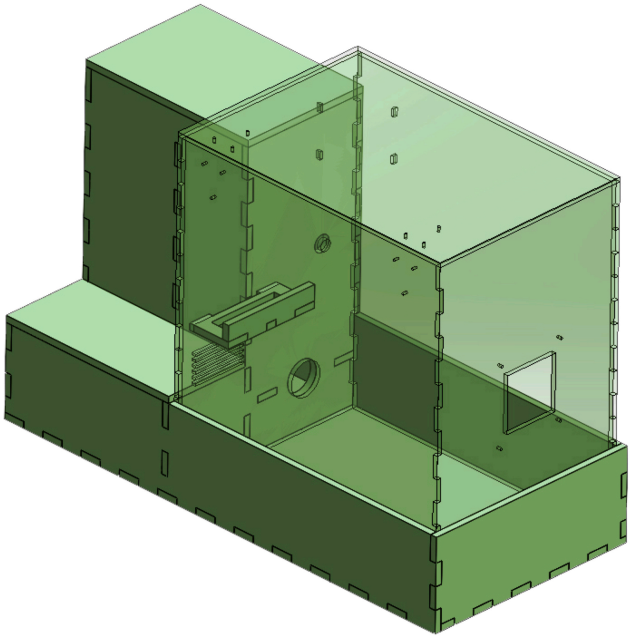
Wiki

EE5 - ee-plant 🌱

Introduction

ee-plant is a smart IoT system that takes care of your plants without the need of your time. It runs autonomously and you can monitor everything through our mobile app.



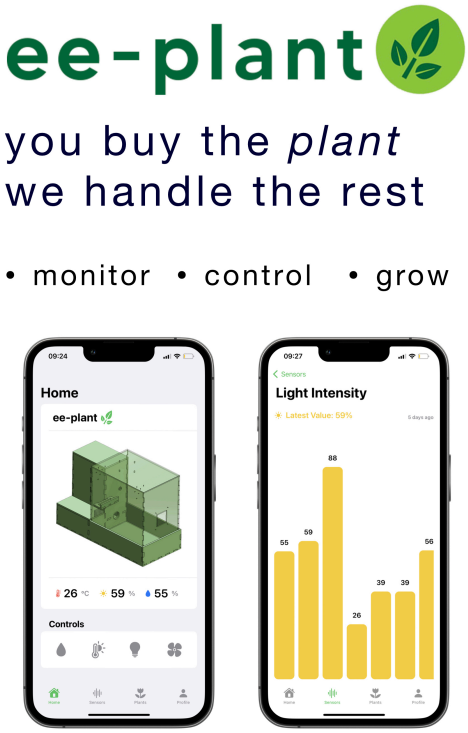


Table of contents

1. Requirements
2. Design
3. Hardware
    1. Actuators
    2. Case
    3. PCBs
    4. Sensors
4. Software
    1. App

    2. Database
    3. ESP32
    4. Server
5. Videos

1. Requirements

The EE-Plant🌱

EE5 Project - Team IoT 02

Intro

If you do not have the time to take care of your plants or when you are away from home for a few days, the EE-Plant can help you! The system can look after different plant species, from common house plants to exotic ones. Specialised algorithms are designed with that in mind, tailored for every kind.

The EE-Plant comes with an Android app that updates the user about the plant. Is it too cold? The temperature goes up! Is it too dry? The system will water the plant. If the reservoir's water level is low, the app will notify the user.

Of course, you can take matters into your own hands and adjust every command accordingly.

Sensors

The following sensors monitor the system 24/7:

- the soil moisture
- the air humidity
- temperature
- light intensity

Actuators

The following actuators regulate and control the system:

- a heater warms up the air
- a fan cools down the environment and regulates the humidity
- a spray machine, connected to a servo motor, sprays water and increase the humidity
- the water reservoir, attached with a pump and a servo motor, waters the soil

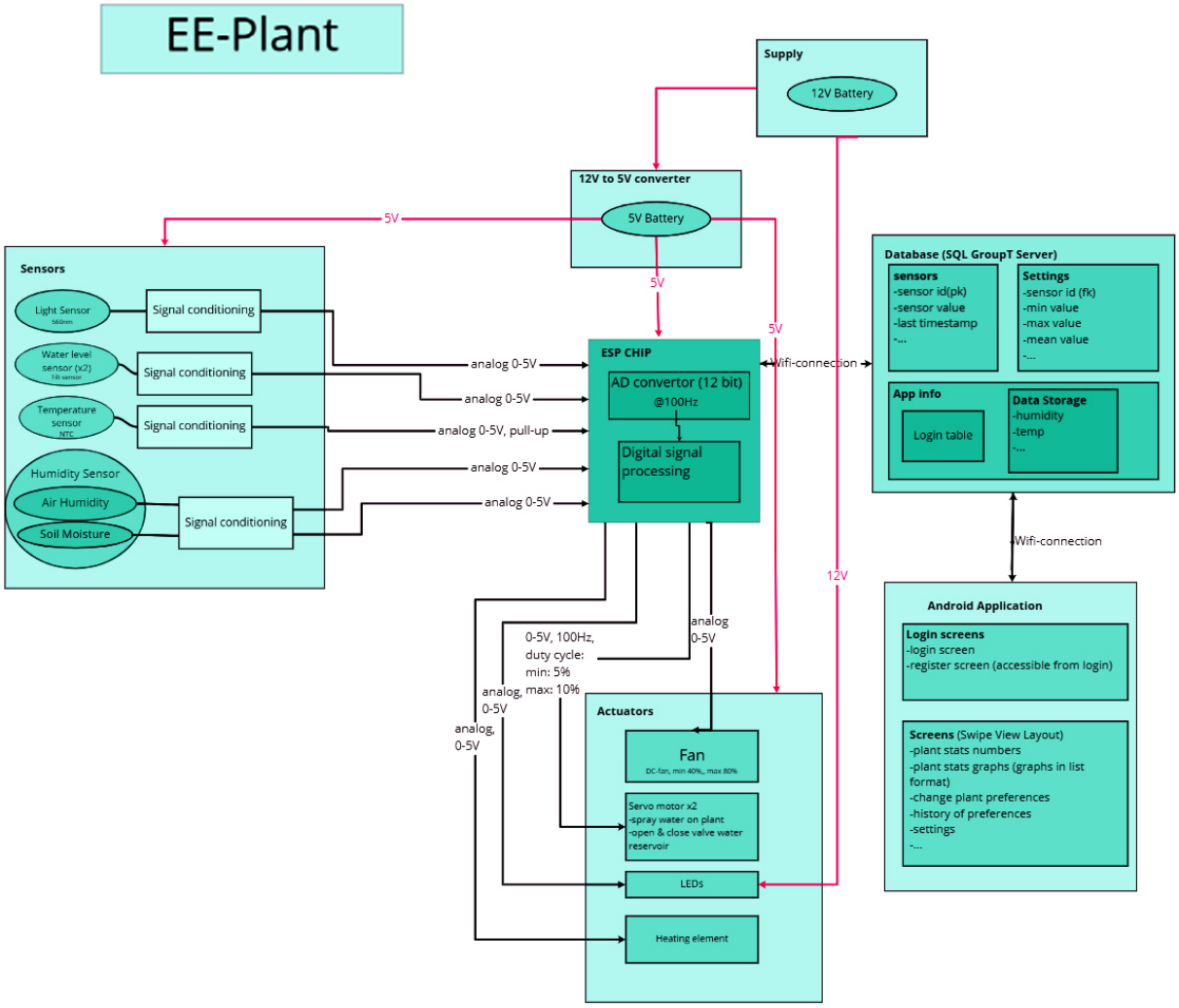
Finally

The system's exterior is made out of a plexiglass box, MDF and 3D printed components. A power bank supplies the system with 5V. The ESP32 communicates with the database in the server through its Wi-Fi module. Data from the sensors is saved in these databases. It is then made available to the user via the app in form of graphs over time

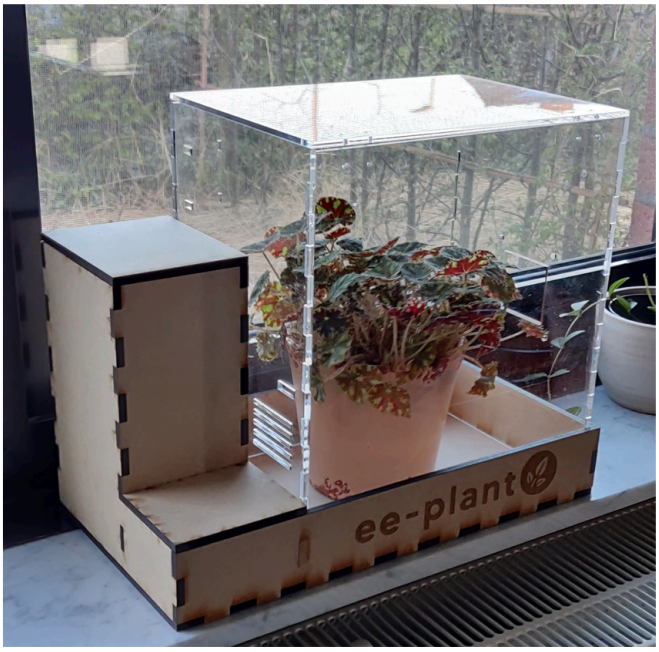
> *With the EE-Plant everything is about nature and comfort.*

