures the resistence and depends on the amount of water in the soil because water is a natural conductor for electricity (the lower the measured resistance, the higher is the amount of water in the soil)	• measures the changes in capacitance caused by the changes in the dielectric contrast between water and soil
• due to the contact with water, electrolysis damages the sensor and makes the sensor inaccurate in a relatively short period of time	• the major advantage of the capacitive sensor is that there is no direct exposure of the metal electrodes and therefore there is no electrolysis that damages the sensor through corrosion

Although the price for a resistive soil moisture sensor is very small, the corrosion interferes

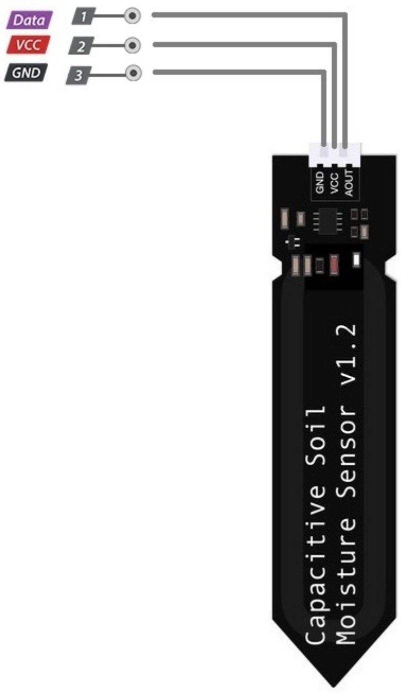


Figure 3.4: Pinout of a capacitive soil moisture sensor [24]

with its accuracy, making it good for rather less frequent readings, but not a good option for this project where we had to read data from plants for a few weeks. Regarding the capacitive soil moisture sensor, the lack of corrosion makes it a great option for frequent readings and therefore for our project.

After deciding that a capacitive soil moisture sensor that can be seen in Figure 3.4 suits our needs best, we connected it to NodeMCU and tested it in different extreme environments and made the observations that can be seen in Table 3.2.

Table 3.2: Soil moisture (raw values) in some special cases

Environment	**Raw value**
air	740
water	300
wet soil	286
dry soil	560

Since the raw values seem a little counter-intuitive because a higher raw value actually represents a low moisture and a lower value actually means high moisture and also because we are only interested in measuring the moisture of the soil (not air or water), we wanted to calibrate the sensor for soil only. For example the 286 value obtained in wet soil should be mapped to 100% and the 560 value obtained in dry soil should be mapped to 0%. For this we used the *map()* function provided by Arduino [2] which maps an interval into another interval by taking a value of fromLow and mapping it to toLow, a value fromHigh and mapping it to toHigh and also mapping the values in-between:

$$map(value, fromLow, fromHigh, toLow, toHigh)$$

does the following:

$$(value - fromLow) * (toHigh - toLow)/(fromHigh - fromLow) + toLow$$

where:

- *value*: the number to map
- *fromLow*: the lower bound of the value's current range
- *fromHigh*: the upper bound of the value's current range
- *toLow*: the lower bound of the value's target range
- *toHigh*: the upper bound of the value's target range

3.1.4 Air humidity and temperature sensor: DHT22

There are two popular air humidity and temperature sensors that are used in similar projects: DHT11 and DHT22. Both are lab calibrated, accurate, stable, have digital outputs and are relatively inexpensive for the given performance. They both operate on 3.3V to 5V, have three or four pins and can be used with our NodeMCU. Both of them measure the relative humidity which is the ratio of how much water vapour is in the air and how much water vapour the air could potentially contain at a given temperature and is given by the formula:

$$RH = \frac{\rho_w}{\rho_s} * 100\%$$

where:

- RH: the relative humidity
- ρ_w: actual water vapor density
- ρ_s: saturation water vapor density

Humans usually prefer a air humidity between 30-50%, while the ideal humidity for houseplants is 40-60%. At 100% relative humidity, the air is saturated and is at its dew point.

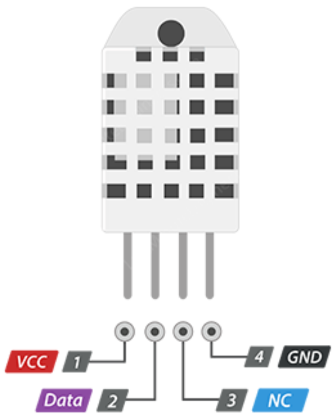


Figure 3.5: Pinout of a DHT22 soil moisture sensor [24]

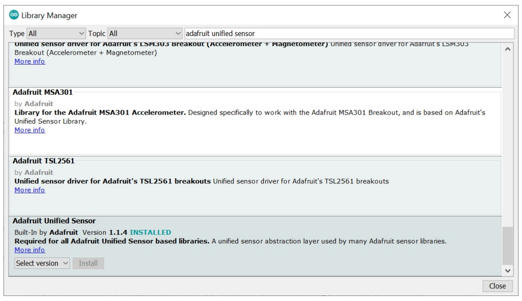
The main difference between these two is on the temperature and humidity range and sampling rate as you can see in Table 3.3 which was inspired by [20].

Table 3.3: Main differences between the DHT sensors

Parameter	**DHT11**	**DHT22**
Temperature range	0 to 50°C / ±2°C	-40 to 125°C / ±0.5°C
Humidity range	20 to 80% ±5%	0 to 100% $\pm 2 - 5\%$
Sampling range	Once a second	Twice a second

As can be seen, the DHT22 is an upgrade over the DHT11, being a little more accurate and good over a slightly larger range. We started the project with a DHT11 sensor but then when we made another prototype of the irrigation system we switched to the DHT22 sensor for more accuracy.

To use these we have to use Adafruit's DHT22 library. It can be installed using Arduino's library manager. It comes in two components: the Adafruit Unified Sensor library and the DHT sensor library as can be seen in Figure 3.6.


Library Manager

Type All Topic All adafruit unified sensor

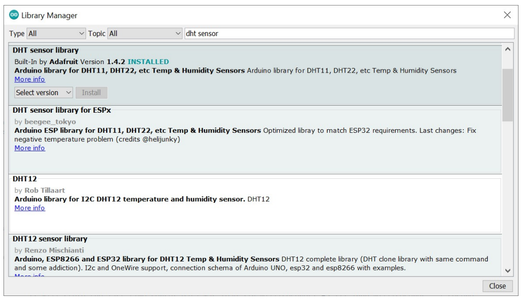
Unified sensor driver for Adafruit's LSM303 Breakout (Accelerometer + Magnetometer) Unified sensor driver for Adafruit's LSM303 Breakout (Accelerometer + Magnetometer)
More info

Adafruit MSA301
by **Adafruit**
Library for the Adafruit MSA301 Accelerometer. Designed specifically to work with the Adafruit MSA301 Breakout, and is based on Adafruit's Unified Sensor Library.
More info

Adafruit TSL2561
by **Adafruit**
Unified sensor driver for Adafruit's TSL2561 breakouts Unified sensor driver for Adafruit's TSL2561 breakouts
More info

Adafruit Unified Sensor
Built-In by **Adafruit** Version **1.1.4** INSTALLED
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

((a)) Adafruit Unified Sensor Library


Library Manager

Type All Topic All dht sensor

DHT sensor library
Built-In by **Adafruit** Version **1.4.2** INSTALLED
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
More info

DHT sensor library for ESPx
by **beegee_tokyo**
Arduino ESP library for DHT11, DHT22, etc Temp & Humidity Sensors Optimized libray to match ESP32 requirements. Last changes: Fix negative temperature problem (credits @helijunky)
More info

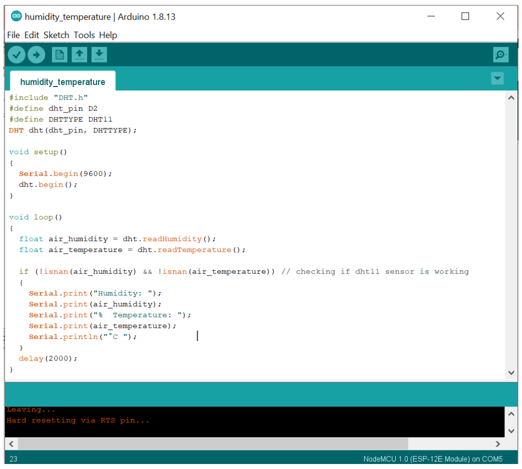
DHT12
by **Rob Tillaart**
Arduino library for I2C DHT12 temperature and humidity sensor. DHT12
More info

DHT12 sensor library
by **Renzo Mischianti**
Arduino, ESP8266 and ESP32 library for DHT12 Temp & Humidity Sensors DHT12 complete library (DHT clone library with same command and some addiction). I2c and OneWire support, connection schema of Arduino UNO, esp32 and esp8266 with examples.

