UniTBv

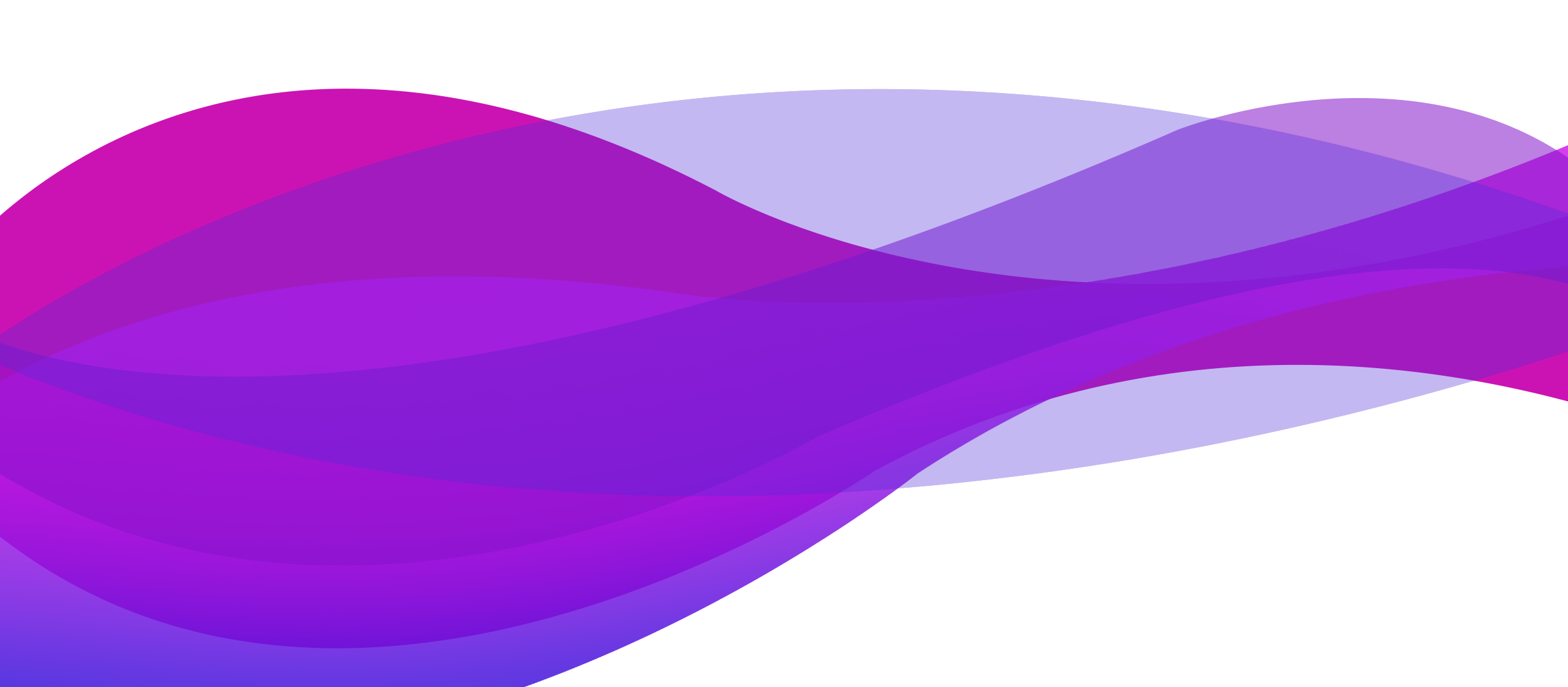
Bănucu GH. George

Brezuică S. Andreea-Alexandra

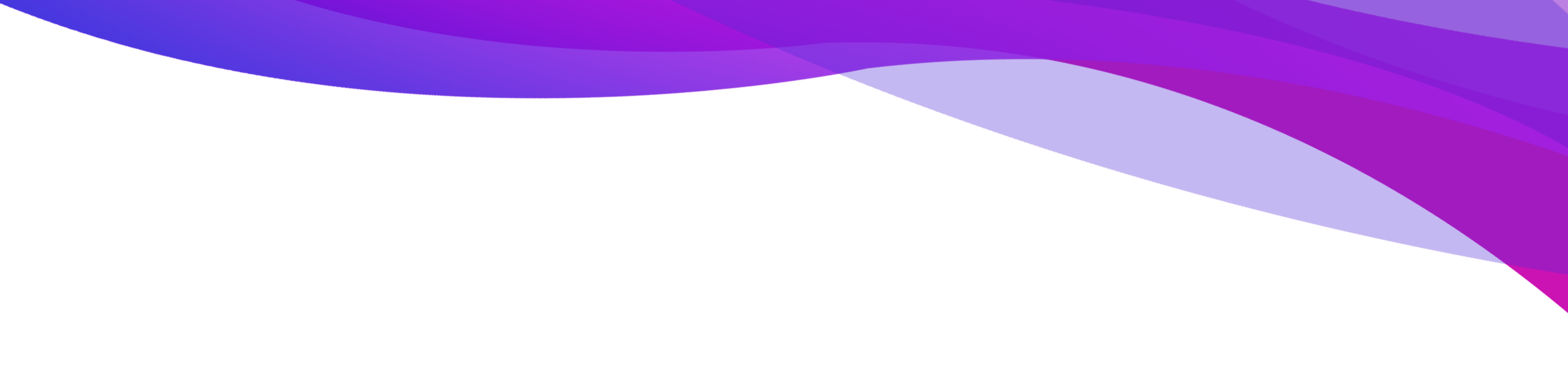
Bularca A. Alexandru

Cârnu F. Ferencz

Chivoiu C. Ștefan Gabriel



Botani Talk



**2024**

Laboratory Work – CIP

Contents

[1.Objectives of the work 3](#_Toc156393294)

[1.1. Meet and Greet 3](#_Toc156393295)

[1.2. Project functionality 4](#_Toc156393296)

[2. Component description 5](#_Toc156393297)

[2.1. Technical Features 5](#_Toc156393298)

[3. Description of the work 8](#_Toc156393299)

[3.1. Overview 8](#_Toc156393300)

[3.1.1. Core functionality 8](#_Toc156393301)

[3.1.2. User-friendly Design 8](#_Toc156393302)

[3.1.3. Relevance 8](#_Toc156393303)

[3.1.4. Applications 8](#_Toc156393304)

[3.2. Electric diagram 9](#_Toc156393305)

[3.3. Electric scheme 10](#_Toc156393306)

[3.4. Software Diagram 11](#_Toc156393307)

[3.5. Connections 12](#_Toc156393308)

[3.6. Our evolution 16](#_Toc156393309)

[4. Working with registers 20](#_Toc156393310)

[4.1. Water Pump and LEDs pins 20](#_Toc156393311)

[4.2. Sensor temperature pins 21](#_Toc156393312)

[5. Code 22](#_Toc156393313)

[5.1. config.h 22](#_Toc156393314)

[5.2. display.h 23](#_Toc156393315)

[5.3. network.h 27](#_Toc156393316)

[5.4. main\_sketch.ino 29](#_Toc156393317)

[5.4.1. Setup Function 29](#_Toc156393318)

[5.4.2. Loop function 31](#_Toc156393319)

[6. Working with IFTTT 34](#_Toc156393320)

[7. Working with GitHub 36](#_Toc156393321)

[8. Bibliography 36](#_Toc156393322)

[9. Finally… 36](#_Toc156393323)

# 1.Objectives of the work

## Meet and Greet

Welcome to Botani Talk, an Arduino-based system designed to revolutionize indoor environmental management. Developed by third-year students eager to delve into sensor technology, this project seamlessly blends simplicity with sophistication. Botani Talk focuses on optimizing your living space by intelligently monitoring and managing crucial factors such as soil moisture, light intensity, and temperature.

At its core, Botani Talk employs a diverse array of sensors, including moisture sensors for precise plant hydration, light sensors to ensure optimal illumination, and temperature sensors for maintaining a comfortable indoor climate. Complemented by a vibrant LED strip, this system not only enhances energy efficiency but also creates a visually appealing atmosphere.