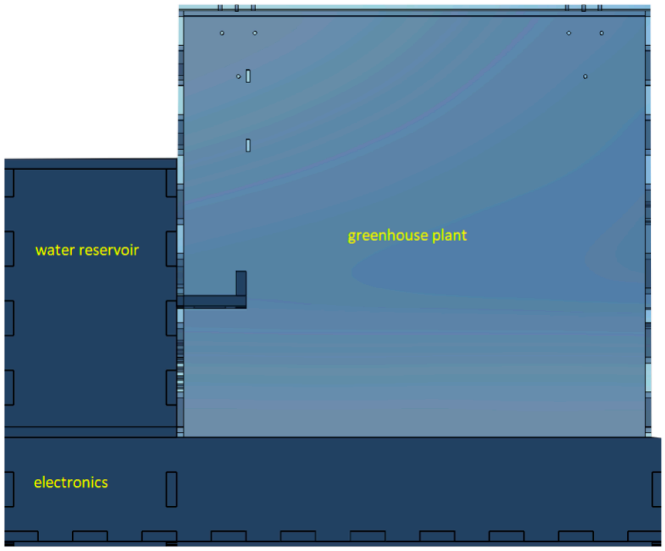
2. Design

The General Connection

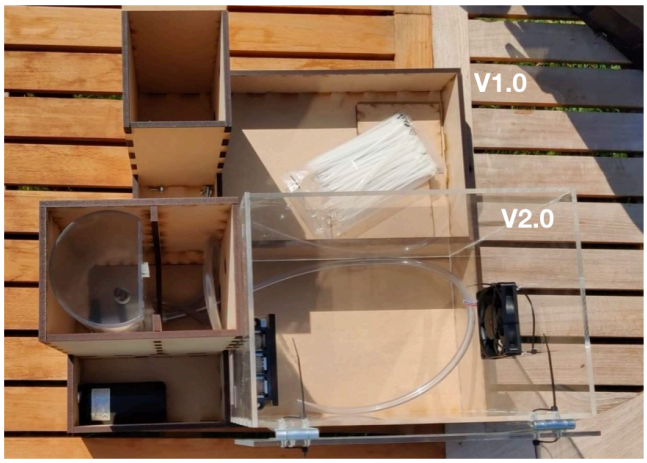


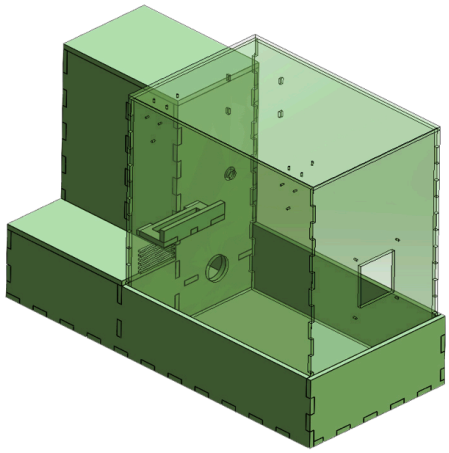
Version 1.0





Version 2.0

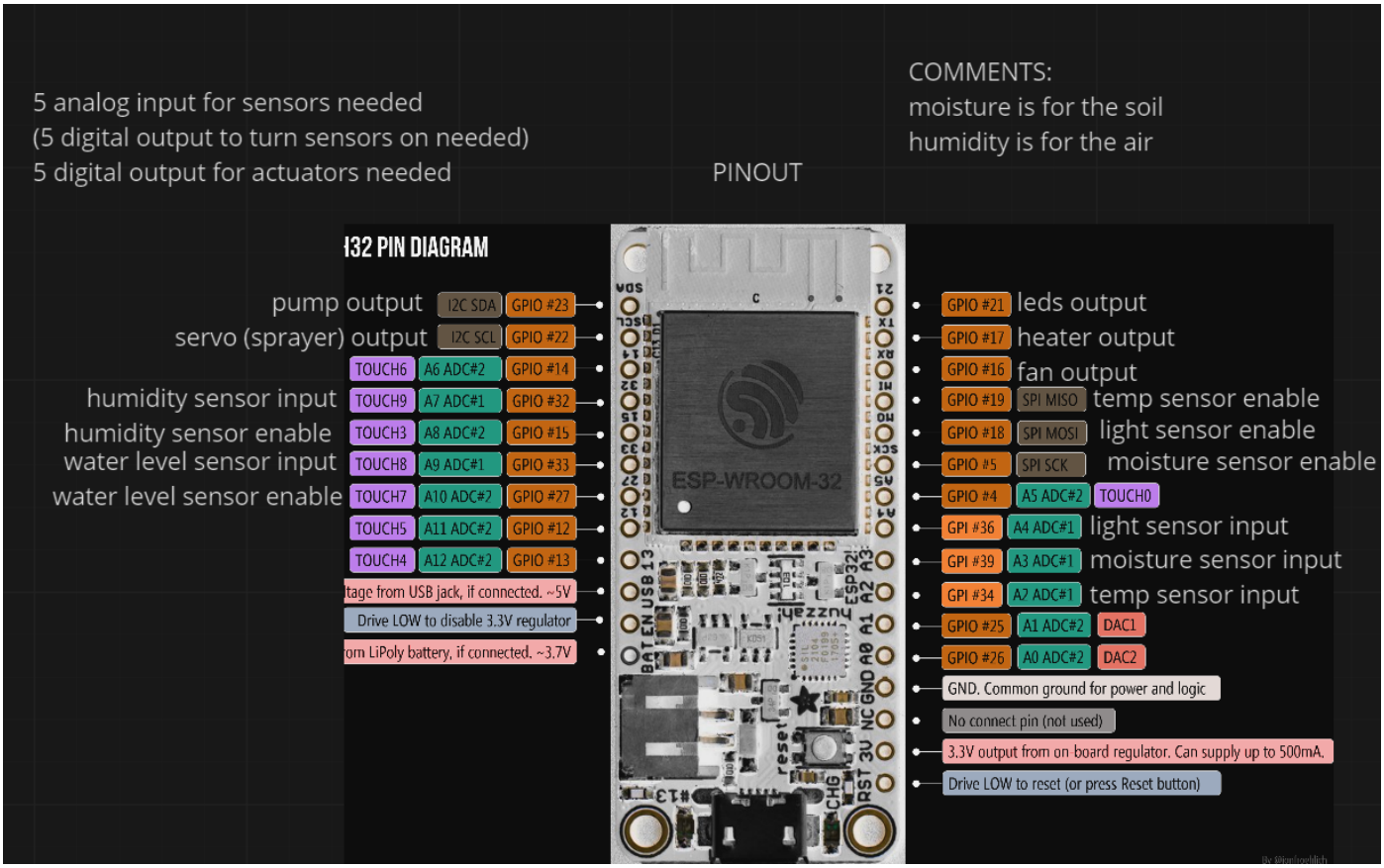




Version 2.0 is larger, to provide more space for the electronics and the water reservoir.