((b)) DHT Sensor Library

Figure 3.6: Adding the libraries needed for the DHT sensors

After installation, we then proceeded to test the sensor which was done with a relatively simple sketch and can be seen in Figure 3.7.

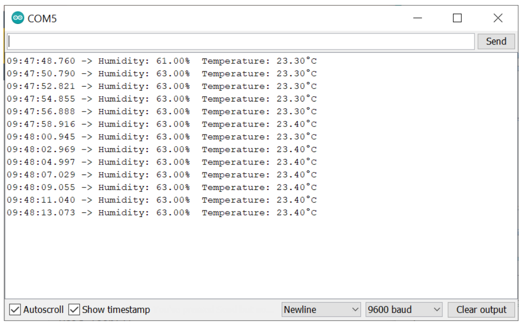

humidity_temperature | Arduino 1.8.13

```
#include "DHT.h"
#define dht_pin D2
#define DHTTYPE DHT11
DHT dht(dht_pin, DHTTYPE);

void setup()
{
  Serial.begin(9600);
  dht.begin();
}

void loop()
{
  float air_humidity = dht.readHumidity();
  float air_temperature = dht.readTemperature();

  if (!isnan(air_humidity) && !isnan(air_temperature)) // checking if dht11 sensor is working
  {
    Serial.print("Humidity: ");
    Serial.print(air_humidity);
    Serial.print("%  Temperature: ");
    Serial.print(air_temperature);
    Serial.println("°C ");
  }
  delay(2000);
}
```

```
Leaving...
Hard resetting via RTS pin...
```

23 NodeMCU 1.0 (ESP-12E Module) on COM5

((a)) Code


COM5

```
09:47:48.760 -> Humidity: 61.00%  Temperature: 23.30°C
09:47:50.790 -> Humidity: 63.00%  Temperature: 23.30°C
09:47:52.821 -> Humidity: 63.00%  Temperature: 23.30°C
09:47:54.855 -> Humidity: 63.00%  Temperature: 23.30°C
09:47:56.888 -> Humidity: 63.00%  Temperature: 23.30°C
09:47:58.916 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:00.945 -> Humidity: 63.00%  Temperature: 23.30°C
09:48:02.969 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:04.997 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:07.029 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:09.055 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:11.040 -> Humidity: 63.00%  Temperature: 23.40°C
09:48:13.073 -> Humidity: 63.00%  Temperature: 23.40°C
```

✓ Autoscroll ✓ Show timestamp Newline 9600 baud

((b)) Sensor readings

Figure 3.7: Testing the DHT sensor

3.1.5 Relay

A relay is an electrically operated switch. Relays are used when needing to control a high current circuit by a low current one. Since our NodeMCU operates at 5V, it cannot control a higher voltage directly, but we can use a 5V relay to manage it. One of the most useful things we can do with a board is to program it to manage the relay. Although our pump only operates on 3-6V, it has to be turned on and off when needed. The NodeMCU can be programmed to turn on the relay for example when the soil moisture drops below a certain level, which in turn will control the pump that will water the plant.

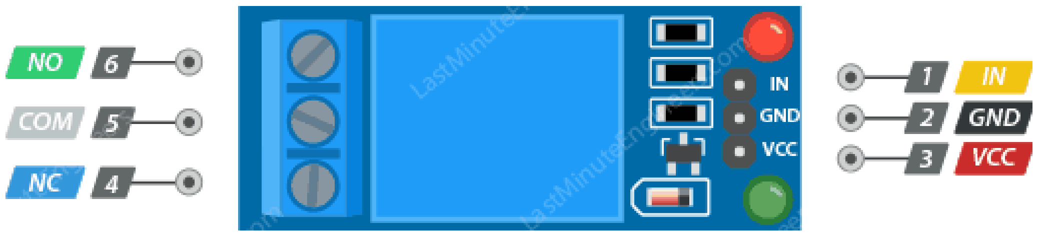


Figure 3.8: Pinout of a one channel relay module [24]

The relay can work in two modes: normally open and normally closed. The one that suits our project best is the normally open mode. For this we had to connect the pump and the battery to the COM (Common) and NO (Normally Open) pins that can be seen in Figure 3.8.

In the normally open relay configuration the default is the open position, meaning that there is no contact between the circuits. When power is supplied, the first circuit is pulled into contact with the second by an electromagnet, closing the circuit and allowing power to flow through. When power is turned off, the circuit is again opened and there is no current flowing. In code, if the relay receives a LOW signal, the switch closes and allows current to flow. A HIGH signal opens the circuit again, stopping the current flow.

3.1.6 Submersible water pump

The mini submersible water pump used in our prototype operates on 3 to 6V and is said to pump up to 120 liters per hour. Since we only needed to pump a little water for a single plant in a pot for our project, we had to do a small experiment to find out the daily needs of a plant and the actual interval of time in which that specific amount of water is delivered by the pump, so that our plant will be properly irrigated.

To decide how much water does a plant need in a day we consulted the guide for determining water needs proposed by [17] which successfully grew petunias with less then 2 litres of water applied over 40 days. We thoroughly watered each of the plants and let them drain for 30 minutes. Then we weighted the pots, came back 24 and 48 hours later and weighted the pots again. The decrease in weight is the amount of water that has been used by the plant and evaporated. The main environmental factors that play a role in this are temperature, relative

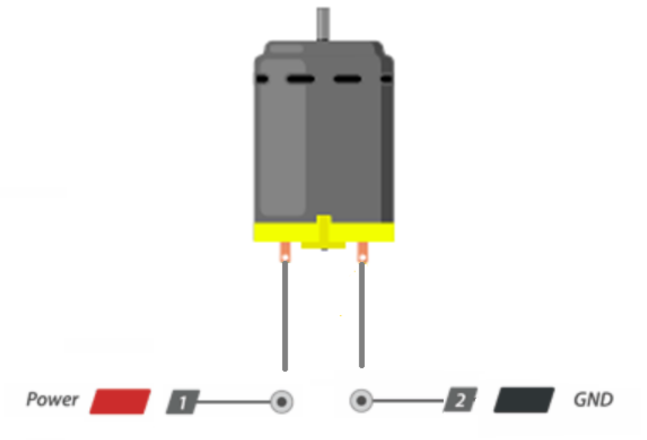


Figure 3.9: Pinout of the submersible water pump [24]

humidity, light and plant size. The pot size also plays an important role in the amount of water needed, but we eliminated that variable by using same sized pots. The water needs of our plants can be seen in Table 3.4. The plants were chosen to be herbs because they are resilient and can be used in recipes, combining our two passions: gardening and cooking.

Table 3.4: Water needs of our herbs

Herb	**Initial weight**	**After 24h**	**After 48h**	**Avg. needs**
Mint	272g	232g	196g	48g
Sage	439g	367g	301g	69g
Oregano	325g	275g	237g	44g
Basil	292g	262g	236g	28g
Rosemary	349g	329g	311g	19g
Tarragon	295g	271g	253g	21g

After finding out the water needs for each of our plants, we also had to find out the amount of water that is delivered in a specific amount of time. Our finding was that the system delivers 45-50 ml of water in 2 seconds when the plant and the pot were on the same level.

3.1.7 The circuit

The circuit was assembled one component at a time and after adding each sensor and actuator to the NodeMCU board taking into account their pinouts that could be seen in the previous figures, we wrote the sketch for each component and checked if it was working. The final result can be seen in Figure 3.10.