Imagine a world where your indoor environment is not just monitored but actively adjusted to meet your comfort needs. Botani Talk achieves this through an intuitive interface and a control module powered by Arduino, making it accessible to users with varying technical backgrounds. This project introduces you to the realm of sensors, microcontrollers, and actuators, providing a hands-on experience in crafting intelligent, user-friendly systems.

But that's not all! Botani Talk goes a step further by incorporating an automated email notification system. Picture receiving timely alerts and personalized recommendations when environmental conditions deviate from the optimal range. It's like having a smart companion for your plants and indoor space, ensuring they thrive in the best possible conditions.

As a student embarking on this exciting journey, Botani Talk aims to be your gateway into the world of sensor-based technology. Together, let's explore how technology can seamlessly integrate with our daily lives, creating smarter, eco-friendly, and more comfortable living spaces. Join us on this adventure, where innovation meets simplicity, and discover the endless possibilities of intelligent indoor environmental management!

## Project functionality

Considering the project’s functionality, some essential modules include:

* Communication module

**Functionality:** - Facilitates communication between the Arduino and the user through email notifications. It ensures timely alerts and recommendations, enhancing the system’s accessibility and user engagement.

* Control module

**Functionality:** - “The brain” of the project, process sensor data and execute control logic. The Arduino microcontroller interprets sensor inputs and triggers actions such as adjusting lights or activating the watering motor.

* Power Supply

**Functionality:** - Manages the power distribution to various components, ensuring a stable and reliable power supply. This module is crucial for the continuous operation of the sensors, microcontroller, and the other peripherals.

* Actuation Module - MOS Transistor

**Functionality:** - Controls the motorized watering mechanism based on the system’s decisions. It interprets commands from the microcontroller and regulates the motor’s speed to efficiently water the plant as needed.

* LED Control Module

**Functionality:** - Manages the state and behavior of the LED strips. It interprets commands from the microcontroller to adjust the lighting conditions based on the light sensor input, contributing to energy efficiency and creating a comfortable indoor environment.

* LCD Display Module

**Functionality:** - Provides a visual interface for real-time feedback and system status. The LCD display communicates information such as sensor readings, status of the plant.

* Sensor Modules

**Functionality:** - Acts as a hub for various sensors (moisture, light, temperature) by collecting and relaying data to the microcontroller. This module is critical for continuous environmental monitoring, enabling the system to make informed decisions for optimal conditions.

These modules work together to ensure the seamless operation of the Botani Talk project, integrating sensor data and automated responses to create an intelligent and user-friendly indoor environmental management system.

# Component description

The laboratory project will use Arduino UNO development board with the ATMega328p microcontroller.

## Technical Features

1. **Arduino UNO development board + USB A-B cable**

* Operating voltage: 5V;
* Power Jack voltage: 7V - 12V;
* I/O pins: 14;
* PWM pins: 6 (out of 14 I/O pins);
* ADC pins: 8;
* Flash memory: 32kB (8 occupied by bootloader);
* TWI, SPI and UART communication;
* Operating frequency: 16MHz.

1. **ATMega328p Microcontroller**

* Supply voltage: 1.8V - 5.5V;
* Capsule: 28DIP;
* Operating frequency: up to 20MHz;
* Flash: 32kB;
* 23 I/O pins;
* 1 UART, 2 SPI and 1 I2C;
* 6 channels of PWM;
* 8 x 10 bit ADCs.

1. **Arduino Ethernet Shield 2**

* Operating voltage: 5 V (supplied from the Arduino Board);
* Ethernet Controller: W5500 with internal 32K buffer;
* Connection speed: 10/100Mb;
* Connection with Arduino: on SPI port;

PoE module features:

* IEEE802.3af compliant
* Input voltage range 36 to 57 V;
* Overload and Short-circuit protection;
* Output: 12V;
* High efficiency DC/DC Converter: typ 85% @ 80% load;
* Isolation: 1500V (input to output);
* Shield: standard Rj45 Ethernet jack;
* Reset: resets both the W5500 and the Arduino Board;
* LEDs: ON – Board and Shield – powered;

13 – Arduino standard built in LED;

ACT – flashes when RX or TX activity is present;

LINK – the presence of a network link and flashes when the Shield transmits or receives data.

1. **1.44inch Arduino SPI Module MAR1441**

* Display Color: 16BIT RGB 65K color;
* SKU: MAR1441;
* Screen Size: 1.44[inch];
* Type: TFT;
* Driver IC: ST7735S;
* Resolution: 128\*128 [pixel];
* Module Interface: 4-Wire SPI interface;
* Backlight: 1 White LED;
* Active Area: 26.2 x 27.2 [mm];
* Module PCB Size: 31.49 x 43.95 [mm];
* Angle of view: <=60;
* Operating Temperature: -10[0C]- 60[0C];
* Storage Temperature: -20[0C] – 70[0C];
* Operating Voltage: 3.3[V] /5V;
* Power Consumption: About 90mA;
* Product Weight: About 25(g);
* Module Pins: VCC- LCD power supply positive pin;

2x GND- LCD Power ground pin;

2x NC – not defined, reserved;

LED – LCD backlight control pin (high level lighting, if no control is required, connect directly to 5V/3.3V)

CLK -LCD SPI bus clock pin;

SDI – LCD SPI bus write data pin;

RS – LCD data/ command selection control pin (low level: command; high level: data);

RST – LCD reset control pin (reset at low level);

CS – LCD chip select control pin(enabled at low level).

1. **LDR Sensor**

* Diameter: 201;
* Max. DC Voltage: 150;
* Power Dissipation: 90 [mW];
* Light Resistance(10Lux) (K) : 50-100;
* Dark Resistance (M): 0.2;
* Response Times (ms): Increase: 30; Decrease: 30;
* Ambient Temperature Range: -30 to +70[0C].

1. **NTC Temperature Sensor Breakout (Digital Out) Lm393**

* Operating Voltage: 3.3 to 5 [V];
* Max Operating Current: 15[mA];
* Temperature Range: -20[0C] -> +105[0C];
* Output Digital: -0V to 5V, Adjustable trigger level from preset;
* Output Analog: -0V to 5V based on temperature on ambience;
* PCB size: 3.2 [cm] x 1.4 [cm].

1. **DIY Moisture Sensor**

* Operating Voltage: 5[V];
* Resistance: 1 [MΩ];

1. **LEDs**

**Blue White**

* Voltage Forward @IF=20 mA 3.0-3.4 3.0-3.4
* Wavelength (nm) or TC(K) 460-470 6000-6500K
* IV (mcd or lm) 200-300 3-5LM
* Viewing Angle (0) 120 120

1. **Water Pump**

* Cable length: approx.. 100[cm];
* Flow: max. 10l/min;
* Pressure: max 0,5 bar;
* Consumption: 10-18 [W];
* Color: blue;
* Operating Voltage: 12 [V];
* Diameter: 3,8 [cm];
* Height: 10,4 [cm];
* Pumping Height: max. 5 [m];

# Description of the work

## Overview

Botani Talk is an Arduino-based system designed for the intelligent monitoring and management of indoor environmental conditions. It leverages an array of sensors, including moisture, light, and temperature sensors, to create a comprehensive solution that ensures optimal indoor conditions for plants and human comfort. The project integrates an automated email notification system, providing users with timely reminders and recommendations for maintaining a healthy and energy-efficient indoor environment.

### Core functionality

1. Sensor integration – Monitors key environmental parameters, including soil moisture, light intensity, and temperature, ensuring optimal conditions for plant growth and human comfort.
2. LED control – Manages a band of LEDs that dynamically adjusts based on ambient light levels, contributing to energy efficiency, and creating a comfortable living space.
3. Motorized Plant Watering – Incorporates a motorized watering mechanism to autonomously water plants based on soil moisture levels, promoting efficient plant care.
4. Automated Email Notifications with HTTP requests using IFTTT(If This Then That) – Alerts users via email, when environmental conditions fall outside predefined optimal ranges. Provides notes for starting the water pump and turning on the LEDs.

### User-friendly Design

Intuitive interface for easy monitoring with the LCD, making the system accessible to user with varying technical expertise.