3. Hardware

Pinout

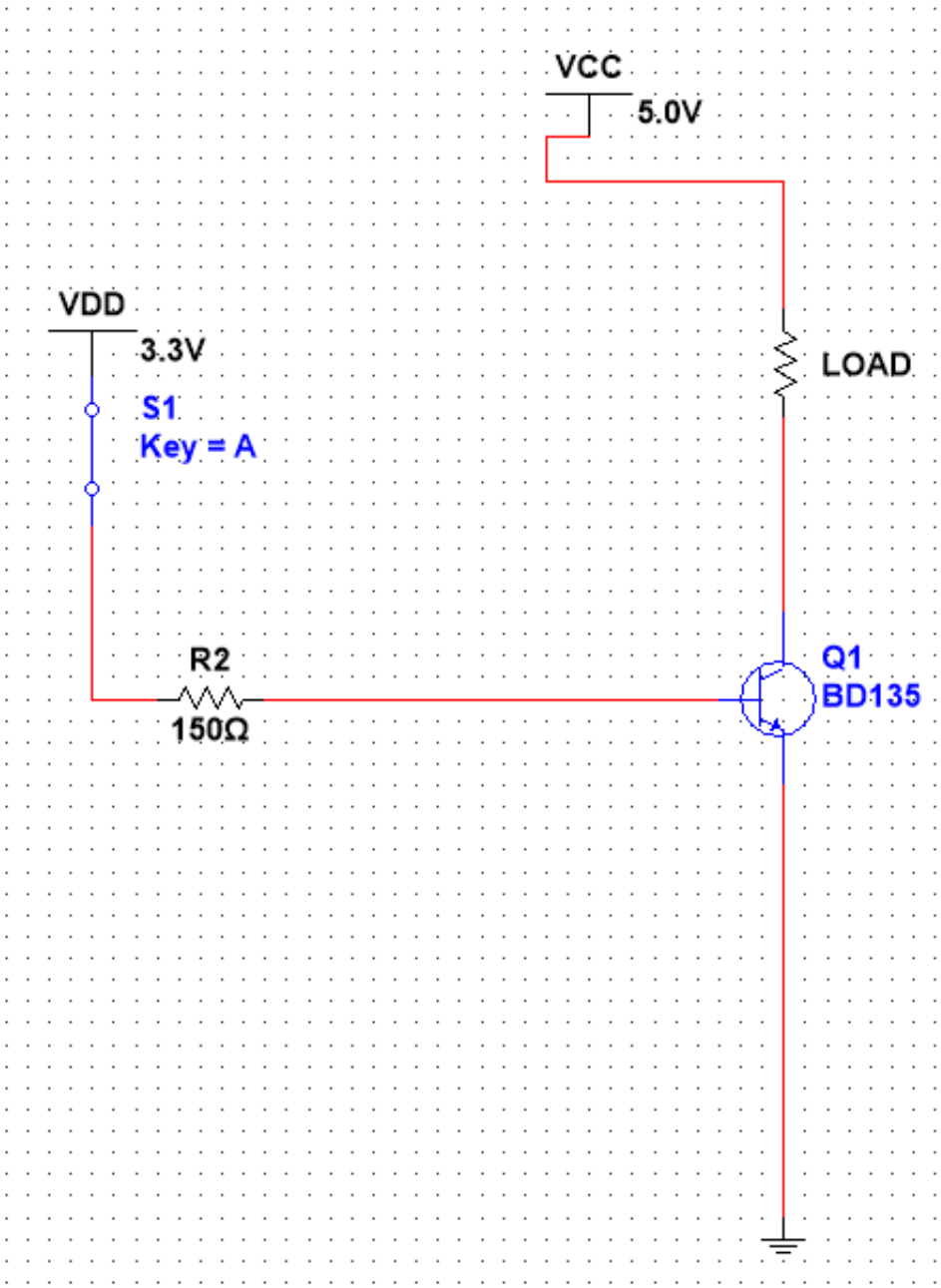


3.1 Actuators

Actuators

Heater

A Heater, a DC Fan and a Pump control the system's environment. Transistors are used since I/O pins cannot power these components. The transistors should be able to withstand the current. They are used in saturation mode to dissipate as little power as possible.



The heater pulls around 2A at 5V, which the battery cannot supply, so a PWM is used to limit

the average current.

Pump

The voltage range of the pump is between 5V and 12V. On a working voltage of 5V directly connected to the pins of the pump, it will deliver water at an acceptable speed and amount. The pump operates for around 5seconds every time the pump is turned on. A tube is connected to the pump which will deliver water to the plants. To get the effect of rain and to water the entire surface, the end of the tube is stuffed with glue to create an inside pressure which will press the water out of small holes that are made in the tube.

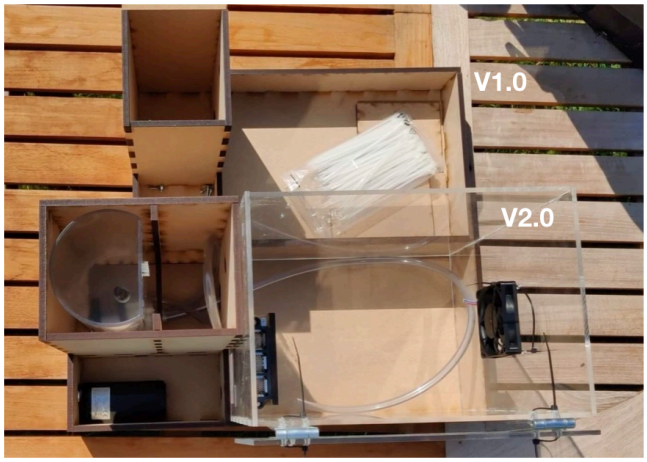
Servo Motor

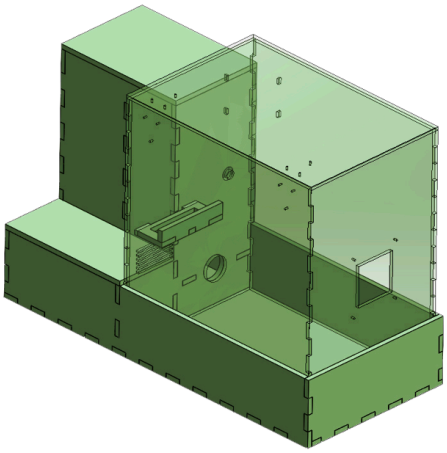
A servo motor is added in order to pull a lever of a water dispenser. The motor has 3 wires: GROUND , SIGNAL and 5V. Via the ESP32 the signal to the servo will be send. The servo will pull the water dispenser multiple times to create rain into the plant box. Because of poor performance and it takes a lot of space, this component is not used in the project.

Fan

The fan is powered with 5V. Transistors are used since I/O pins cannot power these components. The transistors should be able to withstand the current. They are used in saturation mode to dissipate as little power as possible.