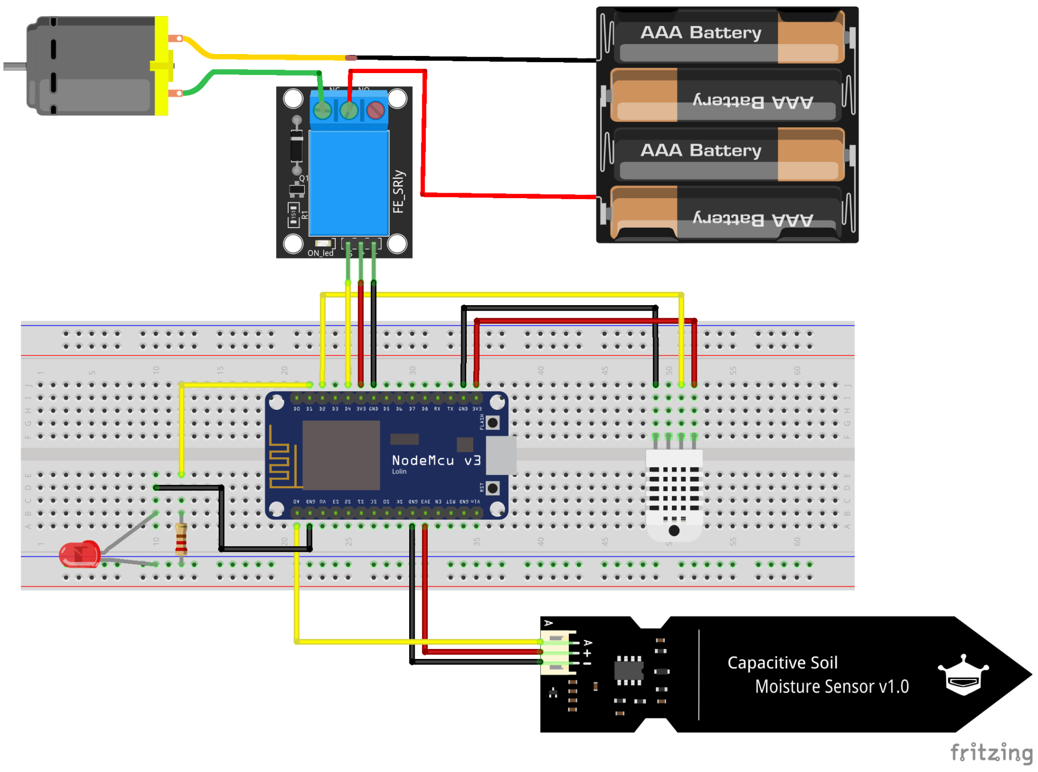


Figure 3.10: The circuit diagram of the system

3.2 Software description

To program our prototype we used Arduino IDE. Our implementation is trying to irrigate plants in an intelligent manner taking into account the soil moisture sensor readings and only watering the plant when the soil is drying. After implementing the intelligent irrigation system, we also want to be able to remotely monitor and control it, so we build a web server. For this we store the HTML, CSS and JS code as separate files from the sketch using the LittleFS filesystem. The web server allows the user to control the water pump on and off by pushing two buttons and displays the real-time sensor readings, also plotting the last few readings in real time using data that is stored in Firebase. It also displays the weather using the API from OpenWeatherMap. All this can be seen in Figure 3.11.

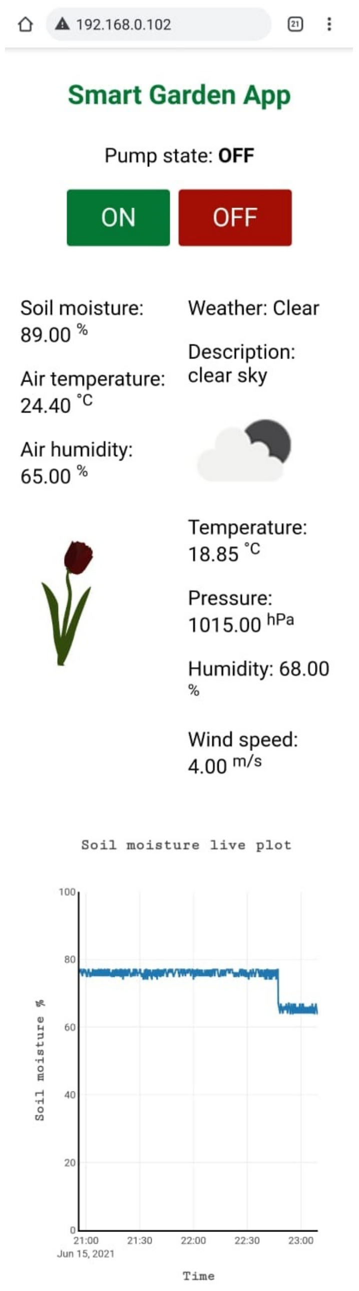


Figure 3.11: Our Web Server

3.2.1 Arduino IDE and Arduino Programming Language

The open-source Arduino Software Integrated Development Environment (IDE) provides us with a programming editor with integrated libraries support that make it easy to write code and with a simple way to upload our programs to Arduino compatible boards that are connected to the computer. It can be used with any Arduino board and, moreover, the ESP8266 Community provides a package [11] that can be added using Additional Boards Manager in Arduino IDE allowing us to program our NodeMCU ESP8266.

The IDE supports a language called Arduino Programming Language which is actually a framework built on top of C++. A program written in this language is called a sketch and is saved with the .ino extension. The main difference from C/C++ is that here we wrap our code into at least two main functions. The two mandatory functions are setup() which is called once and loop() which is repeatedly called while the program is running. Everything else is normal C/C++ code, but we do not have a main() function as we used to have in C/C++. Another feature of the Arduino Programming Language are the built-in libraries that allow the developer to integrate the functionality provided by the Arduino board [9].

The Arduino Programming Language has three main parts: **functions** for controlling the board and performing computations like the ones handling digital and analog I/O or the time functions such as delay() and millis(), **variables** (data types and constants) such as the built-in constants HIGH and LOW for the voltage levels and **structure** such as the program lifecycle functions setup() and loop() [4].

3.2.2 Firebase

Firebase is “Google’s mobile application development platform that helps you build, improve, and grow your app” [7]. Firebase is a Backend-as-a-Service (Baas) built on Google’s infrastructure. It provides a range of tools and services that help programmers develop quality apps and expand them based on demand. Firebase is currently among the top app development platforms relied upon by developers across the globe.

Some of the most important features of Firebase include databases, authentication, file storage, analytics, hosting and more. Furthermore, the services are cloud-hosted. With this platform Android, iOS and Web applications can be developed. Some of its advantages are the fact that it is free to start with a Google account, serverless, secure, has automatic regular backups, offers machine learning capabilities and much more.

For our project we chose the Firebase Realtime Database [8] which is a NoSQL cloud-hosted database. With it we can store and sync data in realtime even when the app goes offline. Once the connection is reestablished, the data is synchronized. The data is stored as JSON-like documents. A document is a set of key-value pairs and group of documents makes up a collection. The data is synchronized in realtime to every connected client and even in cross-platform apps all of the clients share one Realtime Database instance and automatically receive updates with the newest data, making it a great option for scaling our app in the future.

3.2.3 HTML, CSS and JS

The HyperText Markup Language (HTML) is the one of the fundamental technologies of the Web. HTML is a standard markup language used for documents that are meant to be displayed in a web browser. Web browsers receive HTML files from web servers or local storage and render those documents into multimedia web pages. HTML describes the content and structure of the web content. HTML provides a way to build structured documents with the help of HTML elements that are delimited by tags which are not displayed by the browsers, but interpreted [34].

Cascading Style Sheets (CSS) can be included alongside HTML to describe the web page's look and layout. CSS is a style sheet language that is used to specify the appearance of a document written in markup language, including layout, colors and fonts. It is designed to separate the HTML content from its design to enable multiple web pages to share the same formatting, reducing repetition and providing more accessibility, control and flexibility [33].

JavaScript programs can also be embedded in HTML to describe the web page's functionality/behaviour. JavaScript (JS) is a high-level programming language that is compiled during execution. It is object-oriented and has a curly-bracket syntax similar to Java. It has API's that help with working with text, dates, regular expressions and data structures [35].

3.2.4 LittleFS

LittleFS is a lightweight file system created for microcontrollers that allows the access to the flash memory just like in a standard file system on the computer without the need of any external memory. On a board, while the file system is stored on the same flash chip as the program, the file system contents are not modified by programming a new sketch, but we can still use the file system to store sketch data, configuration files or documents for the web server on the NodeMCU ESP8266 using a plugin [10] in Arduino IDE. Our other option would be to store the content needed for the web server as strings in the sketch, but that would not be best practice.

SPIFFS (Serial Peripheral Interface Flash File System) and LittleFS (Little File System) are two file systems for using the on board flash on our NodeMCU ESP8266. The original file system, SPIFFS, is currently deprecated, while LittleFS is under active development, having higher performance and directory support and being many times faster for most operations [3], making it the best choice for our application. The same methods used for SPIFFS still apply to LittleFS, so we can just replace the inclusion of SPIFFS.h with LittleFS.h and the keyword SPIFFS with LittleFS in methods.

3.3 Our approach

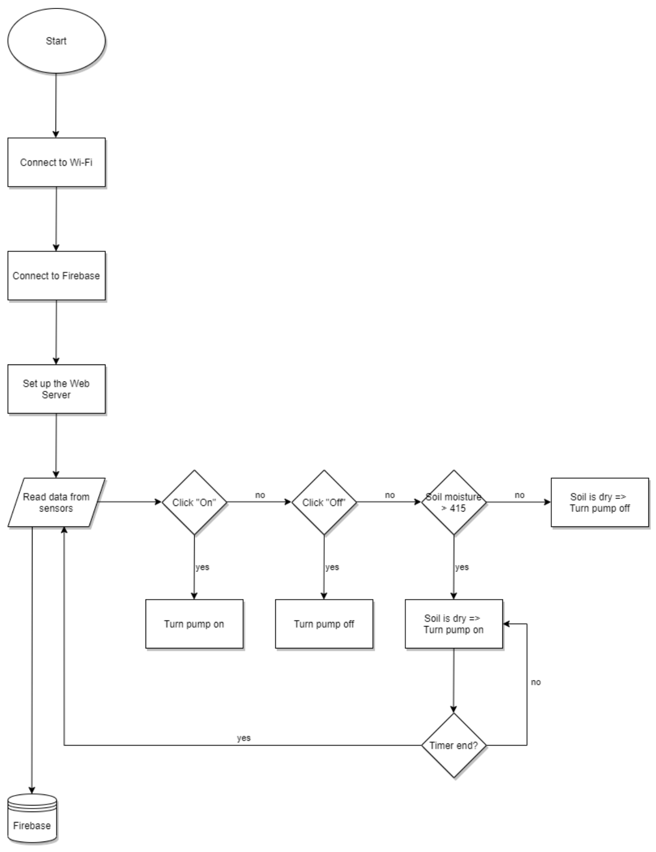


Figure 3.12: Flow diagram of the system

Many irrigation systems are programmed to water the plants according to a schedule that is based on experience and observation, turning the pump on at regular intervals for an amount of time knowing the quantity of water that is delivered in that time and having to take into account the plant's specific needs. However, this approach is not adaptable to environmental conditions and also has to be programmed according to each individual crop, needing frequent adjustments.