The project is designed to enhance the quality of life by contributing to energy efficiency, cost savings, and creating a healthier and more comfortable living environment.

### Relevance

Addresses the growing importance of home automation and the need for optimizing indoor environmental quality.

Aligns with the global shift toward sustainability and energy conservation by promoting resource-conscious living and environmental responsibility.

### Applications

Ideal for smart homes, indoor gardens, and spaces where plants and human comfort are a priority.

## Electric diagram

A diagram of a circuit board

Description automatically generated

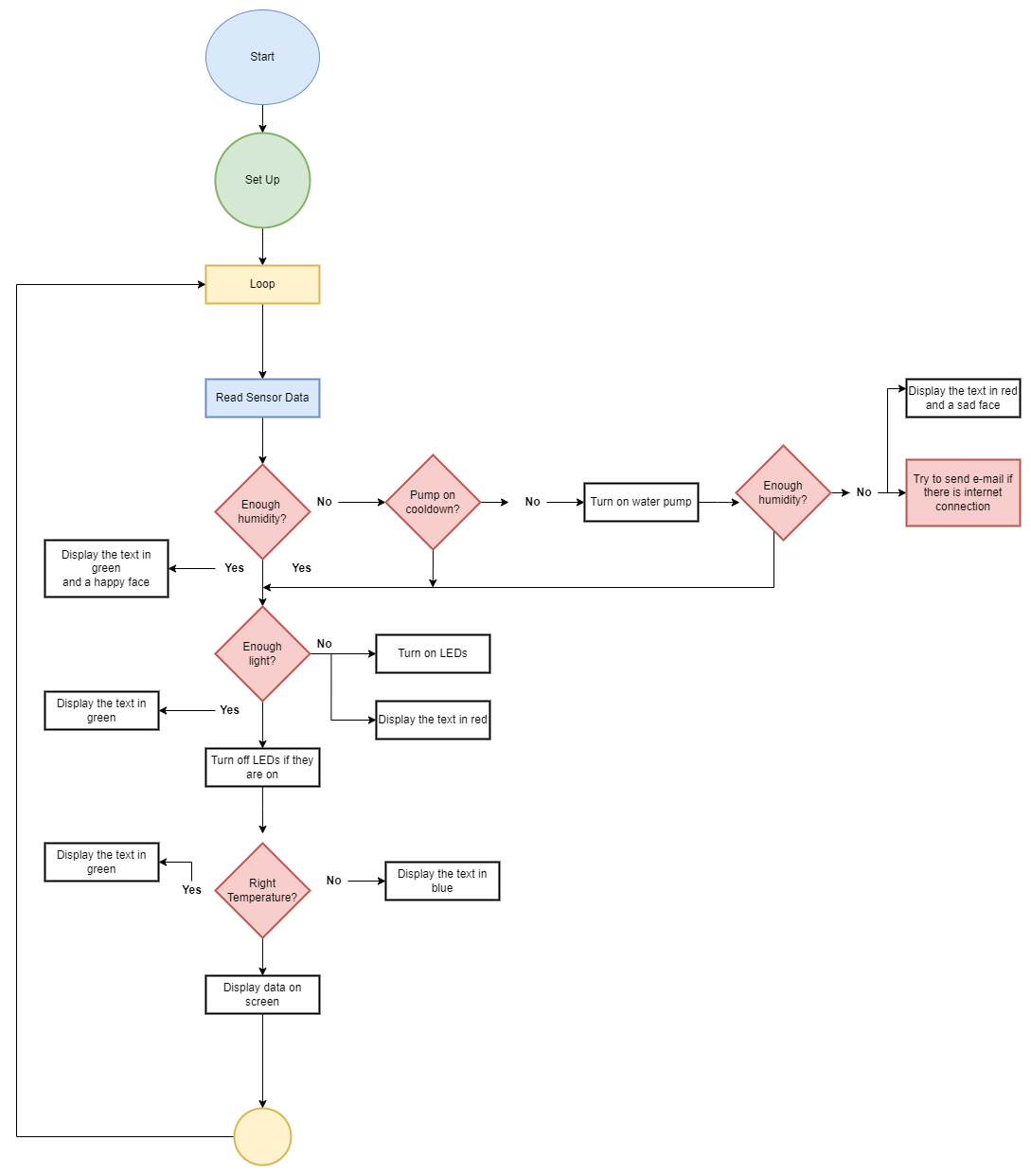
## Electric scheme

## Software Diagram

In the setup was done the configuration of the pins, initialization of the display along with its coloring, and last but not least the connection with the Internet, using the interface of Ethernet shield 2.

In the loop block, the sensors are read and, based on the results, actions are performing.

Cooldown is referring to a time that must pass before the pump starts. It is used so that the plant is not watered too often.



## Connections

* LCD Module

A small rectangular electronic device

Description automatically generated

A red electronic device with black wires

Description automatically generated

* GND -> GND;
* LED -> 5V;
* SCL ->PIN 8;
* SDA ->PIN 9;
* RS ->PIN 10;
* RST -> PIN 11;
* CS ->PIN 12;
* Water Pump

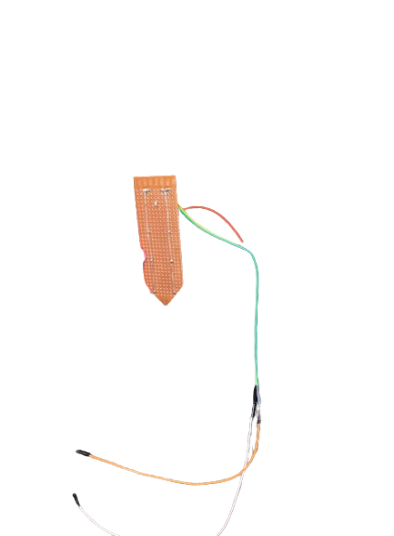
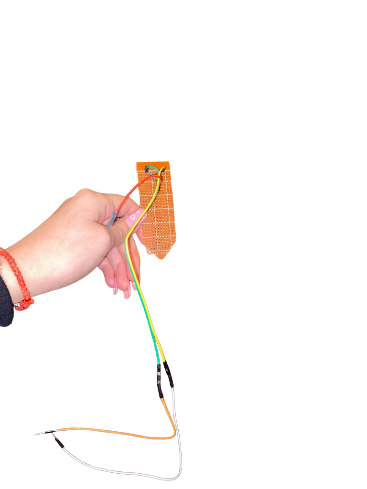


The ac-DC adaptor:

* Positive (12V): Drain;
* Negative: Ground.

The pump:

* Positive: Anode;
* Negative: Cathode.
* DIY Soil Humidity Sensor

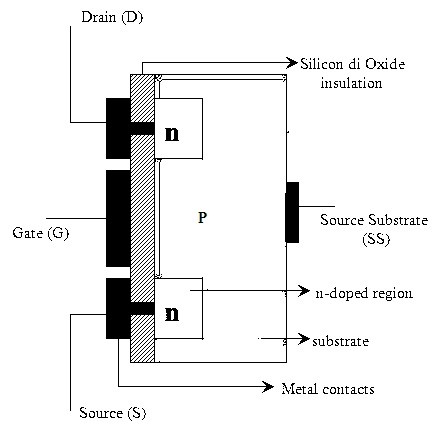


* Positive: 5V;
* Negative: GND;
* Reading: A4.

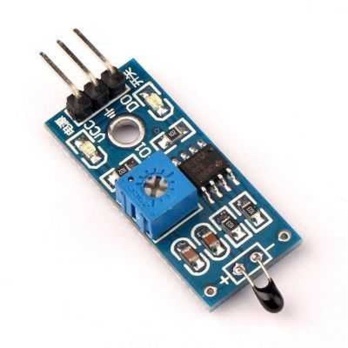
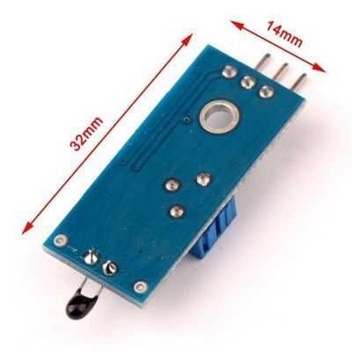
By placing two wires in a soil pot, a variable resistor is formed, and its resistance changes based on the moisture content of the soil. This variable resistor is incorporated into a voltage divider setup, where an Arduino measures a voltage that is proportionate to the resistance between the two wires. Essentially, as the soil becomes more humid, the Arduino detects a lower voltage. With the use of a 1 Mega Ohm resistor and two wires, one can construct a homemade soil moisture sensor.