3.2 Case

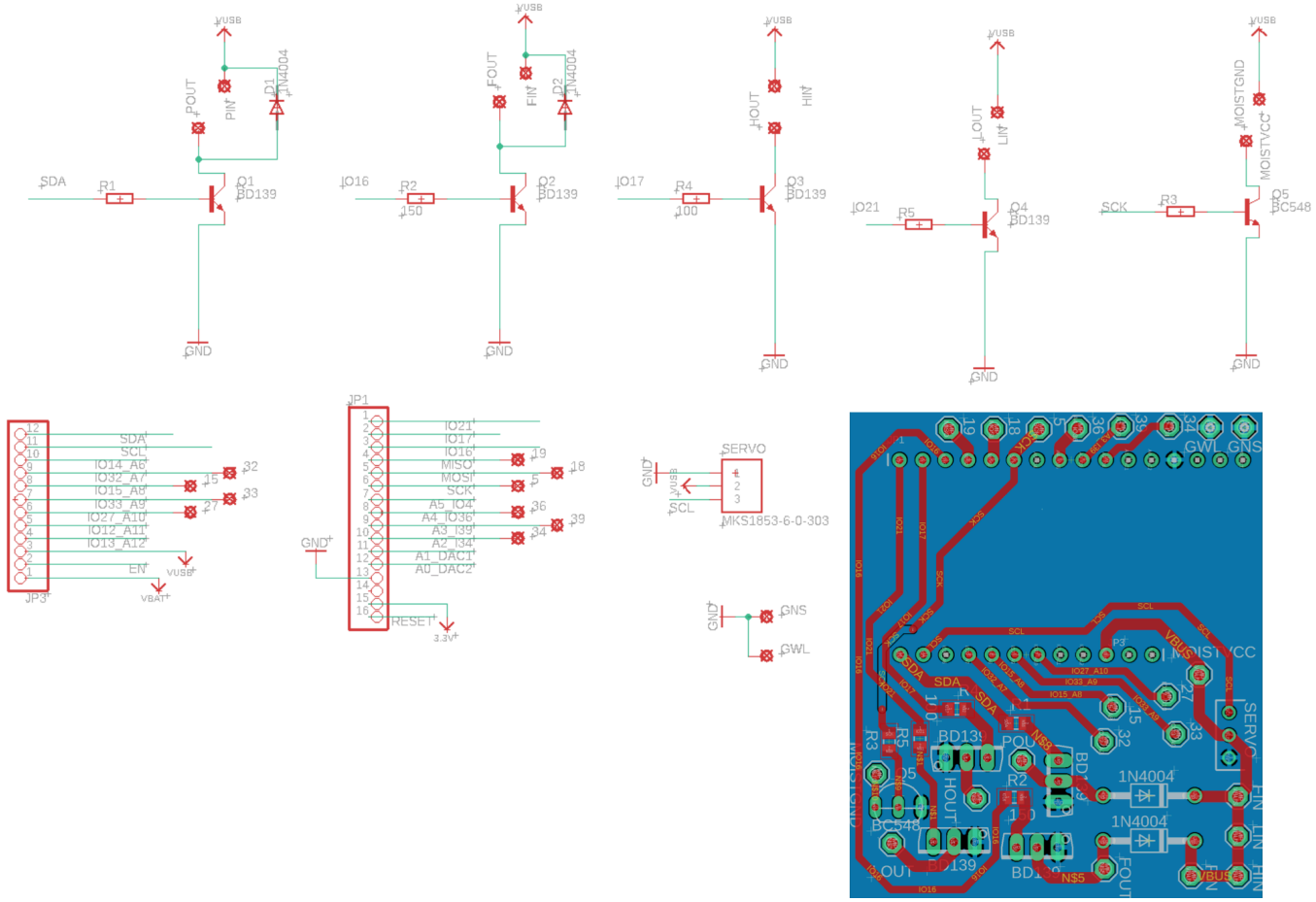




3.3 PCBs

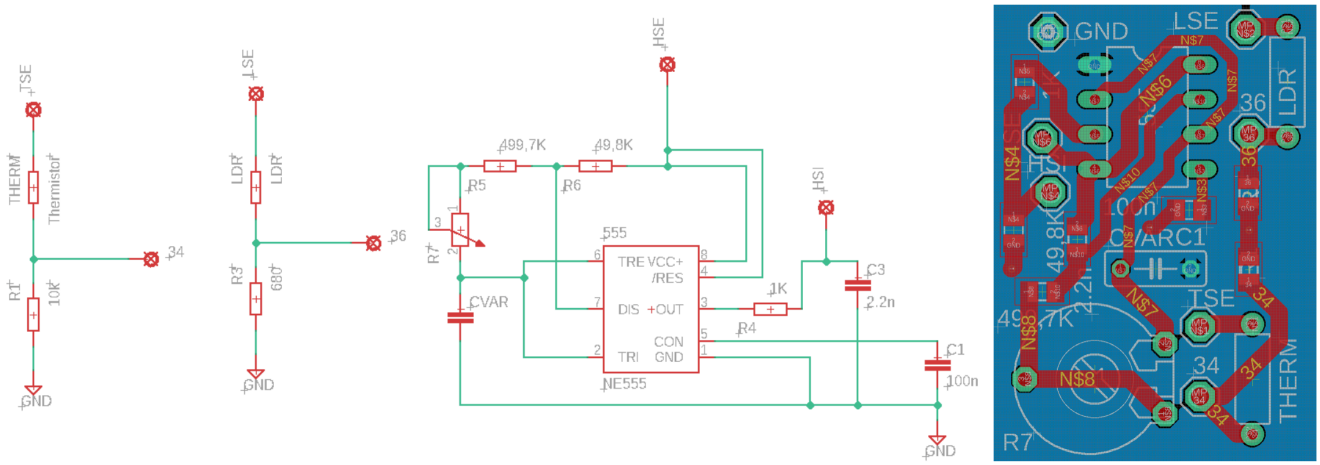
PCBs

Main PCB



The Main PCB holds the ESP32 and transistors to drive the actuators.

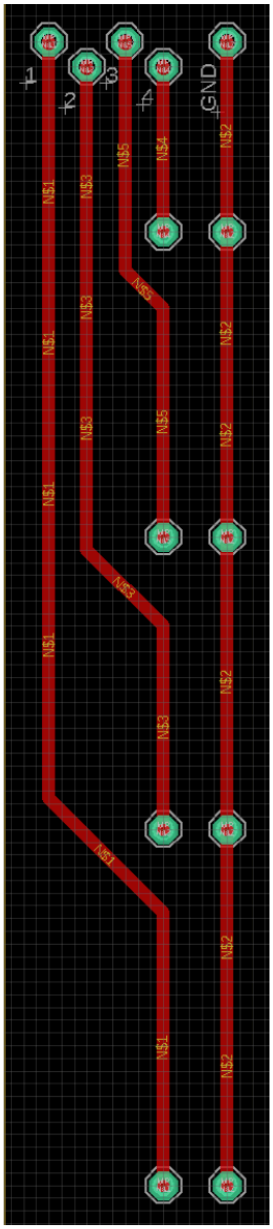
Sensors PCB

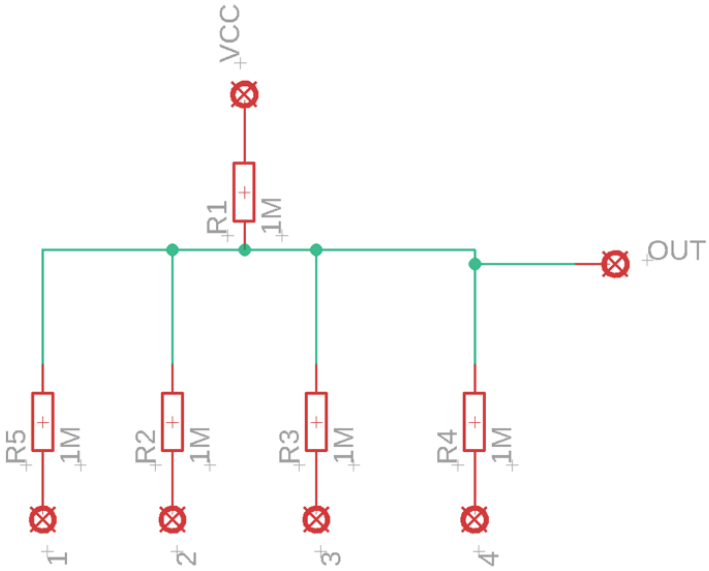


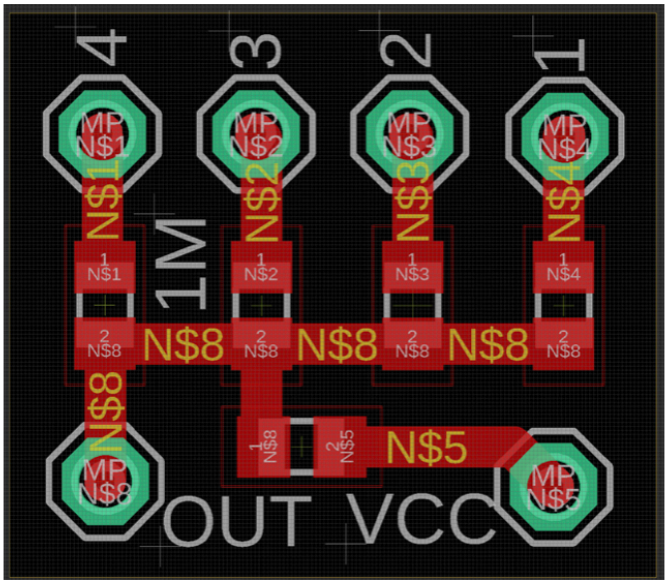
The Sensor PCB contains two voltage dividers, to measure the temperature and light change and a circuit to convert humidity sensor's capacity change to a frequency, that the ESP32 can measure.

Water Level Sensor PCB

This water level sensor has two PCBs: the long one shorts inside the water reservoir at different levels; the second one processes this signal through a voltage divider.


1
2
3
4
GND
N$1
N$2
N$3
N$4
N$5


VCC
R1
1M
OUT
R5
R2
R3
R4
1M
1
2
3
4


4
3
2
1
MP
N$1
N$2
N$3
N$4
N$8
N$5
1M
OUT
VCC

3.4 Sensors

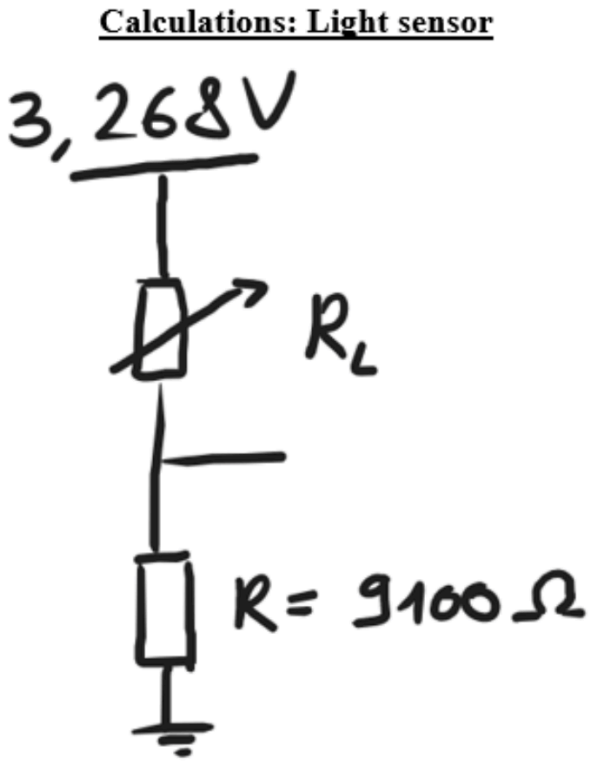
Sensors

The sensors measure the temperature, light intensity, air humidity, soil moisture and water level.