Our approach tries to solve these problems by using a soil moisture sensor and watering the plant only when needed, that is, when the value read from the sensor indicates that the soil is starting to dry. For this we only need to know the approximate value that shows that the soil is drying and the amount of water delivered by the system in a time interval.

Furthermore, we want to be able to see the real time sensor data values in a web server where we can also turn the pump on and off, see real time plots of our sensor data that we stored in Firebase's Realtime Database and consult the current weather allowing the user to take the decision of watering the plant now or to wait for it to rain in order to save water. By having a web server, we limit the need for physical human interaction, saving time and effort. How the system works can be seen in the flow diagram of the system in Figure 3.12.

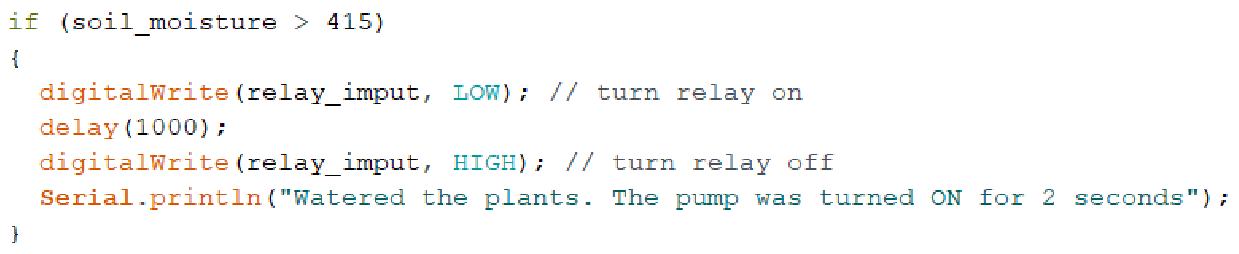
As you will see in Chapter 4 our approach is more efficient because it is not only automated, but also adaptable to environmental conditions such as rainfall and temperature and to the individual plant's needs by monitoring the soil moisture. Moreover, it also saves water which is a vital resource that needs to be used wisely.

The program was built incrementally as it follows:

3.3.1 Write the intelligent irrigation code

We started this project by connecting each of the sensors and actuators to the NodeMCU ESP8266 board one at the time and then installing needed sensor libraries for Arduino IDE and writing some code to test each component individually as seen in Section 3.1. After successfully being able to read each sensor data and print it on the serial monitor from Arduino IDE, we proceed to write the code for the intelligent irrigation.

Our irrigation program is fairly simple. It reads sensor data every 10 minutes, then compares the current soil moisture to the limit that indicates that the soil is drying. By observation we found this limit to be somewhere around 415 raw value. If the soil moisture is 415 or above, then the soil is in need of watering, therefore we send a LOW signal to the relay that will close the circuit, turning the water pump on for 2 seconds as can be seen in Figure 3.13, which will deliver approximately 40-50 ml of water to the plant, amount of water which we found to be somewhere around the average daily need for our potted herbs. However, if this amount of water is not enough, 10 minutes later when sensor data is read again, if the soil is still too dry, the system will deliver water for 2 seconds again, and so on, making the approach adaptable to the plant's needs.



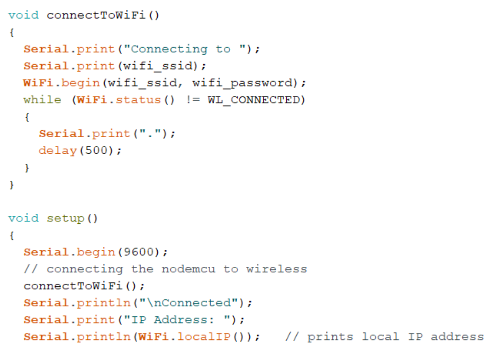
```
if (soil_moisture > 415)
{
  digitalWrite(relay_imput, LOW); // turn relay on
  delay(1000);
  digitalWrite(relay_imput, HIGH); // turn relay off
  Serial.println("Watered the plants. The pump was turned ON for 2 seconds");
}
```

Figure 3.13: Programming the relay

3.3.2 Connecting to Wi-Fi

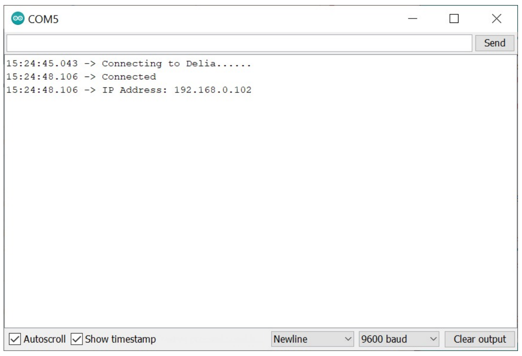
The next important step is connecting to the Wireless Network because the rest of the project requires an internet connection. We can do this thanks to the ESP8266 Wi-Fi microchip that is included in our NodeMCU board.

To write the actual code that allows us to connect to Wi-Fi we have to include the ESP8266.h and declare our Wi-Fi Service Set Identifier (SSID) and Password. We then write a method for this that we call in the setup() function and print the IP address which will be useful when we will want to access the web server. All this can be seen in Figure 3.14.



```
void connectToWiFi()
{
  Serial.print("Connecting to ");
  Serial.print(wifi_ssid);
  WiFi.begin(wifi_ssid, wifi_password);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
}

void setup()
{
  Serial.begin(9600);
  // connecting the nodemcu to wireless
  connectToWiFi();
  Serial.println("\nConnected");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());   // prints local IP address
```

((a)) Code



COM5

```
15:24:45.043 -> Connecting to Delia......
15:24:48.106 -> Connected
15:24:48.106 -> IP Address: 192.168.0.102
```

((b)) Serial monitor

Figure 3.14: Testing the Wi-Fi connection

3.3.3 Connecting to Firebase

Another step is connecting to Firebase so that we can store our sensor readings in a database. For this we used the FirebaseArduino.h [12] library which we had to download and add to the libraries folder of the Arduino app. We also needed the ArduinoJSON.h library that we could directly install via the Library Manager provided by Arduino IDE. This library helps us with JSON parsing so that we can directly use C/C++ types such as strings.

Then we had to go to the official Firebase website [7], sign up with a Google account, create a new project, create a Realtime Database and get its link and secret key to declare them in our code to be able to connect to our database.

After doing this, we had to call the *Firebase.begin(firebase_host, firebase_key)* function, check the Firebase connection, push the sensor data in C/C++ String format, letting the ArduinoJSON handle the parsing and check if the data could be pushed. The connection was working, however, the push wasn't. After checking the Firebase credentials, creating a few more databases and reinstalling the libraries again, pushing data still wasn't working. After some research on why it wasn't working, we found a solution [25] and realized it was a common problem due to the library being under constant development and all we had to do was generate a new fingerprint for our own database using [16] and go to the FirebaseHttpClient.h file in our libraries>firebase-arduino-master>src and replace the old "static const char kFirebaseFingerprint[]" with our new fingerprint. This solved the problem we struggled with for a few hours.