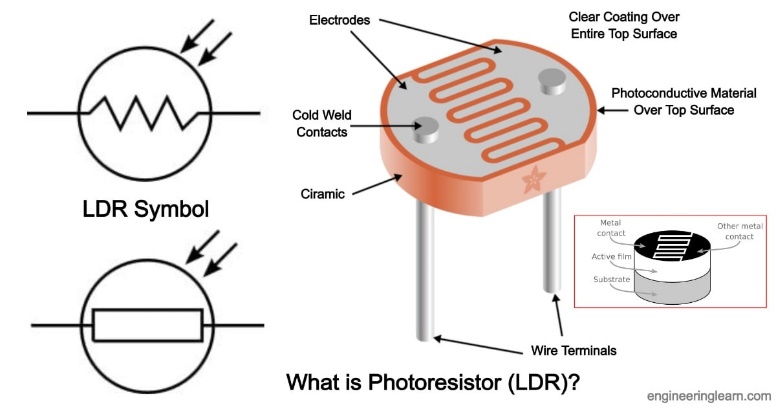
* NMOS

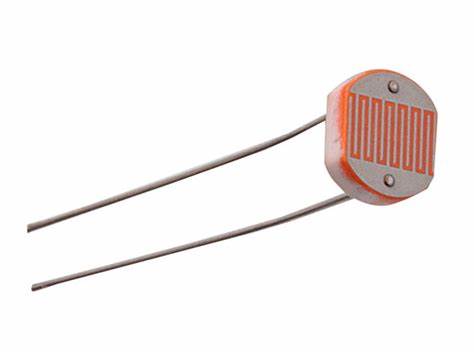




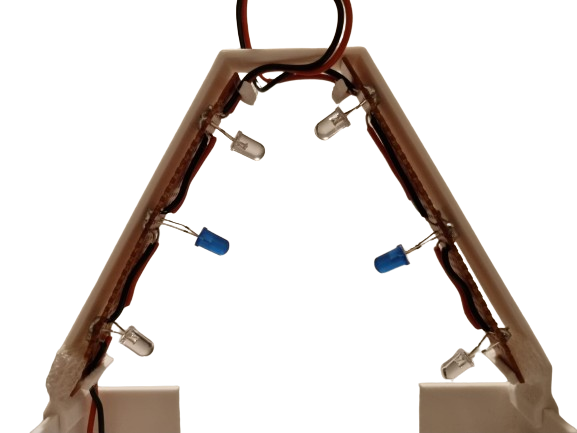
* Gate: 7;
* Drain: Anode of the diode + Negative of the water pump;
* Source: GND.
* NTC Temperature Sensor Breakout (Digital Out) Lm393

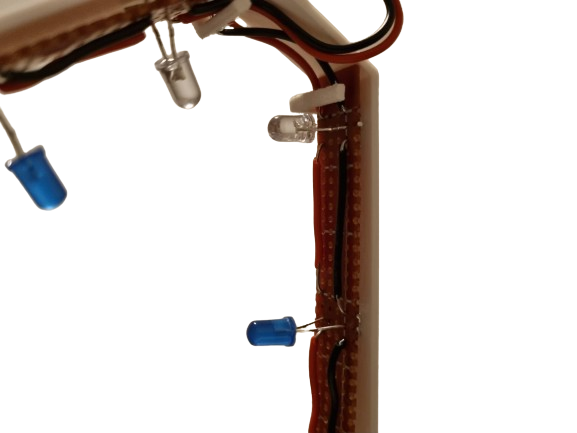


* Vcc: 5V;
* GND: GND;
* DO: PIN 5.
* LDR



* 5V -> LDR -> Resistor of 10 [kΩ] output A3 -> GND;
* Band of LEDs





* Positive: PIN 6;
* GND: GND.
* Diode



* Anode: Drain of the NMOS;
* Cathode: Positive of the water pump.

## Our evolution

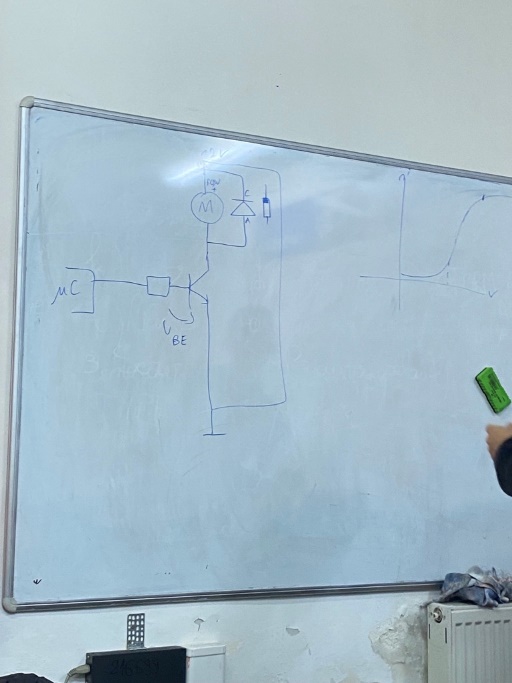
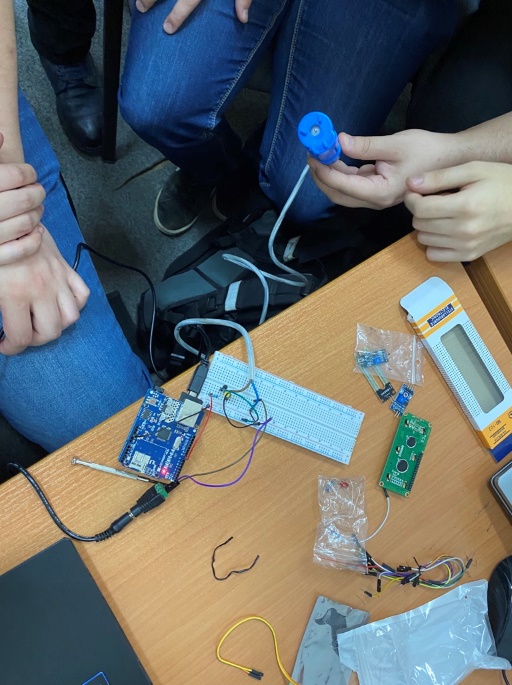


Figure 2. We are trying to figure out what we are doing

Figure 1. Only a scheme, nothing practical yet.

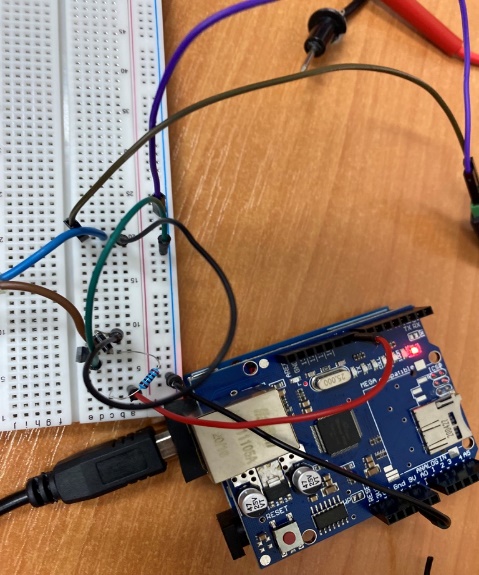


Figure 3. We did the circuit for the water pump, it seemed fine, but we burnt the transistor.