Thermistor and LDR

LDR to measure the light intensity. Thermistor to measure temperature. We use a voltage divider for each sensor to measure the changing voltage.

LDR in series with 9.1kΩ.



$$R_{\min} = 3k\Omega$$

$$R_{\max} = 300k\Omega$$

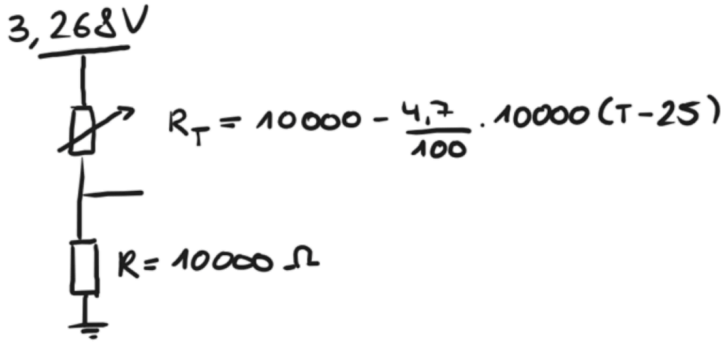
$$V_{out} = \frac{9100}{9100 + 3000} * 3.268 = 2400\ mV$$

$$V_{out} = \frac{9100}{9100 + 300000} * 3.268 = 96\ mV$$

$$\frac{V_{out} - 96}{2304} * 100 = \cdots \%$$

Thermistor in series with 10kΩ. Voltage is only applied while measuring to not heat the thermistor.

Calculations: Temperature sensor



$$V_{out} = 3.268 * \left(\frac{R}{R + R_T}\right)$$

$$\frac{V_{out}}{10000 * 3.268} = \frac{1}{10000 + 10000 - \frac{4.7}{100} 10000(T - 25)}$$

$$\frac{32680}{V_{out}} = 20000 - 470T + 11750$$

$$-470T = \frac{32680}{V_o} - 20000 - 11750$$

$$T = -\frac{69.5319}{V_o} + 67,55$$

Soil Moisture Sensor

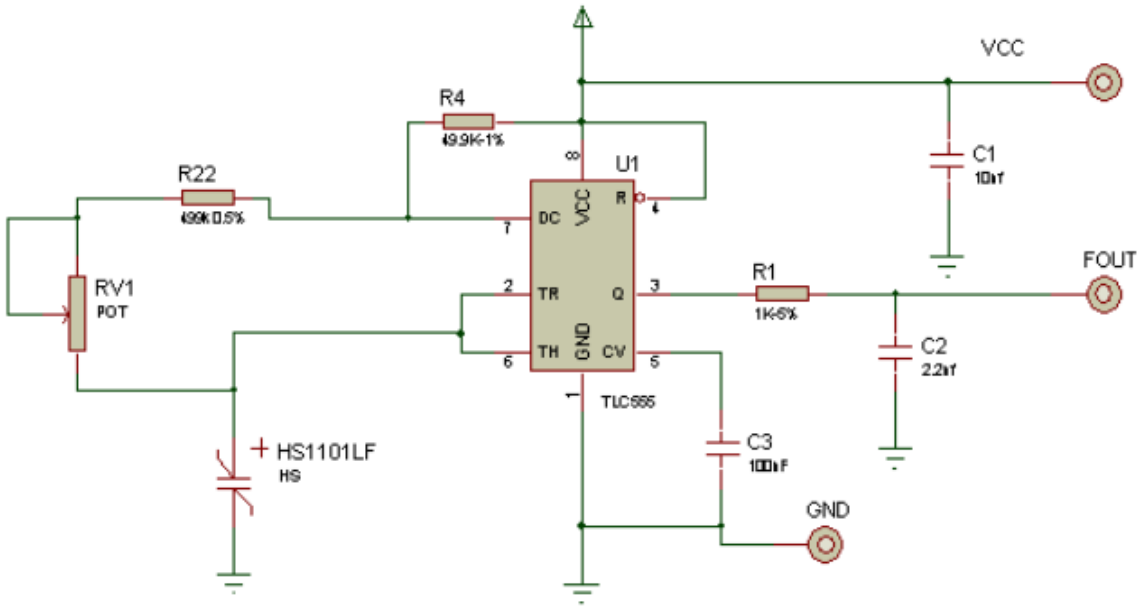
The moisture sensor applies a voltage over the soil. The wetter the soil, the more current flows. The sensor then returns the value as a voltage.

Moisture sensor (humidity)

The Humidity Sensor is a changing capacitance. The capacitance only changes by a few picofarads, which is hard to do an RC measurement. Instead, a 555 time converts the capacitance into a frequency, which the microcontroller can then read.

FREQUENCY OUTPUT CIRCUIT

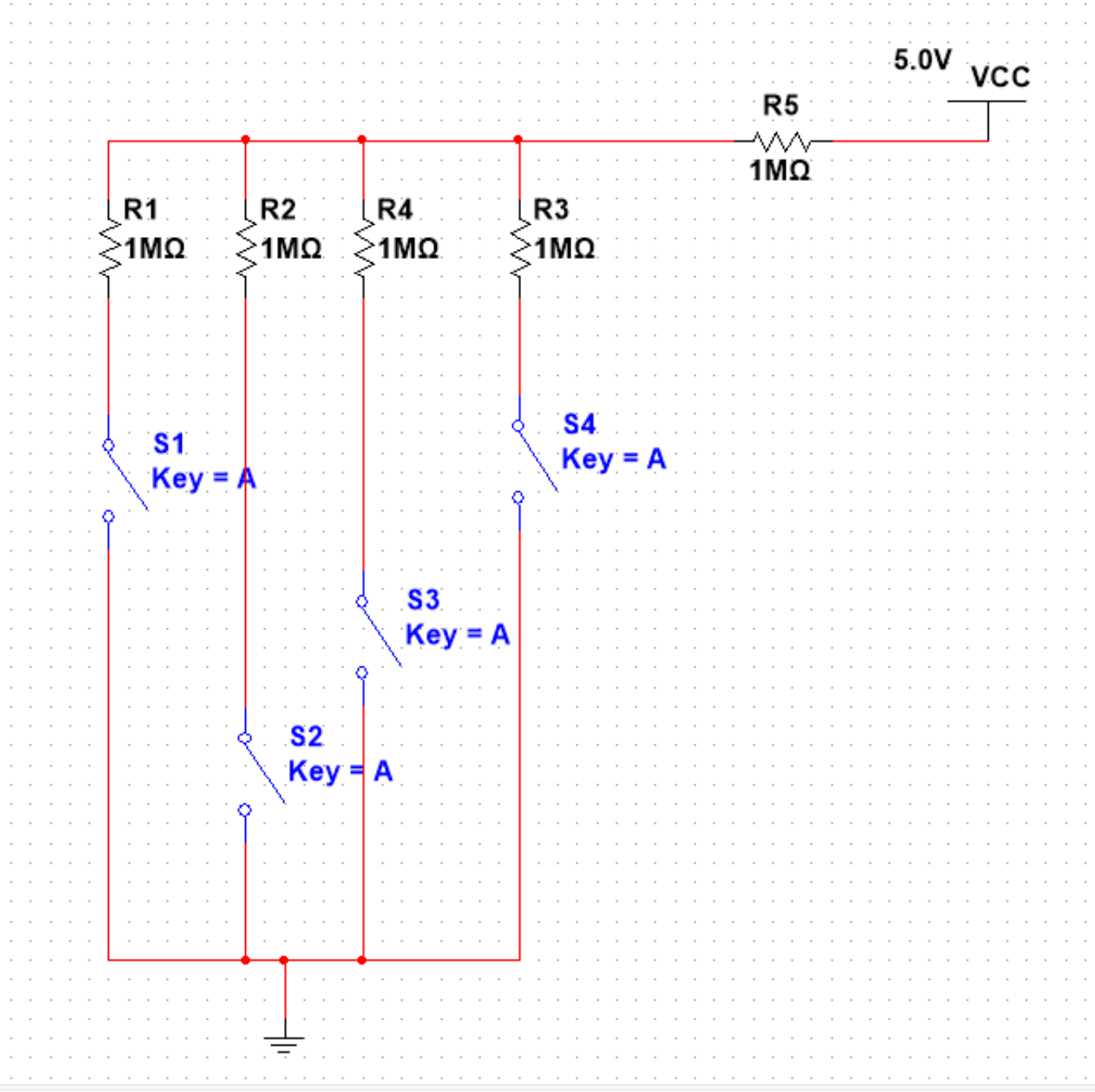
CIRCUIT



Note: R22=499kΩ / R4=49.9kΩ / R1=1 kΩ / RV1=50 kΩ potentiometer / C1=10nF / C2=2.2nF / C3=100nF

Water level sensor

The Water Level Sensor uses the water's ability to conduct electricity. The water shorts the soldering pins, and the output voltage drops. In the schematic below, the switches represent a closed circuit when there is water in the soldering pin.