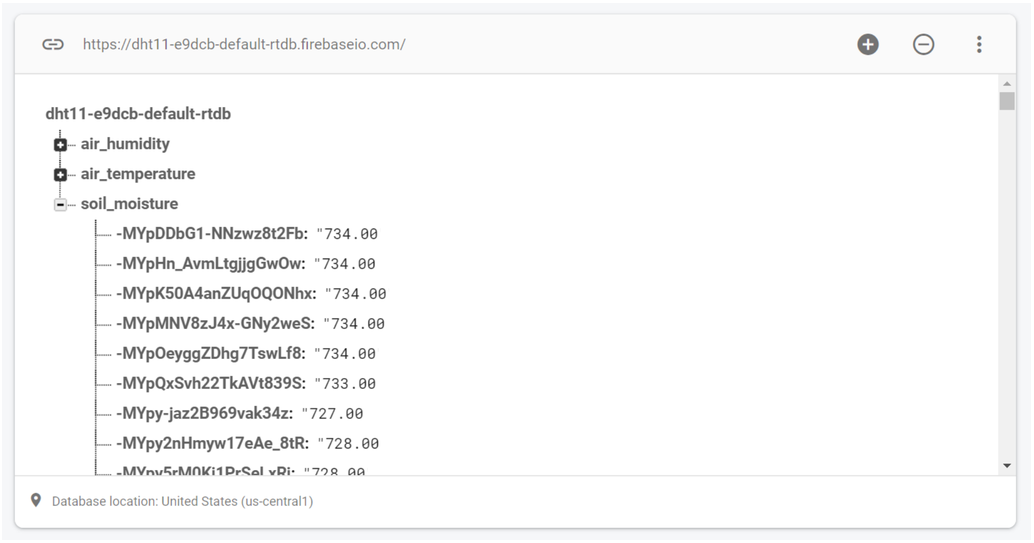
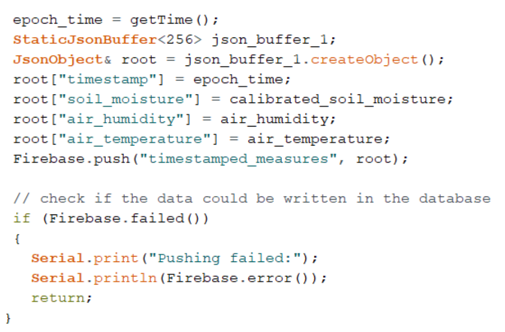
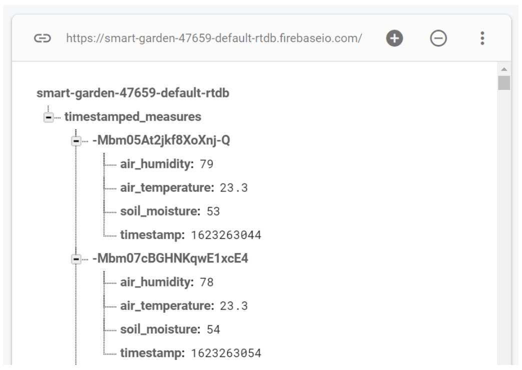

```
https://dht11-e9dcb-default-rtdb.firebaseio.com/

dht11-e9dcb-default-rtdb
  air_humidity
  air_temperature
  soil_moisture
      -MYpDDbG1-NNzwz8t2Fb: "734.00
      -MYpHn_AvmLtgjjgGwOw: "734.00
      -MYpK50A4anZUqOQONhx: "734.00
      -MYpMNV8zJ4x-GNy2weS: "734.00
      -MYpOeyggZDhg7TswLf8: "734.00
      -MYpQxSvh22TkAVt839S: "733.00
      -MYpy-jaz2B969vak34z: "727.00
      -MYpy2nHmyw17eAe_8tR: "728.00
      -MYpy5rM0Kj1PrSeLxRi: "728.00

Database location: United States (us-central1)
```

Figure 3.15: Our old real-time database



```
epoch_time = getTime();
StaticJsonBuffer<256> json_buffer_1;
JsonObject& root = json_buffer_1.createObject();
root["timestamp"] = epoch_time;
root["soil_moisture"] = calibrated_soil_moisture;
root["air_humidity"] = air_humidity;
root["air_temperature"] = air_temperature;
Firebase.push("timestamped_measures", root);

// check if the data could be written in the database
if (Firebase.failed())
{
  Serial.print("Pushing failed:");
  Serial.println(Firebase.error());
  return;
}
```

((a)) Code



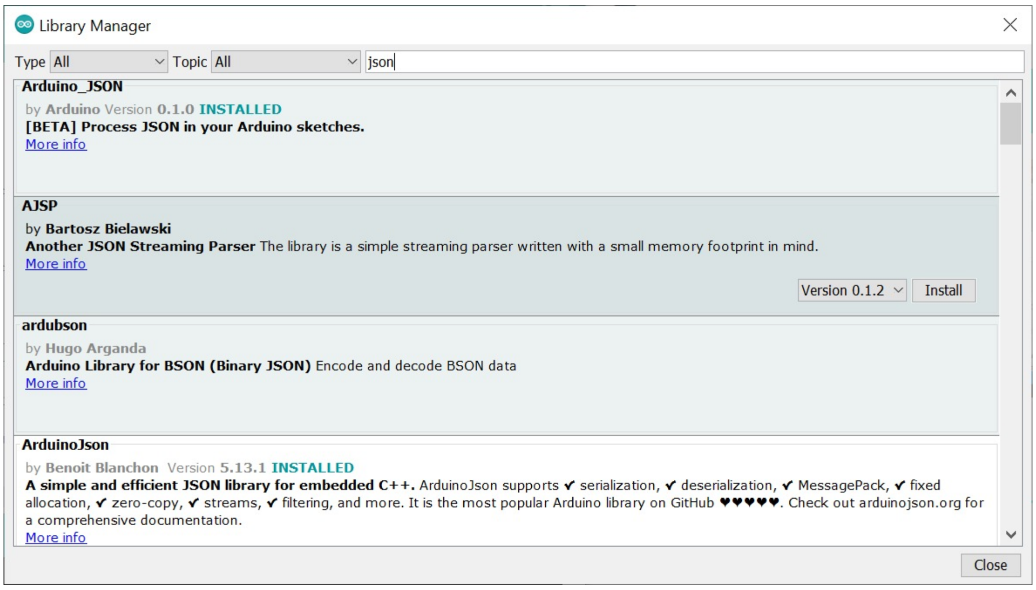
https://smart-garden-47659-default-rtdb.firebaseio.com/

```
smart-garden-47659-default-rtdb
  timestamped_measures
    -Mbm05At2jkf8XoXnj-Q
      air_humidity: 79
      air_temperature: 23.3
      soil_moisture: 53
      timestamp: 1623263044
    -Mbm07cBGHNKqwE1xcE4
      air_humidity: 78
      air_temperature: 23.3
      soil_moisture: 54
      timestamp: 1623263054
```

((b)) Firebase

Figure 3.16: Our new real-time database that includes timestamps

After deciding to plot our data, however, we wanted to store all our sensor readings and add a timestamp to our data and keep them all under the same key. We also wanted to keep the soil moisture as percentage mapping the raw values in the 0-100% interval to ease the plotting. The previous ArduinoJSON library would store the temperature, humidity and raw soil moisture sensor data separately and without a timestamp as can be seen in Figure 3.15 and plotting it wouldn't have shown us the time and date. To store it under the same key as seen in Figure 3.16 we had to use the Arduino_JSON.h library instead to send a JSON object directly to Firebase and also add the WiFiUDP.h and NTPClient.h libraries to get the UNIX Epoch time. The JSON libraries we used can be seen in Figure 3.17.



Library Manager

Type All Topic All json

Arduino_JSON
by **Arduino** Version **0.1.0** **INSTALLED**
[BETA] Process JSON in your Arduino sketches.
More info

AJSP
by **Bartosz Bielawski**
Another JSON Streaming Parser The library is a simple streaming parser written with a small memory footprint in mind.
More info
Version 0.1.2 Install

ardubson
by **Hugo Arganda**
Arduino Library for BSON (Binary JSON) Encode and decode BSON data
More info

ArduinoJson
by **Benoit Blanchon** Version **5.13.1** **INSTALLED**
A simple and efficient JSON library for embedded C++. ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ MessagePack, ✔ fixed allocation, ✔ zero-copy, ✔ streams, ✔ filtering, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation.
More info

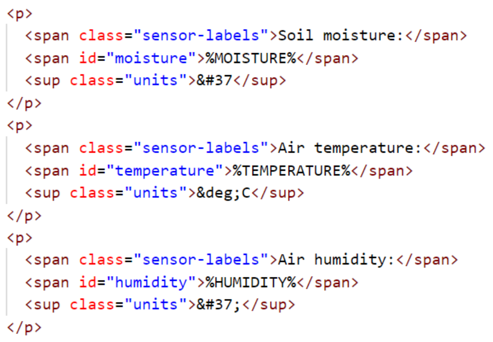
Close

Figure 3.17: The Arduino JSON libraries

3.3.4 Build a web server using HTML, CSS and JS and store these files using LittleFS

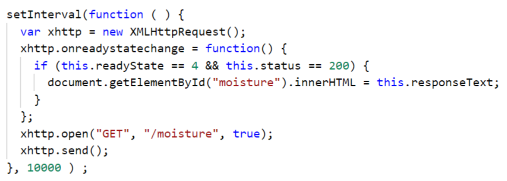
The next feature we wanted to add to our project was a web server that shows the current state of the pump and allows the user to control it, turning it on or off. We want to display the real-time sensor data too and plot it and also the current weather to help the user decide whether to turn on the pump or not in case of rain, for example. We decided to use the aforementioned LittleFS filesystem for storing the code for our website as separate files on the NodeMCU and not as strings in the sketch because that would complicate the code a lot and would not be best practice.