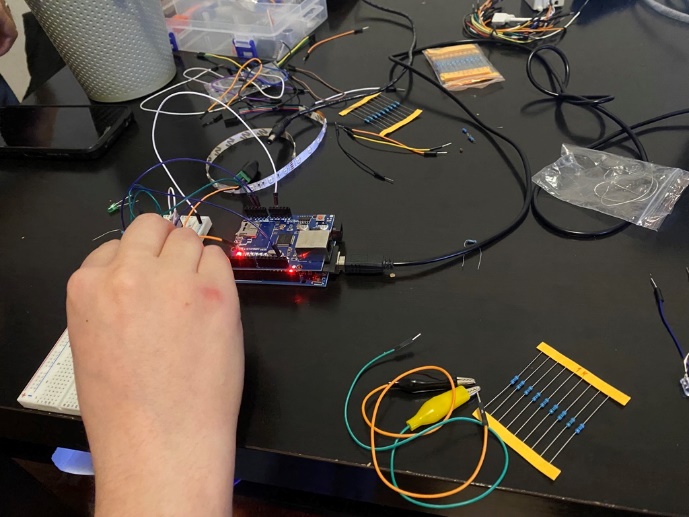


Figure 4. We are doing some tests at Feri’s home. We learn, we code.

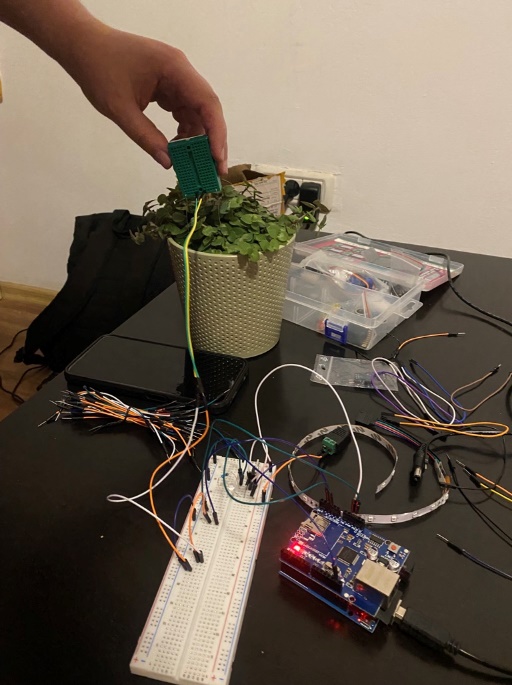
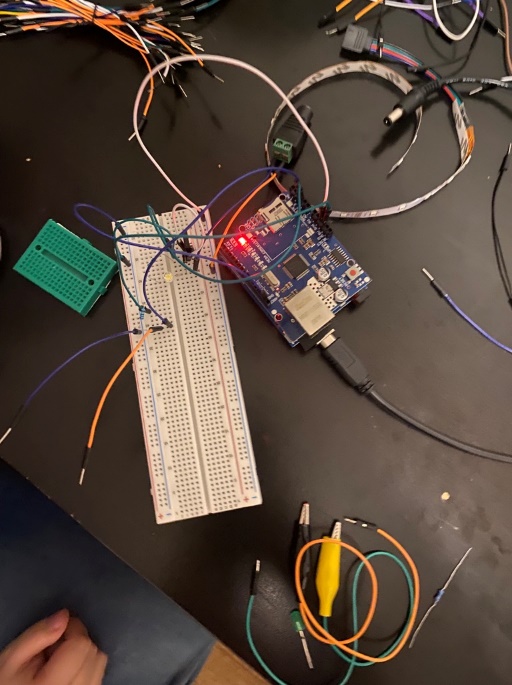


Figure 5. We were happy that it worked like we wanted to, but we had problems with emails, http connection and SMTP. (We solved them later).

Figure 6. We had success in making the “DIY moisture sensor”, but we were so tired and atill made some mistakes



Figure 7. That’s how “the workbench” looked.

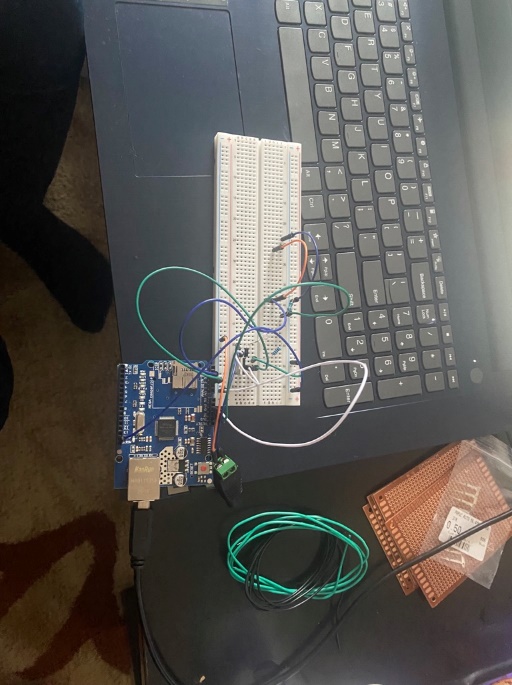
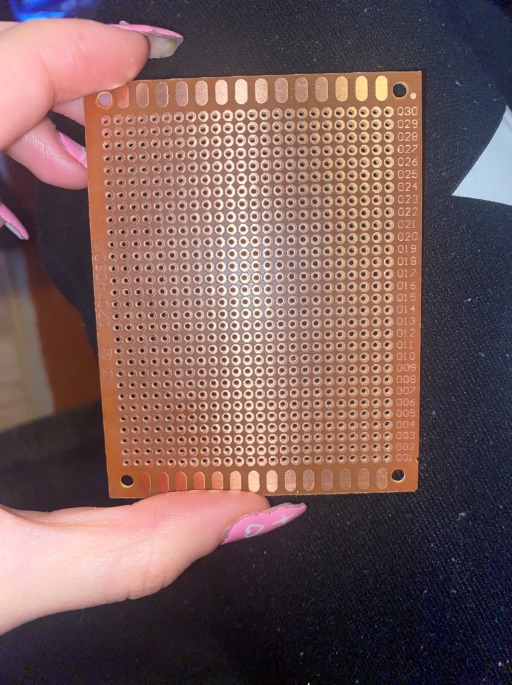


Figure 8. We were still figuring out some things….

Figure 9. We bought some new elements, which helps us to divide the project into smaller parts

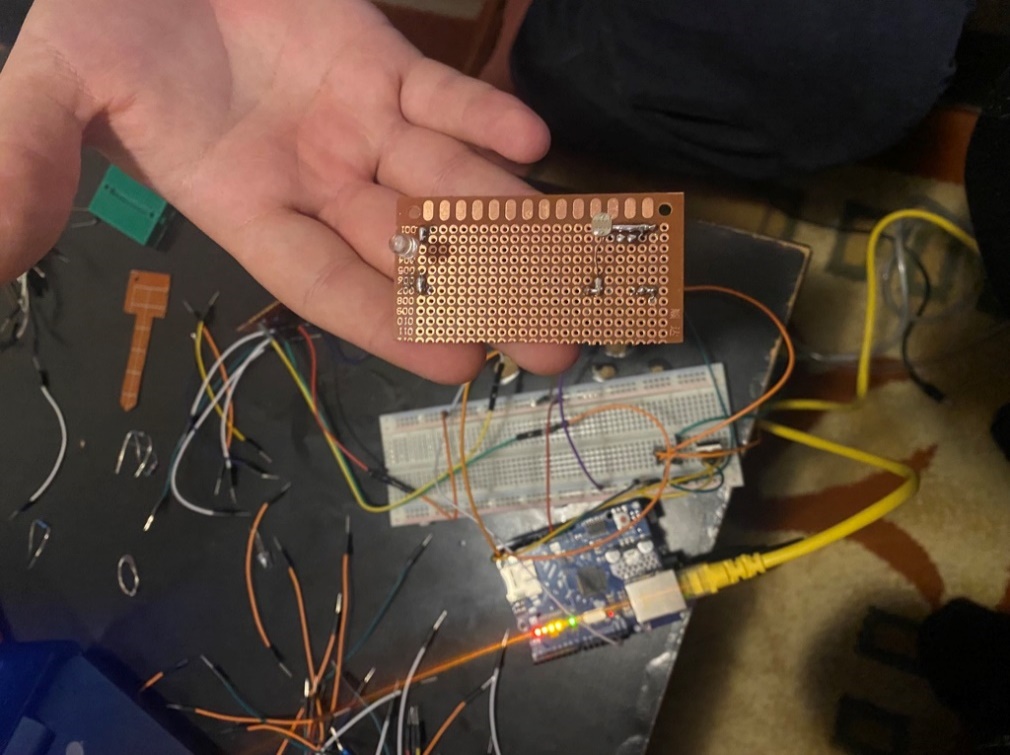


Figure 10. The part with the photoresistor.