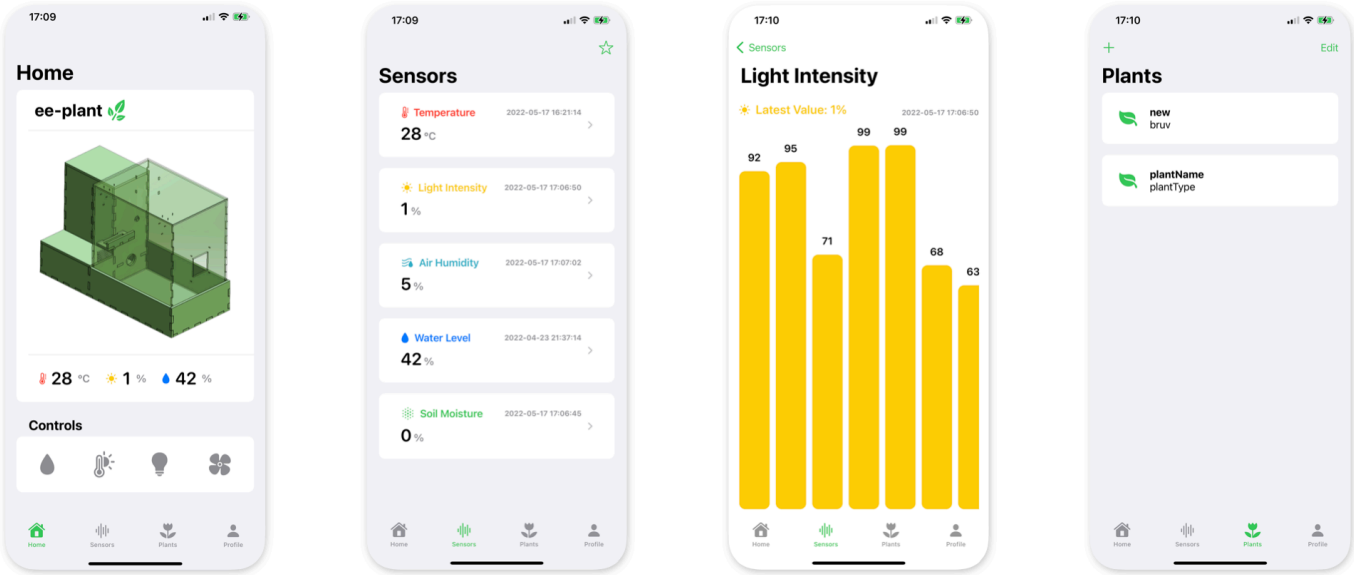


4. Software

Link to gitlab:
https://gitlab.groept.be/ee5/a21iot02

4.1 App

App Overview



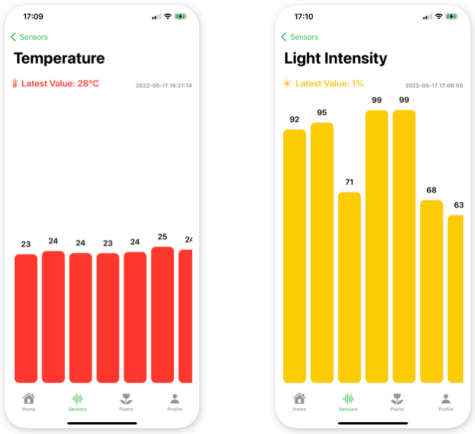
The app was developed in Xcode for iOS using Swift and the SwiftUI library.

Home

The Home Screen shows an overview of the system, values from *favorited* sensor and buttons to control the actuators. The user can manually heat, ventilate, toggle the LED strips and water the system.

Sensors

The Sensor Screen shows data from the Temperature, Light Intensity, Air Humidity, Water Level and Soli Moisture sensors. Further information is available via additional graphs.



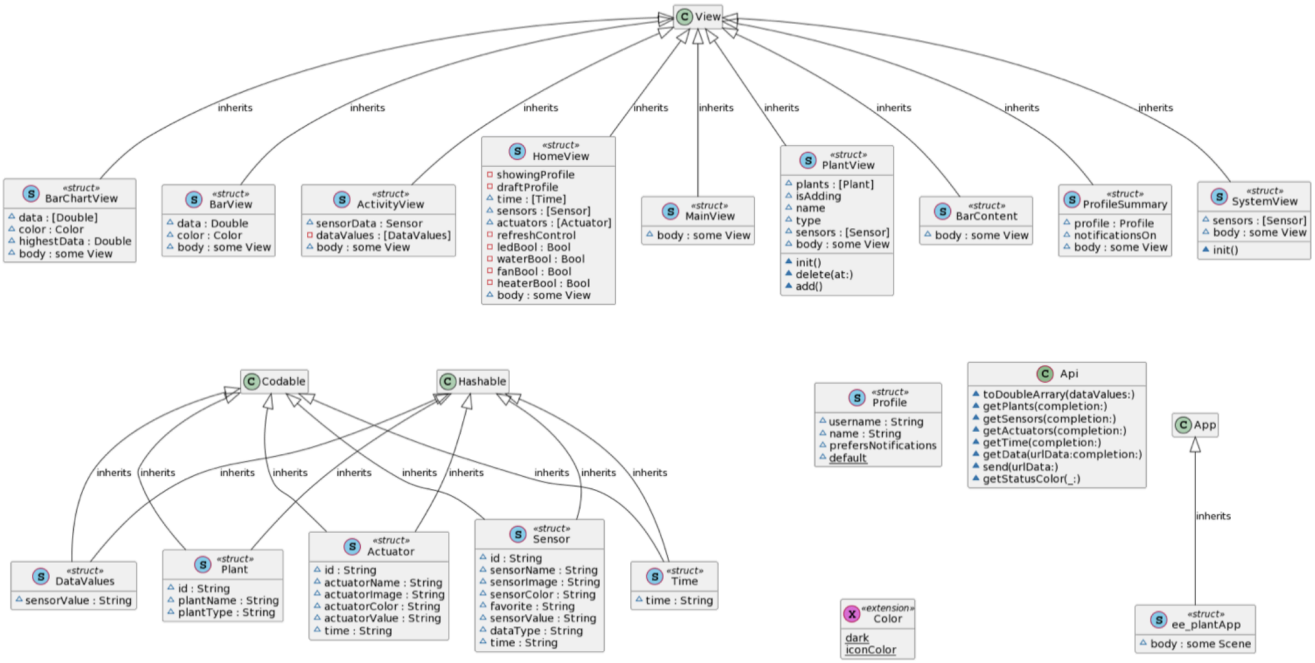
The bars change color based on the Sensor's color.

Plants

The Plants Screen allows the user to add and delete plants from the system. The user can give their plant a name and also add the plant's type. In future releases, the system can adjust the environment to be as close to the optimal conditions for each plant type.

Extra

UML Diagram



Generated using SwiftPlantUML

Dark Mode

API

Most of the API requests are handled through similar methods and return JSON data like:

```
func getSensors(completion: @escaping ([Sensor]) -> ()){
        guard let url = URL(string: "https://a21iot02.studev.groept.be/database/getSensor.php")
        else { return }
        @State var sensorDef: [Sensor] = []

        URLSession.shared.dataTask(with: url) { (data, _, _) in
            let sensors = try! JSONDecoder().decode([Sensor].self, from: data!)
            DispatchQueue.main.async {
                completion(sensors)
            }
        }
        .resume()
    }
```

```
{
  "id":"1",
  "sensorName":"Temperature",
  "sensorImage":"thermometer",
  "sensorColor":"red",
  "favorite":"1",
  "sensorValue":"25.939048",
  "dataType":"\u00b0C",
  "time":"2022-05-24 14:52:35"
  }
```

The `sensorImage` is not necessarily a real image. Instead, for example, by using `Image(systemName: "thermometer")` we receive a temperature icon from Apple's SF Symbols. These system's images' weight, color and size can change accordingly.