For this part we had to download the LittleFS plugin from [10] and add it to the arduino>tools folder. After restarting the Arduino IDE going to Tools section we now had the option "ESP8266 LittleFS Data Upload", which uploads the files on the board and which we will have to run every time we make a change in the HTML, CSS and JS files that we need to store in a folder called "data" that we keep in the same folder as our sketch. To build the web server using files stored in the filesystem we also used the ESPAsyncWebServer [14] and the ESPAsyncTCP [13] libraries that we manually downloaded and added to the arduino>libraries folder.



```html
<p>
  <span class="sensor-labels">Soil moisture:</span>
  <span id="moisture">%MOISTURE%</span>
  <sup class="units">&#37</sup>
</p>
<p>
  <span class="sensor-labels">Air temperature:</span>
  <span id="temperature">%TEMPERATURE%</span>
  <sup class="units">&deg;C</sup>
</p>
<p>
  <span class="sensor-labels">Air humidity:</span>
  <span id="humidity">%HUMIDITY%</span>
  <sup class="units">&#37;</sup>
</p>
```

((a)) HTML code



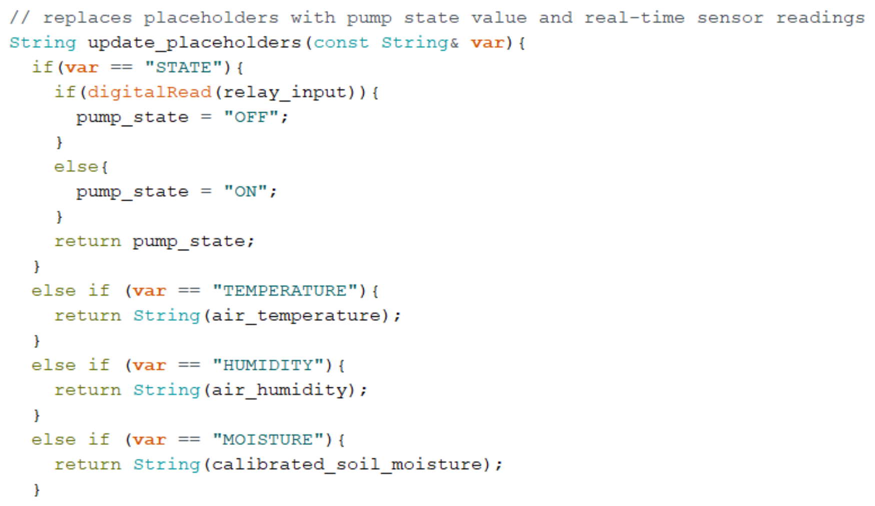
```javascript
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("moisture").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/moisture", true);
  xhttp.send();
}, 10000 ) ;
```

((b)) JavaScript code

Figure 3.18: Asynchronously updating the soil moisture

For the web server we keep the HTML, CSS and JS code in separate files and we reference the CSS and JS in the HTML file. To set values in the HTML file for the pump state and the sensor data from the Arduino sketch and send them to the web server we use placeholders in between "%" signs and label them with an id such as id="moisture" for %MOISTURE%. To automatically update those values every 10 seconds without the need for refreshing the page we use some JavaScript to make a request in the "/moisture" URL, for example, to get the last soil moisture value read we update the element with the id="moisture". This way, the moisture is updated asynchronously as can be seen in Figure 3.18. The CSS file only contains some basic designing such as style, colors, font size, align and buttons.

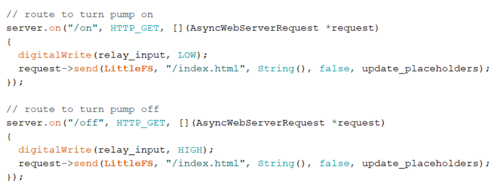
Then, in the Arduino sketch we need to include the previously mentioned libraries to be able to create an AsyncWebServer that listens on port 80. We also want to have a method that sets values to the placeholders we defined in the HTML file. The method has as argument a placeholder and just replaces it by returning the corresponding String that represents the value read from the sensor as can be seen in Figure 3.19.



```
// replaces placeholders with pump state value and real-time sensor readings
String update_placeholders(const String& var){
  if(var == "STATE"){
    if(digitalRead(relay_input)){
      pump_state = "OFF";
    }
    else{
      pump_state = "ON";
    }
    return pump_state;
  }
  else if (var == "TEMPERATURE"){
    return String(air_temperature);
  }
  else if (var == "HUMIDITY"){
    return String(air_humidity);
  }
  else if (var == "MOISTURE"){
    return String(calibrated_soil_moisture);
  }
```

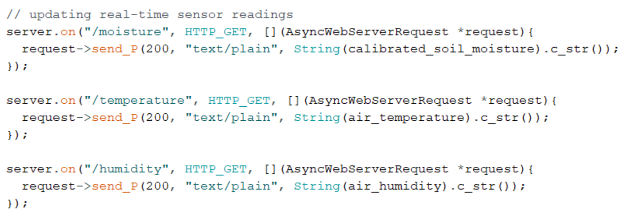
Figure 3.19: Method that replaces the HTML placeholders with Arduino values

In the setup() we need to initialize LittleFS and check if it works. Here we also want to set up the routes that the server will listen to for HTTP requests such as send "index.html" for "/" URL and make the requests for the CSS and JS files too. We also want to specify what needs to be done on the "/on" and "/off" routes, that is, turn the pump on or off by setting the relay input to LOW or HIGH. Then we define what happens on the sensor readings routes that we requested every 10 seconds in the JavaScript code such as the "/moisture" URL. Here we only have to send the new sensor readings via their corresponding variables that are updated at a specified time interval in the Arduino code. All this can be seen in Figure 3.20.



```
// route to turn pump on
server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request)
{
  digitalWrite(relay_input, LOW);
  request->send(LittleFS, "/index.html", String(), false, update_placeholders);
});

// route to turn pump off
server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request)
{
  digitalWrite(relay_input, HIGH);
  request->send(LittleFS, "/index.html", String(), false, update_placeholders);
});
```

((a)) Routes to turn the pump on and off



```
// updating real-time sensor readings
server.on("/moisture", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", String(calibrated_soil_moisture).c_str());
});

server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", String(air_temperature).c_str());
});

server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", String(air_humidity).c_str());
});
```

((b)) Routes to update the sensor readings

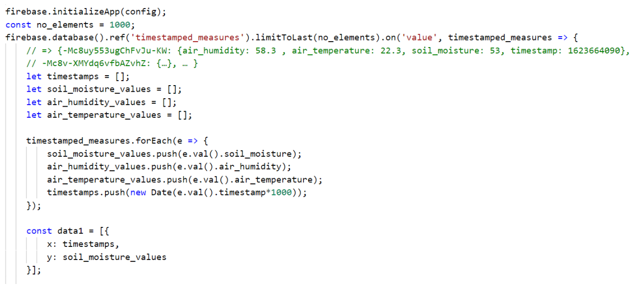
Figure 3.20: Setting the routes

3.3.5 Plot the real-time data using Plotly.js

Another feature that we wanted to add to our application are plots to help the user visualize their data that was collected over some time and stored in a Firebase Realtime Database. The data is plotted in real time by only representing a number of instances and when a new sensor reading is available, we shift the graph to the left such that the oldest data disappears, allowing the new data to be shown. As mentioned earlier, to plot the data it would be easier to store all our current sensor readings with their timestamp under the same key in Firebase.

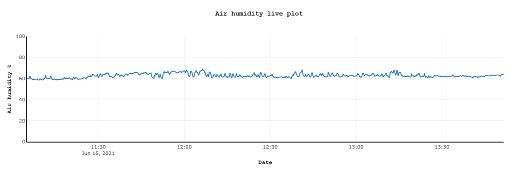
To actually plot the data we used Plotly.js that is a free, open source high-level JavaScript data visualization library. It has over 40 chart types, including statistical charts, 3D graphs, SVG and tile maps and more. For this we have to include the plotly.js [5] library in the HTML file.