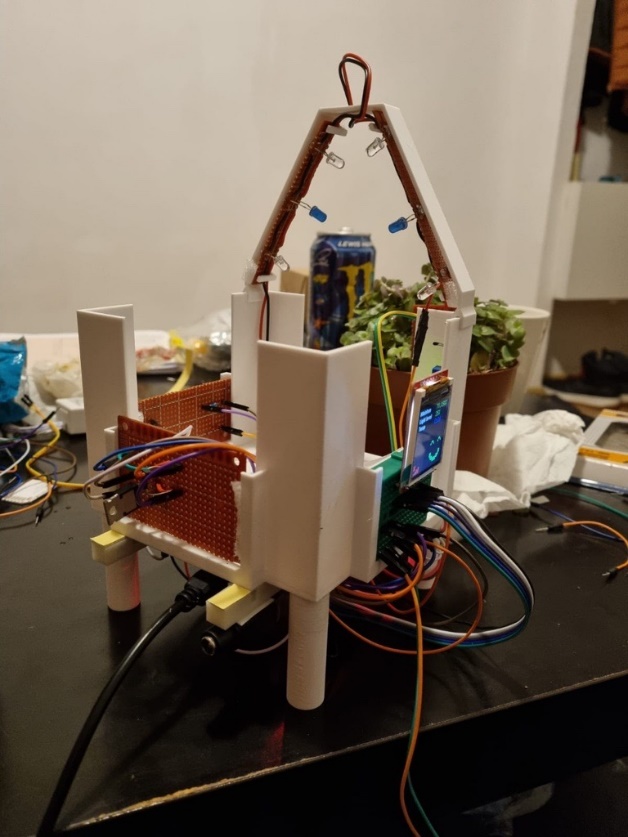




Figure 11. Ta-daaaaa

# Working with registers

## Water Pump and LEDs pins

A black line with black text

Description automatically generated

In AVR microcontrollers, "DDRD" is the Data Direction Register for Port D. In microcontroller programming, the DDRD register is used to set the direction of the individual pins in Port D. Each bit in the DDRD register corresponds to a specific pin in Port D. When a bit is set to 1, it configures the corresponding pin as an output, and when set to 0, it configures the pin as an input.

A screen shot of a computer

Description automatically generated

Our code is setting the bits associated with ‘light\_out’ and ‘pump\_out’ in the ‘DDRD’ register to 1, which configures those pins as outputs.

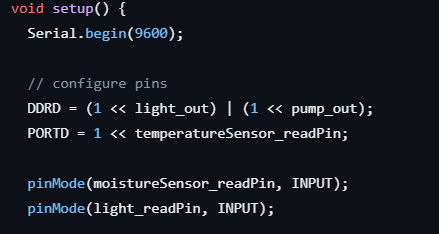
* 1 << light\_out: This expression shifts the value 1 to the left by the number of positions specified by light\_out. It effectively sets the bit at the position indicated by light\_out to 1.
* 1 << pump\_out: Similarly, this expression sets the bit at the position indicated by pump\_out to 1.
* | (bitwise OR): The | operator combines the results of the two expressions using bitwise OR. This means that if either bit (at the position of light\_out or pump\_out) is 1, the corresponding bit in the DDRD register will be set to 1.

## Sensor temperature pins

A black line with white text

Description automatically generated

The PORTD register is often used in microcontroller programming to control the state (high or low) of the individual pins in Port D. The PORTD register, commonly used in microcontroller programming, is part of the I/O (Input/Output) system and plays a crucial role in controlling the state of digital output pins. In the context of AVR microcontrollers (like those used in Arduino), PORTD specifically refers to Port D, which is a collection of digital pins.



Our code is setting a specific bit in the ‘PORTD’ register, based on the value of ‘temperatureSensor\_readPin’.

* temperatureSensor\_readPin: This variable represents the pin number.
* 1 << temperatureSensor\_readPin: This expression shifts the value 1 to the left by the number of positions specified by temperatureSensor\_readPin. In other words, it sets the bit at the position indicated by temperatureSensor\_readPin to 1.
* PORTD = 1 << temperatureSensor\_readPin: This line assigns the result of the above expression to the PORTD register. It means that the specific pin on PORTD corresponding to temperatureSensor\_readPin is set to HIGH (logic level 1), and all other pins remain unchanged.

# Code

The code was divided into 4 files:

* 3 headers;
* 1 main file.

The division was implemented to enhance the clarity and visual organization of the code.

## config.h

#ifndef CONFIG\_H

#define CONFIG\_H

const uint8\_t pump\_out PROGMEM = 7; // gate signal to pump transistor

const uint8\_t light\_out PROGMEM = 6; // power the DIY LED strip

const uint8\_t light\_readPin PROGMEM = A3; // light sensor (photoresistor) input pin

const uint8\_t moistureSensor\_readPin PROGMEM = A4; // moisture sensor (DIY) input pin

const uint8\_t temperatureSensor\_readPin PROGMEM = 5; // digital temperature sensor input pin

const short sensor\_pollRate = 2000; // interval (in ms) at which the sensor values are updated

const uint8\_t moisture\_threshold PROGMEM = 60; // under this limit (in percents [%]), the the pump is turned on

const uint8\_t light\_threshold PROGMEM = 1 5; // under this limit (in percents [%]), the led strip is turned on

const int minEmailInterval = 30000; // time (in ms) after which we try to send an e-mail if there is no water

unsigned long timeSinceLastMail = 0;

unsigned long sensor\_lastPoll = 0;

unsigned long pump\_lastStart = 0;

const short pump\_coolDown = 20000; // cooldown (ms) until the pump can start again (to give the plant time to absorb the water)

const short pump\_activeDuration = 1000; // amount of time (in ms) for which the pump is on before stopping

const uint8\_t mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC address from which to send the mail HTTP request

const char HOST\_NAME[] = "maker.ifttt.com";

const char PATH\_NAME[] = "/trigger/send-email/with/key/heQXcCZGAUOnHDjYqOk7d2zmYVCYitI7pZkzSlRJC-B";

#endif // end config.h

* This configuration file, named "config.h," defines constants and parameters for our project. The key elements include pin assignments for various sensors and actuators, interval settings for sensor polling, threshold values for sensor-triggered actions, and parameters governing the operation of a pump. Additionally, it specifies network-related details such as the MAC address for sending HTTP requests and the server path for triggering email notifications. The file enhances code readability and maintainability by centralizing configuration parameters, promoting a modular structure in the overall code design.

## display.h

#ifndef DISPLAY\_H

#define DISPLAY\_H

#define LAN\_ICON\_H 6

#define LAN\_ICON\_W 8

#define FACE\_ICON\_H 16

#define FACE\_ICON\_W 16

// display object constructor

Ucglib\_ST7735\_18x128x160\_SWSPI ucg(/\*scl=\*/ 8, /\*data=\*/ 9, /\*cd=\*/ 10, /\*cs=\*/12, /\*reset=\*/ 11);

/\*

The icons we display are arrays of booleans

Each digit represents a pixel as follows:

- 0: off (black) pixel

- 1: on: (colored) pixel

We use bools so we don't waste space for each color channel (3 channels \* 4 bytes = 12 bytes per pixle)

We also use a scaled down version and scale it up programatically when rendering it for the same reason as above

\*/

// display an icon showing there is no internet connection

const bool noInternet[LAN\_ICON\_H][LAN\_ICON\_W] = {

{1,0,1,0,0,0,0,0},

{0,1,0,0,0,0,0,0},

{1,0,1,0,0,0,0,1},

{0,0,0,0,0,1,0,1},

{0,0,0,1,0,1,0,1},

{0,1,0,1,0,1,0,1}

};