Relative Date

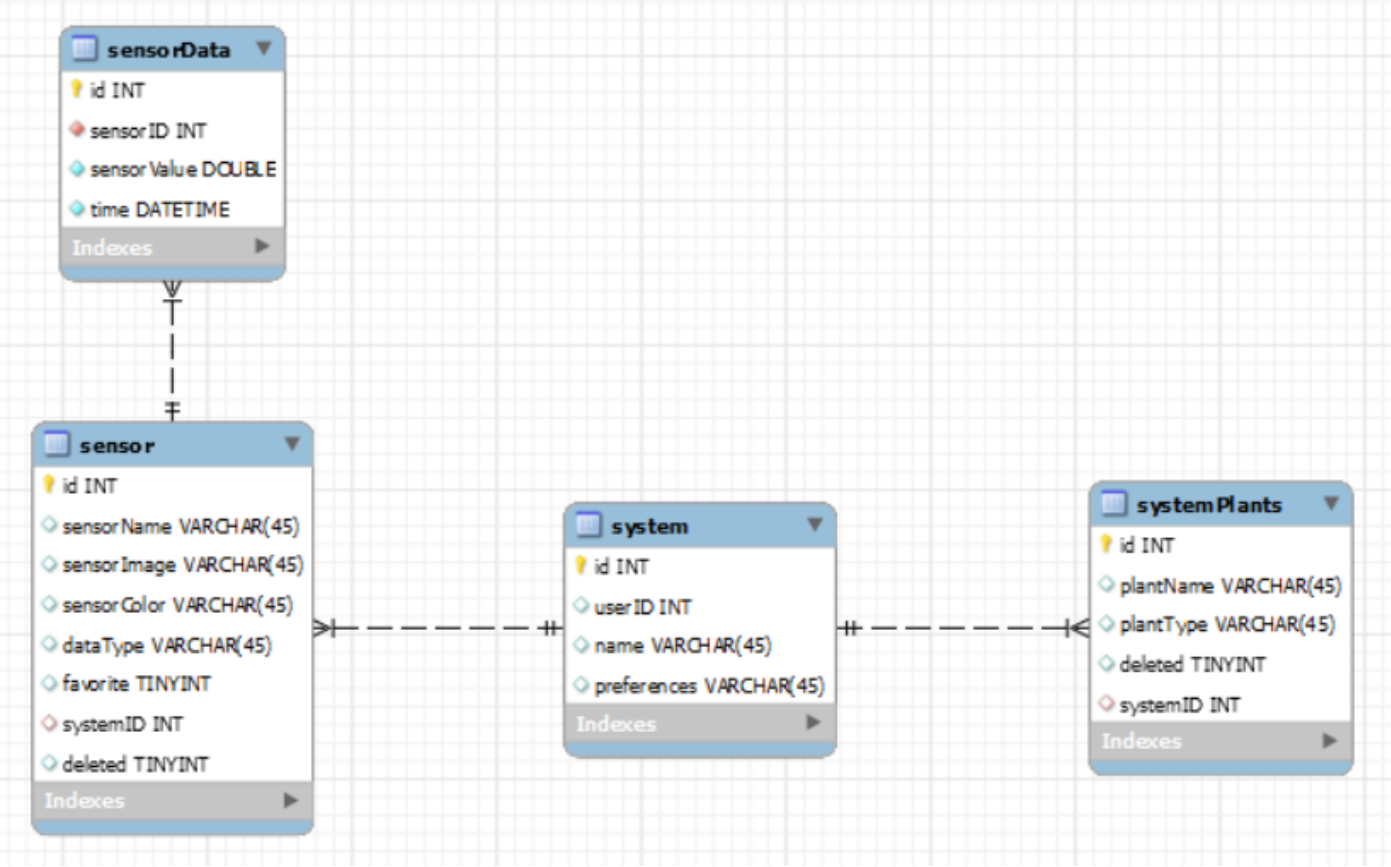
```swift
func getRelativeTime(time: String) -> String{
        let date = Date(timeIntervalSince1970: Double(time) ?? 0.0)
        let formatter = RelativeDateTimeFormatter()
        formatter.unitsStyle = .short
        let relativeDate = formatter.localizedString(for: date as Date, relativeTo: Date.now)
        return relativeDate
    }
```

From Absolute Time to Relative Time

4.2 Database

Database

Database Structure



system

- id: INT--> system's primary key
- userID: INT --> which user the system belongs to
- name: VARCHAR(45) --> plant's name, useful when multiple systems

systemPlants

- id: INT --> plant's primary key
- plantName: VARCHAR(45) --> plant's name, useful when multiple plants of the same type

- plantType: VARCHAR(45) --> plant's type
- deleted: TINYINT --> non-permanently deleting plants
- systemID: INT --> which system the plant belongs to

sensor

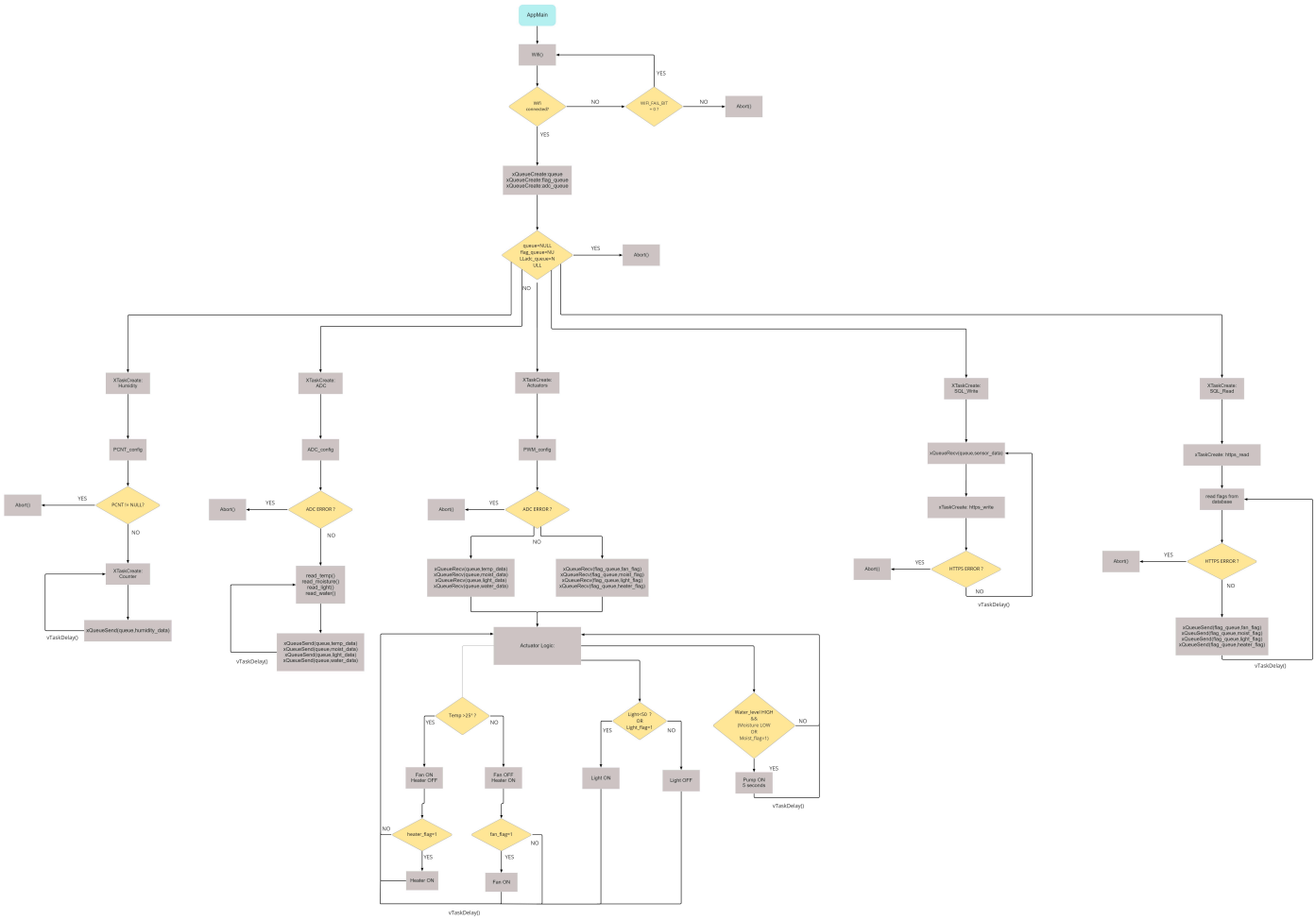
- id: INT --> sensor's primary key
- sensorName: VARCHAR(45) --> the sensor's name, usually its function
- sensorImage: VARCHAR(45) --> the SF Symbols used internally in the app
- sensorColor: VARCHAR(45) --> the color used internally in the app
- dataType: VARCHAR(45) --> Sensor's output unit (e.g. %, °C)
- favorite: TINYINT --> favorite sensors are shown in the Home Screen
- systemID: INT --> which system the sensor belongs to
- deleted: TINYINT --> non-permanently deleting sensors

sensorData

- id: INT --> sensorData's primary key
- sensorID: INT --> which sensor the data belongs to
- sensorValue: DOUBLE --> sensor output
- time: DATETIME --> server time of when the data was inserted

4.3 ESP32

Flowchart



Data communication

The ESP32 is connected through wi-fi to pass/fetch data to/from the server and database. The event handler connects the ESP with a local Access Point. At this moment SSID and password are hardcoded into the code for the correct login credentials. A wi-fi fail bit is set when the connection fails or is disrupted. A wi-fi connect bit is set when the connection is set-up. The maximal number of retries to re-connect to the AP is 10.