Another important thing that we had to do when implementing this feature was to install the Firebase Command Line Interface. We then went to the Firebase console and obtained and included the project configuration into our JavaScript file and two Firebase scripts [18] [19] in our HTML file. To be able to have the JavaScript code separately we also need to include the script.js file in our HTML code.



```
firebase.initializeApp(config);
const no_elements = 1000;
firebase.database().ref('timestamped_measures').limitToLast(no_elements).on('value', timestamped_measures => {
    // => {-Mc8uy553ugChFvJu-KW: {air_humidity: 58.3 , air_temperature: 22.3, soil_moisture: 53, timestamp: 1623664090},
    // -Mc8v-XMYdq6vfbAZvhZ: {…}, … }
    let timestamps = [];
    let soil_moisture_values = [];
    let air_humidity_values = [];
    let air_temperature_values = [];

    timestamped_measures.forEach(e => {
        soil_moisture_values.push(e.val().soil_moisture);
        air_humidity_values.push(e.val().air_humidity);
        air_temperature_values.push(e.val().air_temperature);
        timestamps.push(new Date(e.val().timestamp*1000));
    });

    const data1 = [{
        x: timestamps,
        y: soil_moisture_values
    }];
```

((a)) The JavaScript code for plotting our graphs



((b)) Real time plot of the air humidity sensor data

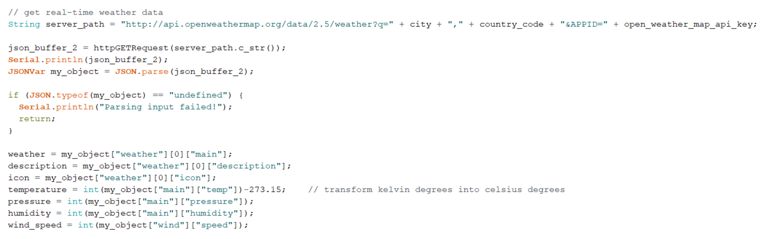
Figure 3.21: Plotting real time data with Plotly.js

Then we define how many values we want to plot and we make three lists with that number of elements for soil moisture, air humidity and temperature for the y axis and one list with the corresponding timestamps for the x axis. The lists will be modified each time a value is added to our database to contain the last number of elements from it, hence giving the graph a real time plot aspect. How we did this and the resulting plot for the air humidity can be seen in Figure 3.21.

3.3.6 Get weather data using OpenWeatherMap API

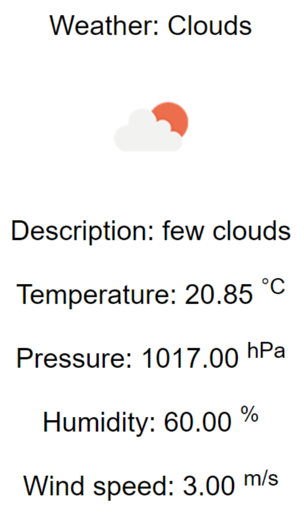
To help the user decide whether to start the pump or not we also wanted to display the current weather in our web server. For this we used the OpenWeatherMap API that allows us to request weather data. OpenWeatherMap provides many options to choose from and we decided that the current day's weather forecast for the user's location is the most appropriate one given the context.

To do this we had to go to the OpenWeatherMap website [28], make an account and get our API key. To get the current weather we used the HTTP GET Request Method. The way this works is really easy to understand: the client (NodeMCU ESP8266) makes a HTTP request to the server (OpenWeatherMap), then the server returns a response containing the requested weather information as a JSON object to the client.



```
// get real-time weather data
String server_path = "http://api.openweathermap.org/data/2.5/weather?q=" + city + "," + country_code + "&APPID=" + open_weather_map_api_key;

json_buffer_2 = httpGETRequest(server_path.c_str());
Serial.println(json_buffer_2);
JSONVar my_object = JSON.parse(json_buffer_2);

if (JSON.typeof(my_object) == "undefined") {
  Serial.println("Parsing input failed!");
  return;
}

weather = my_object["weather"][0]["main"];
description = my_object["weather"][0]["description"];
icon = my_object["weather"][0]["icon"];
temperature = int(my_object["main"]["temp"])-273.15;    // transform kelvin degrees into celsius degrees
pressure = int(my_object["main"]["pressure"]);
humidity = int(my_object["main"]["humidity"]);
wind_speed = int(my_object["wind"]["speed"]);
```

((a)) Arduino code for requesting the weather from the OpenWeatherMap API



Weather: Clouds

Description: few clouds

Temperature: 20.85 °C

Pressure: 1017.00 hPa

Humidity: 60.00 %

Wind speed: 3.00 m/s

((b)) Real time display of the current weather forecast

Figure 3.22: Displaying the current weather using the OpenWeatherMap API

After declaring the API key in our code, we also needed to define the city and country for which we wanted to get the information. Since we want to update the weather periodically, we have to put the HTTP request in the loop() section. There we call the httpGETRequest() method that makes a request to OpenWeatherMap and returns a JSON object as a string containing the current weather information for the location that we specified. The next thing that did was to decode the JSON object and put each of the values that are relevant in some variables to be able to display them in our web server. To display them we used placeholders and updated them asynchronously the same way we did for showing the real time sensor readings. Another thing we did was display a small image that would change according to the weather. We did this using the icon name field in the JSON object and the icons provided by OpenWeatherMap [29]. All this can be seen in Figure 3.22.

Chapter 4

Experiment

To test whether our approach is better than an experience based scheduled irrigation we made a little experiment that involved modifying the sketch a bit to not take soil moisture into account and just water according to a timetable that is heuristically determined by knowing each plants individual needs and the amount of water the system delivers in a given time. For the purpose of the experiment, we abstracted a backyard garden with a couple plants in pots that have different water needs: a mint plant that needs less watering and a sage plant that needs more water. We monitored each of these plants for a few weeks using the two different approaches to irrigation: scheduled irrigation and intelligent irrigation. During the entire experiment we monitored the soil moisture every 10 minutes in both cases to be able to make a comparison between the two types of irrigation and draw conclusions regarding which is better in terms of taking care of the plants needs and saving resources.

4.1 Scheduled irrigation

The first approach is scheduled irrigation. This means watering the plant at regular time intervals for a specific amount of time, knowing how much water the system delivers in that amount of time and taking into account the plant's individual needs.

To define our scheduled irrigation program, we had to weight the plants to see what their average water needs are as could be seen in Table 3.4 from Section 3.1.6. Then we had to find out the time in which the system delivers that amount of water and modify the sketch according to each plant which was time consuming and also error prone. This method was highly heuristic and chaotic as you will see down below.

For the mint plant we scheduled the irrigation once in three days for 5 seconds which delivered approximately 120 ml of water which was a little less than the plant needed (approximately 150 ml), and yet the result was overwatering the mint plant. In a week, the mint was watered twice and as it can be seen in color red in Figure 4.1 the plant had a rather extreme variation in the soil moisture during that week, going from dry to a little too moist. Because of that, the plant started showing clear signs of being overwatered: some of the leaves started to yellow and wilt, the roots became brown which is a symptom of root rot and lack of oxygen and the plant did not thrive.

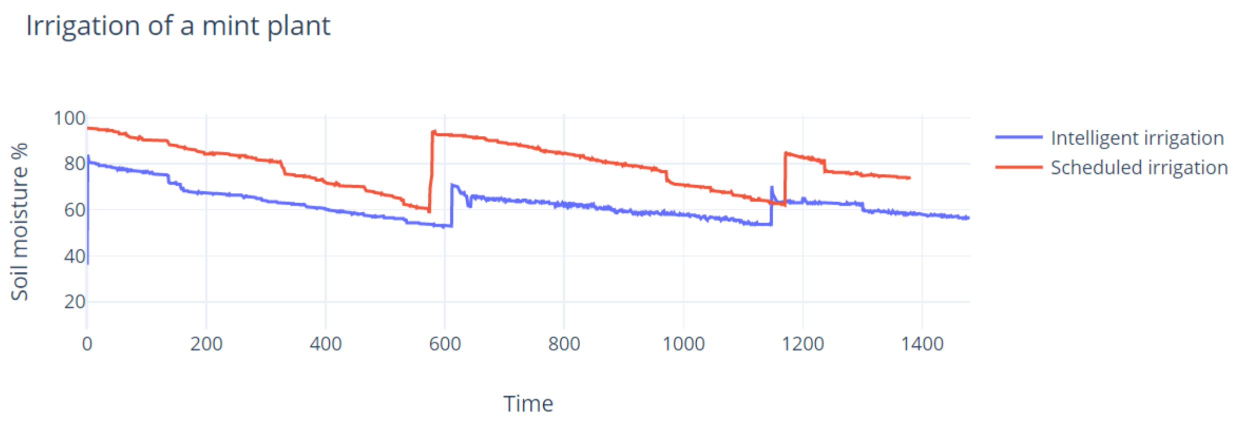


Figure 4.1: Irrigation of a mint plant

A similar thing happened with the sage plant. We scheduled the program to water every two days for 4 seconds which delivered approximately 100 ml of water which was, again, a little less than the plant needed (approximately 140 ml). This time we only monitored the plant for 4 days, but the result was, again, the plant showing clear overwatering signs and a great variation in the soil moisture which jumped from dry to overly moist as can be seen in color red in Figure 4.2. The plots were made using [6].

With this approach, the soil has a tendency to dry out and then is flooded with water, resulting in a high increase in the soil moisture levels. These extreme variations are not beneficial to the plants that can die from lack of oxygen to the roots due to water saturated soil. Moreover, dry soil lets water pour out through the bottom draining holes of the pot, wasting the water.