// happy face used when soil is moist

const bool happyFace[FACE\_ICON\_H][FACE\_ICON\_W] = {

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0},

{0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0},

{0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0},

{0,0,0,1,1,1,0,0,0,0,1,1,1,0,0,0},

{0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0},

{0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

};

// sad face when dry

const bool sadFace[FACE\_ICON\_H][FACE\_ICON\_W] = {

{0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0},

{0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0},

{0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0},

{1,0,0,1,1,1,0,0,0,0,1,1,1,0,0,1},

{0,0,1,1,0,1,1,0,0,1,1,0,1,1,0,0},

{0,0,1,1,0,0,1,0,0,1,1,0,0,1,0,0},

{0,0,1,0,0,1,1,0,0,1,0,0,1,1,0,0},

{0,0,0,1,1,1,0,0,0,0,1,1,1,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0},

{0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0},

{0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0},

{0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

};

/\*\*

\*@brief display a face on the lower right corner of the display

\*@param scale: factor for upscaling the image

\*@param happy: display happyFace if true, sadFace otherwise

\*\*/

void displayFace(int scale, bool happy) {

for (uint8\_t i = 0; i < FACE\_ICON\_H; i++) {

for (uint8\_t j = 0; j < FACE\_ICON\_W; j++) {

if (happy)

ucg.setColor(0, happyFace[i][j] \* 255, 0); // green if happy

else

ucg.setColor(0, sadFace[i][j] \* 255, sadFace[i][j] \* 255); // yellow if sad

ucg.drawBox(j+j\*scale + ucg.getWidth() - 64, i+i\*scale + 64, scale + 1, scale + 1); // scale it by the scale factor and shift it to the left

}

}

}

/\*\*

\*@brief display a small icon in the lower left corner showing that there is no internet connection

\*@param scale: factor for upscaling the image

\*@param enable: show if true, hide if false

\*\*/

void displayNoInternet(uint8\_t scale, bool enable = true) {

for (uint8\_t i = 0; i < LAN\_ICON\_H; i++) {

for (uint8\_t j = 0; j < LAN\_ICON\_W; j++) {

ucg.setColor(0, 0, enable ? noInternet[i][j] \* 255 : 0); // red

ucg.drawBox(j+j\*scale + 4, i+i\*scale + ucg.getHeight() - 20 - 32, scale + 1, scale + 1);

}

}

}

/\*\*

\*@brief display a numeric sensor entry on the screen

\*@param pos: x and y positions

\*@param color: [blue, green, red]

\*@param title: title for the value entry (will be white and bolded)

\*@param value: value of the sensor, its color will be given by the parameter

\*@param unit: a character to be displayed after the value (e.g. '%', 'm', 'Hz' etc.); 0 for none

\*@param toInt: cast the value to integer if true

\*\*/

void displaySensorValue(uint8\_t pos[2], uint8\_t color[3], char\* title, float value, char\* unit = 0, bool toInt = false) {

ucg.setColor(255, 255, 255);

ucg.setPrintPos(pos[0], pos[1]);

ucg.setFont(ucg\_font\_helvB08\_tr);

ucg.print(title);

ucg.setFont(ucg\_font\_7x13\_mr);

ucg.setColor(0, color[0], color[1], color[2]);

ucg.setPrintPos(ucg.getWidth() - 52, pos[1]);

if (toInt) {

ucg.print((int) value);

}

else {

ucg.print(value);

}

if (unit) {

ucg.print(unit);

}

// print some empty space so no stray pixels are left if the number of characters lowers after a refresh

ucg.print(" ");

}

/\*\*

\*@brief display a text-value sensor entry on the screen

\*@param pos: x and y positions

\*@param color: [blue, green, red]

\*@param title: title for the value entry (will be white and bolded)

\*@param status: the status obtained by reading the sensor (e.g. "cold", "warm", "ok", "on", "off" etc.), its color will be given by the parameter

\*\*/

void displaySensorValue(uint8\_t pos[2], uint8\_t color[3], char\* title, char\* status) {

ucg.setColor(255, 255, 255);

ucg.setPrintPos(pos[0], pos[1]);

ucg.setFont(ucg\_font\_helvB08\_tr);

ucg.print(title);

ucg.setFont(ucg\_font\_7x13\_mr);

ucg.setColor(0, color[0], color[1], color[2]);

ucg.setPrintPos(ucg.getWidth() - 52, pos[1]);

ucg.print(status);

}

#endif // end display.h

* The "display.h" file contains definitions and functions related to visual elements for “LCD of 1.44” for STC, STM 32 and Arduino (5V)” display. It includes constants for icon dimensions, instantiation of a display object (Ucglib\_ST7735\_18x128x160\_SWSPI), and arrays representing graphical icons for indicating internet connectivity and emotional states (happy and sad faces). Additionally, it provides functions for displaying these icons on the screen, along with numeric and text-based sensor values in specified positions and formats.

## network.h

#ifndef NET\_H

#define NET\_H

#include "config.h"

EthernetClient client;

bool sendEmail(float moisture, int light, bool isWarm) {

// connect to IFTTT server on port 80:

if (client.connect(HOST\_NAME, 80)) {

// if connected:

Serial.println(F("Connected to server"));

// make a HTTP request:

client.print("GET ");

client.print(PATH\_NAME);

client.print(F("?value1="));

client.print(moisture);

client.print(F("&value2="));

client.print(light);

client.print(F("&value3="));

client.print(isWarm ? "Warm" : "Cold");

client.println(" HTTP/1.1");

client.print("Host: ");

client.println(HOST\_NAME);

client.println("Connection: close");

client.println();

while (client.connected()) {

if (client.available()) {

// read an incoming byte from the server and print it to serial monitor:

char c = client.read();

Serial.print(c);

}

}

// the server disconnected, stop the client:

client.stop();

Serial.println();

Serial.println(F("disconnected"));

return true;

}

else {// if not connected:

Serial.println(F("connection failed"));

return false;

}

}

#endif // end network.h

The "net.h" file handles network-related functionalities for “Botani-Talk” project, utilizing the Ethernet library. It establishes a connection to an IFTTT server to send an email with sensor data. The sendEmail function takes parameters such as moisture level, light intensity, and a boolean indicating whether it's warm or cold. It constructs a GET request with these parameters and sends it to the specified server path.

## main\_sketch.ino

We included the necessary libraries for our work and, also, the header files explained above.

We declared some global variables, used in functions explained later in this presentation.

#include <SPI.h>

#include <Ethernet.h>

#include "Ucglib.h"

#include "config.h"

#include "display.h"

#include "network.h"

// sensor variables

float moisture = 0;

uint8\_t light = -1;

bool isWarm = false;

bool internetAvailable;

### Setup Function

void setup() {

Serial.begin(9600);

// configure pins

DDRD = (1 << light\_out) | (1 << pump\_out);

PORTD = 1 << temperatureSensor\_readPin;

pinMode(moistureSensor\_readPin, INPUT);

pinMode(light\_readPin, INPUT);

// initialize display

ucg.begin(0);

ucg.clearScreen();

// make screen black

ucg.setColor(0,0,0);

ucg.drawBox(4,0, ucg.getWidth() + 4, ucg.getHeight());

ucg.setColor(255, 255, 255);

ucg.setPrintPos(4, 16);

ucg.setFont(ucg\_font\_helvB08\_tr);

ucg.println(F("Connecting to internet..."));

// try to connect to the Internet (timeout = 10s)

internetAvailable = Ethernet.begin(mac, 10000);

ucg.setColor(0,0,0);

ucg.drawBox(0,0, ucg.getWidth() + 4, ucg.getHeight() - LAN\_ICON\_H - 4);

// handle connection failure

if (!internetAvailable) {

Serial.println(F("Failed to obtain an IP address using DHCP"));

displayNoInternet(2, true); // display an icon scaled x2

}

}

* The ‘setup’ function initializes serial communication, configures pins for input and output, initializes the display, and attempts to establish an internet connection using DHCP. It displays a connection message on the screen and handles failures by printing messages and showing a "no internet" icon if necessary.