Through wi-fi with a https protocol, php files can be requested to fetch data from the server (see Server).

Main

The ESP32 process starts in the appMain function. This function starts by initializing the wifi and setting up a connection. If a connection is successfully setup the 3 queues for inter-task communication will be created. All the different Tasks will now be created, these tasks are: HumidityTask, ADCTask, SQLWriteTask, SQLReadTask, ActuatorTask. Each of these tasks will now run in parallel and do their separate things.

Tasks

HumidityTask

The HumidityTask is the task that is responsible for the acquisition of the air humidity data. The task is initialized with the function start_humdity which can be found in the airHumidity.c file.

The start_humidity function start by initializing the necessary GPIO pins which are GPIO_32 for the sensor enable pin and GPIO_22 for the sensor input pin. This function also configures the PCNT (Pulse Counter). If there are no errors this task will create a new task called counterTask. This task counts the amount of rising edges the air humidity sensor produces in x amount of seconds at regular intervals. From these counts we can derive the frequency from which we can derive the air humidity value using the graphs from the datasheet. This value is then send to the queue to be handled by the SQL task and to the adc_queue to be handled by the ActuatorTask.

The air humidity is read out every 30 seconds but this can be easily changed by changing the amount of time the task is delayed between each read.

ADCTask

The ADCTask is the task that is responsible for the aquisition of data for the temperature sensor, light sensor, soil moisture sensor and the water level sensor. All these sensors are done in this one tasks because they all need to use the built in ADC. This tasks is initialized with the start_adc function which can be found in the adc.c file. The function starts of by configuring the ADC which is in our use case the ADC1 module.

The GPIO pins that are used by this task are split into two categories. First we have the enable pins, the goal of these pins is to enable the sensor so it is only active when we want

to read from them and not when we are waiting unil our next read. This both saves power and increases sensor life time. The enable pins are as follows: GPIO_19 for temperature enable, GPIO_18 for light enable, GPIO_5 for moisture enable and GPIO_27 for level enable. For the second category we have the sensor input pins which are connected to the ADC channels. We use the ADC1 module so we need to use adc1 channels. For the channels the GPIO pins are as follows: GPIO_36 for light input, GPIO_39 for moisture input, GPIO_33 for water level input and GPIO_34 for temperature input.

The ADC task will read out each sensor one by one with a delay of half a second between each read. The raw values read out by the ADC will be converted to their appropriate values like temperature or light intensity. These values will then be send to the queue where they will be handled by the SQL task.

All sensors will be read out every 30 seconds but this can also easily be changed by changing the amount of time the task is delayed between each read.

SQLWriteTask

The SQLWriteTask is the task that reads out the data from queue that was inserted by both the ADCTask and HumidityTask. The task is initialized with the start_sql_write function which can be found in the appMain.c file.

The task continiously looks for new data in the queue, when new data arrives it is concatenated into the base URL string which will then be used by https_write_task to send the data to the database. The https_write_task is created by the SQLWriteTask. After the data is send by the https_write_task, the task will self delete and the cycle will start again. There is a delay of 4 seconds between each datapoint that is send to the database to give the http_write_task time to send the data.

SQLReadTask

The SQLReadTask is the task that reads out data from the database. The task is initialized with the start_sql_read function which can be found in the appMain.c file.

This task will create the https_read_task that will use the provided URL to read data from the database. In this case the task reads out the flags from the database so that when a flag is set in the app the flag will also be updated in the c code. Once the https_read_task has read out the flag values it will put these values in the flag_queue which will be used by the ActuatorTask to enable actuators. After each read the https_read_task will self delete and will be created again by the SQLReadTask.

This task reads out data from the database every 4 seconds.

ActuatorTask

The ActuatorTask is the task that reads out data from the adc_queue and flag_queue. The adc_queue contains the read out values from ADCTask and the flag_queue contains the flag values that were read out by the SQLReadTask. This task is initialized with the start_actuators function which can be found in the actuators.c file.

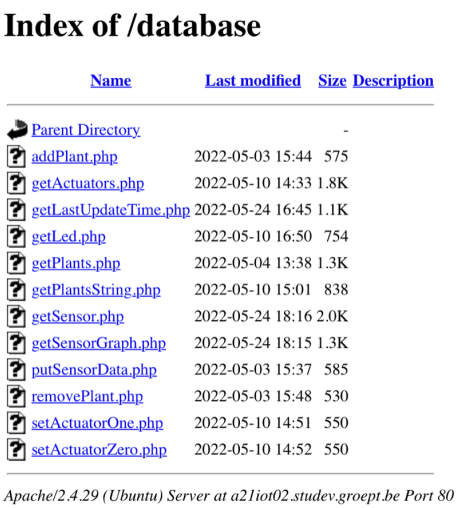
The task start by initializing the PWM module that will be used by the heater. After the config is done it will start reading data from the queues and based on this data will enable/disable certain actuators. The heater/fan will react to the temperature value and temperature flag. If the temperature value is higher than a certain value the fan will turn on, it is lower the heater will turn on. When an actuator is off a flag can overwrite this and turn it on nonetheless. The LEDs will react to the light sensor value. The pump will react to the soil moisture value and the water level. If the soil moisture is low the pump will turn on but only if there is enough water left in the reservoir. The pump is turned on for 5 seconds after which it disables itself.

The actuator task has no delay and will just wait on new values to come into the queues and change it's outputs based on this.

4.4 Server

PHP

For database communication from our ESP to the database, we are using PHP scripts. These scripts are stored on the server and can be executed by running an HTTP request on the ESP32. Below is an overview of the PHP files we are using and an example from getSensorGraph.php



Index of /database

Name	Last modified	Size	Description
Parent Directory		-	
addPlant.php	2022-05-03 15:44	575	
getActuators.php	2022-05-10 14:33	1.8K	
getLastUpdateTime.php	2022-05-24 16:45	1.1K	
getLed.php	2022-05-10 16:50	754	
getPlants.php	2022-05-04 13:38	1.3K	
getPlantsString.php	2022-05-10 15:01	838	
getSensor.php	2022-05-24 18:16	2.0K	
getSensorGraph.php	2022-05-24 18:15	1.3K	
putSensorData.php	2022-05-03 15:37	585	
removePlant.php	2022-05-03 15:48	530	
setActuatorOne.php	2022-05-10 14:51	550	
setActuatorZero.php	2022-05-10 14:52	550	

Apache/2.4.29 (Ubuntu) Server at a21iot02.studev.groept.be Port 80

```php
<?php
$servername = "mysql.studev.groept.be";
$username = "a21iot02";
$password = "dontstealourserverpls";

// Create connection
$conn = new mysqli($servername, $username, $password, "", 3306);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$id = $_GET['id'];

$sql = "SELECT sensorValue, UNIX_TIMESTAMP(time) FROM a21iot02.sensorDataWHERE sensorID = $id & sensorValue < 101;";

$result = $conn->query($sql);

if ($result->num_rows > 0) {
    $array = array();
    while($row = mysqli_fetch_assoc($result))
    {
        $sensorValue = $row["sensorValue"];
        $time = $row["UNIX_TIMESTAMP(time)"];

        $myObj = new stdClass();
        $myObj->sensorValue = $sensorValue;
        $myObj->time = $time;

        $myJSON = json_encode($myObj);
        array_push($array, $myJSON);
    }
    echo "[ ";
    foreach ($array as $key => $value) {
        echo $value;
        if (next($array )) {
            echo ',';
        }
    }
    echo " ]";

  } else {
    echo "0 results";
  }
$conn->close();
?>
```

5. Videos

Sprint 1

In sprint 1 we worked mainly on the system's sensors.

The following user stories were closed:

- As a user I want to organise the data in databases
- As a user I want to know the temperature
- As a user I want to know the humidity and the moisture
- As a user I want to know the water level of the reservoir
- As a user I want to know if the plant is in light

Video Sprint 1

Sprint 2

The following user stories were closed:

- As a user I want to know the water level of the reservoir
- As a user I want to know if the plant is in light
- As a user I want to know the humidity and the moisture
- As a user I want to know the temperature
- As a user I want to organise the data in databases
- As a user I want a nice exterior design/case

Progress was made in the following:

- As a user I want to have a custom REST API
- As a user I want an Android App to control the device
- As a user I want to control the lighting

Video Sprint 2

Sprint 3

The following user stories were closed:

- As a user I want an iOS app to control the device
- As a user I want to have a smart device
- As a user I want everything to fit in the same case
- As a user I want to water the plant
- As a user I want to have web connectivity
- As a user I want to have as minimal wiring as possible (PCB)

Video Sprint 3

Sprint 4

The following user stories were closed:

- As a user I want a reliable product