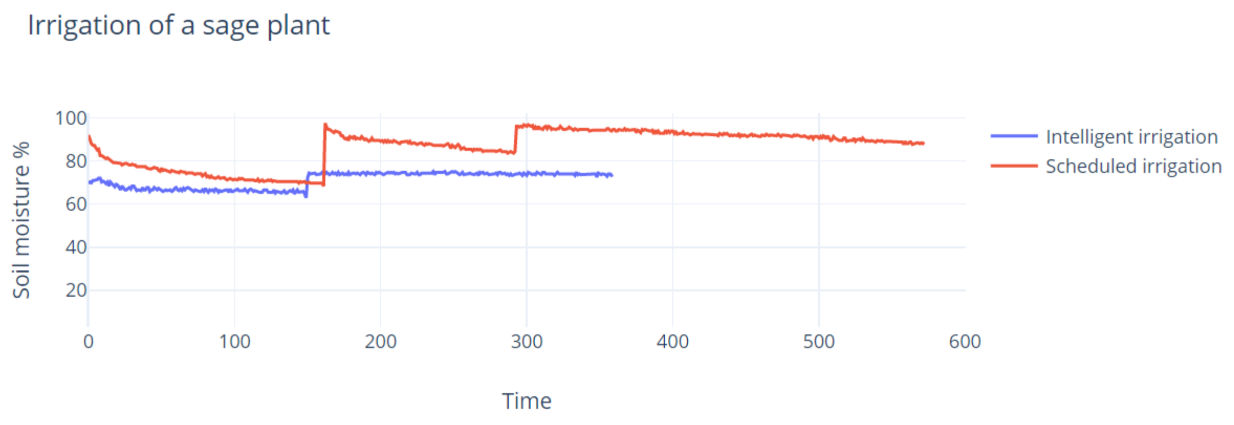


Figure 4.2: Irrigation of a sage plant

4.2 Intelligent irrigation

The better approach is intelligent irrigation. This method takes into account the soil moisture measured with a capacitive sensor and only starts the pump to water the plant when the soil moisture value drops below a lower limit that indicates that the soil is drying out. For this we only need to set the aforementioned soil moisture limit and to know the amount of water delivered by the system in a given time.

In this sketch, we want to read data from the sensors every 10 minutes and if the soil reaches the *limit* that implies that it is starting to dry, turn on the irrigation for 2 seconds providing the plant with approximately 50 ml of water. We experimentally found the *limit* to be around 415 raw value, so when the sensor returns a value higher than that, the pump is turned on and the plant is watered. Then the system waits for 10 minutes, reads data from the sensors again and checks the value of the soil moisture to decide if another round of watering is necessary.

This approach provides a watering scheme that adapts to the specific needs of each plant, making the requirement of knowing the amount of water necessary for each individual plant obsolete and designing a standard (universal) system that is suitable for every plant type by supplying only a small amount of water and checking again a few minutes later to see if there is need for more water. The program also adapts to the environmental conditions by constantly monitoring the soil moisture levels. Regardless of the weather, air humidity and temperature, the system will adjust to maintain the proper level of moisture.

As a result, the intelligent irrigation approach manages to maintain the soil moisture at relatively same levels, preventing great variations and therefore not allowing the plant to be under or overwatered as can be seen in color blue in Figure 4.1 and Figure 4.2, while also using the minimum amount of water that the plants need to thrive. In fact, for this we only needed 2/3 of the quantity we used in the case of scheduled irrigation, therefore saving water which is an important resource that needs to be used sparingly.

Chapter 5

Conclusion and future work

In this paper we presented a hardware prototype and the software implementation of a smart irrigation system. The proposed system consisted of a NodeMCU board to which we connected a capacitive soil moisture sensor, an air humidity and temperature sensor and a relay that controls a submersible water pump. The board was programmed in Arduino IDE to perform an intelligent irrigation that took into account the level of the soil moisture and sent real-time data to Firebase and to a web server that allowed the user to see the values read from the sensors displayed and plotted, to remotely control the water pump and to see the weather forecast.

We also made a comparison between our intelligent system and a scheduled irrigation system and found out that our approach manages to maintain the soil moisture around the same levels which provides us with an universal system that adapts to both the plant needs and the environmental conditions without the need of constant code adjustments according to each specific plant's needs, while also saving water and keeping the plants healthy. By also setting up a web server we provided the user with an automated watering system that can be monitored and controlled remotely, limiting the need for physical human interaction, therefore saving time and effort.

As for future work, concerning the software implementation, we would like to use a fuzzy algorithm that also takes into account the air humidity and temperature when deciding whether to water the plants and compare it with our current best approach. Regarding the hardware part, we want to scale the prototype for a garden bed by replacing the current mini hose with a drip irrigation tubes and letting the pump run for a longer period of time. Furthermore, to be more environmentally friendly we plan to use solar panels to power our system and collected rain water to irrigate the plants.

Bibliography

[1] Constantinos Marios Angelopoulos, Sotiris Nikoletseas, and Georgios Constantinos Theofanopoulos. A smart system for garden watering using wireless sensor networks. In *Proceedings of the 9th ACM international symposium on Mobility management and wireless access*, pages 167–170, 2011.

[2] `https://www.arduino.cc/reference/en/language/functions/math/map/`.

[3] `https://arduino-esp8266.readthedocs.io/en/latest/filesystem.html`.

[4] `https://www.arduino.cc/reference/en/`.

[5] `https://cdn.plot.ly/plotly-latest.min.js`.

[6] `https://chart-studio.plotly.com/create/#/`.

[7] `https://firebase.google.com/`.

[8] `https://firebase.google.com/docs/database`.

[9] `https://flaviocopes.com/arduino-programming-language/`.

[10] `https://github.com/earlephilhower/arduino-esp8266littlefs-plugin/releases`.

[11] `https://github.com/esp8266/Arduino`.

[12] `https://github.com/FirebaseExtended/firebase-arduino`.

[13] `https://github.com/me-no-dev/ESPAsyncTCP`.

[14] `https://github.com/me-no-dev/ESPAsyncWebServer`.

[15] `https://www.globalagriculture.org/report-topics/water.html`.

[16] `https://www.grc.com/fingerprints.htm`.

[17] `https://www.greenhousemag.com/article/gmpro-0310-water-plants-automating-irrigation/`.

[18] `https://www.gstatic.com/firebasejs/5.5.9/firebase-app.js`.

[19] `https://www.gstatic.com/firebasejs/5.5.9/firebase-database.js`.

[20] `https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino/`.

[21] SN Ishak, NNN Abd Malik, NM Abdul Latiff, N Effiyana Ghazali, and MA Baharudin. Smart home garden irrigation system using raspberry pi. In *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, pages 101–106. IEEE, 2017.

[22] Nurulisma Ismail, Sheegillshah Rajendran, Wong Chee Tak, Tham Ker Xin, Nur Shazatushima Shahril Anuar, Fadhil Aiman Zakaria, Yahya Mohammed Salleh Al Quhaif, Hussein Amer M Hasan Karakhan, and Hasliza A Rahim. Smart irrigation system based on internet of things (iot). In *Journal of Physics: Conference Series*, volume 1339, page 012012. IOP Publishing, 2019.

[23] R Santhana Krishnan, E Golden Julie, Y Harold Robinson, S Raja, Raghvendra Kumar, Pham Huy Thong, et al. Fuzzy logic based smart irrigation system using internet of things. *Journal of Cleaner Production*, 252:119902, 2020.

[24] `https://lastminuteengineers.com/`.

[25] `https://medium.com/@bimxra/how-to-fix-nodemcu-arduino-firebase-library-not-working-issue-5c8ff5a4a1ff`.

[26] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7):1497–1516, 2012.

[27] M Safdar Munir, Imran Sarwar Bajwa, M Asif Naeem, and Bushra Ramzan. Design and implementation of an iot system for smart energy consumption and smart irrigation in tunnel farming. *Energies*, 11(12):3427, 2018.

[28] https://openweathermap.org/.

[29] https://openweathermap.org/weather-conditions#Icon-list.

[30] Birgit Penzenstadler, Jason Plojo, Marinela Sanchez, Ruben Marin, Lam Tran, and Jayden Khakurel. The diy resilient smart garden kit. In *Proceedings of the Workshop on Computing within Limits (LIMITS), Calgary, AB, Canada*, pages 6–7, 2018.

[31] Ipin Prasojo, Andino Maseleno, Nishith Shahu, et al. Design of automatic watering system based on arduino. *Journal of Robotics and Control (JRC)*, 1(2):59–63, 2020.

[32] https://www.statista.com/statistics/617136/digital-population-worldwide/.

[33] https://en.wikipedia.org/wiki/CSS.

[34] https://en.wikipedia.org/wiki/HTML.

[35] https://en.wikipedia.org/wiki/JavaScript.

[36] https://en.wikipedia.org/wiki/NodeMCU.