### Loop function

void loop() {

// get the time in ms since the program started

const unsigned long currentTime = millis();

const uint8\_t text\_height = 14; // how many pixels to go down when printing anouther text row

uint8\_t pos[2] = {6, text\_height};

uint8\_t color[3];

// check the sensor once at every <sensor\_pollrate> ms

if (currentTime - sensor\_lastPoll >= sensor\_pollRate) {

// get percent values for moisture and light level

moisture = ((1023.0 - analogRead(moistureSensor\_readPin)) / 1023.0) \* 100.0; // resistance decreases when wet, so subtract it from max (1023)

light = ((analogRead(light\_readPin)) / 1023.0) \* 100.0; // resistance increases with light

// read temperature sensor from PIND readPin

isWarm = !((PIND & (1 << temperatureSensor\_readPin)) >> temperatureSensor\_readPin); // HIGH = cold; LOW = warm

Serial.print(F("Moisture: "));

Serial.print(moisture);

Serial.print(F("%"));

Serial.print(F(" | Light: "));

Serial.print(light);

Serial.print(F("%"));

Serial.print(F(" | Temp: "));

Serial.println(isWarm ? "Warm" : "Cold");

sensor\_lastPoll = currentTime; // keep track of this sensor polling

color[0] = 0;

color[1] = moisture <= moisture\_threshold ? 0 : 255; // green if moist

color[2] = moisture > moisture\_threshold ? 0 : 255; // red if dry

displaySensorValue(pos, color, "Moisture", moisture, "%");

pos[1] += text\_height;

color[0] = 0;

color[1] = light < light\_threshold ? 0 : 255; // green if well lit

color[2] = light >= light\_threshold ? 0 : 255; // red if dark

displaySensorValue(pos, color, "Light level", light, "%", true);

pos[1] += text\_height;

color[0] = !isWarm ? 255 : 0; // blue if cold

color[1] = isWarm ? 255 : 0; // green if warm

color[2] = 0;

displaySensorValue(pos, color, "Temp", isWarm ? "Warm" : "Cold");

// happy face if moist, sad face if dry

displayFace(3, moisture > moisture\_threshold);

}

// turn on the pump if the soil is dry and the pump is not on cooldown

if (moisture <= moisture\_threshold && currentTime - pump\_lastStart >= pump\_coolDown) {

// digitalWrite(pump\_pin, HIGH);

PORTD = (1 << pump\_out) | PORTD;

pump\_lastStart = currentTime;

Serial.print(F("Pump turned ON at: "));

Serial.println(currentTime);

}

// turn the pump off after pump\_activeDuration has passed

int isPumpOn = (PIND & (1 << pump\_out)) >> pump\_out;

if (isPumpOn && currentTime - pump\_lastStart >= pump\_activeDuration) {

// digitalWrite(pump\_pin, LOW);

PORTD = ~(1 << pump\_out) & PORTD;

Serial.print(F("Pump turned OFF at: "));

Serial.println(currentTime);

}

// turn on the led strip if dark, otherwise turn it off

if (light != -1 && light < light\_threshold) {

// digitalWrite(light\_out, HIGH);

PORTD = (1 << light\_out) | PORTD;

} else {

// digitalWrite(light\_out, LOW);

PORTD = ~(1 << light\_out) & PORTD;

}

// try to send an e-mail if the soil has been dry for a while

if (moisture <= moisture\_threshold && pump\_lastStart - timeSinceLastMail >= minEmailInterval) {

Serial.println(F("Trying to send e-mail!"));

if (!internetAvailable)

internetAvailable = Ethernet.begin(mac, 10000);

// try to send an emai; and if it fails, show an icon telling there is no internet

displayNoInternet(2, !sendEmail(moisture, light, isWarm));

timeSinceLastMail = currentTime;

}

}

* This ‘loop’ function continuously runs, monitoring sensor values, controlling actuators (pump and LED strip), and attempting to send email notifications based on specified conditions.

Parts of execution:

1. Records the current time in milliseconds since the program started.
2. Checks if enough time has passed to poll the sensors. Reads moisture, light, and temperature values, updates the display, and sets color codes based on sensor readings.
3. Turns on the pump if the soil is dry and the cooldown period has passed.
4. Turns off the pump after a specified active duration.
5. Controls the LED strip based on the ambient light conditions.
6. Attempts to send an email notification if the soil has been dry for a while, and updates the time since the last email attempt.

# Working with IFTTT

A screenshot of a computer screen

Description automatically generatedWe created an applet on IFTTT, which receive a web request and send an email via an WEBHOOK.

You can edit your subject and body, and, also change the mail receiver.

A screenshot of a chat

Description automatically generated

# Working with GitHub

A screenshot of a computer

Description automatically generated

Using GitHub for our project provided version control, collaborative development, code review, issue tracking, documentation, continuous integration, access control, community collaboration, availability, and integrations. It enhanced communication, transparency, and efficiency in our team-work.

# Difficulties encountered

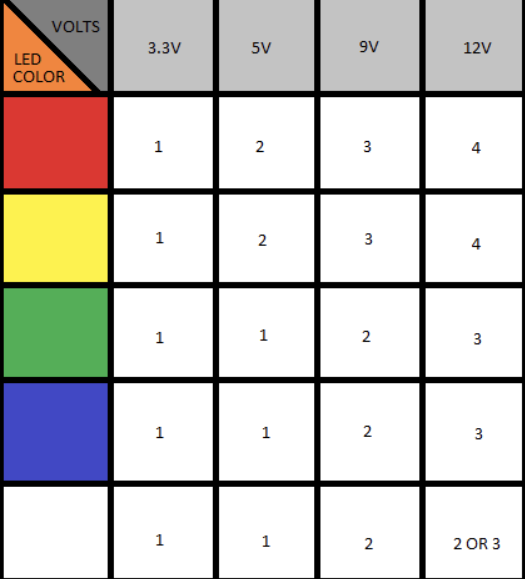
## Memory Optimization

During the development of the software, SRAM management started to become an issue. After we implemented the small face an display, showing if soil was wet or dry, we realized that the bitmaps use too much memory, and when we try to send an e-mail, the URL field will overflow and we get a BAD REQUEST response.

We fixed it by changing some variable types with smaller range (e.g. int -> uint8\_t), saving constant string literals and other constant variables in flash storage instead of memory.

The bulk of memory usage was the pixel bitmaps. For each face we chose a 16×16 bitmap (256 bytes) and upscaled it programmatically instead.

## LED Color Management



In the first place, we put random color LEDs just so we can test if they will light, but we had a mega-surprise. Only the yellow ones lit up, but with the help of this sheet, we figured that the voltage drop of the LEDs is different from color to color. Finally, we put the white and blue LEDs because we use a 5V supply and VOILA, everything was fine in the end.

## Burnt display and transistor

In the process of assembling our electronic circuit on the breadboard, we unfortunately encountered an issue with a bad connection that ultimately resulted in the burning of our display. The problem stemmed from a faulty connection on the breadboard, where either a wire was not securely inserted or a component wasn't properly aligned. This led to an erratic flow of electrical current, causing excessive heat and ultimately damaging the display.

In the course of our project, a critical oversight occurred when we inadvertently substituted a transistor for an IGBT (Insulated Gate Bipolar Transistor). This substitution resulted in the burning of the transistor due to its inability to handle the higher power demands and voltage spikes associated with the IGBT's intended role.

# Bibliography

1. [andreeabrezuica/Botani-Talk (github.com)](https://github.com/andreeabrezuica/Botani-Talk);
2. [Arduino port register manipulation control tutorial (electronoobs.com)](http://electronoobs.com/eng_arduino_tut130.php);
3. [IFTTT - Automate business & home](https://ifttt.com/);
4. [ALLDATASHEET.COM - Electronic Parts Datasheet Search](https://www.alldatasheet.com/);
5. [Curs: Computers Interfaces and Peripherals - III IEC | unitbv.ro - e-Learning 2023-2024](https://elearning.unitbv.ro/course/view.php?id=1931);
6. [Arduino Forum](https://forum.arduino.cc/);
7. [Tinkercad | Create 3D digital designs with online CAD](https://www.tinkercad.com/);
8. [draw.io (drawio.com)](https://www.drawio.com/)

# Finally…

Thank you for listening to us!

Thanks to my amazing team!